FinancePy 0.33

Dominic O'Kane

December 4, 2023

Contents

Chapter 1

Introduction to FinancePy

Latest News and Versions

1. 33 released

- Tidied up key rate code - Fixed unit tests for pytest - Fixed vectorisation of barrier options - Various pep8 fixes

1. 32 released

- Fixed bug in Bond OAS and ASW

1. 31 released

Schedule - Corrected bug in schedule generation - Corrected bug in CDS protection leg integral

Many Bond Classes have been amended - Changed FULL price to DIRTY price in functions UPDATE YOUR CODE PLEASE. APOLS for inconvenience. - Removed face amount from bond class - how much you buy is not intrinsic to a bond - Made number of ex-dividend days a member of bond class - Added adjustment for ex-dividend dates to yield calculations - Revised acccrued and principal functions to take face amount as input - Updated document

1. 30 released

- Added PrettyPrint to required dependencies

1. 22 Nov 22

Version 0.260 has been released and pushed to PyPI - Create Date from python datetime - Zero coupon bond class - Fixed bug in bond payment date

1. 31-Aug-2022

Version 0.240 has just been released and pushed to PyPI with changes - Negative terms in date class - Recovery rates do not default to standard value for CDS curves

DISCLAIMER

This software is distributed FREE AND WITHOUT ANY WARRANTY.

Report any bugs or suggestions here as an issue.

CONTRIBUTORS WANTED !

If you have a knowledge of Quantitative Finance and a reasonable knowledge of Python, then please consider contributing to this project. There are small tasks and big tasks to be done. Just look in the list of Issues and you may find something you can do. Before you begin, please comment in the issue thread in case someone else may be working on that issue. Or you can contact me directly at dominic.okane at edhec.edu.

If you are a user and require some additional functionality, then please add it as an issue.

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 LLVMLite 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 â€“ just use the default â€“ 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.

The Library Design

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

- Utils - These are utility functions used to assist you with modelling a security. These include dates (Date), 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 over 15 years of academic experience.

Contact me at dominic.okane at edhec.edu.

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:

- The code should be Pep8 compliant.
- Comments are required for every class and function and they should be a clear description.
- At least one broad test case and a set of unit tests 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.
- Calendar is a class for determining which dates are not business dates in a specific region or country.
- DayCount is a class for determining accrued interest in bonds and also accrual factors in ISDA swap-like contracts.
- Error is a class which handles errors in the calculations done within FinancePy
- annual_frequency takes in a annual_frequency type and then returns the number of payments per year
- global_vars holds the value of constants used across the whole of FinancePy
- helper_functions is a set of helpful functions that can be used in a number of places
- math 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 generates a sequence of cashflow payment dates in accordance with financial market standards
- 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 day count 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.
- ACT 365 F - 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 amount

Class: Amount

Amount

Create Amount object.

```
Amount(notional: float = ONE_MILLION,
       currency_type: CurrencyTypes = CurrencyTypes.NONE):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
notional	float	-	ONE_MILLION
currency_type	CurrencyTypes	-	CurrencyTypes.NONE

__repr__

Print out the details of the schedule and the actual dates. This can be used for providing transparency on schedule calculations.

```
__repr__():
```

The function arguments are described in the following table.

amount

PLEASE ADD A FUNCTION DESCRIPTION

```
amount():
```

The function arguments are described in the following table.

_print

Print out the details of the schedule and the actual dates. This can be used for providing transparency on schedule calculations.

```
_print():
```

The function arguments are described in the following table.

2.2 calendar

Enumerated Type: BusDayAdjustTypes

This enumerated type has the following values:

- NONE
- FOLLOWING
- MODIFIED_FOLLOWING
- PRECEDING
- MODIFIED_PRECEDING

Enumerated Type: CalendarTypes

This enumerated type has the following values:

- NONE
- WEEKEND
- AUSTRALIA
- CANADA
- FRANCE
- GERMANY
- ITALY
- JAPAN
- NEW_ZEALAND
- NORWAY
- SWEDEN
- SWITZERLAND
- TARGET
- UNITED_STATES
- UNITED_KINGDOM

Enumerated Type: DateGenRuleTypes

This enumerated type has the following values:

- FORWARD
- BACKWARD

Class: Calendar

Class to manage designation of payment dates as holidays according to a regional or country-specific calendar convention specified by the user. It also supplies an adjustment method which takes in an adjustment convention and then applies that to any date that falls on a holiday in the specified calendar.

Calendar

Create a calendar based on a specified calendar type.

```
Calendar(cal_type: CalendarTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cal_type	CalendarTypes	-	-

adjust

Adjust a payment date if it falls on a holiday according to the specified business day convention.

```
adjust(dt: Date,
       bd_type: BusDayAdjustTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-
bd_type	BusDayAdjustTypes	-	-

add_business_days

Returns a new date that is numDays business days after Date. All holidays in the chosen calendar are assumed not business days.

```
add_business_days(start_dt: Date,
                  numDays: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_dt	Date	-	-
numDays	int	-	-

is_business_day

Determines if a date is a business day according to the specified calendar. If it is it returns True, otherwise False.

```
is_business_day(dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-

is_holiday

Determines if a date is a Holiday according to the specified calendar. Weekends are not holidays unless the holiday falls on a weekend date.

```
is_holiday(dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-

holiday_weekend

Weekends by themselves are a holiday.

```
holiday_weekend():
```

The function arguments are described in the following table.

holiday_australia

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_australia():
```

The function arguments are described in the following table.

holiday_united_kingdom

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_united_kingdom():
```

The function arguments are described in the following table.

holiday_france

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_france():
```

The function arguments are described in the following table.

holiday_sweden

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_sweden():
```

The function arguments are described in the following table.

holiday_germany

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_germany():
```

The function arguments are described in the following table.

holiday_switzerland

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_switzerland():
```

The function arguments are described in the following table.

holiday_japan

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_japan():
```

The function arguments are described in the following table.

holiday_new_zealand

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_new_zealand():
```

The function arguments are described in the following table.

holiday_norway

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_norway():
```

The function arguments are described in the following table.

holiday_united_states

Only bank holidays. Weekends by themselves are not a holiday. This is a generic US calendar that contains the superset of holidays for bond markets, NYSE, and public holidays. For each of these and other categories there will be some variations.

```
holiday_united_states():
```

The function arguments are described in the following table.

holiday_canada

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_canada():
```

The function arguments are described in the following table.

holiday_italy

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_italy():
```

The function arguments are described in the following table.

holiday_target

Only bank holidays. Weekends by themselves are not a holiday.

```
holiday_target():
```

The function arguments are described in the following table.

holiday_none

No day is a holiday.

```
holiday_none():
```

The function arguments are described in the following table.

get_holiday_list

generates a list of holidays in a specific year for the specified calendar. Useful for diagnostics.

```
get_holiday_list(year: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
year	float	-	-

easter_monday

Get the day in a given year that is Easter Monday. This is not easy to compute so we rely on a pre-calculated array.

```
easter_monday(year: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
year	float	-	-

__str__

PLEASE ADD A FUNCTION DESCRIPTION

```
__str__():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

2.3 currency

Enumerated Type: CurrencyTypes

This enumerated type has the following values:

- USD
- EUR
- GBP
- CHF
- CAD
- AUD
- NZD
- DKK
- SEK
- HKD
- NONE

2.4 date

Enumerated Type: DateFormatTypes

This enumerated type has the following values:

- BLOOMBERG
- US_SHORT
- US_MEDIUM
- US_LONG
- US_LONGEST
- UK_SHORT
- UK_MEDIUM
- UK_LONG
- UK_LONGEST
- DATETIME

Class: Date()

A date class to manage dates that is simple to use and includes a number of useful date functions used frequently in Finance.

Date

Create a date given a day of month, month and year. The arguments must be in the order of day (of month), month number and then the year. The year must be a 4-digit number greater than or equal to 1900. The user can also supply an hour, minute and second for intraday work. Example Input: start_dt = Date(1, 1, 2018)

```
Date(d, m, y, hh=0, mm=0, ss=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
d	-	-	-
m	-	-	-
y	-	-	-
hh	-	-	0
mm	-	-	0
ss	-	-	0

from_string

Create a Date from a date and format string. Example Input: start_dt = Date('1-1-2018', '

```
from_string(cls, date_string, formatString):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cls	-	-	-
date_string	-	-	-
formatString	-	-	-

from_date

Create a Date from a python datetime.date object or from a Numpy datetime64 object. Example Input: start_dt = Date.from_dt(datetime.date(2022, 11, 8))

```
from_date(cls, date: [datetime.date, np.datetime64]):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cls	-	-	-
date	[datetime.date,np.datetime64]	-	-

_refresh

Update internal representation of date as number of days since the 1st Jan 1900. This is same as Excel convention.

```
_refresh():
```

The function arguments are described in the following table.

__gt__

PLEASE ADD A FUNCTION DESCRIPTION

```
__gt__(other):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
other	-	-	-

__lt__

PLEASE ADD A FUNCTION DESCRIPTION

```
__lt__(other):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
other	-	-	-

__ge__

PLEASE ADD A FUNCTION DESCRIPTION

```
__ge__(other):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
other	-	-	-

__le__

PLEASE ADD A FUNCTION DESCRIPTION

```
__le__(other):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
other	-	-	-

__sub__

PLEASE ADD A FUNCTION DESCRIPTION

```
__sub__(other):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
other	-	-	-

__rsub__

PLEASE ADD A FUNCTION DESCRIPTION

```
__rsub__(other):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
other	-	-	-

__eq__

PLEASE ADD A FUNCTION DESCRIPTION

```
__eq__(other):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
other	-	-	-

is_weekend

returns True if the date falls on a weekend.

```
is_weekend():
```

The function arguments are described in the following table.

is_eom

returns True if this date falls on a month end.

```
is_eom():
```

The function arguments are described in the following table.

eom

returns last date of month of this date.

```
eom():
```

The function arguments are described in the following table.

add_hours

Returns a new date that is h hours after the Date.

```
add_hours(hours):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
hours	-	-	-

add_days

Returns a new date that is numDays after the Date. I also make it possible to go backwards a number of days.

```
add_days(numDays: int = 1):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numDays	int	-	1

add_weekdays

Returns a new date that is numDays working days after Date. Note that only weekends are taken into account. Other Holidays are not. If you want to include regional holidays then use add_business_days from the FinCalendar class.

```
add_weekdays(numDays: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numDays	int	-	-

add_months

Returns a new date that is mm months after the Date. If mm is an integer or float you get back a single date. If mm is a vector you get back a vector of dates.

```
add_months(mm: (list, int)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
mm	list or int	-	-

add_years

Returns a new date that is yy years after the Date. If yy is an integer or float you get back a single date. If yy is a list you get back a vector of dates.

```
add_years(yy: (np.ndarray, float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
yy	np.ndarray or float	-	-

next_cds_date

Returns a CDS date that is mm months after the Date. If no argument is supplied then the next CDS date after today is returned.

```
next_cds_date(mm: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
mm	int	-	0

third_wednesday_of_month

For a specific month and year this returns the day number of the 3rd Wednesday by scanning through dates in the third week.

```
third_wednesday_of_month(m: int,  # Month number
                         y: int):  # Year number
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
m	int	Month number	-
y	int	Year number	-

next_imm_date

This function returns the next IMM date after the current date This is a 3rd Wednesday of Jun, March, Sep or December. For an IMM contract the IMM date is the First Delivery Date of the futures contract.

```
next_imm_date():
```

The function arguments are described in the following table.

add_tenor

Return the date following the Date by a period given by the tenor which is a string consisting of a number and a letter, the letter being d, w, m , y for day, week, month or year. This is case independent. For example 10Y means 10 years while 120m also means 10 years. The date is NOT weekend or holiday calendar adjusted. This must be done AFTERWARDS.

```
add_tenor(tenor: (list, str)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
tenor	list or str	-	-

datetime

Returns a datetime of the date

```
datetime():
```

The function arguments are described in the following table.

str

returns a formatted string of the date

```
str(format):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
format	-	-	-

__repr__

returns a formatted string of the date

```
__repr__():
```

The function arguments are described in the following table.

_print

prints formatted string of the date.

```
_print():
```

The function arguments are described in the following table.

set_date_format

Function that sets the global date format type.

```
set_date_format(format_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
format_type	-	-	-

is_leap_year

Test whether year y is a leap year - if so return True, else False

```
is_leap_year(y: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
y	int	-	-

parse_dt

PLEASE ADD A FUNCTION DESCRIPTION

```
parse_dt(date_str, date_format):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
date_str	-	-	-
date_format	-	-	-

calculate_list

Calculate list of dates so that we can do quick lookup to get the number of dates since 1 Jan 1900 (inclusive) BUT TAKING INTO ACCOUNT THE FACT THAT EXCEL MISTAKENLY CALLS 1900 A LEAP YEAR. For us, agreement with Excel is more important than this leap year error and in any case, we will not usually be calculating day differences with start dates before 28 Feb 1900. Note that Excel inherited this "BUG" from LOTUS 1-2-3.

```
calculate_list():
```

The function arguments are described in the following table.

date_index

PLEASE ADD A FUNCTION DESCRIPTION

```
date_index(d, m, y):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
d	-	-	-
m	-	-	-
y	-	-	-

date_from_index

Reverse mapping from index to date. Take care with numba as it can do weird rounding on the integer. Seems OK now.

```
date_from_index(idx):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
idx	-	-	-

weekday

PLEASE ADD A FUNCTION DESCRIPTION

```
weekday(day_count):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
day_count	-	-	-

vectorisation_helper

PLEASE ADD A FUNCTION DESCRIPTION

```
vectorisation_helper(func):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
func	-	-	-

daily_working_day_schedule

Returns a list of working dates between start_dt and end_dt. This function should be replaced by dateRange once add_tenor allows for working days.

```
daily_working_day_schedule(self,
                           start_dt: Date,
                           end_dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
self	-	-	-
start_dt	Date	-	-
end_dt	Date	-	-

datediff

Calculate the number of days between two Findates.

```
datediff(d1: Date,
         d2: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
d1	Date	-	-
d2	Date	-	-

from_datetime

Construct a Date from a datetime as this is often needed if we receive inputs from other Python objects such as Pandas dataframes.

```
from_datetime(dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-

days_in_month

Get the number of days in the month (1-12) of a given year y.

```
days_in_month(m, y):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
m	-	-	-
y	-	-	-

date_range

Returns a list of dates between start_dt (inclusive) and end_dt (inclusive). The tenor represents the distance between two consecutive dates and is set to daily by default.

```
date_range(start_dt: Date,
           end_dt: Date,
           tenor: str = "1D"):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_dt	Date	-	-
end_dt	Date	-	-
tenor	str	-	”1D”

test_type

PLEASE ADD A FUNCTION DESCRIPTION

```
test_type():
```

The function arguments are described in the following table.

2.5 day_count

Enumerated Type: DayCountTypes

This enumerated type has the following values:

- ZERO
- THIRTY_360_BOND
- THIRTY_E_360
- THIRTY_E_360_ISDA
- THIRTY_E_PLUS_360
- ACT_ACT_ISDA
- ACT_ACT_ICMA
- ACT_365F
- ACT_360
- ACT_365L
- SIMPLE

Class: DayCount

Calculate the fractional day count between two dates according to a specified day count convention.

DayCount

Create Day Count convention by passing in the Day Count Type.

```
DayCount(dccType: DayCountTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dccType	DayCountTypes	-	-

year_frac

This method performs two functions: 1) It calculates the year fraction between dates dt1 and dt2 using the specified day count convention which is useful for calculating year fractions for Libor products whose flows are day count adjusted. In this case we will set dt3 to be None 2) This function is also for calculating bond accrued where dt1 is the last coupon date, dt2 is the settlement date of the bond and date dt3 must be set to the next coupon date. You will also need to provide a coupon frequency for some conventions. Note that if the date is intraday, i.e. hh,mm and ss do not equal zero then that is used in the calculation of the year frac. This avoids discontinuities for short dated intra day products. It should not affect normal dates for which hh=mm=ss=0. This seems like a useful source: https://www.eclipsesoftware.biz/DayCountConventions.html Wikipedia also has a decent survey of the conventions https://en.wikipedia.org/wiki/Day_count_convention and http://data.cbonds.info/files/cbondscalc/Calculator.pdf

```
year_frac(dt1: Date,  # Start of coupon period
          dt2: Date,  # Settlement (for bonds) or period end(swaps)
          dt3: Date = None,  # End of coupon period for accrued
```

```
        freq_type: FrequencyTypes = FrequencyTypes.ANNUAL,
        isTerminationDate: bool = False):  # Is dt2 a termination date
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt1	Date	Start of coupon period	-
dt2	Date	Settlement (for bonds) or period end(swaps)	-
dt3	Date	End of coupon period for accrued	None
freq_type	FrequencyTypes	-	FrequencyTypes.ANNUAL
isTerminationDate	bool	Is dt2 a termination date	False

__repr__

Returns the calendar type as a string.

```
__repr__():
```

The function arguments are described in the following table.

is_last_day_of_feb

PLEASE ADD A FUNCTION DESCRIPTION

```
is_last_day_of_feb(dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-

2.6 distribution

Class: FinDistribution()

FinDistribution

Initialise FinDistribution with x values and associated vector of density times dx values.

```
FinDistribution(x, y):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
y	-	-	-

sum

This should equal 1.0 for the entire distribution.

```
sum():
```

The function arguments are described in the following table.

2.7 error

Class: FinError(Exception)

Simple error class specific to FinPy. Need to decide how to handle FinancePy errors. Work in progress.

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	-	-

_print

PLEASE ADD A FUNCTION DESCRIPTION

```
_print():
```

The function arguments are described in the following table.

_hide_traceback

PLEASE ADD A FUNCTION DESCRIPTION

```
_hide_traceback(exc_tuple=None, filename=None, tb_offset=None,
                exception_only=False, running_compiled_code=False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
exc_tuple	-	-	None
filename	-	-	None
tb_offset	-	-	None
exception_only	-	-	False
running_compiled_code	-	-	False

func_name

PLEASE ADD A FUNCTION DESCRIPTION

```
func_name():
```

The function arguments are described in the following table.

suppress_traceback

PLEASE ADD A FUNCTION DESCRIPTION

```
suppress_traceback():
```

The function arguments are described in the following table.

2.8 frequency

Enumerated Type: FrequencyTypes

This enumerated type has the following values:

- ZERO
- SIMPLE
- ANNUAL
- SEMI_ANNUAL
- TRI_ANNUAL
- QUARTERLY
- MONTHLY
- CONTINUOUS

annual_frequency

This is a function that takes in a Frequency Type and returns a float value for the number of times a year a payment occurs.

```
annual_frequency(freq_type: FrequencyTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
freq_type	FrequencyTypes	-	-

2.9 global_types

Enumerated Type: FinLongShort

This enumerated type has the following values:

- LONG
- SHORT

Enumerated Type: OptionTypes

This enumerated type has the following values:

- EUROPEAN_CALL
- EUROPEAN_PUT
- AMERICAN_CALL
- AMERICAN_PUT
- DIGITAL_CALL
- DIGITAL_PUT
- ASIAN_CALL
- ASIAN_PUT
- COMPOUND_CALL
- COMPOUND_PUT

Enumerated Type: EquityBarrierTypes

This enumerated type has the following values:

- DOWN_AND_OUT_CALL
- DOWN_AND_IN_CALL
- UP_AND_OUT_CALL
- UP_AND_IN_CALL
- UP_AND_OUT_PUT
- UP_AND_IN_PUT
- DOWN_AND_OUT_PUT
- DOWN_AND_IN_PUT

Enumerated Type: FinCapFloorTypes

This enumerated type has the following values:

- CAP
- FLOOR

Enumerated Type: SwapTypes

This enumerated type has the following values:

- PAY
- RECEIVE

Enumerated Type: ReturnTypes

This enumerated type has the following values:

- TOTAL_RETURN
- PRICE_RETURN

Enumerated Type: FinExerciseTypes

This enumerated type has the following values:

- EUROPEAN
- BERMUDAN
- AMERICAN

Enumerated Type: FinSolverTypes

This enumerated type has the following values:

- CONJUGATE_GRADIENT
- NELDER_MEAD
- NELDER_MEAD_NUMBA

Enumerated Type: TouchOptionTypes

This enumerated type has the following values:

- DOWN_AND_IN_CASH_AT_HIT
- UP_AND_IN_CASH_AT_HIT
- DOWN_AND_IN_CASH_AT_EXPIRY
- UP_AND_IN_CASH_AT_EXPIRY
- DOWN_AND_OUT_CASH_OR_NOTHING
- UP_AND_OUT_CASH_OR_NOTHING
- DOWN_AND_IN_ASSET_AT_HIT
- UP_AND_IN_ASSET_AT_HIT
- DOWN_AND_IN_ASSET_AT_EXPIRY
- UP_AND_IN_ASSET_AT_EXPIRY
- DOWN_AND_OUT_ASSET_OR_NOTHING
- UP_AND_OUT_ASSET_OR_NOTHING

2.10 global_vars

2.11 helpers

_func_name

Extract calling function name - using a protected method is not that advisable but calling inspect.stack is so slow it must be avoided.

```
_func_name():
```

The function arguments are described in the following table.

grid_index

PLEASE ADD A FUNCTION DESCRIPTION

```
grid_index(t, grid_times):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
grid_times	-	-	-

beta_vector_to_corr_matrix

Convert a one-factor vector of factor weights to a square correlation matrix.

```
beta_vector_to_corr_matrix(betas):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
betas	-	-	-

pv01_times

Calculate a bond style pv01 by calculating remaining coupon times for a bond with t years to maturity and a coupon frequency of f. The order of the list is reverse time order - it starts with the last coupon date and ends with the first coupon date.

```
pv01_times(t: float,
           f: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	float	-	-
f	float	-	-

times_from_dates

If a single date is passed in then return the year from valuation date but if a whole vector of dates is passed in then convert to a vector of times from the valuation date. The output is always a numpy vector of times which has only one element if the input is only one date.

```
times_from_dates(dt: (Date, list),
                 value_dt: Date,
                 day_count_type: DayCountTypes = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date or list	-	-
value_dt	Date	-	-
day_count_type	DayCountTypes	-	None

check_vector_differences

Compare two vectors elementwise to see if they are more different than tolerance.

```
check_vector_differences(x: np.ndarray,
                         y: np.ndarray,
                         tol: float = 1e-6):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	np.ndarray	-	-
y	np.ndarray	-	-
tol	float	-	1e-6

check_date

Check that input d is a Date.

```
check_date(d: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
d	Date	-	-

dump

PLEASE ADD A FUNCTION DESCRIPTION

```
dump(obj):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
obj	-	-	-

print_tree

Function that prints a binomial or trinonial tree to screen for the purpose of debugging.

```
print_tree(array: np.ndarray,
           depth: int = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
array	np.ndarray	-	-
depth	int	-	None

input_time

Validates a time input in relation to a curve. If it is a float then it returns a float as long as it is positive. If it is a Date then it converts it to a float. If it is a Numpy array then it returns the array as long as it is all positive.

```
input_time(dt: Date,
           curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-
curve	-	-	-

listdiff

Calculate a vector of differences between two equal sized vectors.

```
listdiff(a: np.ndarray,
         b: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	np.ndarray	-	-
b	np.ndarray	-	-

dotproduct

Fast calculation of dot product using Numba.

```
dotproduct(xVector: np.ndarray,
           yVector: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
xVector	np.ndarray	-	-
yVector	np.ndarray	-	-

frange

fast range function that takes start value, stop value and step.

```
frange(start: int,
       stop: int,
       step: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start	int	-	-
stop	int	-	-
step	int	-	-

normalise_weights

Normalise a vector of weights so that they sum up to 1.0.

```
normalise_weights(wt_vector: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
wt_vector	np.ndarray	-	-

label_to_string

Format label/value pairs for a unified formatting.

```
label_to_string(label: str,
                value: (float, str),
                separator: str = "\\n",
                list_format: bool = False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
label	str	-	-
value	float or str	-	-
separator n”	str	-	”
list_format	bool	-	False

table_to_string

Format a 2D array into a table-like string.

```
table_to_string(header: str,
                value_table,
                float_precision="10.7f"):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
header	str	-	-
value_table	-	-	-
float_precision	-	-	”10.7f”

format_table

Format a 2D array into a table-like string. Similar to ”table_to_string”, but using a wrapper around PrettyTable to get a nice formatting.

```
format_table(header: (list, tuple),
             rows: (list, tuple)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
header	list or tuple	-	-
rows	list or tuple	-	-

to_usable_type

Convert a type such that it can be used with ‘isinstance‘

```
to_usable_type(t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-

uniform_to_default_time

Fast mapping of a uniform random variable to a default time given a survival probability curve.

```
uniform_to_default_time(u, t, v):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
u	-	-	-
t	-	-	-
v	-	-	-

accrued_tree

Fast calulation of accrued interest using an Actual/Actual type of convention. This does not calculate according to other conventions.

```
accrued_tree(grid_times: np.ndarray,
             grid_flows: np.ndarray,
             face: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
grid_times	np.ndarray	-	-
grid_flows	np.ndarray	-	-
face	float	-	-

check_argument_types

Check that all values passed into a function are of the same type as the function annotations. If a value has not been annotated, it will not be checked.

```
check_argument_types(func, values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
func	-	-	-
values	-	-	-

2.12 latex

convertToLatexTable

PLEASE ADD A FUNCTION DESCRIPTION

```
convertToLatexTable(txt, sep="␣", header_list=[]):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
txt	-	-	-
sep	-	-	” ”
header_list	-	-	[]

2.13 math

accrued_interpolator

Fast calculation of accrued interest using an Actual/Actual type of convention. This does not calculate according to other conventions.

```
accrued_interpolator(tset: float,  # Settlement time in years
                     cpn_times: np.ndarray,
                     cpn_amounts: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
tset	float	Settlement time in years	-
cpn_times	np.ndarray	-	-
cpn_amounts	np.ndarray	-	-

is_leap_year

Test whether year y is a leap year - if so return True, else False

```
is_leap_year(y: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
y	int	-	-

scale

Scale all of the elements of an array by the same amount factor.

```
scale(x: np.ndarray,
      factor: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	np.ndarray	-	-
factor	float	-	-

test_monotonicity

Check that an array of doubles is monotonic and strictly increasing.

```
test_monotonicity(x: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	np.ndarray	-	-

test_range

Check that all of the values of an array fall between a lower and upper bound.

```
test_range(x: np.ndarray,
           lower: float,
           upper: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	np.ndarray	-	-
lower	float	-	-
upper	float	-	-

maximum

Determine the array in which each element is the maximum of the corresponding element in two equally length arrays a and b.

```
maximum(a: np.ndarray,
        b: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	np.ndarray	-	-
b	np.ndarray	-	-

maxaxis

Perform a search for the vector of maximum values over an axis of a 2D Numpy Array

```
maxaxis(s: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	np.ndarray	-	-

minaxis

Perform a search for the vector of minimum values over an axis of a 2D Numpy Array

```
minaxis(s: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	np.ndarray	-	-

covar

Calculate the Covariance of two arrays of numbers. TODO: check that this works well for Numpy Arrays and add NUMBA function signature to code. Do test of timings against Numpy.

```
covar(a: np.ndarray,
      b: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	np.ndarray	-	-
b	np.ndarray	-	-

pair_gcd

Determine the Greatest Common Divisor of two integers using Euclid's algorithm. TODO - compare this with math.gcd(a,b) for speed. Also examine to see if I should not be declaring inputs as integers for NUMBA.

```
pair_gcd(v1: float,
         v2: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
v1	float	-	-
v2	float	-	-

nprime

Calculate the first derivative of the Cumulative Normal CDF which is simply the PDF of the Normal Distribution

```
nprime(x: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	float	-	-

heaviside

Calculate the Heaviside function for x

```
heaviside(x: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	float	-	-

frange

Calculate a range of values from start in steps of size step. Ends as soon as the value equals or exceeds stop.

```
frange(start: int,
       stop: int,
       step: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start	int	-	-
stop	int	-	-
step	int	-	-

normpdf

Calculate the probability density function for a Gaussian (Normal) function at value x

```
normpdf(x: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	float	-	-

N

Fast Normal CDF function based on Hull OFAODS 4th Edition Page 252. This function is accurate to 6 decimal places.

```
N(x):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-

n_vect

PLEASE ADD A FUNCTION DESCRIPTION

```
n_vect(x):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-

n_prime_vect

PLEASE ADD A FUNCTION DESCRIPTION

```
n_prime_vect(x):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-

normcdf_integrate

Calculation of Normal Distribution CDF by simple integration which can become exact in the limit of the number of steps tending towards infinity. This function is used for checking as it is slow since the number of integration steps is currently hardcoded to 10,000.

```
normcdf_integrate(x: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	float	-	-

normcdf_slow

Calculation of Normal Distribution CDF accurate to 1d-15. This method is faster than integration but slower than other approximations. Reference: J.L. Schonfelder, Math Comp 32(1978), pp 1232-1240.

```
normcdf_slow(z: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
z	float	-	-

phi3

Bivariate Normal CDF function to upper limits $b1$ and $b2$ which uses integration to perform the inner-most integral. This may need further refinement to ensure it is optimal as the current range of integration is from -7 and the integration steps are dx = 0.001. This may be excessive.

```
phi3(b1: float,
     b2: float,
     b3: float,
     r12: float,
     r13: float,
     r23: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
b1	float	-	-
b2	float	-	-
b3	float	-	-
r12	float	-	-
r13	float	-	-
r23	float	-	-

norminvcdf

This algorithm computes the inverse Normal CDF and is based on the algorithm found at (http:#home.online.no/ pjacklam/notes/invnorm/) which is by John Herrero (3-Jan-03)

```
norminvcdf(p):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
p	-	-	-

M

PLEASE ADD A FUNCTION DESCRIPTION

```
M(a, b, c):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
b	-	-	-
c	-	-	-

phi2

Drezner and Wesolowsky implementation of bi-variate normal

```
phi2(h1, hk, r):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
h1	-	-	-
hk	-	-	-
r	-	-	-

cholesky

Numba-compliant wrapper around Numpy cholesky function.

```
cholesky(rho):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rho	-	-	-

corr_matrix_generator

Utility function to generate a full rank n x n correlation matrix with a flat correlation structure and value rho.

```
corr_matrix_generator(rho, n):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rho	-	-	-
n	-	-	-

npv

Function to calculate the npv given irr and cashflow. It can be used to do root search in IRR. times_cfs is a list of tuples. The tuple is in the form of (years from first date, cashflow)

```
npv(irr: float, times_cfs: list):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
irr	float	-	-
times_cfs	list	-	-

band_matrix_multiplication

PLEASE ADD A FUNCTION DESCRIPTION

```
band_matrix_multiplication(A, m1, m2, b):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
A	-	-	-
m1	-	-	-
m2	-	-	-
b	-	-	-

solve_tridiagonal_matrix

Solve A u = r for vector u when A is tridiagonal The matrix A is split into vectors a, b, and c contain the three non-zero elements of each row of A, in order. i.e. the vector b is the main diagonal of A, with a and c the elements either side of the main diagonal. Note that a[0] and c[-1] are not used, and so can be any value.

```
solve_tridiagonal_matrix(A, r):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
A	-	-	-
r	-	-	-

transpose_tridiagonal_matrix

PLEASE ADD A FUNCTION DESCRIPTION

```
transpose_tridiagonal_matrix(A):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
A	-	-	-

2.14 polyfit

_coeff_mat

PLEASE ADD A FUNCTION DESCRIPTION

```
_coeff_mat(x, deg):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
deg	-	-	-

_fit_x

PLEASE ADD A FUNCTION DESCRIPTION

```
_fit_x(a, b):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
b	-	-	-

fit_poly

PLEASE ADD A FUNCTION DESCRIPTION

```
fit_poly(x, y, deg):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
y	-	-	-
deg	-	-	-

eval_polynomial

Compute polynomial P(x) where P is a vector of coefficients, highest order coefficient at P[0]. Uses Horner's Method.

```
eval_polynomial(P, x):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
P	-	-	-
x	-	-	-

2.15 schedule

Class: Schedule

A schedule is a set of dates generated according to ISDA standard rules which starts on the next date after the effective date and runs up to a termination date. Dates are adjusted to a provided calendar. The zeroth element is the previous coupon date (PCD) and the first element is the Next Coupon Date (NCD). We reference ISDA 2006.

Schedule

*Create Schedule object which calculates a sequence of dates following the ISDA convention for fixed income products, mainly swaps. If the date gen rule type is FORWARD we get the unadjusted dates by stepping forward from the effective date in steps of months determined by the period tenor - i.e. the number of months between payments. We stop before we go past the termination date. If the date gen rule type is BACKWARD we get the unadjusted dates by stepping backward from the termination date in steps of months determined by the period tenor - i.e. the number of months between payments. We stop before we go past the effective date. - If the EOM flag is false, and the start date is on the 31st then the the unadjusted dates will fall on the 30 if a 30 is a previous date. - If the EOM flag is false and the start date is 28 Feb then all unadjusted dates will fall on the 28th. - If the EOM flag is false and the start date is 28 Feb then all unadjusted dates will fall on their respective EOM. We then adjust all of the flow dates if they fall on a weekend or holiday according to the calendar specified. These dates are adjusted in accordance with the business date adjustment. The effective date is never adjusted as it is not a payment date. The termination date is not automatically business day adjusted in a swap - assuming it is a holiday date. This must be explicitly stated in the trade confirm. However, it is adjusted in a CDS contract as standard. Inputs first_dt and next_to_last_dt are for managing long payment stubs at the start and end of the swap but *have not yet been implemented*. All stubs are currently short, either at the start or end of swap.*

```
Schedule(effective_dt: Date,  # Also known as the start date
         # This is UNADJUSTED (set flag to adjust it)
         termination_dt: Date,
         freq_type: FrequencyTypes = FrequencyTypes.ANNUAL,
         cal_type: CalendarTypes = CalendarTypes.WEEKEND,
         bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
         dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD,
         adjust_termination_dt: bool = True,  # Default is to adjust
         end_of_month: bool = False,  # All flow dates are EOM if True
         first_dt=None,  # First coupon date
         next_to_last_dt=None):  # Penultimate coupon date
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	Also known as the start date	-
termination_dt	Date	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.ANNUAL
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD
adjust_termination_dt	bool	Default is to adjust	True
end_of_month	bool	All flow dates are EOM if True	False
first_dt	-	First coupon date	None
next_to_last_dt	-	Penultimate coupon date	None

schedule_dts

Returns a list of the schedule of Dates.

```
schedule_dts():
```

The function arguments are described in the following table.

_generate

Generate schedule of dates according to specified date generation rules and also adjust these dates for holidays according to the specified business day convention and the specified calendar.

```
_generate():
```

The function arguments are described in the following table.

__repr__

Print out the details of the schedule and the actual dates. This can be used for providing transparency on schedule calculations.

```
__repr__():
```

The function arguments are described in the following table.

_print

Print out the details of the schedule and the actual dates. This can be used for providing transparency on schedule calculations.

```
_print():
```

The function arguments are described in the following table.

2.16 singleton

Class: Singleton(type)

Singleton type which is used to ensure there is only one instance across the whole project.

__call__

PLEASE ADD A FUNCTION DESCRIPTION

```
__call__(cls, *args, **kwargs):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cls	-	-	-
*args	-	-	-
**kwargs	-	-	-

2.17 solver_1d

_results

rSelect from a tuple of(root, funccalls, iterations, flag)

```
_results(r):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r	-	-	-

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...), where a,b,c... 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

PLEASE ADD A FUNCTION DESCRIPTION

```
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

PLEASE ADD A FUNCTION DESCRIPTION

```
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_powel

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 minimium 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

2.18 solver_cg

Class: OptimizeResult(dict)

Represents the optimization result. Attributes ——— x : ndarray The solution of the optimization. success : bool Whether or not the optimizer exited successfully. status : int Termination status of the optimizer. Its value depends on the underlying solver. Refer to 'message' for details. message : str Description of the cause of the termination. fun, jac, hess: ndarray Values of objective function, its Jacobian and its Hessian (if available). The Hessians may be approximations, see the documentation of the function in question. hess_inv : object Inverse of the objective functions Hessian; may be an approximation. Not available for all solvers. The type of this attribute may be either np.ndarray or scipy.sparse.linalg.LinearOperator. nfev, njev, nhev : int Number of evaluations of the objective functions and of its Jacobian and Hessian. nit : int Number of iterations performed by the optimizer. maxcv : float The maximum constraint violation. Notes —– There may be additional attributes not listed above depending of the specific solver. Since this class is essentially a subclass of dict with attribute accessors, one can see which attributes are available using the 'keys()' method.

__getattr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__getattr__(name):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
name	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

__dir__

PLEASE ADD A FUNCTION DESCRIPTION

```
__dir__():
```

The function arguments are described in the following table.

polak_ribiere_powell_step

PLEASE ADD A FUNCTION DESCRIPTION

```
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

PLEASE ADD A FUNCTION DESCRIPTION

```
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

PLEASE ADD A FUNCTION DESCRIPTION

```
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

_check_unknown_options

PLEASE ADD A FUNCTION DESCRIPTION

```
_check_unknown_options(unknown_options):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
unknown_options	-	-	-

_prepare_scalar_function

*Creates a ScalarFunction object for use with scalar minimizers (BFGS/LBFGSB/SLSQP/TNC/CG/etc). Parameters ———- fun : callable The objective function to be minimized. “fun(x, *args) -¿ float“ where “x“ is an 1-D array with shape (n,) and “args“ is a tuple of the fixed parameters needed to completely specify the function. x0 : ndarray, shape (n,) Initial guess. Array of real elements of size (n,), where ’n’ is the number of independent variables. jac : callable, ’2-point’, ’3-point’, ’cs’, None, optional Method for computing the gradient vector. If it is a callable, it should be a function that returns the gradient vector: “jac(x, *args) -¿ array_like, shape (n,)“ If one of ‘’2-point’, ’3-point’, ’cs’‘ is selected then the gradient is calculated with a relative step for finite differences. If ‘None‘, then two-point finite differences with an absolute step is used. args : tuple, optional Extra arguments passed to the objective function and its derivatives (‘fun‘, ‘jac‘ functions). bounds : sequence, optional Bounds on variables. ’new-style’ bounds are required. eps : float or ndarray If ‘jac is None‘ the absolute step size used for numerical approximation of the jacobian via forward differences. finite_diff_rel_step : None or array_like, optional If ‘jac in [’2-point’, ’3-point’, ’cs’]‘ the relative step size to use for numerical approximation of the jacobian. The absolute step size is computed as “h = rel_step * sign(x0) * max(1, abs(x0))“, possibly adjusted to fit into the bounds. For “method=’3-point’“ the sign of ‘h‘ is ignored. If None (default) then step is selected automatically. hess : callable, ’2-point’, ’3-point’, ’cs’, None Computes the Hessian matrix. If it is callable, it should return the Hessian matrix: “hess(x, *args) -¿ LinearOperator, spmatrix, array, (n, n)“ Alternatively, the keywords ’2-point’, ’3-point’, ’cs’ select a finite difference scheme for numerical estimation. Whenever the gradient is estimated via finite-differences, the Hessian cannot be estimated with options ’2-point’, ’3-point’, ’cs’ and needs to be estimated using one of the quasi-Newton strategies. Returns ——- sf : ScalarFunction*

```
_prepare_scalar_function(fun, x0, jac=None, args=(), bounds=None,
                         epsilon=None, finite_diff_rel_step=None,
                         hess=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fun	-	-	-
x0	-	-	-
jac	-	-	None
args	-	-	()
bounds	-	-	None
epsilon	-	-	None
finite_diff_rel_step	-	-	None
hess	-	-	None

vecnorm

PLEASE ADD A FUNCTION DESCRIPTION

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

The function arguments are described in the following table.

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

_line_search_wolfe12

Same as line_search_wolfe1, but fall back to line_search_wolfe2 if suitable step length is not found, and raise an exception if a suitable step length is not found. Raises —— _LineSearchError If no suitable step size is found

```
_line_search_wolfe12(f, fprime, xk, pk, gfk, old_fval, old_old_fval,
                     **kwargs):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
f	-	-	-
fprime	-	-	-
xk	-	-	-
pk	-	-	-
gfk	-	-	-
old_fval	-	-	-
old_old_fval	-	-	-
**kwargs	-	-	-

_minimize_cg

Minimization of scalar function of one or more variables using the conjugate gradient algorithm. Options ——- disp : bool Set to True to print convergence messages. maxiter : int Maximum number of iterations *to perform. gtol : float Gradient norm must be less than 'gtol' before successful termination. norm : float Order of norm (Inf is max, -Inf is min). eps : float or ndarray If 'jac is None' the absolute step size used for numerical approximation of the jacobian via forward differences. return_all : bool, optional Set to True to return a list of the best solution at each of the iterations. finite_diff_rel_step : None or array_like, optional If 'jac in ['2-point', '3-point', 'cs']' the relative step size to use for numerical approximation of the jacobian. The absolute step size is computed as "h = rel_step * sign(x0) * max(1, abs(x0))", possibly adjusted to fit into the bounds. For "method='3-point'" the sign of 'h' is ignored. If None (default) then step is selected automatically.*

```
_minimize_cg(fun, x0, args=(), jac=None, callback=None,
             gtol=1e-5, norm=Inf, eps=_epsilon, maxiter=None,
             disp=False, return_all=False, finite_diff_rel_step=None,
             **unknown_options):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fun	-	-	-
x0	-	-	-
args	-	-	()
jac	-	-	None
callback	-	-	None
gtol	-	-	1e-5
norm	-	-	Inf
eps	-	-	_epsilon
maxiter	-	-	None
disp	-	-	False
return_all	-	-	False
finite_diff_rel_step	-	-	None
**unknown_options	-	-	-

2.19 solver_nm

nelder_mead

PLEASE ADD A FUNCTION DESCRIPTION

```
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

_initialize_simplex

Generates an initial simplex for the Nelder-Mead method. JIT-compiled in ‘nopython‘ mode using Numba. Parameters ——— x0 : ndarray(float, ndim=1) Initial guess. Array of real elements of size (n,), where ‘n’ is the number of independent variables. bounds: ndarray(float, ndim=2) Sequence of (min, max) pairs for each element in x0. Returns ——— vertices : ndarray(float, ndim=2) Initial simplex with shape (n+1, n).

```
_initialize_simplex(x0, bounds):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x0	-	-	-
bounds	-	-	-

_check_params

Checks whether the parameters for the Nelder-Mead algorithm are valid. JIT-compiled in ‘nopython‘ mode using Numba. Parameters ——— rho : scalar(float) Reflection parameter. Must be strictly greater than 0. chi : scalar(float) Expansion parameter. Must be strictly greater than max(1, roh). v : scalar(float) Contraction parameter. Must be stricly between 0 and 1. sigma : scalar(float) Shrinkage parameter. Must be strictly between 0 and 1. bounds: ndarray(float, ndim=2) Sequence of (min, max) pairs for each element in x. n : scalar(int) Number of independent variables.

```
_check_params(rho, chi, v, sigma, bounds, n):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rho	-	-	-
chi	-	-	-
v	-	-	-
sigma	-	-	-
bounds	-	-	-
n	-	-	-

_check_bounds

Checks whether ‘x‘ is within ‘bounds‘. JIT-compiled in ‘nopython‘ mode using Numba. Parameters ——-—- x : ndarray(float, ndim=1) 1-D array with shape (n,) of independent variables. bounds: ndarray(float, ndim=2) Sequence of (min, max) pairs for each element in x. Returns ———- bool ‘True‘ if ‘x‘ is within ‘bounds‘, ‘False‘ otherwise.

```
_check_bounds(x, bounds):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
bounds	-	-	-

2.20 stats

mean

Calculate the arithmetic mean of a vector of numbers x.

```
mean(x: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	float	-	-

stdev

Calculate the standard deviation of a vector of numbers x.

```
stdev(x: ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	ndarray	-	-

stderr

Calculate the standard error estimate of a vector of numbers x.

```
stderr(x: ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	ndarray	-	-

var

Calculate the variance of a vector of numbers x.

```
var(x: ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	ndarray	-	-

moment

Calculate the m-th moment of a vector of numbers x.

```
moment(x: ndarray,
       m: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	ndarray	-	-
m	int	-	-

correlation

Calculate the correlation between two series x1 and x2.

```
correlation(x1: ndarray,
            x2: ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x1	ndarray	-	-
x2	ndarray	-	-

2.21 __init__

Chapter 3

financepy.market.curves

Discount Curves

These modules create a family of discount curve types related to the term structures of interest rates. 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. These are curves which supply a discount factor that can be used to present-value future payments.

DiscountCurve

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.

DiscountCurveFlat

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

DiscountCurveNS

Implementation of the Nelson-Siegel curve parametrisation.

DiscountCurveNSS

Implementation of the Nelson-Siegel-Svensson curve parametrisation.

DiscountCurveZeros

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

Interpolate

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 x R(T) where R(T) is the zero rate. So this is not linear interpolation of R(T) but of T x 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.

3.1 discount_curve

Class: DiscountCurve

class DiscountCurve:

DiscountCurve

Create the discount curve from a vector of times and discount factors with an anchor date and specify an interpolation scheme. As we are explicitly linking dates and discount factors, we do not need to specify any compounding convention or day count calculation since discount factors are pure prices. We do however need to specify a convention for interpolating the discount factors in time.

```
DiscountCurve(value_dt: Date,
              df_dts: list,
              df_values: np.ndarray,
              interp_type: InterpTypes = InterpTypes.FLAT_FWD_RATES):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
df_dts	list	-	-
df_values	np.ndarray	-	-
interp_type	InterpTypes	-	InterpTypes.FLAT_FWD_RATES

_zero_to_df

Convert a zero with a specified compounding frequency and day count convention to a discount factor for a single maturity date or a list of dates. The day count is used to calculate the elapsed year fraction.

```
_zero_to_df(value_dt: Date,        #TODO: why is value_date not used ?
            rates: (float, np.ndarray),
            times: (float, np.ndarray),
            freq_type: FrequencyTypes,
            dc_type: DayCountTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	TODO: why is value_date not used ?	-
rates	float or np.ndarray	-	-
times	float or np.ndarray	-	-
freq_type	FrequencyTypes	-	-
dc_type	DayCountTypes	-	-

_df_to_zero

Given a dates this first generates the discount factors. It then converts the discount factors to a zero rate with a chosen compounding frequency which may be continuous, simple, or compounded at a specific frequency which are all choices of FrequencyTypes. Returns a list of discount factor.

```
_df_to_zero(dfs: (float, np.ndarray),
            maturity_dts: (Date, list),
            freq_type: FrequencyTypes,
            dc_type: DayCountTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dfs	float or np.ndarray	-	-
maturity_dts	Date or list	-	-
freq_type	FrequencyTypes	-	-
dc_type	DayCountTypes	-	-

zero_rate

Calculation of zero rates with specified frequency. This function can return a vector of zero rates given a vector of dates so must use Numpy functions. Default frequency is a continuously compounded rate.

```
zero_rate(dts: (list, Date),
          freq_type: FrequencyTypes = FrequencyTypes.CONTINUOUS,
          dc_type: DayCountTypes = DayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dts	list or Date	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.CONTINUOUS
dc_type	DayCountTypes	-	DayCountTypes.ACT_360

cc_rate

Calculation of zero rates with continuous compounding. This function can return a vector of cc rates given a vector of dates so must use Numpy functions.

```
cc_rate(dts: (list, Date),
        dc_type: DayCountTypes = DayCountTypes.SIMPLE):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dts	list or Date	-	-
dc_type	DayCountTypes	-	DayCountTypes.SIMPLE

swap_rate

Calculate the swap rate to maturity date. This is the rate paid by a swap that has a price of par today. This is the same as a Libor swap rate except that we do not do any business day adjustments.

```
swap_rate(effective_dt: Date,
          maturity_dt: (list, Date),
          freq_type=FrequencyTypes.ANNUAL,
          dc_type: DayCountTypes = DayCountTypes.THIRTY_E_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	-	-
maturity_dt	list or Date	-	-
freq_type	-	-	FrequencyTypes.ANNUAL
dc_type	DayCountTypes	-	DayCountTypes.THIRTY_E_360

df

Function to calculate a discount factor from a date or a vector of dates. The day count determines how dates get converted to years. I allow this to default to ACT_ACT_ISDA unless specified.

```
df(dt: (list, Date),
   day_count=DayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	list or Date	-	-
day_count	-	-	DayCountTypes.ACT_ACT_ISDA

_df

Hidden function to calculate a discount factor from a time or a vector of times. Discourage usage in favour of passing in dates.

```
_df(t: (float, np.ndarray)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	float or np.ndarray	-	-

survival_prob

This returns a survival probability to a specified date based on the assumption that the continuously compounded rate is a default hazard rate in which case the survival probability is directly analogous to a discount factor.

```
survival_prob(dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-

fwd

Calculate the continuously compounded forward rate at the forward Date provided. This is done by perturbing the time by one day only and measuring the change in the log of the discount factor divided by the time increment dt. I am assuming continuous compounding over the one date.

```
fwd(dts: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dts	Date	-	-

_fwd

Calculate the continuously compounded forward rate at the forward time provided. This is done by perturbing the time by a small amount and measuring the change in the log of the discount factor divided by the time increment dt.

```
_fwd(times: (np.ndarray, float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
times	np.ndarray or float	-	-

bump

Adjust the continuously compounded forward rates by a perturbation upward equal to the bump size and return a curve objet with this bumped curve. This is used for interest rate risk.

```
bump(bump_size: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bump_size	float	-	-

fwd_rate

Calculate the forward rate between two forward dates according to the specified day count convention. This defaults to Actual 360. The first date is specified and the second is given as a date or as a tenor which is added to the first date.

```
fwd_rate(start_dt: (list, Date),
         date_or_tenor: (Date, str),
         dc_type: DayCountTypes = DayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_dt	list or Date	-	-
date_or_tenor	Date or str	-	-
dc_type	DayCountTypes	-	DayCountTypes.ACT_360

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

3.2 discount_curve_flat

Class: DiscountCurveFlat(DiscountCurve)

A very simple discount curve based on a single zero rate with its own specified compounding method. Hence the curve is assumed to be flat. It is used for quick and dirty analysis and when limited information is available. It inherits several methods from FinDiscountCurve.

DiscountCurveFlat

Create a discount curve which is flat. This is very useful for quick testing and simply requires a curve date a rate and a compound frequency. As we have entered a rate, a corresponding day count convention must be used to specify how time periods are to be measured. As the curve is flat, no interpolation scheme is required.

```
DiscountCurveFlat(value_dt: Date,
                  flat_rate: (float, np.ndarray),
                  freq_type: FrequencyTypes = FrequencyTypes.CONTINUOUS,
                  dc_type: DayCountTypes = DayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
flat_rate	float or np.ndarray	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.CONTINUOUS
dc_type	DayCountTypes	-	DayCountTypes.ACT_ACT_ISDA

bump

Creates a new FinDiscountCurveFlat object with the entire curve bumped up by the bumpsize. All other parameters are preserved.

```
bump(bump_size: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bump_size	float	-	-

df

Return discount factors given a single or vector of dts. The discount factor depends on the rate and this in turn depends on its compounding frequency and it defaults to continuous compounding. It also depends on the day count convention. This was set in the construction of the curve to be ACT_ACT_ISDA.

```
df(dts: (Date, list)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dts	Date or list	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

3.3 discount_curve_ns

Class: DiscountCurveNS(DiscountCurve)

Implementation of Nelson-Siegel parametrisation of a discount curve. The internal rate is a continuously compounded rate but you can calculate alternative frequencies by providing a corresponding compounding frequency. A day count convention is needed to ensure that dates are converted to the correct time in years. The class inherits methods from FinDiscountCurve.

DiscountCurveNS

Creation of a FinDiscountCurveNS object. Parameters are provided individually for beta0, beta1, beta2 and tau. The zero rates produced by this parametrisation have an implicit compounding convention that defaults to continuous but which can be overridden.

```
DiscountCurveNS(value_dt: Date,
                beta0: float,
                beta1: float,
                beta2: float,
                tau: float,
                freq_type: FrequencyTypes = FrequencyTypes.CONTINUOUS,
                dc_type: DayCountTypes = DayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
beta0	float	-	-
beta1	float	-	-
beta2	float	-	-
tau	float	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.CONTINUOUS
dc_type	DayCountTypes	-	DayCountTypes.ACT_ACT_ISDA

zero_rate

Calculation of zero rates with specified frequency according to NS parametrisation. This method overrides that in FinDiscountCurve. The parametrisation is not strictly in terms of continuously compounded zero rates, this function allows other compounding and day counts. This function returns a single or vector of zero rates given a vector of dates so must use Numpy functions. The default frequency is a continuously compounded rate and ACT ACT day counting.

```
zero_rate(dates: (list, Date),
          freq_type: FrequencyTypes = FrequencyTypes.CONTINUOUS,
          dc_type: DayCountTypes = DayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dates	list or Date	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.CONTINUOUS
dc_type	DayCountTypes	-	DayCountTypes.ACT_360

_zero_rate

Zero rate for Nelson-Siegel curve parametrisation. This means that the t vector must use the curve day count.

```
_zero_rate(times: (float, np.ndarray)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
times	float or np.ndarray	-	-

df

Return discount factors given a single or vector of dates. The discount factor depends on the rate and this in turn depends on its compounding frequency and it defaults to continuous compounding. It also depends on the day count convention. This was set in the construction of the curve to be ACT_ACT_ISDA.

```
df(dates: (Date, list)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dates	Date or list	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

3.4 discount_curve_nss

Class: DiscountCurveNSS(DiscountCurve)

Implementation of Nelson-Siegel-Svensson parametrisation of the zero rate curve. The zero rate is assumed to be continuously compounded. This can be changed when calling for zero rates. A day count convention is needed to ensure that dates are converted to the correct time in years. The class inherits methods from FinDiscountCurve.

DiscountCurveNSS

Create a FinDiscountCurveNSS object by passing in curve valuation date plus the 4 different beta values and the 2 tau values. The zero rates produced by this parametrisation have an implicit compounding convention that defaults to continuous but can be overriden.

```
DiscountCurveNSS(value_dt: Date,
                 beta_0: float,
                 beta_1: float,
                 beta_2: float,
                 beta_3: float,
                 tau_1: float,
                 tau_2: float,
                 freq_type: FrequencyTypes = FrequencyTypes.CONTINUOUS,
                 day_count_type: DayCountTypes = DayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
beta_0	float	-	-
beta_1	float	-	-
beta_2	float	-	-
beta_3	float	-	-
tau_1	float	-	-
tau_2	float	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.CONTINUOUS
day_count_type	DayCountTypes	-	DayCountTypes.ACT_ACT_ISDA

zero_rate

Calculation of zero rates with specified frequency according to NSS parametrisation. This method overrides that in FinDiscountCurve. The NSS parametrisation is no strictly terms of continuously compounded zero rates, this function allows other compounding and day counts. This function returns a single or vector of zero rates given a vector of dates so must use Numpy functions. The default frequency is a continuously compounded rate and ACT ACT day counting.

```
zero_rate(dates: (list, Date),
          freq_type: FrequencyTypes = FrequencyTypes.CONTINUOUS,
          dc_type: DayCountTypes = DayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dates	list or Date	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.CONTINUOUS
dc_type	DayCountTypes	-	DayCountTypes.ACT_360

_zero_rate

Calculation of zero rates given a single time or a numpy vector of times. This function can return a single zero rate or a vector of zero rates. The compounding frequency must be provided.

```
_zero_rate(times: (float, np.ndarray)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
times	float or np.ndarray	-	-

df

Return discount factors given a single or vector of dates. The discount factor depends on the rate and this in turn depends on its compounding frequency and it defaults to continuous compounding. It also depends on the day count convention. This was set in the construction of the curve to be ACT_ACT_ISDA.

```
df(dates: (Date, list)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dates	Date or list	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

3.5 discount_curve_poly

Class: DiscountCurvePoly(DiscountCurve)

Zero Rate Curve of a specified frequency parametrised using a cubic polynomial. The zero rate is assumed to be continuously compounded but this can be amended by providing a frequency when extracting zero rates. We also need to specify a Day count convention for time calculations. The class inherits all of the methods from FinDiscountCurve.

DiscountCurvePoly

Create zero rate curve parametrised using a cubic curve from coefficients and specifying a compounding frequency type and day count convention.

```
DiscountCurvePoly(value_dt: Date,
                  coefficients: (list, np.ndarray),
                  freq_type: FrequencyTypes = FrequencyTypes.CONTINUOUS,
                  dc_type: DayCountTypes = DayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
coefficients	list or np.ndarray	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.CONTINUOUS
dc_type	DayCountTypes	-	DayCountTypes.ACT_ACT_ISDA

zero_rate

Calculation of zero rates with specified frequency according to polynomial parametrisation. This method overrides FinDiscountCurve. The parametrisation is not strictly in terms of continuously compounded zero rates, this function allows other compounding and day counts. This function returns a single or vector of zero rates given a vector of dates so must use Numpy functions. The default frequency is a continuously compounded rate and ACT ACT day counting.

```
zero_rate(dts: (list, Date),
          freq_type: FrequencyTypes = FrequencyTypes.CONTINUOUS,
          dc_type: DayCountTypes = DayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dts	list or Date	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.CONTINUOUS
dc_type	DayCountTypes	-	DayCountTypes.ACT_360

_zero_rate

Calculate the zero rate to maturity date but with times as inputs. This function is used internally and should be discouraged for external use. The compounding frequency defaults to that specified in the constructor of the curve object. Which may be annual to continuous.

```
_zero_rate(times: (float, np.ndarray)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
times	float or np.ndarray	-	-

df

Calculate the fwd rate to maturity date but with times as inputs. This function is used internally and should be discouraged for external use. The compounding frequency defaults to that specified in the constructor of the curve object.

```
df(dates: (list, Date)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dates	list or Date	-	-

__repr__

Display internal parameters of curve.

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

3.6 discount_curve_pwf

Class: DiscountCurvePWF(DiscountCurve)

Curve is made up of a series of zero rates sections with each having a piecewise flat zero rate. The default compounding assumption is continuous. The class inherits methods from FinDiscountCurve.

DiscountCurvePWF

Creates a discount curve using a vector of times and zero rates that assumes that the zero rates are piecewise flat.

```
DiscountCurvePWF(value_dt: Date,
                 zero_dts: list,
                 zero_rates: (list, np.ndarray),
                 freq_type: FrequencyTypes = FrequencyTypes.CONTINUOUS,
                 day_count_type: DayCountTypes = DayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
zero_dts	list	-	-
zero_rates	list or np.ndarray	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.CONTINUOUS
day_count_type	DayCountTypes	-	DayCountTypes.ACT_ACT_ISDA

_zero_rate

The piecewise flat zero rate is selected and returned.

```
_zero_rate(times: (float, np.ndarray, list)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
times	float or np.ndarray,list	-	-

_fwd

Calculate the continuously compounded forward rate at the forward time provided. This is done by perturbing the time by a small amount and measuring the change in the log of the discount factor divided by the time increment dt.

```
_fwd(times: (np.ndarray, list)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
times	np.ndarray or list	-	-

df

Return discount factors given a single or vector of dates. The discount factor depends on the rate and this in turn depends on its compounding frequency and it defaults to continuous compounding. It also depends on the day count convention. This was set in the construction of the curve to be ACT_ACT_ISDA.

```
df(dates: (Date, list)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dates	Date or list	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

3.7 discount_curve_pwl

Class: DiscountCurvePWL(DiscountCurve)

Curve is made up of a series of sections assumed to each have a piece-wise linear zero rate. The zero rate has a specified frequency which defaults to continuous. This curve inherits all of the extra methods from FinDiscountCurve.

DiscountCurvePWL

Curve is defined by a vector of increasing times and zero rates.

```
DiscountCurvePWL(value_dt: Date,
                 zero_dts: (Date, list),
                 zero_rates: (list, np.ndarray),
                 freq_type: FrequencyTypes = FrequencyTypes.CONTINUOUS,
                 dc_type: DayCountTypes = DayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
zero_dts	Date or list	-	-
zero_rates	list or np.ndarray	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.CONTINUOUS
dc_type	DayCountTypes	-	DayCountTypes.ACT_ACT_ISDA

_zero_rate

Calculate the piecewise linear zero rate. This is taken from the initial inputs. A simple linear interpolation scheme is used. If the user supplies a frequency type then a conversion is done.

```
_zero_rate(times: (list, np.ndarray)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
times	list or np.ndarray	-	-

df

Return discount factors given a single or vector of dates. The discount factor depends on the rate and this in turn depends on its compounding frequency and it defaults to continuous compounding. It also depends on the day count convention. This was set in the construction of the curve to be ACT_ACT_ISDA.

```
df(dates: (Date, list)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dates	Date or list	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

3.8 discount_curve_zeros

Class: DiscountCurveZeros(DiscountCurve)

This is a curve calculated from a set of dates and zero rates. As we have rates as inputs, we need to specify the corresponding compounding frequency. Also to go from rates and dates to discount factors we need to compute the year fraction correctly and for this we require a day count convention. Finally, we need to interpolate the zero rate for the times between the zero rates given and for this we must specify an interpolation convention. The class inherits methods from FinDiscountCurve.

DiscountCurveZeros

Create the discount curve from a vector of dates and zero rates factors. The first date is the curve anchor. Then a vector of zero dates and then another same-length vector of rates. The rate is to the corresponding date. We must specify the compounding frequency of the zero rates and also a day count convention for calculating times which we must do to calculate discount factors. Finally we specify the interpolation scheme for off-grid dates.

```
DiscountCurveZeros(value_dt: Date,
                   zero_dts: list,
                   zero_rates: (list, np.ndarray),
                   freq_type: FrequencyTypes = FrequencyTypes.ANNUAL,
                   dc_type: DayCountTypes = DayCountTypes.ACT_ACT_ISDA,
                   interp_type: InterpTypes = InterpTypes.FLAT_FWD_RATES):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
zero_dts	list	-	-
zero_rates	list or np.ndarray	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.ANNUAL
dc_type	DayCountTypes	-	DayCountTypes.ACT_ACT_ISDA
interp_type	InterpTypes	-	InterpTypes.FLAT_FWD_RATES

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

3.9 interpolator

Enumerated Type: InterpTypes

This enumerated type has the following values:

- FLAT_FWD_RATES
- LINEAR_FWD_RATES
- LINEAR_ZERO_RATES
- FINCUBIC_ZERO_RATES
- NATCUBIC_LOG_DISCOUNT
- NATCUBIC_ZERO_RATES
- PCHIP_ZERO_RATES
- PCHIP_LOG_DISCOUNT

Class: Interpolator()

class Interpolator():

Interpolator

PLEASE ADD A FUNCTION DESCRIPTION

```
Interpolator(interpolatorType: InterpTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
interpolatorType	InterpTypes	-	-

fit

 # Second derivatives at left is zero and first derivative at # right is clamped to zero.

```
fit(times: np.ndarray,
    dfs: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
times	np.ndarray	-	-
dfs	np.ndarray	-	-

interpolate

Interpolation of discount factors at time x given discount factors at times provided using one of the methods in the enum InterpTypes. The value of x can be an array so that the function is vectorised.

```
interpolate(t: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	float	-	-

interpolate

Fast interpolation of discount factors at time x given discount factors at times provided using one of the methods in the enum InterpTypes. The value of x can be an array so that the function is vectorised.

```
interpolate(t: (float, np.ndarray),  # time or array of times
            times: np.ndarray,  # Vector of times on grid
            dfs: np.ndarray,  # Vector of discount factors
            method: int):  # Interpolation method which is value of enum
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	float or np.ndarray	time or array of times	-
times	np.ndarray	Vector of times on grid	-
dfs	np.ndarray	Vector of discount factors	-
method	int	Interpolation method which is value of enum	-

_uinterpolate

Return the interpolated value of y given x and a vector of x and y. The values of x must be monotonic and increasing. The different schemes for interpolation are linear in y (as a function of x), linear in log(y) and piecewise flat in the continuously compounded forward y rate.

```
_uinterpolate(t, times, dfs, method):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
times	-	-	-
dfs	-	-	-
method	-	-	-

_vinterpolate

Return the interpolated values of y given x and a vector of x and y. The values of x must be monotonic and increasing. The different schemes for interpolation are linear in y (as a function of x), linear in log(y) and piecewise flat in the continuously compounded forward y rate.

```
_vinterpolate(xValues,
              xvector,
              dfs,
              method):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
xValues	-	-	-
xvector	-	-	-
dfs	-	-	-
method	-	-	-

3.10 __init__

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 parametric 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.

FinIborCapFloorVol

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.

FinIborCapFloorVolFn

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

4.1 equity_vol_curve

Class: EquityVolCurve()

Class to manage a smile or skew in volatility at a single maturity horizon. It fits the volatility using a polynomial. Includes analytics to extract the implied pdf of the underyling at maturity. THIS NEEDS TO BE SUBSTITUTED WITH FINEQUITYVOLSURFACE.

EquityVolCurve

PLEASE ADD A FUNCTION DESCRIPTION

```
EquityVolCurve(curve_dt,
               expiry_dt,
               strikes,
               volatilities,
               polynomial=3):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
curve_dt	-	-	-
expiry_dt	-	-	-
strikes	-	-	-
volatilities	-	-	-
polynomial	-	-	3

volatility

Return the volatility for a strike using a given polynomial interpolation.

```
volatility(strike):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
strike	-	-	-

calculate_pdf

calculate the probability density function of the underlying using the volatility smile or skew curve following the approach set out in Breedon and Litzenberger.

```
calculate_pdf():
```

The function arguments are described in the following table.

4.2 equity_vol_surface

Class: EquityVolSurface

Class to perform a calibration of a chosen parametrised surface to the prices of equity options at different strikes and expiry tenors. There is0 a choice of volatility function from cubic in delta to full SABR and SSVI. Check out VolFuncTypes. Visualising the volatility curve is useful. Also, there is no guarantee that the implied pdf will be positive.

EquityVolSurface

Create the EquitySurface object by passing in market vol data for a list of strikes and expiry dates.

```
EquityVolSurface(value_dt: Date,
                 stock_price: float,
                 discount_curve: DiscountCurve,
                 dividend_curve: DiscountCurve,
                 expiry_dts: (list),
                 strikes: (list, np.ndarray),
                 volatility_grid: (list, np.ndarray),
                 volatility_function_type: VolFuncTypes = VolFuncTypes.CLARK,
                 finSolverType: FinSolverTypes = FinSolverTypes.NELDER_MEAD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
expiry_dts	list or (list	-	-
strikes	list or np.ndarray	-	-
volatility_grid	list or np.ndarray	-	-
volatility_function_type	VolFuncTypes	-	VolFuncTypes.CLARK
finSolverType	FinSolverTypes	-	NELDER_MEAD

vol_from_strike_dt

Interpolates the Black-Scholes volatility from the volatility surface given call option strike and expiry date. Linear interpolation is done in variance space. The smile strikes at bracketed dates are determined by determining the strike that reproduces the provided delta value. This uses the calibration delta convention, but it can be overriden by a provided delta convention. The resulting volatilities are then determined for each bracketing expiry time and linear interpolation is done in variance space and then converted back to a lognormal volatility.

```
vol_from_strike_dt(K, expiry_dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-
expiry_dt	-	-	-

vol_from_delta_dt

Interpolates the Black-Scholes volatility from the volatility surface given a call option delta and expiry date. Linear interpolation is done in variance space. The smile strikes at bracketed dates are determined by determining the strike that reproduces the provided delta value. This uses the calibration delta convention, but it can be overriden by a provided delta convention. The resulting volatilities are then determined for each bracketing expiry time and linear interpolation is done in variance space and then converted back to a lognormal volatility.

```
vol_from_delta_dt(call_delta, expiry_dt, delta_method=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
call_delta	-	-	-
expiry_dt	-	-	-
delta_method	-	-	None

_build_vol_surface

Main function to construct the vol surface.

```
_build_vol_surface(finSolverType=FinSolverTypes.NELDER_MEAD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
finSolverType	-	-	NELDER_MEAD

check_calibration

Compare calibrated vol surface with market and output a report which sets out the quality of fit to the ATM and 10 and 25 delta market strangles and risk reversals.

```
check_calibration(verbose: bool, tol: float = 1e-6):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
verbose	bool	-	-
tol	float	-	1e-6

implied_dbns

Calculate the pdf for each tenor horizon. Returns a list of FinDistribution objects, one for each tenor horizon.

```
implied_dbns(lowS, highS, numIntervals):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
lowS	-	-	-
highS	-	-	-
numIntervals	-	-	-

plot_vol_curves

Generates a plot of each of the vol discount implied by the market and fitted.

```
plot_vol_curves():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

_obj

Return a value that is minimised when the ATM, MS and RR vols have been best fitted using the parametric volatility curve represented by params and specified by the vol_type_value. We fit at one time slice only.

```
_obj(params, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
*args	-	-	-

_solve_to_horizon

PLEASE ADD A FUNCTION DESCRIPTION

```
_solve_to_horizon(s, t, r, q,
                  strikes,
                  time_index,
                  volatility_grid,
                  vol_type_value,
                  x_inits,
                  finSolverType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
r	-	-	-
q	-	-	-
strikes	-	-	-
time_index	-	-	-
volatility_grid	-	-	-
vol_type_value	-	-	-
x_inits	-	-	-
finSolverType	-	-	-

vol_function

Return the volatility for a strike using a given polynomial interpolation following Section 3.9 of Iain Clark book.

```
vol_function(vol_function_type_value, params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
vol_function_type_value	-	-	-
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

_delta_fit

PLEASE ADD A FUNCTION DESCRIPTION

```
_delta_fit(k, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
*args	-	-	-

_solver_for_smile_strike

Solve for the strike that sets the delta of the option equal to the target value of delta allowing the volatility to be a function of the strike.

```
_solver_for_smile_strike(s, t, r, q,
                         option_type_value,
                         volatilityTypeValue,
                         delta_target,
                         initial_guess,
                         parameters):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
r	-	-	-
q	-	-	-
option_type_value	-	-	-
volatilityTypeValue	-	-	-
delta_target	-	-	-
initial_guess	-	-	-
parameters	-	-	-

4.3 fx_vol_surface

Class: FXVolSurface()

Class to perform a calibration of a chosen parametrised surface to the prices of FX options at different strikes and expiry tenors. The calibration inputs are the ATM and 25 Delta volatilities given in terms of the market strangle amd risk reversals. There is a choice of volatility function ranging from polynomial in delta to a limited version of SABR.

FXVolSurface

Create the FinFXVolSurface object by passing in market vol data for ATM and 25 Delta Market Strangles and Risk Reversals.

```
FXVolSurface(value_dt: Date,
             spot_fx_rate: float,
             currency_pair: str,
             notional_currency: str,
             dom_discount_curve: DiscountCurve,
             for_discount_curve: DiscountCurve,
             tenors: (list),
             atm_vols: (list, np.ndarray),
             mktStrangle25DeltaVols: (list, np.ndarray),
             riskReversal25DeltaVols: (list, np.ndarray),
             atmMethod: FinFXATMMethod = FinFXATMMethod.FWD_DELTA_NEUTRAL,
             delta_method: FinFXDeltaMethod = FinFXDeltaMethod.SPOT_DELTA,
             volatility_function_type: VolFuncTypes = VolFuncTypes.CLARK):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
spot_fx_rate	float	-	-
currency_pair	str	-	-
notional_currency	str	-	-
dom_discount_curve	DiscountCurve	-	-
for_discount_curve	DiscountCurve	-	-
tenors	list or (list	-	-
atm_vols	list or np.ndarray	-	-
mktStrangle25DeltaVols	list or np.ndarray	-	-
riskReversal25DeltaVols	list or np.ndarray	-	-
atmMethod	FinFXATMMethod	-	FWD_DELTA_NEUTRAL
delta_method	FinFXDeltaMethod	-	SPOT_DELTA
volatility_function_type	VolFuncTypes	-	VolFuncTypes.CLARK

volatility

Interpolate the Black-Scholes volatility from the volatility surface given the option strike and expiry date. Linear interpolation is done in variance x time.

```
volatility(K, expiry_dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-
expiry_dt	-	-	-

build_vol_surface

PLEASE ADD A FUNCTION DESCRIPTION

```
build_vol_surface():
```

The function arguments are described in the following table.

solver_for_smile_strike

Solve for the strike that sets the delta of the option equal to the target value of delta allowing the volatility to be a function of the strike.

```
solver_for_smile_strike(option_type_value,
                        delta_target,
                        tenor_index,
                        initialValue):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
option_type_value	-	-	-
delta_target	-	-	-
tenor_index	-	-	-
initialValue	-	-	-

check_calibration

PLEASE ADD A FUNCTION DESCRIPTION

```
check_calibration(verbose: bool, tol: float = 1e-6):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
verbose	bool	-	-
tol	float	-	1e-6

implied_dbns

Calculate the pdf for each tenor horizon. Returns a list of FinDistribution objects, one for each tenor horizon.

```
implied_dbns(lowFX, highFX, numIntervals):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
lowFX	-	-	-
highFX	-	-	-
numIntervals	-	-	-

plot_vol_curves

PLEASE ADD A FUNCTION DESCRIPTION

```
plot_vol_curves():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

g

PLEASE ADD A FUNCTION DESCRIPTION

```
g(K, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-
*args	-	-	-

obj_fast

Return a function that is minimised when the ATM, MS and RR vols have been best fitted using the parametric volatility curve represented by cvec

```
obj_fast(params, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
*args	-	-	-

solve_to_horizon_fast

PLEASE ADD A FUNCTION DESCRIPTION

```
solve_to_horizon_fast(s, t,
                      rd, rf,
                      K_ATM, atm_vol,
                      ms25DVol, rr25DVol,
                      delta_method_value, vol_type_value,
                      xopt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
rd	-	-	-
rf	-	-	-
K_ATM	-	-	-
atm_vol	-	-	-
ms25DVol	-	-	-
rr25DVol	-	-	-
delta_method_value	-	-	-
vol_type_value	-	-	-
xopt	-	-	-

vol_function

Return the volatility for a strike using a given polynomial interpolation following Section 3.9 of Iain Clark book.

```
vol_function(vol_function_type_value, params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
vol_function_type_value	-	-	-
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

delta_fit

PLEASE ADD A FUNCTION DESCRIPTION

```
delta_fit(K, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-
*args	-	-	-

solver_for_smile_strike_fast

Solve for the strike that sets the delta of the option equal to the target value of delta allowing the volatility to be a function of the strike.

```
solver_for_smile_strike_fast(s, t, rd, rf,
                             option_type_value,
                             volatilityTypeValue,
                             delta_target,
                             delta_method_value,
                             initial_guess,
                             parameters):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
rd	-	-	-
rf	-	-	-
option_type_value	-	-	-
volatilityTypeValue	-	-	-
delta_target	-	-	-
delta_method_value	-	-	-
initial_guess	-	-	-
parameters	-	-	-

solve_for_strike

This function determines the implied strike of an FX option given a delta and the other option details. It uses a one-dimensional Newton root search algorithm to determine the strike that matches an input volatility.

```
solve_for_strike(spot_fx_rate,
                 tdel, rd, rf,
                 option_type_value,
                 delta_target,
                 delta_method_value,
                 volatility):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spot_fx_rate	-	-	-
tdel	-	-	-
rd	-	-	-
rf	-	-	-
option_type_value	-	-	-
delta_target	-	-	-
delta_method_value	-	-	-
volatility	-	-	-

4.4 fx_vol_surface_plus

Class: FXVolSurfacePlus()

Class to perform a calibration of a chosen parametrised surface to the prices of FX options at different strikes and expiry tenors. The calibration inputs are the ATM and 25 and 10 Delta volatilities in terms of the market strangle amd risk reversals. There is a choice of volatility function from cubic in delta to full SABR. Check out VolFuncTypes. Parameter alpha [0,1] is used to interpolate between fitting only 25D when alpha=0 to fitting only 10D when alpha=1.0. Alpha=0.5 assigns equal weights A vol function with more parameters will give a better fit. Of course. But it might also overfit. Visualising the volatility curve is useful. Also, there is no guarantee that the implied pdf will be positive.

FXVolSurfacePlus

Create the FinFXVolSurfacePlus object by passing in market vol data for ATM, 25 Delta and 10 Delta strikes. The alpha weight shifts the fitting between 25D and 10D. Alpha = 0.0 is 100is 100

```
FXVolSurfacePlus(value_dt: Date,
                 spot_fx_rate: float,
                 currency_pair: str,
                 notional_currency: str,
                 dom_discount_curve: DiscountCurve,
                 for_discount_curve: DiscountCurve,
                 tenors: (list),
                 atm_vols: (list, np.ndarray),
                 mktStrangle25DeltaVols: (list, np.ndarray),
                 riskReversal25DeltaVols: (list, np.ndarray),
                 mktStrangle10DeltaVols: (list, np.ndarray),
                 riskReversal10DeltaVols: (list, np.ndarray),
                 alpha: float,
                 atmMethod: FinFXATMMethod = FinFXATMMethod.FWD_DELTA_NEUTRAL,
                 delta_method: FinFXDeltaMethod = FinFXDeltaMethod.SPOT_DELTA,
                 volatility_function_type: VolFuncTypes = VolFuncTypes.CLARK,
                 finSolverType: FinSolverTypes = FinSolverTypes.NELDER_MEAD,
                 tol: float = 1e-8):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
spot_fx_rate	float	-	-
currency_pair	str	-	-
notional_currency	str	-	-
dom_discount_curve	DiscountCurve	-	-
for_discount_curve	DiscountCurve	-	-
tenors	list or (list	-	-
atm_vols	list or np.ndarray	-	-
mktStrangle25DeltaVols	list or np.ndarray	-	-
riskReversal25DeltaVols	list or np.ndarray	-	-
mktStrangle10DeltaVols	list or np.ndarray	-	-
riskReversal10DeltaVols	list or np.ndarray	-	-
alpha	float	-	-
atmMethod	FinFXATMMethod	-	FWD_DELTA_NEUTRAL
delta_method	FinFXDeltaMethod	-	SPOT_DELTA
volatility_function_type	VolFuncTypes	-	VolFuncTypes.CLARK
finSolverType	FinSolverTypes	-	NELDER_MEAD
tol	float	-	1e-8

vol_from_strike_dt

Interpolates the Black-Scholes volatility from the volatility surface given call option strike and expiry date. Linear interpolation is done in variance space. The smile strikes at bracketed dates are determined by determining the strike that reproduces the provided delta value. This uses the calibration delta convention, but it can be overriden by a provided delta convention. The resulting volatilities are then determined for each bracketing expiry time and linear interpolation is done in variance space and then converted back to a lognormal volatility.

```
vol_from_strike_dt(K, expiry_dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-
expiry_dt	-	-	-

delta_to_strike

Interpolates the strike at a delta and expiry date. Linear time to expiry interpolation is used in strike.

```
delta_to_strike(call_delta, expiry_dt, delta_method):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
call_delta	-	-	-
expiry_dt	-	-	-
delta_method	-	-	-

vol_from_delta_dt

Interpolates the Black-Scholes volatility from the volatility surface given a call option delta and expiry date. Linear interpolation is done in variance space. The smile strikes at bracketed dates are determined by determining the strike that reproduces the provided delta value. This uses the calibration delta convention, but it can be overriden by a provided delta convention. The resulting volatilities are then determined for each bracketing expiry time and linear interpolation is done in variance space and then converted back to a lognormal volatility.

```
vol_from_delta_dt(call_delta,
                  expiry_dt,
                  delta_method=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
call_delta	-	-	-
expiry_dt	-	-	-
delta_method	-	-	None

_build_vol_surface

Main function to construct the vol surface.

```
_build_vol_surface(finSolverType=FinSolverTypes.NELDER_MEAD,
                   tol=1e-8):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
finSolverType	-	-	NELDER_MEAD
tol	-	-	1e-8

check_calibration

Compare calibrated vol surface with market and output a report which sets out the quality of fit to the ATM and 10 and 25 delta market strangles and risk reversals.

```
check_calibration(verbose: bool, tol: float = 1e-6):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
verbose	bool	-	-
tol	float	-	1e-6

implied_dbns

Calculate the pdf for each tenor horizon. Returns a list of FinDistribution objects, one for each tenor horizon.

```
implied_dbns(lowFX, highFX, numIntervals):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
lowFX	-	-	-
highFX	-	-	-
numIntervals	-	-	-

plot_vol_curves

Generates a plot of each of the vol discount implied by the market and fitted.

```
plot_vol_curves():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

_g

PLEASE ADD A FUNCTION DESCRIPTION

```
_g(K, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-
*args	-	-	-

_interpolate_gap

PLEASE ADD A FUNCTION DESCRIPTION

```
_interpolate_gap(k, strikes, gaps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
strikes	-	-	-
gaps	-	-	-

_obj

Return a function that is minimised when the ATM, MS and RR vols have been best fitted using the parametric volatility curve represented by params and specified by the vol_type_value

```
_obj(params, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
*args	-	-	-

_obj_gap

Return a function that is minimised when the ATM, MS and RR vols have been best fitted using the parametric volatility curve represented by params and specified by the vol_type_value

```
_obj_gap(gaps, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
gaps	-	-	-
*args	-	-	-

_solve_to_horizon

PLEASE ADD A FUNCTION DESCRIPTION

```
_solve_to_horizon(s, t, rd, rf,
                  K_ATM, atm_vol,
                  ms25DVol, rr25DVol,
                  ms10DVol, rr10DVol,
                  delta_method_value, vol_type_value,
                  alpha,
                  x_inits,
                  ginits,
                  finSolverType,
                  tol):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
rd	-	-	-
rf	-	-	-
K_ATM	-	-	-
atm_vol	-	-	-
ms25DVol	-	-	-
rr25DVol	-	-	-
ms10DVol	-	-	-
rr10DVol	-	-	-
delta_method_value	-	-	-
vol_type_value	-	-	-
alpha	-	-	-
x_inits	-	-	-
ginits	-	-	-
finSolverType	-	-	-
tol	-	-	-

vol_function

Return the volatility for a strike using a given polynomial interpolation following Section 3.9 of Iain Clark book.

```
vol_function(vol_function_type_value, params, strikes, gaps, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
vol_function_type_value	-	-	-
params	-	-	-
strikes	-	-	-
gaps	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

_delta_fit

PLEASE ADD A FUNCTION DESCRIPTION

```
_delta_fit(k, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
*args	-	-	-

_solver_for_smile_strike

Solve for the strike that sets the delta of the option equal to the target value of delta allowing the volatility to be a function of the strike.

```
_solver_for_smile_strike(s, t, rd, rf,
                         option_type_value,
                         volatilityTypeValue,
                         delta_target,
                         delta_method_value,
                         initial_guess,
                         parameters,
                         strikes,
                         gaps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
rd	-	-	-
rf	-	-	-
option_type_value	-	-	-
volatilityTypeValue	-	-	-
delta_target	-	-	-
delta_method_value	-	-	-
initial_guess	-	-	-
parameters	-	-	-
strikes	-	-	-
gaps	-	-	-

solve_for_strike

This function determines the implied strike of an FX option given a delta and the other option details. It uses a one-dimensional Newton root search algorithm to determine the strike that matches an input volatility.

```
solve_for_strike(spot_fx_rate,
                 tdel, rd, rf,
                 option_type_value,
                 delta_target,
                 delta_method_value,
                 volatility):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spot_fx_rate	-	-	-
tdel	-	-	-
rd	-	-	-
rf	-	-	-
option_type_value	-	-	-
delta_target	-	-	-
delta_method_value	-	-	-
volatility	-	-	-

4.5 ibor_cap_vol_curve

Class: IborCapVolCurve()

Class to manage a term structure of cap (flat) volatilities and to do the conversion to caplet (spot) volatilities. This does not manage a strike dependency, only a term structure. The cap and caplet volatilies are keyed off the cap and caplet maturity dates. However this volatility only applies to the evolution of the Ibor rate out to the caplet start dates. Note also that this class also handles floor vols.

IborCapVolCurve

Create a cap/floor volatility curve given a curve date, a list of cap maturity dates and a vector of cap volatilities. To avoid confusion first date of the capDates must be equal to the curve date and first cap volatility for this date must equal zero. The internal times are calculated according to the provided day count convention. Note cap and floor volatilities are the same for the same strike and tenor, I just refer to cap volatilities in the code for code simplicity.

```
IborCapVolCurve(curve_dt,  # Valuation date for cap volatility
                cap_maturity_dts,  # curve date + maturity dates for caps
                cap_sigmas,  # Flat cap volatility for cap maturity dates
                dc_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
curve_dt	-	Valuation date for cap volatility	-
cap_maturity_dts	-	curve date + maturity dates for caps	-
cap_sigmas	-	Flat cap volatility for cap maturity dates	-
dc_type	-	-	-

generate_caplet_vols

Bootstrap caplet volatilities from cap volatilities using similar notation to Hull's book (page 32.11). The first volatility in the vector of caplet vols is zero.

```
generate_caplet_vols():
```

The function arguments are described in the following table.

caplet_vol

PLEASE ADD A FUNCTION DESCRIPTION

```
caplet_vol(dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-

cap_vol

Return the cap flat volatility for a specific cap maturity date for the last caplet/floorlet in the cap/floor. The volatility interpolation is piecewise flat.

```
cap_vol(dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

4.6 ibor_cap_vol_curve_fn

Class: IborCapVolCurveFn()

Class to manage a term structure of caplet volatilities using the parametric form suggested by Rebonato (1999).

IborCapVolCurveFn

PLEASE ADD A FUNCTION DESCRIPTION

```
IborCapVolCurveFn(curve_dt,
                  a,
                  b,
                  c,
                  d):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
curve_dt	-	-	-
a	-	-	-
b	-	-	-
c	-	-	-
d	-	-	-

cap_floorlet_vol

Return the caplet volatility.

```
cap_floorlet_vol(dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-

4.7 swaption_vol_surface

Class: SwaptionVolSurface()

Class to perform a calibration of a chosen parametrised surface to the prices of swaptions at different expiry dates and swap tenors. There is a choice of volatility function from cubic in delta to full SABR and SSVI. Check out VolFuncTypes. Visualising the volatility curve is useful. Also, there is no guarantee that the implied pdf will be positive.

SwaptionVolSurface

Create the FinSwaptionVolSurface object by passing in market vol data for a list of strikes and expiry dates.

```
SwaptionVolSurface(value_dt: Date,
                   expiry_dts: (list),
                   fwd_swap_rates: (list, np.ndarray),
                   strike_grid: (np.ndarray),
                   volatility_grid: (np.ndarray),
                   volatility_function_type: VolFuncTypes = VolFuncTypes.SABR,
                   finSolverType: FinSolverTypes = FinSolverTypes.NELDER_MEAD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
expiry_dts	list or (list	-	-
fwd_swap_rates	list or np.ndarray	-	-
strike_grid	np.ndarray or (np.ndarray	-	-
volatility_grid	np.ndarray or (np.ndarray	-	-
volatility_function_type	VolFuncTypes	-	VolFuncTypes.SABR
finSolverType	FinSolverTypes	-	NELDER_MEAD

vol_from_strike_dt

Interpolates the Black-Scholes volatility from the volatility surface given call option strike and expiry date. Linear interpolation is done in variance space. The smile strikes at bracketed dates are determined by determining the strike that reproduces the provided delta value. This uses the calibration delta convention, but it can be overriden by a provided delta convention. The resulting volatilities are then determined for each bracketing expiry time and linear interpolation is done in variance space and then converted back to a lognormal volatility.

```
vol_from_strike_dt(K, expiry_dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-
expiry_dt	-	-	-

_build_vol_surface

Main function to construct the vol surface.

```
_build_vol_surface(finSolverType=FinSolverTypes.NELDER_MEAD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
finSolverType	-	-	NELDER_MEAD

check_calibration

Compare calibrated vol surface with market and output a report which sets out the quality of fit to the ATM and 10 and 25 delta market strangles and risk reversals.

```
check_calibration(verbose: bool, tol: float = 1e-6):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
verbose	bool	-	-
tol	float	-	1e-6

plot_vol_curves

Generates a plot of each of the vol discount implied by the market and fitted.

```
plot_vol_curves():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

_obj

Return a value that is minimised when the ATM, MS and RR vols have been best fitted using the parametric volatility curve represented by params and specified by the vol_type_value at a single time slice only.

```
_obj(params, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
*args	-	-	-

_solve_to_horizon

PLEASE ADD A FUNCTION DESCRIPTION

```
_solve_to_horizon(t, f,
                  strikes_grid,
                  time_index,
                  volatility_grid,
                  vol_type_value,
                  x_inits,
                  finSolverType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
f	-	-	-
strikes_grid	-	-	-
time_index	-	-	-
volatility_grid	-	-	-
vol_type_value	-	-	-
x_inits	-	-	-
finSolverType	-	-	-

vol_function

Return the volatility for a strike using a given polynomial interpolation following Section 3.9 of Iain Clark book.

```
vol_function(vol_function_type_value, params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
vol_function_type_value	-	-	-
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

4.8 __init__

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

5.1 equity_american_option

Class: EquityAmericanOption(EquityOption)

Class for American (and European) style options on simple vanilla calls and puts - a tree valuation model is used that can handle both.

EquityAmericanOption

Class for American style options on simple vanilla calls and puts. Specify the expiry date, strike price, whether the option is a call or put and the number of options.

```
EquityAmericanOption(expiry_dt: Date,
                     strike_price: float,
                     option_type: OptionTypes,
                     num_options: float = 1.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
strike_price	float	-	-
option_type	OptionTypes	-	-
num_options	float	-	1.0

value

Valuation of an American option using a CRR tree to take into account the value of early exercise.

```
value(value_dt: Date,
      stock_price: (np.ndarray, float),
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	np.ndarray or float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

5.2 equity_asian_option

Enumerated Type: AsianOptionValuationMethods

This enumerated type has the following values:

- GEOMETRIC
- TURNBULL_WAKEMAN
- CURRAN

Class: EquityAsianOption

Class for an Equity Asian Option. This is an option with a final payoff linked to the averaging of the stock price over some specified period before the option expires. The valuation is done for both an arithmetic and a geometric average but the former can only be done either using an analytical approximation of the arithmetic average distribution or by using Monte-Carlo simulation.

EquityAsianOption

Create an EquityAsian option object which takes a start date for the averaging, an expiry date, a strike price, an option type and a number of observations.

```
EquityAsianOption(startAveragingDate: Date,
                  expiry_dt: Date,
                  strike_price: float,
                  option_type: OptionTypes,
                  numberOfObservations: int = 100):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startAveragingDate	Date	-	-
expiry_dt	Date	-	-
strike_price	float	-	-
option_type	OptionTypes	-	-
numberOfObservations	int	-	100

value

Calculate the value of an Asian option using one of the specified analytical approximations for an average rate option. These are the three enumerated values in the enum AsianOptionValuationMethods. The choices of approximation are (i) GEOMETRIC - the average is a geometric one as in paper by Kenna and Worst (1990), (ii) TURNBULL_WAKEMAN - this is a value based on an edgeworth expansion of the moments of the arithmetic average, and (iii) CURRAN - another approximative approach by Curran based on conditioning on the geometric mean price. Just choose the corresponding enumerated value to switch between these different approaches. Note that the accrued average is only required if the value date is inside the averaging period for the option.

```
value(value_dt: Date,
```

```
        stock_price: float,
        discount_curve: DiscountCurve,
        dividend_curve: DiscountCurve,
        model,
        method: AsianOptionValuationMethods,
        accrued_average: float = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-
method	AsianOptionValuationMethods	-	-
accrued_average	float	-	None

_value_geometric

This option valuation is based on paper by Kemna and Vorst 1990. It calculates the Geometric Asian option price which is a lower bound on the Arithmetic option price. This should not be used as a valuation model for the Arithmetic Average option but can be used as a control variate for other approaches.

```
_value_geometric(value_dt, stock_price, discount_curve,
                 dividend_curve, model, accrued_average):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
model	-	-	-
accrued_average	-	-	-

_value_curran

Valuation of an Asian option using the result by Vorst.

```
_value_curran(value_dt, stock_price, discount_curve,
              dividend_curve, model, accrued_average):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
model	-	-	-
accrued_average	-	-	-

_value_turnbull_wakeman

Asian option valuation based on paper by Turnbull and Wakeman 1991 which uses the edgeworth expansion to find the first two moments of the arithmetic average.

```
_value_turnbull_wakeman(value_dt, stock_price, discount_curve,
                        dividend_curve, model, accrued_average):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
model	-	-	-
accrued_average	-	-	-

_value_mc

Monte Carlo valuation of the Asian Average option using standard Monte Carlo code enhanced by Numba. I have discontinued the use of this as it is both slow and has limited variance reduction.

```
_value_mc(value_dt: Date,
          stock_price: float,
          discount_curve: DiscountCurve,
          dividend_curve: DiscountCurve,
          model,
          num_paths: int,
          seed: int,
          accrued_average: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-
num_paths	int	-	-
seed	int	-	-
accrued_average	float	-	-

_value_mc_fast

Monte Carlo valuation of the Asian Average option. This method uses a lot of Numpy vectorisation. It is also helped by Numba.

```
_value_mc_fast(value_dt,
               stock_price,
               discount_curve,
               dividend_curve,  # Yield
               model,          # Model
               num_paths,       # Numpaths integer
               seed,
               accrued_average):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	-	-
discount_curve	-	-	-
dividend_curve	-	Yield	-
model	-	Model	-
num_paths	-	Numpaths integer	-
seed	-	-	-
accrued_average	-	-	-

value_mc

Monte Carlo valuation of the Asian Average option using a control variate method that improves accuracy and reduces the variance of the price. This uses Numpy and Numba. This is the standard MC pricer.

```
value_mc(value_dt: Date,
         stock_price: float,
         discount_curve: DiscountCurve,
         dividend_curve: DiscountCurve,
         model,
         num_paths: int,
         seed: int,
         accrued_average: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-
num_paths	int	-	-
seed	int	-	-
accrued_average	float	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_value_mc_numba

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_mc_numba(t0,
                t,
                tau,
                K,
                n,
                option_type,
                stock_price,
                interest_rate,
                dividend_yield,
                volatility,
                num_paths,
                seed,
                accrued_average):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t0	-	-	-
t	-	-	-
tau	-	-	-
K	-	-	-
n	-	-	-
option_type	-	-	-
stock_price	-	-	-
interest_rate	-	-	-
dividend_yield	-	-	-
volatility	-	-	-
num_paths	-	-	-
seed	-	-	-
accrued_average	-	-	-

_value_mc_fast_numba

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_mc_fast_numba(t0: float,
                     t: float,
                     tau: float,
                     K: float,
                     n: int,
                     option_type: int,
                     stock_price: float,
                     interest_rate: float,
                     dividend_yield: float,
                     volatility: float,
                     num_paths: int,
                     seed: int,
                     accrued_average: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t0	float	-	-
t	float	-	-
tau	float	-	-
K	float	-	-
n	int	-	-
option_type	int	-	-
stock_price	float	-	-
interest_rate	float	-	-
dividend_yield	float	-	-
volatility	float	-	-
num_paths	int	-	-
seed	int	-	-
accrued_average	float	-	-

_value_mc_fast_cv_numba

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_mc_fast_cv_numba(t0, t, tau, K, n, option_type, stock_price,
                        interest_rate, dividend_yield, volatility, num_paths,
                        seed, accrued_average, v_g_exact):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t0	-	-	-
t	-	-	-
tau	-	-	-
K	-	-	-
n	-	-	-
option_type	-	-	-
stock_price	-	-	-
interest_rate	-	-	-
dividend_yield	-	-	-
volatility	-	-	-
num_paths	-	-	-
seed	-	-	-
accrued_average	-	-	-
v_g_exact	-	-	-

5.3 equity_barrier_option

Class: EquityBarrierOption(EquityOption)

Class to hold details of an Equity Barrier Option. It also calculates the option price using Black Scholes for 8 different variants on the Barrier structure in enum EquityBarrierTypes.

EquityBarrierOption

Create the EquityBarrierOption by specifying the expiry date, strike price, option type, barrier level, the number of observations per year and the notional.

```
EquityBarrierOption(expiry_dt: Date,
                    strike_price: float,
                    option_type: EquityBarrierTypes,
                    barrier_level: float,
                    num_obs_per_year: (int, float) = 252,
                    notional: float = 1.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
strike_price	float	-	-
option_type	EquityBarrierTypes	-	-
barrier_level	float	-	-
num_obs_per_year	int or float	-	252
notional	float	-	1.0

value

This prices an Equity Barrier option using the formulae given in the paper by Clewlow, Llanos and Strickland December 1994 which can be found at https://warwick.ac.uk/fac/soc/wbs/subjects/finance/research/wpaperseries/1994/94-54.pdf

```
value(value_dt: Date,
      stock_price: (float, np.ndarray),
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float or np.ndarray	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

value_mc

A Monte-Carlo based valuation of the barrier option which simulates the evolution of the stock price of at a specified number of annual observation times until expiry to examine if the barrier has been crossed and the corresponding value of the final payoff, if any. It assumes a GBM model for the stock price.

```
value_mc(t: float,
         k,
         option_type: int,
         b,
         notional,
         s: float,
         r: float,
         process_type,
         model_params,
         num_ann_obs: int = 252,
         num_paths: int = 10000,
         seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	float	-	-
k	-	-	-
option_type	int	-	-
b	-	-	-
notional	-	-	-
s	float	-	-
r	float	-	-
process_type	-	-	-
model_params	-	-	-
num_ann_obs	int	-	252
num_paths	int	-	10000
seed	int	-	4242

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

5.4 equity_basket_option

Class: EquityBasketOption

A EquityBasketOption is a contract to buy a put or a call option on an equally weighted portfolio of different stocks, each with its own price, volatility and dividend yield. An analytical and monte-carlo pricing model have been implemented for a European style option.

EquityBasketOption

Define the EquityBasket option by specifying its expiry date, its strike price, whether it is a put or call, and the number of underlying stocks in the basket.

```
EquityBasketOption(expiry_dt: Date,
                   strike_price: float,
                   option_type: OptionTypes,
                   num_assets: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
strike_price	float	-	-
option_type	OptionTypes	-	-
num_assets	int	-	-

_validate

PLEASE ADD A FUNCTION DESCRIPTION

```
_validate(stock_prices,
          dividend_yields,
          volatilities,
          correlations):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stock_prices	-	-	-
dividend_yields	-	-	-
volatilities	-	-	-
correlations	-	-	-

value

Basket valuation using a moment matching method to approximate the effective variance of the underlying basket value. This approach is able to handle a full rank correlation structure between the individual assets.

```
value(value_dt: Date,
      stock_prices: np.ndarray,
      discount_curve: DiscountCurve,
      dividend_curves: (list),
      volatilities: np.ndarray,
      correlations: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_prices	np.ndarray	-	-
discount_curve	DiscountCurve	-	-
dividend_curves	list or (list	-	-
volatilities	np.ndarray	-	-
correlations	np.ndarray	-	-

value_mc

Valuation of the EquityBasketOption using a Monte-Carlo simulation of stock prices assuming a GBM distribution. Cholesky decomposition is used to handle a full rank correlation structure between the individual assets. The num_paths and seed are pre-set to default values but can be overwritten.

```
value_mc(value_dt: Date,
         stock_prices: np.ndarray,
         discount_curve: DiscountCurve,
         dividend_curves: (list),
         volatilities: np.ndarray,
         corr_matrix: np.ndarray,
         num_paths: int = 10000,
         seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_prices	np.ndarray	-	-
discount_curve	DiscountCurve	-	-
dividend_curves	list or (list	-	-
volatilities	np.ndarray	-	-
corr_matrix	np.ndarray	-	-
num_paths	int	-	10000
seed	int	-	4242

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

5.5 equity_binomial_tree

Enumerated Type: EquityTreePayoffTypes

This enumerated type has the following values:
- FWD_CONTRACT
- VANILLA_OPTION
- DIGITAL_OPTION
- POWER_CONTRACT
- POWER_OPTION
- LOG_CONTRACT
- LOG_OPTION

Enumerated Type: EquityTreeExerciseTypes

This enumerated type has the following values:
- EUROPEAN
- AMERICAN

Class: EquityBinomialTree()

class EquityBinomialTree():

EquityBinomialTree

PLEASE ADD A FUNCTION DESCRIPTION

```
EquityBinomialTree():
```

The function arguments are described in the following table.

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(stock_price,
      discount_curve,
      dividend_curve,
      volatility,
      num_steps,
      value_dt,
      payoff,
      expiry_dt,
      payoff_type,
      exercise_type,
      payoff_params):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stock_price	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
volatility	-	-	-
num_steps	-	-	-
value_dt	-	-	-
payoff	-	-	-
expiry_dt	-	-	-
payoff_type	-	-	-
exercise_type	-	-	-
payoff_params	-	-	-

_validate_payoff

PLEASE ADD A FUNCTION DESCRIPTION

```
_validate_payoff(payoff_type, payoff_params):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
payoff_type	-	-	-
payoff_params	-	-	-

_payoff_value

PLEASE ADD A FUNCTION DESCRIPTION

```
_payoff_value(s, payoff_type, payoff_params):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
payoff_type	-	-	-
payoff_params	-	-	-

_value_once

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_once(stock_price,
            r,
            q,
            volatility,
            num_steps,
```

```
            time_to_expiry,
            payoff_type,
            exercise_type,
            payoff_params):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stock_price	-	-	-
r	-	-	-
q	-	-	-
volatility	-	-	-
num_steps	-	-	-
time_to_expiry	-	-	-
payoff_type	-	-	-
exercise_type	-	-	-
payoff_params	-	-	-

5.6 equity_chooser_option

Class: EquityChooserOption(EquityOption)

A EquityChooserOption is an option which allows the holder to either enter into a call or a put option on a later expiry date, with both strikes potentially different and both expiry dates potentially different. This is known as a complex chooser. All the option details are set at trade initiation.

EquityChooserOption

Create the EquityChooserOption by passing in the chooser date and then the put and call expiry dates as well as the corresponding put and call strike prices.

```
EquityChooserOption(choose_dt: Date,
                    call_expiry_dt: Date,
                    put_expiry_dt: Date,
                    call_strike_price: float,
                    put_strike_price: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
choose_dt	Date	-	-
call_expiry_dt	Date	-	-
put_expiry_dt	Date	-	-
call_strike_price	float	-	-
put_strike_price	float	-	-

value

Value the complex chooser option using an approach by Rubinstein (1991). See also Haug page 129 for complex chooser options.

```
value(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

value_mc

Value the complex chooser option Monte Carlo.

```
value_mc(value_dt: Date,
         stock_price: float,
         discount_curve: DiscountCurve,
         dividend_curve: DiscountCurve,
         model,
         num_paths: int = 10000,
         seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-
num_paths	int	-	10000
seed	int	-	4242

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_f

Complex chooser option solve for critical stock price that makes the forward starting call and put options have the same price on the chooser date.

```
_f(ss, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
ss	-	-	-
*args	-	-	-

5.7 equity_cliquet_option

Class: EquityCliquetOption(EquityOption)

A EquityCliquetOption is a series of options which start and stop at successive times with each subsequent option resetting its strike to be ATM at the start of its life. This is also known as a reset option.

EquityCliquetOption

Create the EquityCliquetOption by passing in the start date and the end date and whether it is a call or a put. Some additional data is needed in order to calculate the individual payments.

```
EquityCliquetOption(start_dt: Date,
                    final_expiry_dt: Date,
                    option_type: OptionTypes,
                    freq_type: FrequencyTypes,
                    day_count_type: DayCountTypes = DayCountTypes.THIRTY_E_360,
                    cal_type: CalendarTypes = CalendarTypes.WEEKEND,
                    bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
                    dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_dt	Date	-	-
final_expiry_dt	Date	-	-
option_type	OptionTypes	-	-
freq_type	FrequencyTypes	-	-
day_count_type	DayCountTypes	-	DayCountTypes.THIRTY_E_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value

Value the cliquet option as a sequence of options using the Black- Scholes model.

```
value(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-

print_payments

PLEASE ADD A FUNCTION DESCRIPTION

```
print_payments():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

5.8 equity_compound_option

Class: EquityCompoundOption(EquityOption)

A EquityCompoundOption is a compound option which allows the holder to either buy or sell another underlying option on a first expiry date that itself expires on a second expiry date. Both strikes are set at trade initiation.

EquityCompoundOption

Create the EquityCompoundOption by passing in the first and second expiry dates as well as the corresponding strike prices and option types.

```
EquityCompoundOption(cExpiryDate: Date,  # Compound Option expiry date
                     cOptionType: OptionTypes,  # Compound option type
                     cStrikePrice: float,  # Compound option strike
                     uExpiryDate: Date,  # Underlying option expiry date
                     uOptionType: OptionTypes,  # Underlying option type
                     uStrikePrice: float):  # Underlying option strike price
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cExpiryDate	Date	Compound Option expiry date	-
cOptionType	OptionTypes	Compound option type	-
cStrikePrice	float	Compound option strike	-
uExpiryDate	Date	Underlying option expiry date	-
uOptionType	OptionTypes	Underlying option type	-
uStrikePrice	float	Underlying option strike price	-

value

Value the compound option using an analytical approach if it is entirely European style. Otherwise use a Tree approach to handle the early exercise. Solution by Geske (1977), Hodges and Selby (1987) and Rubinstein (1991). See also Haug page 132.

```
value(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model,
      num_steps: int = 200):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-
num_steps	int	-	200

_value_tree

This function is called if the option has American features.

```
_value_tree(value_dt,
            stock_price,
            discount_curve,
            dividend_curve,
            model,
            num_steps=200):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
model	-	-	-
num_steps	-	-	200

_implied_stock_price

PLEASE ADD A FUNCTION DESCRIPTION

```
_implied_stock_price(stock_price,
                     expiry_dt1,
                     expiry_dt2,
                     strike_price1,
                     strike_price2,
                     option_type2,
                     interest_rate,
                     dividend_yield,
                     model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stock_price	-	-	-
expiry_dt1	-	-	-
expiry_dt2	-	-	-
strike_price1	-	-	-
strike_price2	-	-	-
option_type2	-	-	-
interest_rate	-	-	-
dividend_yield	-	-	-
model	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_f

PLEASE ADD A FUNCTION DESCRIPTION

```
_f(s0, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s0	-	-	-
*args	-	-	-

_value_once

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_once(s,
            r,
            q,
```

```
            volatility,
            t1, t2,
            option_type1, option_type2,
            k1, k2,
            num_steps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
r	-	-	-
q	-	-	-
volatility	-	-	-
t1	-	-	-
t2	-	-	-
option_type1	-	-	-
option_type2	-	-	-
k1	-	-	-
k2	-	-	-
num_steps	-	-	-

5.9 equity_digital_option

Enumerated Type: FinDigitalOptionTypes

This enumerated type has the following values:

- CASH_OR_NOTHING
- ASSET_OR_NOTHING

Class: EquityDigitalOption(EquityOption)

A EquityDigitalOption is an option in which the buyer receives some payment if the stock price has crossed a barrier ONLY at expiry and zero otherwise. There are two types: cash-or-nothing and the asset-or-nothing option. We do not care whether the stock price has crossed the barrier today, we only care about the barrier at option expiry. For a continuously- monitored barrier, use the EquityOneTouchOption class.

EquityDigitalOption

Create the digital option by specifying the expiry date, the barrier price and the type of option which is either a EUROPEAN_CALL or a EUROPEAN_PUT or an AMERICAN_CALL or AMERICAN_PUT. There are two types of underlying - cash or nothing and asset or nothing.

```
EquityDigitalOption(expiry_dt: Date,
                    barrier: float,
                    call_put_type: OptionTypes,
                    digital_type: FinDigitalOptionTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
barrier	float	-	-
call_put_type	OptionTypes	-	-
digital_type	FinDigitalOptionTypes	-	-

value

Digital Option valuation using the Black-Scholes model assuming a barrier at expiry. Handles both cash-or-nothing and asset-or-nothing options.

```
value(value_dt: Date,
      s: (float, np.ndarray),
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
s	float or np.ndarray	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

value_mc

Digital Option valuation using the Black-Scholes model and Monte Carlo simulation. Product assumes a barrier only at expiry. Monte Carlo handles both a cash-or-nothing and an asset-or-nothing option.

```
value_mc(value_dt: Date,
         stock_price: float,
         discount_curve: DiscountCurve,
         dividend_curve: DiscountCurve,
         model,
         num_paths: int = 10000,
         seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-
num_paths	int	-	10000
seed	int	-	4242

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

5.10 equity_fixed_lookback_option

Class: EquityFixedLookbackOption(EquityOption)

This is an equity option in which the strike of the option is fixed but the value of the stock price used to determine the payoff is the maximum in the case of a call option, and a minimum in the case of a put option.

EquityFixedLookbackOption

Create the FixedLookbackOption by specifying the expiry date, the option type and the option strike.

```
EquityFixedLookbackOption(expiry_dt: Date,
                          option_type: OptionTypes,
                          strike_price: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
option_type	OptionTypes	-	-
strike_price	float	-	-

value

Valuation of the Fixed Lookback option using Black-Scholes using the formulae derived by Conze and Viswanathan (1991). One of the inputs is the minimum of maximum of the stock price since the start of the option depending on whether the option is a call or a put.

```
value(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      volatility: float,
      stock_min_max: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
volatility	float	-	-
stock_min_max	float	-	-

value_mc

Monte Carlo valuation of a fixed strike lookback option using a Black-Scholes model that assumes the stock follows a GBM process.

```
value_mc(value_dt: Date,
         stock_price: float,
         discount_curve: DiscountCurve,
         dividend_curve: DiscountCurve,
         volatility: float,
         stock_min_max: float,
         num_paths: int = 10000,
         num_steps_per_year: int = 252,
         seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
volatility	float	-	-
stock_min_max	float	-	-
num_paths	int	-	10000
num_steps_per_year	int	-	252
seed	int	-	4242

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

5.11 equity_float_lookback_option

Class: EquityFloatLookbackOption(EquityOption)

This is 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.

EquityFloatLookbackOption

Create the FloatLookbackOption by specifying the expiry date and the option type. The strike is determined internally as the maximum or minimum of the stock price depending on whether it is a put or a call option.

```
EquityFloatLookbackOption(expiry_dt: Date,
                          option_type: OptionTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
option_type	OptionTypes	-	-

value

Valuation of the Floating Lookback option using Black-Scholes using the formulae derived by Goldman, Sosin and Gatto (1979).

```
value(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      volatility: float,
      stock_min_max: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
volatility	float	-	-
stock_min_max	float	-	-

value_mc

Monte Carlo valuation of a floating strike lookback option using a Black-Scholes model that assumes the stock follows a GBM process.

```
value_mc(value_dt: Date,
         stock_price: float,
         discount_curve: DiscountCurve,
         dividend_curve: DiscountCurve,
         volatility: float,
         stock_min_max: float,
         num_paths: int = 10000,
         num_steps_per_year: int = 252,
         seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
volatility	float	-	-
stock_min_max	float	-	-
num_paths	int	-	10000
num_steps_per_year	int	-	252
seed	int	-	4242

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

5.12 equity_forward

Class: EquityForward()

EquityForward

Creates a EquityForward which allows the owner to buy the stock at a price agreed today. Need to specify if LONG or SHORT.

```
EquityForward(expiry_dt: Date,
              forward_price: float,  # PRICE OF 1 UNIT OF FOREIGN IN DOM CCY
              notional: float,
              long_short: FinLongShort = FinLongShort.LONG):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
forward_price	float	PRICE OF 1 UNIT OF FOREIGN IN DOM CCY	-
notional	float	-	-
long_short	FinLongShort	-	LONG

value

Calculate the value of an equity forward contract from the stock price and discount and dividend discount.

```
value(value_dt,
      stock_price,  # Current stock price
      discount_curve,
      dividend_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	Current stock price	-
discount_curve	-	-	-
dividend_curve	-	-	-

forward

Calculate the forward price of the equity forward contract.

```
forward(value_dt,
        stock_price,  # Current stock price
        discount_curve,
        dividend_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	Current stock price	-
discount_curve	-	-	-
dividend_curve	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

5.13 equity_index_option

Class: EquityIndexOption

Class for managing plain vanilla European/American calls and puts on equity indices.

EquityIndexOption

Create the Equity Index option object by specifying the expiry date, the option strike, the option type and the number of options.

```
EquityIndexOption(expiry_dt: Union[Date, list],
                  strike_price: Union[float, np.ndarray],
                  option_type: OptionTypes,
                  num_options: Optional[float] = 1.0,
                  ):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date or list	-	-
strike_price	float or np.ndarray	-	-
option_type	OptionTypes	-	-
num_options	Optional[float]	-	1.0

value

Equity Index Option valuation using Black model.

```
value(value_dt: Union[Date, list],
      forward_price: float,
      discount_curve: DiscountCurve,
      model: Model,
      ):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date or list	-	-
forward_price	float	-	-
discount_curve	DiscountCurve	-	-
model	Model	-	-

delta

Calculate delta of a European/American Index option.

```
delta(value_dt: Date,
      forward_price: float,
```

```
      discount_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
forward_price	float	-	-
discount_curve	DiscountCurve	-	-
model	-	-	-

gamma

Calculate gamma of a European/American Index option.

```
gamma(value_dt: Date,
      forward_price: float,
      discount_curve: DiscountCurve,
      model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
forward_price	float	-	-
discount_curve	DiscountCurve	-	-
model	Model	-	-

vega

Calculate vega of a European/American Index option.

```
vega(value_dt: Date,
     forward_price: float,
     discount_curve: DiscountCurve,
     model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
forward_price	float	-	-
discount_curve	DiscountCurve	-	-
model	Model	-	-

theta

Calculate theta of a European/American Index option.

```
theta(value_dt: Date,
      forward_price: float,
      discount_curve: DiscountCurve,
      model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
forward_price	float	-	-
discount_curve	DiscountCurve	-	-
model	Model	-	-

implied_volatility

Calculate the Black implied volatility of a European/American Index option.

```
implied_volatility(value_dt: Date,
                   forward_price: Union[float, list, np.ndarray],
                   discount_curve: DiscountCurve,
                   model: Model,
                   price: float,
                   ):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
forward_price	float or list,np.ndarray	-	-
discount_curve	DiscountCurve	-	-
model	Model	-	-
price	float	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

5.14 equity_model_types

Class: EquityModel

EquityModel

PLEASE ADD A FUNCTION DESCRIPTION

```
EquityModel():
```

The function arguments are described in the following table.

Class: EquityModelHeston(EquityModel)

class EquityModelHeston(EquityModel):

EquityModelHeston

PLEASE ADD A FUNCTION DESCRIPTION

```
EquityModelHeston(volatility, mean_reversion):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	-	-	-
mean_reversion	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

5.15 equity_one_touch_option

Class: EquityOneTouchOption(EquityOption)

A EquityOneTouchOption is an option in which the buyer receives one unit of cash OR stock if the stock price touches a barrier at any time before the option expiry date and zero otherwise. The choice of cash or stock is made at trade initiation. The single barrier payoff must define whether the option pays or cancels if the barrier is touched and also when the payment is made (at hit time or option expiry). All of these variants are all members of the FinTouchOptionTypes enumerated type.

EquityOneTouchOption

Create the one touch option by defining its expiry date and the barrier level and a payment size if it is a cash
.

```
EquityOneTouchOption(expiry_dt: Date,
                     option_type: TouchOptionTypes,
                     barrier_price: float,
                     payment_size: float = 1.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
option_type	TouchOptionTypes	-	-
barrier_price	float	-	-
payment_size	float	-	1.0

value

Equity One-Touch Option valuation using the Black-Scholes model assuming a continuous (American) barrier from value date to expiry. Handles both cash-or-nothing and asset-or-nothing options.

```
value(value_dt: Date,
      stock_price: (float, np.ndarray),
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float or np.ndarray	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

value_mc

Touch Option valuation using the Black-Scholes model and Monte Carlo simulation. Accuracy is not great when compared to the analytical result as we only observe the barrier a finite number of times. The convergence is slow.

```
value_mc(value_dt: Date,
         stock_price: float,
         discount_curve: DiscountCurve,
         dividend_curve: DiscountCurve,
         model,
         num_paths: int = 10000,
         num_steps_per_year: int = 252,
         seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-
num_paths	int	-	10000
num_steps_per_year	int	-	252
seed	int	-	4242

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_barrier_pay_one_at_hit_pv_down

Pay $1 if the stock crosses the barrier H from above. PV payment.

```
_barrier_pay_one_at_hit_pv_down(s, H, r, dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
H	-	-	-
r	-	-	-
dt	-	-	-

_barrier_pay_one_at_hit_pv_up

Pay $1 if the stock crosses the barrier H from below. PV payment.

```
_barrier_pay_one_at_hit_pv_up(s, H, r, dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
H	-	-	-
r	-	-	-
dt	-	-	-

_barrier_pay_asset_at_expiry_down_out

Pay $1 if the stock crosses the barrier H from above. PV payment.

```
_barrier_pay_asset_at_expiry_down_out(s, H):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
H	-	-	-

_barrier_pay_asset_at_expiry_up_out

Pay $1 if the stock crosses the barrier H from below. PV payment.

```
_barrier_pay_asset_at_expiry_up_out(s, H):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
H	-	-	-

5.16 equity_option

Enumerated Type: EquityOptionModelTypes

This enumerated type has the following values:

- BLACKSCHOLES
- ANOTHER

Class: EquityOption

This class is a parent class for all option classes that require any perturbatory risk.

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_yield: float,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_yield	float	-	-
model	-	-	-

delta

Calculation of option delta by perturbation of stock price and revaluation.

```
delta(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

gamma

Calculation of option gamma by perturbation of stock price and revaluation.

```
gamma(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

vega

Calculation of option vega by perturbing vol and revaluation.

```
vega(value_dt: Date,
     stock_price: float,
     discount_curve: DiscountCurve,
     dividend_curve: DiscountCurve,
     model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

vanna

Calculation of option vanna by perturbing delta with respect to the stock price volatility.

```
vanna(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

theta

Calculation of option theta by perturbing value date by one calendar date (not a business date) and then doing revaluation and calculating the difference divided by dt = 1 / gDaysInYear.

```
theta(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

rho

Calculation of option rho by perturbing interest rate and revaluation.

```
rho(value_dt: Date,
    stock_price: float,
    discount_curve: DiscountCurve,
    dividend_curve: DiscountCurve,
    model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

5.17 equity_rainbow_option

Enumerated Type: EquityRainbowOptionTypes

This enumerated type has the following values:

- CALL_ON_MAXIMUM
- PUT_ON_MAXIMUM
- CALL_ON_MINIMUM
- PUT_ON_MINIMUM
- CALL_ON_NTH
- PUT_ON_NTH

Class: EquityRainbowOption(EquityOption)

class EquityRainbowOption(EquityOption):

EquityRainbowOption

PLEASE ADD A FUNCTION DESCRIPTION

```
EquityRainbowOption(expiry_dt: Date,
                    payoff_type: EquityRainbowOptionTypes,
                    payoff_params: List[float],
                    num_assets: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
payoff_type	EquityRainbowOptionTypes	-	-
payoff_params	List[float]	-	-
num_assets	int	-	-

_validate

PLEASE ADD A FUNCTION DESCRIPTION

```
_validate(stock_prices,
          dividend_curves,
          volatilities,
          betas):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stock_prices	-	-	-
dividend_curves	-	-	-
volatilities	-	-	-
betas	-	-	-

_validate_payoff

PLEASE ADD A FUNCTION DESCRIPTION

```
_validate_payoff(payoff_type, payoff_params, num_assets):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
payoff_type	-	-	-
payoff_params	-	-	-
num_assets	-	-	-

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(value_dt: Date,
      stock_prices: np.ndarray,
      discount_curve: DiscountCurve,
      dividend_curves: (list),
      volatilities: np.ndarray,
      corr_matrix: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_prices	np.ndarray	-	-
discount_curve	DiscountCurve	-	-
dividend_curves	list or (list	-	-
volatilities	np.ndarray	-	-
corr_matrix	np.ndarray	-	-

value_mc

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc(value_dt,
         stock_prices,
         discount_curve,
         dividend_curves,
         volatilities,
         corr_matrix,
         num_paths=10000,
         seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_prices	-	-	-
discount_curve	-	-	-
dividend_curves	-	-	-
volatilities	-	-	-
corr_matrix	-	-	-
num_paths	-	-	10000
seed	-	-	4242

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

payoff_value

PLEASE ADD A FUNCTION DESCRIPTION

```
payoff_value(s, payoff_typeValue, payoff_params):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
payoff_typeValue	-	-	-
payoff_params	-	-	-

value_mc_fast

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc_fast(t,
              stock_prices,
              discount_curve,
```

```
                dividend_curves,
                volatilities,
                betas,
                num_assets,
                payoff_type,
                payoff_params,
                num_paths=10000,
                seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
stock_prices	-	-	-
discount_curve	-	-	-
dividend_curves	-	-	-
volatilities	-	-	-
betas	-	-	-
num_assets	-	-	-
payoff_type	-	-	-
payoff_params	-	-	-
num_paths	-	-	10000
seed	-	-	4242

5.18 equity_swap

Class: EquitySwap

Class for managing a standard Equity vs Float leg swap. This is a contract in which an equity payment leg is exchanged for a series of floating rates payments. There is no exchange of principal. The contract is entered into at zero initial cost when spreads are zero. The contract lasts from an effective date to a specified maturity date.

The equity payments are not known fully until the end of the payment period.

The floating rate is not known fully until the end of the preceding payment period. It is set in advance and paid in arrears.

The value of the contract is the NPV of the two coupon streams. Discounting is done on a supplied discount curve which is separate from the curve from which the implied index rates are extracted.

EquitySwap

Create an equity swap contract given the contract effective date, its maturity, underlying price and quantity, day count convention and return type and other details. The equity 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.

```
EquitySwap(effective_dt: Date,  # Date contract starts or last Equity Reset
           term_dt_or_tenor: (Date, str),  # Date contract ends
           eq_leg_type: SwapTypes,
           eq_freq_type: FrequencyTypes,
           eq_dc_type: DayCountTypes,
           strike: float,  # Price at effective date
           quantity: float = 1.0,  # Quantity at effective date
           eq_payment_lag: int = 0,
           eq_return_type: ReturnTypes = ReturnTypes.TOTAL_RETURN,
           rate_freq_type: FrequencyTypes = FrequencyTypes.MONTHLY,
           rate_dc_type: DayCountTypes = DayCountTypes.ACT_360,
           rate_spread: float = 0.0,
           rate_payment_lag: int = 0,
           cal_type: CalendarTypes = CalendarTypes.WEEKEND,
           bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
           dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD,
           end_of_month: bool = False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	Date contract starts or last Equity Reset	-
term_dt_or_tenor	Date or str	Date contract ends	-
eq_leg_type	SwapTypes	-	-
eq_freq_type	FrequencyTypes	-	-
eq_dc_type	DayCountTypes	-	-
strike	float	Price at effective date	-
quantity	float	Quantity at effective date	1.0
eq_payment_lag	int	-	0
eq_return_type	ReturnTypes	-	ReturnTypes.TOTAL_RETURN
rate_freq_type	FrequencyTypes	-	FrequencyTypes.MONTHLY
rate_dc_type	DayCountTypes	-	DayCountTypes.ACT_360
rate_spread	float	-	0.0
rate_payment_lag	int	-	0
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD
end_of_month	bool	-	False

value

Value the Equity swap on a valuation date.

```
value(value_dt: Date,
      discount_curve: DiscountCurve,
      index_curve: DiscountCurve = None,
      dividend_curve: DiscountCurve = None,
      current_price: float = None,
      firstFixingRate=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
discount_curve	DiscountCurve	-	-
index_curve	DiscountCurve	-	None
dividend_curve	DiscountCurve	-	None
current_price	float	-	None
firstFixingRate	-	-	None

_fill_rate_notional_array

In an equity swap, at every equity reset, the notional of the contract is updated to reflect the new underlying price. This is a helper function that takes the Equity Notional list from Equity Leg and convert it to a Notional array that fits the payment schedule defined for the rate leg.

```
_fill_rate_notional_array():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

5.19 equity_swap_leg

Class: EquitySwapLeg

Class for managing the equity leg of an equity swap. An equity leg is a leg with a sequence of flows calculated according to an ISDA schedule and follows the economics of a collection of equity forward contracts.

EquitySwapLeg

Create the equity leg of a swap contract giving the contract start date, its maturity, underlying strike price and quantity, payment frequency, day count convention, return type, and other details

```
EquitySwapLeg(effective_dt: Date,  # Contract starts or last equity reset
              term_dt_or_tenor: (Date, str),  # Date contract ends
              leg_type: SwapTypes,
              freq_type: FrequencyTypes,
              dc_type: DayCountTypes,
              strike: float,  # Price at effective date
              quantity: float = 1.0,  # Quantity at effective date
              payment_lag: int = 0,
              return_type: ReturnTypes = ReturnTypes.TOTAL_RETURN,
              cal_type: CalendarTypes = CalendarTypes.WEEKEND,
              bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
              dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD,
              end_of_month: bool = False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	Contract starts or last equity reset	-
term_dt_or_tenor	Date or str	Date contract ends	-
leg_type	SwapTypes	-	-
freq_type	FrequencyTypes	-	-
dc_type	DayCountTypes	-	-
strike	float	Price at effective date	-
quantity	float	Quantity at effective date	1.0
payment_lag	int	-	0
return_type	ReturnTypes	-	ReturnTypes.TOTAL_RETURN
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD
end_of_month	bool	-	False

generate_payment_dts

Generate the Equity leg payment dates and accrual factors. Similar to swap float leg, payment values can't be generated, as we do not have index curve, dividend curve and equity price.

```
generate_payment_dts():
```

The function arguments are described in the following table.

value

Value the equity leg with payments from an equity price, quantity, an index curve and an [optional] dividend curve. Discounting is based on a supplied discount curve as of the valuation date supplied.

```
value(value_dt: Date,
      discount_curve: DiscountCurve,
      index_curve: DiscountCurve,
      dividend_curve: DiscountCurve = None,
      current_price: float = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
discount_curve	DiscountCurve	-	-
index_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	None
current_price	float	-	None

print_payments

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

```
print_payments():
```

The function arguments are described in the following table.

print_valuation

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

```
print_valuation():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

5.20 equity_vanilla_option

Class: EquityVanillaOption()

Class for managing plain vanilla European calls and puts on equities. For American calls and puts see the EquityAmericanOption class.

EquityVanillaOption

Create the Equity Vanilla option object by specifying the expiry date, the option strike, the option type and the number of options.

```
EquityVanillaOption(expiry_dt: (Date, list),
                    strike_price: (float, np.ndarray),
                    option_type: (OptionTypes, list),
                    num_options: float = 1.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date or list	-	-
strike_price	float or np.ndarray	-	-
option_type	OptionTypes or list	-	-
num_options	float	-	1.0

intrinsic

Equity Vanilla Option valuation using Black-Scholes model.

```
intrinsic(value_dt: (Date, list),
          stock_price: (np.ndarray, float),
          discount_curve: DiscountCurve,
          dividend_curve: DiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date or list	-	-
stock_price	np.ndarray or float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-

value

Equity Vanilla Option valuation using Black-Scholes model.

```
value(value_dt: (Date, list),
      stock_price: (np.ndarray, float),
      discount_curve: DiscountCurve,
```

```
      dividend_curve: DiscountCurve,
      model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date or list	-	-
stock_price	np.ndarray or float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-

delta

Calculate the analytical delta of a European vanilla option.

```
delta(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	-	-	-

gamma

Calculate the analytical gamma of a European vanilla option.

```
gamma(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-

vega

Calculate the analytical vega of a European vanilla option.

```
vega(value_dt: Date,
     stock_price: float,
     discount_curve: DiscountCurve,
     dividend_curve: DiscountCurve,
     model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-

theta

Calculate the analytical theta of a European vanilla option.

```
theta(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-

rho

Calculate the analytical rho of a European vanilla option.

```
rho(value_dt: Date,
    stock_price: float,
    discount_curve: DiscountCurve,
    dividend_curve: DiscountCurve,
    model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-

vanna

Calculate the analytical vanna of a European vanilla option.

```
vanna(value_dt: Date,
      stock_price: float,
      discount_curve: DiscountCurve,
      dividend_curve: DiscountCurve,
      model: Model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-

implied_volatility

Calculate the Black-Scholes implied volatility of a European vanilla option.

```
implied_volatility(value_dt: Date,
                   stock_price: (float, list, np.ndarray),
                   discount_curve: DiscountCurve,
                   dividend_curve: DiscountCurve,
                   price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float or list,np.ndarray	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
price	-	-	-

value_mc_numpy_only

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc_numpy_only(value_dt: Date,
                    stock_price: float,
                    discount_curve: DiscountCurve,
                    dividend_curve: DiscountCurve,
                    model: Model,
                    num_paths: int = 10000,
                    seed: int = 4242,
                    use_sobol: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-
num_paths	int	-	10000
seed	int	-	4242
use_sobol	int	-	0

value_mc_numba_only

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc_numba_only(value_dt: Date,
                    stock_price: float,
                    discount_curve: DiscountCurve,
                    dividend_curve: DiscountCurve,
                    model: Model,
                    num_paths: int = 10000,
                    seed: int = 4242,
                    use_sobol: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-
num_paths	int	-	10000
seed	int	-	4242
use_sobol	int	-	0

value_mc_numba_parallel

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc_numba_parallel(value_dt: Date,
                        stock_price: float,
                        discount_curve: DiscountCurve,
                        dividend_curve: DiscountCurve,
                        model: Model,
                        num_paths: int = 10000,
                        seed: int = 4242,
                        use_sobol: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-
num_paths	int	-	10000
seed	int	-	4242
use_sobol	int	-	0

value_mc_numpy_numba

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc_numpy_numba(value_dt: Date,
                     stock_price: float,
                     discount_curve: DiscountCurve,
                     dividend_curve: DiscountCurve,
                     model: Model,
                     num_paths: int = 10000,
                     seed: int = 4242,
                     use_sobol: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-
num_paths	int	-	10000
seed	int	-	4242
use_sobol	int	-	0

value_mc_nonumba_nonumpy

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc_nonumba_nonumpy(value_dt: Date,
                         stock_price: float,
                         discount_curve: DiscountCurve,
                         dividend_curve: DiscountCurve,
                         model: Model,
                         num_paths: int = 10000,
                         seed: int = 4242,
                         use_sobol: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-
num_paths	int	-	10000
seed	int	-	4242
use_sobol	int	-	0

value_mc

Value European style call or put option using Monte Carlo. This is mainly for educational purposes. Sobol numbers can be used.

```
value_mc(value_dt: Date,
         stock_price: float,
         discount_curve: DiscountCurve,
         dividend_curve: DiscountCurve,
         model: Model,
         num_paths: int = 10000,
         seed: int = 4242,
         use_sobol: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
discount_curve	DiscountCurve	-	-
dividend_curve	DiscountCurve	-	-
model	Model	-	-
num_paths	int	-	10000
seed	int	-	4242
use_sobol	int	-	0

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_f

PLEASE ADD A FUNCTION DESCRIPTION

```
_f(v, args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
v	-	-	-
args	-	-	-

_fvega

PLEASE ADD A FUNCTION DESCRIPTION

```
_fvega(v, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
v	-	-	-
*args	-	-	-

5.21 equity_variance_swap

Class: EquityVarianceSwap

EquityVarianceSwap

Create variance swap contract.

```
EquityVarianceSwap(start_dt: Date,
                   maturity_dt_or_tenor: (Date, str),
                   strike_variance: float,
                   notional: float = ONE_MILLION,
                   pay_strike_flag: bool = True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_dt	Date	-	-
maturity_dt_or_tenor	Date or str	-	-
strike_variance	float	-	-
notional	float	-	ONE_MILLION
pay_strike_flag	bool	-	True

value

Calculate the value of the variance swap based on the realised volatility to the valuation date, the forward looking implied volatility to the maturity date using the libor discount curve.

```
value(value_dt,
      realisedVar,
      fair_strikeVar,
      libor_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
realisedVar	-	-	-
fair_strikeVar	-	-	-
libor_curve	-	-	-

fair_strike_approx

This is an approximation of the fair strike variance by Demeterfi et al. (1999) which assumes that sigma(K) = sigma(F) - b(K-F)/F where F is the forward stock price and sigma(F) is the ATM forward vol.

```
fair_strike_approx(value_dt,
                   fwdStockPrice,
                   strikes,
                   volatilities):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
fwdStockPrice	-	-	-
strikes	-	-	-
volatilities	-	-	-

fair_strike

Calculate the implied variance according to the volatility surface using a static replication methodology with a specially weighted portfolio of put and call options across a range of strikes using the approximate method set out by Demeterfi et al. 1999.

```
fair_strike(value_dt,
            stock_price,
            dividend_curve,
            volatility_curve,
            num_call_options,
            num_put_options,
            strike_spacing,
            discount_curve,
            use_forward=True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	-	-
dividend_curve	-	-	-
volatility_curve	-	-	-
num_call_options	-	-	-
num_put_options	-	-	-
strike_spacing	-	-	-
discount_curve	-	-	-
use_forward	-	-	True

f

PLEASE ADD A FUNCTION DESCRIPTION

```
f(x): return (2.0/tmat)*((x-sstar)/sstar-np.log(x/sstar))
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x return (2.0/tmat)*((x-sstar)/sstar-np.log(x/sstar))	-	-	-

realised_variance

Calculate the realised variance according to market standard calculations which can either use log or percentage returns.

```
realised_variance(closePrices, useLogs=True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
closePrices	-	-	-
useLogs	-	-	True

print_weights

Print the list of puts and calls used to replicate the static replication component of the variance swap hedge.

```
print_weights():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

5.22 __init__

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 IborSingleCurve discount curve. It also contains a IborCurve object for discounting. It has methods for fitting the curve and also for extracting survival probabilities.

6.1 cds

Class: CDS

A class which manages a Credit Default Swap. It performs schedule generation and the valuation and risk management of CDS.

CDS

Create a CDS from the step-in date, maturity date and coupon

```
CDS(step_in_dt: Date,  # Date protection starts
    maturity_dt_or_tenor: (Date, str),  # Date or tenor
    running_coupon: float,  # Annualised coupon on premium fee leg
    notional: float = ONE_MILLION,
    long_protection: bool = True,
    freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
    dc_type: DayCountTypes = DayCountTypes.ACT_360,
    cal_type: CalendarTypes = CalendarTypes.WEEKEND,
    bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
    dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
step_in_dt	Date	Date protection starts	-
maturity_dt_or_tenor	Date or str	Date or tenor	-
running_coupon	float	Annualised coupon on premium fee leg	-
notional	float	-	ONE_MILLION
long_protection	bool	-	True
freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
dc_type	DayCountTypes	-	DayCountTypes.ACT_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWIN
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARI

_generate_adjusted_cds_payment_dts

Generate CDS payment dates which have been holiday adjusted.

```
_generate_adjusted_cds_payment_dts():
```

The function arguments are described in the following table.

_calc_flows

Calculate cash flow amounts on premium leg.

```
_calc_flows():
```

The function arguments are described in the following table.

value

Valuation of a CDS contract on a specific valuation date given an issuer curve and a contract recovery rate.

```
value(value_dt,
      issuer_curve,
      contract_recovery_rate,
      pv01_method=0,
      prot_method=0,
      num_steps_per_year=glob_num_steps_per_year):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curve	-	-	-
contract_recovery_rate	-	-	-
pv01_method	-	-	0
prot_method	-	-	0
num_steps_per_year	-	-	glob_num_steps_per_year

credit_dv01

Calculation of the change in the value of the CDS contract for a one basis point change in the level of the CDS curve.

```
credit_dv01(value_dt,
            issuer_curve,
            contract_recovery_rate,
            pv01_method=0,
            prot_method=0,
            num_steps_per_year=glob_num_steps_per_year):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curve	-	-	-
contract_recovery_rate	-	-	-
pv01_method	-	-	0
prot_method	-	-	0
num_steps_per_year	-	-	glob_num_steps_per_year

interest_dv01

Calculation of the interest DV01 based on a simple bump of the discount factors and reconstruction of the CDS curve.

```
interest_dv01(value_dt: Date,
              issuer_curve,
              contract_recovery_rate,
              pv01_method: int = 0,
              prot_method: int = 0,
              num_steps_per_year=glob_num_steps_per_year):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
issuer_curve	-	-	-
contract_recovery_rate	-	-	-
pv01_method	int	-	0
prot_method	int	-	0
num_steps_per_year	-	-	glob_num_steps_per_year

cash_settlement_amount

Value of the contract on the settlement date including accrued interest.

```
cash_settlement_amount(value_dt,
                       settle_dt,
                       issuer_curve,
                       contract_recovery_rate,
                       pv01_method=0,
                       prot_method=0,
                       num_steps_per_year=glob_num_steps_per_year):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
settle_dt	-	-	-
issuer_curve	-	-	-
contract_recovery_rate	-	-	-
pv01_method	-	-	0
prot_method	-	-	0
num_steps_per_year	-	-	glob_num_steps_per_year

clean_price

Value of the CDS contract excluding accrued interest.

```
clean_price(value_dt,
            issuer_curve,
            contract_recovery_rate,
            pv01_method=0,
            prot_method=0,
            num_steps_per_year=glob_num_steps_per_year):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curve	-	-	-
contract_recovery_rate	-	-	-
pv01_method	-	-	0
prot_method	-	-	0
num_steps_per_year	-	-	glob_num_steps_per_year

accrued_days

Number of days between the previous coupon and the currrent step in date.

```
accrued_days():
```

The function arguments are described in the following table.

accrued_interest

Calculate the amount of accrued interest that has accrued from the previous coupon date (PCD) to the step_in_dt of the CDS contract.

```
accrued_interest():
```

The function arguments are described in the following table.

protection_leg_pv

Calculates the protection leg PV of the CDS by calling into the fast NUMBA code that has been defined above.

```
protection_leg_pv(value_dt,
                  issuer_curve,
                  contract_recovery_rate=standard_recovery_rate,
                  num_steps_per_year=glob_num_steps_per_year,
                  protMethod=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curve	-	-	-
contract_recovery_rate	-	-	standard_recovery_rate
num_steps_per_year	-	-	glob_num_steps_per_year
protMethod	-	-	0

risky_pv01

The risky_pv01 is the present value of a risky one dollar paid on the premium leg of a CDS contract.

```
risky_pv01(value_dt,
           issuer_curve,
           pv01_method=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curve	-	-	-
pv01_method	-	-	0

premium_leg_pv

Value of the premium leg of a CDS.

```
premium_leg_pv(value_dt,
               issuer_curve,
               pv01_method=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curve	-	-	-
pv01_method	-	-	0

par_spread

Breakeven CDS coupon that would make the value of the CDS contract equal to zero.

```
par_spread(value_dt,
           issuer_curve,
           contract_recovery_rate=standard_recovery_rate,
           num_steps_per_year=glob_num_steps_per_year,
           pv01_method=0,
           protMethod=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curve	-	-	-
contract_recovery_rate	-	-	standard_recovery_rate
num_steps_per_year	-	-	glob_num_steps_per_year
pv01_method	-	-	0
protMethod	-	-	0

value_fast_approx

Implementation of fast valuation of the CDS contract using an accurate approximation that avoids curve building.

```
value_fast_approx(value_dt,
                  flatContinuousInterestRate,
                  flatCDSCurveSpread,
                  curveRecovery=standard_recovery_rate,
                  contract_recovery_rate=standard_recovery_rate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
flatContinuousInterestRate	-	-	-
flatCDSCurveSpread	-	-	-
curveRecovery	-	-	standard_recovery_rate
contract_recovery_rate	-	-	standard_recovery_rate

print_payments

We only print payments after the current valuation date

```
print_payments(value_dt, issuer_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curve	-	-	-

__repr__

print out details of the CDS contract and all of the calculated cash flows

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_risky_pv01_numba

Fast calculation of the risky PV01 of a CDS using NUMBA. The output is a numpy array of the full and clean risky PV01.

```
_risky_pv01_numba(teff,
                  accrual_factorPCDToNow,
                  paymentTimes,
                  year_fracs,
                  npIborTimes,
                  npIborValues,
                  npSurvTimes,
                  npSurvValues,
                  pv01_method):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
teff	-	-	-
accrual_factorPCDToNow	-	-	-
paymentTimes	-	-	-
year_fracs	-	-	-
npIborTimes	-	-	-
npIborValues	-	-	-
npSurvTimes	-	-	-
npSurvValues	-	-	-
pv01_method	-	-	-

_protection_leg_pv_numba

Fast calculation of the CDS protection leg PV using NUMBA to speed up the numerical integration over time.

```
_protection_leg_pv_numba(teff,
                         tmat,
                         npIborTimes,
                         npIborValues,
                         npSurvTimes,
```

```
                        npSurvValues,
                        contract_recovery_rate,
                        num_steps_per_year,
                        protMethod):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
teff	-	-	-
tmat	-	-	-
npIborTimes	-	-	-
npIborValues	-	-	-
npSurvTimes	-	-	-
npSurvValues	-	-	-
contract_recovery_rate	-	-	-
num_steps_per_year	-	-	-
protMethod	-	-	-

6.2 cds_basket

Class: CDSBasket

CDSBasket

PLEASE ADD A FUNCTION DESCRIPTION

```
CDSBasket(step_in_dt: Date,
          maturity_dt: Date,
          notional: float = ONE_MILLION,
          running_cpn: float = 0.0,
          long_protection: bool = True,
          freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
          dc_type: DayCountTypes = DayCountTypes.ACT_360,
          cal_type: CalendarTypes = CalendarTypes.WEEKEND,
          bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
          dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
step_in_dt	Date	-	-
maturity_dt	Date	-	-
notional	float	-	ONE_MILLION
running_cpn	float	-	0.0
long_protection	bool	-	True
freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
dc_type	DayCountTypes	-	DayCountTypes.ACT_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value_legs_mc

Value the legs of the default basket using Monte Carlo. The default times are an input so this valuation is not model dependent.

```
value_legs_mc(value_dt,
              n_to_default,
              default_times,
              issuer_curves,
              libor_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
n_to_default	-	-	-
default_times	-	-	-
issuer_curves	-	-	-
libor_curve	-	-	-

value_gaussian_mc

Value the default basket using a Gaussian copula model. This depends on the issuer discount and correlation matrix.

```
value_gaussian_mc(value_dt,
                  n_to_default,
                  issuer_curves,
                  correlation_matrix,
                  libor_curve,
                  num_trials,
                  seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
n_to_default	-	-	-
issuer_curves	-	-	-
correlation_matrix	-	-	-
libor_curve	-	-	-
num_trials	-	-	-
seed	-	-	-

value_student_t_mc

Value the default basket using the Student-T copula.

```
value_student_t_mc(value_dt,
                   n_to_default,
                   issuer_curves,
                   correlation_matrix,
                   degrees_of_freedom,
                   libor_curve,
                   num_trials,
                   seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
n_to_default	-	-	-
issuer_curves	-	-	-
correlation_matrix	-	-	-
degrees_of_freedom	-	-	-
libor_curve	-	-	-
num_trials	-	-	-
seed	-	-	-

value_1f_gaussian_homo

Value default basket using 1 factor Gaussian copula and analytical approach which is only exact when all recovery rates are the same.

```
value_1f_gaussian_homo(value_dt,
                       n_to_default,
                       issuer_curves,
                       beta_vector,
                       libor_curve,
                       num_points=50):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
n_to_default	-	-	-
issuer_curves	-	-	-
beta_vector	-	-	-
libor_curve	-	-	-
num_points	-	-	50

__repr__

print out details of the CDS contract and all of the calculated cash flows

```
__repr__():
```

The function arguments are described in the following table.

6.3 cds_curve

Class: CDSCurve

Generate a survival probability curve implied by the value of CDS contracts given a Ibor curve and an assumed recovery rate. The recovery rate corresponds to the seniority of the debt for these CDS. A scheme for the interpolation of the survival probabilities is also required.

CDSCurve

Construct a credit curve from a sequence of maturity-ordered CDS contracts and a Ibor curve using the same recovery rate and the same interpolation method.

```
CDSCurve(value_dt: Date,
         cds_contracts: list,
         libor_curve,
         recovery_rate,
         use_cache: bool = False,
         interp_method: InterpTypes = InterpTypes.FLAT_FWD_RATES):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
cds_contracts	list	-	-
libor_curve	-	-	-
recovery_rate	-	-	-
use_cache	bool	-	False
interp_method	InterpTypes	-	InterpTypes.FLAT_FWD_RATES

_validate

Ensure that contracts are in increasing maturity.

```
_validate(cds_contracts):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cds_contracts	-	-	-

survival_prob

Extract the survival probability to date dt. This function supports vectorisation.

```
survival_prob(dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-

df

Extract the discount factor from the underlying Ibor curve. This function supports vectorisation.

```
df(dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-

_build_curve

Construct the CDS survival curve from a set of CDS contracts

```
_build_curve():
```

The function arguments are described in the following table.

fwd

Calculate the instantaneous forward rate at the forward date dt using the numerical derivative.

```
fwd(dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-

fwd_rate

Calculate the forward rate according between dates date1 and date2 according to the specified day count convention.

```
fwd_rate(date1, date2, day_count_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
date1	-	-	-
date2	-	-	-
day_count_type	-	-	-

zero_rate

Calculate the zero rate to date dt in the chosen compounding frequency where -1 is continuous is the default.

```
zero_rate(dt,
          freq_type=FrequencyTypes.CONTINUOUS):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-
freq_type	-	-	FrequencyTypes.CONTINUOUS

__repr__

Print out the details of the survival probability curve.

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

f

Function that returns zero when the survival probability that gives a zero value of the CDS has been determined.

```
f(q, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
q	-	-	-
*args	-	-	-

6.4 cds_index_option

Class: CDSIndexOption

Class to manage the pricing and risk management of an option to enter into a CDS index. Different pricing algorithms are presented.

CDSIndexOption

PLEASE ADD A FUNCTION DESCRIPTION

```
CDSIndexOption(expiry_dt: Date,
               maturity_dt: Date,
               index_coupon: float,
               strike_coupon: float,
               notional: float = ONE_MILLION,
               long_protection: bool = True,
               freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
               dc_type: DayCountTypes = DayCountTypes.ACT_360,
               cal_type: CalendarTypes = CalendarTypes.WEEKEND,
               bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
               dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
maturity_dt	Date	-	-
index_coupon	float	-	-
strike_coupon	float	-	-
notional	float	-	ONE_MILLION
long_protection	bool	-	True
freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
dc_type	DayCountTypes	-	DayCountTypes.ACT_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value_adjusted_black

This approach uses two adjustments to Black's option pricing model to value an option on a CDS index.

```
value_adjusted_black(value_dt,
                     index_curve,
                     indexRecovery,
                     libor_curve,
                     sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
index_curve	-	-	-
indexRecovery	-	-	-
libor_curve	-	-	-
sigma	-	-	-

value_anderson

This function values a CDS index option following approach by Anderson (2006). This ensures that a no-arbitrage relationship between the constituent CDS contract and the CDS index is enforced. It models the forward spread as a log-normally distributed quantity and uses the credit triangle to compute the forward RPV01.

```
value_anderson(value_dt,
               issuer_curves,
               index_recovery,
               sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curves	-	-	-
index_recovery	-	-	-
sigma	-	-	-

_solve_for_x

Function to solve for the arbitrage free

```
_solve_for_x(value_dt,
             sigma,
             index_coupon,
             indexRecovery,
             libor_curve,
             expH):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
sigma	-	-	-
index_coupon	-	-	-
indexRecovery	-	-	-
libor_curve	-	-	-
expH	-	-	-

_calc_obj_func

An internal function used in the Anderson valuation.

```
_calc_obj_func(x,
               value_dt,
               sigma,
               index_coupon,
               indexRecovery,
               libor_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
value_dt	-	-	-
sigma	-	-	-
index_coupon	-	-	-
indexRecovery	-	-	-
libor_curve	-	-	-

_calc_index_payer_option_price

Calculates the intrinsic value of the index payer swap and the value of the index payer option which are both returned in an array.

```
_calc_index_payer_option_price(value_dt,
                               x,
                               sigma,
                               index_coupon,
                               strike_value,
                               libor_curve,
                               indexRecovery):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
x	-	-	-
sigma	-	-	-
index_coupon	-	-	-
strike_value	-	-	-
libor_curve	-	-	-
indexRecovery	-	-	-

__repr__

print out details of the CDS contract and all of the calculated cash flows

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

6.5 cds_index_portfolio

Class: cds_indexPortfolio

This class manages the calculations associated with an equally weighted portfolio of CDS contracts with the same maturity date.

cds_indexPortfolio

Create Fincds_indexPortfolio object. Note that all of the inputs have a default value which reflects the CDS market standard.

```
cds_indexPortfolio(freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
                   day_count_type: DayCountTypes = DayCountTypes.ACT_360,
                   cal_type: CalendarTypes = CalendarTypes.WEEKEND,
                   bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
                   dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
day_count_type	DayCountTypes	-	DayCountTypes.ACT_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

intrinsic_rpv01

Calculation of the risky PV01 of the CDS portfolio by taking the average of the risky PV01s of each contract.

```
intrinsic_rpv01(value_dt,
                step_in_dt,
                maturity_dt,
                issuer_curves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
step_in_dt	-	-	-
maturity_dt	-	-	-
issuer_curves	-	-	-

intrinsic_protection_leg_pv

Calculation of intrinsic protection leg value of the CDS portfolio by taking the average sum the protection legs of each contract.

```
intrinsic_protection_leg_pv(value_dt,
                            step_in_dt,
                            maturity_dt,
                            issuer_curves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
step_in_dt	-	-	-
maturity_dt	-	-	-
issuer_curves	-	-	-

intrinsic_spread

Calculation of the intrinsic spread of the CDS portfolio as the one which would make the value of the protection legs equal to the value of the premium legs if all premium legs paid the same spread.

```
intrinsic_spread(value_dt,
                 step_in_dt,
                 maturity_dt,
                 issuer_curves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
step_in_dt	-	-	-
maturity_dt	-	-	-
issuer_curves	-	-	-

average_spread

Calculates the average par CDS spread of the CDS portfolio.

```
average_spread(value_dt,
               step_in_dt,
               maturity_dt,
               issuer_curves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
step_in_dt	-	-	-
maturity_dt	-	-	-
issuer_curves	-	-	-

total_spread

Calculates the total CDS spread of the CDS portfolio by summing over all of the issuers and adding the spread with no weights.

```
total_spread(value_dt,
             step_in_dt,
             maturity_dt,
             issuer_curves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
step_in_dt	-	-	-
maturity_dt	-	-	-
issuer_curves	-	-	-

min_spread

Calculates the minimum par CDS spread across all of the issuers in the CDS portfolio.

```
min_spread(value_dt,
           step_in_dt,
           maturity_dt,
           issuer_curves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
step_in_dt	-	-	-
maturity_dt	-	-	-
issuer_curves	-	-	-

max_spread

Calculates the maximum par CDS spread across all of the issuers in the CDS portfolio.

```
max_spread(value_dt,
           step_in_dt,
           maturity_dt,
           issuer_curves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
step_in_dt	-	-	-
maturity_dt	-	-	-
issuer_curves	-	-	-

spread_adjust_intrinsic

Adjust individual CDS discount to reprice CDS index prices. This approach uses an iterative scheme but is slow as it has to use a CDS curve bootstrap required when each trial spread adjustment is made

```
spread_adjust_intrinsic(value_dt,
                        issuer_curves,
                        index_cpns,
                        index_upfronts,
                        index_maturity_dts,
                        index_recovery_rate,
                        tolerance=1e-6):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curves	-	-	-
index_cpns	-	-	-
index_upfronts	-	-	-
index_maturity_dts	-	-	-
index_recovery_rate	-	-	-
tolerance	-	-	1e-6

hazard_rate_adjust_intrinsic

Adjust individual CDS discount to reprice CDS index prices. This approach adjusts the hazard rates and so avoids the slowish CDS curve bootstrap required when a spread adjustment is made.

```
hazard_rate_adjust_intrinsic(value_dt,
                             issuer_curves,
                             index_cpns,
                             index_up_fronts,
                             index_maturity_dts,
                             index_recovery_rate,
                             tolerance=1e-6,
                             max_iterations=200):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curves	-	-	-
index_cpns	-	-	-
index_up_fronts	-	-	-
index_maturity_dts	-	-	-
index_recovery_rate	-	-	-
tolerance	-	-	1e-6
max_iterations	-	-	200

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

6.6 cds_option

Class: CDSOption

Class to manage the pricing and risk-management of an option on a single-name CDS. This is a contract in which the option buyer pays for an option to either buy or sell protection on the underlying CDS at a fixed spread agreed today and to be exercised in the future on a specified expiry date. The option may or may not cancel if there is a credit event before option expiry. This needs to be specified.

CDSOption

Create a FinCDSOption object with the option expiry date, the maturity date of the underlying CDS, the option strike coupon, notional, whether the option knocks out or not in the event of a credit event before expiry and the payment details of the underlying CDS.

```
CDSOption(expiry_dt: Date,
          maturity_dt: Date,
          strike_coupon: float,
          notional: float = ONE_MILLION,
          long_protection: bool = True,
          knockout_flag: bool = True,
          freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
          dc_type: DayCountTypes = DayCountTypes.ACT_360,
          cal_type: CalendarTypes = CalendarTypes.WEEKEND,
          bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
          dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
maturity_dt	Date	-	-
strike_coupon	float	-	-
notional	float	-	ONE_MILLION
long_protection	bool	-	True
knockout_flag	bool	-	True
freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
dc_type	DayCountTypes	-	DayCountTypes.ACT_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value

Value the CDS option using Black's model with an adjustment for any Front End Protection. TODO - Should the CDS be created in the init method ?

```
value(value_dt,
      issuer_curve,
```

```
      volatility):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curve	-	-	-
volatility	-	-	-

implied_volatility

Calculate the implied CDS option volatility from a price.

```
implied_volatility(value_dt,
                   issuer_curve,
                   option_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curve	-	-	-
option_value	-	-	-

fvol

Root searching function in the calculation of the CDS implied volatility.

```
fvol(volatility, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	-	-	-
*args	-	-	-

6.7 cds_tranche

Enumerated Type: FinLossDistributionBuilder

This enumerated type has the following values:

- RECURSION
- ADJUSTED_BINOMIAL
- GAUSSIAN
- LHP

Class: CDSTranche

class CDSTranche:

CDSTranche

PLEASE ADD A FUNCTION DESCRIPTION

```
CDSTranche(step_in_dt: Date,
           maturity_dt: Date,
           k1: float,
           k2: float,
           notional: float = ONE_MILLION,
           running_coupon: float = 0.0,
           long_protection: bool = True,
           freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
           dc_type: DayCountTypes = DayCountTypes.ACT_360,
           cal_type: CalendarTypes = CalendarTypes.WEEKEND,
           bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
           dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
step_in_dt	Date	-	-
maturity_dt	Date	-	-
k1	float	-	-
k2	float	-	-
notional	float	-	ONE_MILLION
running_coupon	float	-	0.0
long_protection	bool	-	True
freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
dc_type	DayCountTypes	-	DayCountTypes.ACT_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value_bc

PLEASE ADD A FUNCTION DESCRIPTION

```
value_bc(value_dt,
         issuer_curves,
         upfront,
         running_coupon,
         corr1,
         corr2,
         num_points=50,
         model=FinLossDistributionBuilder.RECURSION):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
issuer_curves	-	-	-
upfront	-	-	-
running_coupon	-	-	-
corr1	-	-	-
corr2	-	-	-
num_points	-	-	50
model	-	-	RECURSION

6.8 __init__

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.
- BondZero is a zero coupon bond. This is a bond issued at a deep discount that matures at par. Accrued interest is calculated by interpolating the price growth.
- 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.
- Yield Curve is a class to handle bond yield curves. It uses a variety of shapes to best-fit a set of bond yields.
- Zero curve is a class to perform an exact fit to a set of provided bonds using a piece-wise flat zero rate.

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. These are 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. 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

7.1 bond

Enumerated Type: YTMCalcType

This enumerated type has the following values:

- ZERO
- UK_DMO
- US_STREET
- US_TREASURY
- CFETS

Class: Bond

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

Bond

Create Bond object by providing the issue date, maturity Date, coupon frequency, annualised coupon, the accrual convention type, face amount and the number of ex-dividend days. A calendar type is used to determine holidays from which coupon dates might be shifted.

```
Bond(issue_dt: Date,
     maturity_dt: Date,
     coupon: float,  # Annualised bond coupon
     freq_type: FrequencyTypes,
     dc_type: DayCountTypes,
     ex_div_days: int = 0,
     cal_type: CalendarTypes = CalendarTypes.WEEKEND,
     bd_type=BusDayAdjustTypes.FOLLOWING,
     dg_type=DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issue_dt	Date	-	-
maturity_dt	Date	-	-
coupon	float	Annualised bond coupon	-
freq_type	FrequencyTypes	-	-
dc_type	DayCountTypes	-	-
ex_div_days	int	-	0
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	-	-	BusDayAdjustTypes.FOLLOWING
dg_type	-	-	DateGenRuleTypes.BACKWARD

_calculate_cpn_dts

Determine the bond coupon dates. Note that for analytical calculations these are not usually adjusted and so may fall on a weekend or holiday.

```
_calculate_cpn_dts():
```

The function arguments are described in the following table.

_calculate_payment_dts

For the actual payment dates, they are adjusted and so we then use the calendar payment dates. Although payments are calculated as though coupon periods are the same length, payments that fall on a Saturday or Sunday can only be made on the next business day

```
_calculate_payment_dts():
```

The function arguments are described in the following table.

_calculate_flows

Determine the bond cash flow payment amounts without principal. There is no adjustment based on the adjusted payment dates.

```
_calculate_flows():
```

The function arguments are described in the following table.

dirty_price_from_ytm

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

```
dirty_price_from_ytm(settle_dt: Date,
                     ytm: float,
                     convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

principal

Calculate the principal value of the bond based on the face amount from its discount margin and making assumptions about the future Ibor rates.

```
principal(settle_dt: Date,
          ytm: float,
          face: (float),
          convention: YTMCalcType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
face	float or (float	-	-
convention	YTMCalcType	-	-

dollar_duration

Calculate the risk or dP/dy of the bond by bumping. This is also known as the DV01 in Bloomberg.

```
dollar_duration(settle_dt: Date,
                ytm: float,
                convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

macauley_duration

Calculate the Macauley duration of the bond on a settlement date given its yield to maturity.

```
macauley_duration(settle_dt: Date,
                  ytm: float,
                  convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

modified_duration

Calculate the modified duration of the bond on a settlement date given its yield to maturity.

```
modified_duration(settle_dt: Date,
                  ytm: float,
                  convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

key_rate_durations

Calculates the key rate durations for a bond. Parameters ———- bond : FinancePy Bond object settle_dt : FinancePy Date object The settlement date. ytm : float The yield to maturity. key_rate_tenors : list of float, optional The tenors of the key rates, default is None which will generate the tenors from 0.25 to 30 years. shift : float, optional The shift used to calculate the key rate durations, default is None which will set the shift to 0.0001. rates: list of float, optional Corresponding yield curve data in line with key_rate_tenors If None, flat yield curve is used Returns ——- tuple of (numpy array of float, numpy array of float) A tuple containing the key rate tenors and the key rate durations.

```
key_rate_durations(bond,
                   settle_dt: Date,
                   ytm: float,
                   key_rate_tenors: list = None,
                   shift: float = None,
                   rates: list = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bond	-	-	-
settle_dt	Date	-	-
ytm	float	-	-
key_rate_tenors	list	-	None
shift	float	-	None
rates	list	-	None

convexity_from_ytm

Calculate the bond convexity from the yield to maturity. This function is vectorised with respect to the yield input.

```
convexity_from_ytm(settle_dt: Date,
                   ytm: float,
                   convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

clean_price_from_ytm

Calculate the bond clean price from the yield to maturity. This function is vectorised with respect to the yield input.

```
clean_price_from_ytm(settle_dt: Date,
                     ytm: float,
                     convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

clean_price_from_discount_curve

Calculate the clean bond value using some discount curve to present-value the bond's cash flows back to the curve anchor date and not to the settlement date.

```
clean_price_from_discount_curve(settle_dt: Date,
                                discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

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

dirty_price_from_discount_curve

Calculate the bond price using a provided discount curve to PV the bond's cash flows to the settlement date. As such it is effectively a forward bond price if the settlement date is after the valuation date.

```
dirty_price_from_discount_curve(settle_dt: Date,
                                discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

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

current_yield

Calculate the current yield of the bond which is the coupon divided by the clean price (not the full price)

```
current_yield(clean_price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
clean_price	-	-	-

yield_to_maturity

Calculate the bond's yield to maturity by solving the price yield relationship using a one-dimensional root solver.

```
yield_to_maturity(settle_dt: Date,
                  clean_price: float,
                  convention: YTMCalcType = YTMCalcType.US_TREASURY):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
clean_price	float	-	-
convention	YTMCalcType	-	YTMCalcType.US_TREASURY

_calc_pcd_ncd

PLEASE ADD A FUNCTION DESCRIPTION

```
_calc_pcd_ncd(settle_dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-

accrued_interest

Calculate the amount of coupon that has accrued between the previous coupon date and the settlement date. Note that for some day count schemes (such as 30E/360) this is not actually the number of days between the previous coupon payment date and settlement date. If the bond trades with ex-coupon dates then you need to use the number of days before the coupon date the ex-coupon date is. You can specify the calendar to be used in the bond constructor - NONE means only calendar days, WEEKEND is only weekends or you can specify a country calendar for business days.

```
accrued_interest(settle_dt: Date,
                 face: float = 100.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float	-	100.0

asset_swap_spread

Calculate the par asset swap spread of the bond. The discount curve is a Ibor curve that is passed in. This function is vectorised with respect to the clean price.

```
asset_swap_spread(settle_dt: Date,
    clean_price: float,
    discount_curve: DiscountCurve,
    swapFloatDayCountConventionType=DayCountTypes.ACT_360,
    swapFloatFrequencyType=FrequencyTypes.SEMI_ANNUAL,
    swapFloatCalendarType=CalendarTypes.WEEKEND,
    swapFloatBusDayAdjustRuleType=BusDayAdjustTypes.FOLLOWING,
    swapFloatDateGenRuleType=DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
clean_price	float	-	-
discount_curve	DiscountCurve	-	-
swapFloatDayCountConventionType	-	-	DayCountTypes.ACT_360
swapFloatFrequencyType	-	-	FrequencyTypes.SEMI_ANNUAL
swapFloatCalendarType	-	-	CalendarTypes.WEEKEND
swapFloatBusDayAdjustRuleType	-	-	BusDayAdjustTypes.FOLLOWING
swapFloatDateGenRuleType	-	-	DateGenRuleTypes.BACKWARD

dirty_price_from_oas

Calculate the full price of the bond from its OAS given the bond settlement date, a discount curve and the oas as a number.

```
dirty_price_from_oas(settle_dt: Date,
                     discount_curve: DiscountCurve,
                     oas: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
discount_curve	DiscountCurve	-	-
oas	float	-	-

option_adjusted_spread

Return OAS for bullet bond given settlement date, clean bond price and the discount relative to which the spread is to be computed.

```
option_adjusted_spread(settle_dt: Date,
                       clean_price: float,
                       discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
clean_price	float	-	-
discount_curve	DiscountCurve	-	-

bond_payments

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
bond_payments(settle_dt: Date, face: (float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float or (float	-	-

print_payments

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
print_payments(settle_dt: Date,
               face: (float) = 100.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float or (float	-	100.0

dirty_price_from_survival_curve

Calculate discounted present value of flows assuming default model. The survival curve treats the coupons as zero recovery payments while the recovery fraction of the par amount is paid at default. For the defaulting principal we discretize the time steps using the coupon payment times. A finer discretization may handle the time value with more accuracy. I reduce any error by averaging period start and period end payment present values.

```
dirty_price_from_survival_curve(settle_dt: Date,
                                discount_curve: DiscountCurve,
                                survival_curve: DiscountCurve,
                                recovery_rate: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
discount_curve	DiscountCurve	-	-
survival_curve	DiscountCurve	-	-
recovery_rate	float	-	-

clean_price_from_survival_curve

Calculate clean price value of flows assuming default model. The survival curve treats the coupons as zero recovery payments while the recovery fraction of the par amount is paid at default.

```
clean_price_from_survival_curve(settle_dt: Date,
                                discount_curve: DiscountCurve,
                                survival_curve: DiscountCurve,
                                recovery_rate: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
discount_curve	DiscountCurve	-	-
survival_curve	DiscountCurve	-	-
recovery_rate	float	-	-

calc_ror

Calculate the rate of total return(capital return and interest) given a BUY YTM and a SELL YTM of this bond. This function computes the full prices at buying and selling, plus the coupon payments during the period. It returns a tuple which includes a simple rate of return, a compounded IRR and the PnL.

```
calc_ror(begin_dt: Date,
         end_dt: Date,
         begin_ytm: float,
         end_ytm: float,
```

```
        convention: YTMCalcType = YTMCalcType.US_STREET):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
begin_dt	Date	-	-
end_dt	Date	-	-
begin_ytm	float	-	-
end_ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.US_STREET

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

_f

Function used to do root search in price to yield calculation.

```
_f(ytm, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
ytm	-	-	-
*args	-	-	-

_g

Function used to do root search in price to OAS calculation.

```
_g(oas, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
oas	-	-	-
*args	-	-	-

7.2 bond_annuity

Class: BondAnnuity

An annuity is a vector of dates and flows generated according to ISDA standard rules which starts on the next date after the start date (effective date) and runs up to an end date with no principal repayment. Dates are then adjusted according to a specified calendar.

BondAnnuity

PLEASE ADD A FUNCTION DESCRIPTION

```
BondAnnuity(maturity_dt: Date,
            cpn: float,
            freq_type: FrequencyTypes,
            cal_type: CalendarTypes = CalendarTypes.WEEKEND,
            bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
            dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD,
            dc_type: DayCountTypes = DayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
maturity_dt	Date	-	-
cpn	float	-	-
freq_type	FrequencyTypes	-	-
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD
dc_type	DayCountTypes	-	DayCountTypes.ACT_360

clean_price_from_discount_curve

Calculate the bond price using some discount curve to present-value the bond's cash flows.

```
clean_price_from_discount_curve(settle_dt: Date,
                                discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

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

dirty_price_from_discount_curve

Calculate the bond price using some discount curve to present-value the bond's cash flows.

```
dirty_price_from_discount_curve(settle_dt: Date,
                                discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

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

calculate_payments

PLEASE ADD A FUNCTION DESCRIPTION

```
calculate_payments(settle_dt: Date,
                   face: (float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float or (float	-	-

accrued_interest

Calculate the amount of coupon that has accrued between the previous coupon date and the settlement date.

```
accrued_interest(settle_dt: Date,
                 face: (float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float or (float	-	-

print_payments

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
print_payments(settle_dt: Date,
               face: (float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float or (float	-	-

__repr__

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

7.3 bond_callable

Enumerated Type: BondModelTypes

This enumerated type has the following values:

- BLACK
- HO_LEE
- HULL_WHITE
- BLACK_KARASINSKI

Enumerated Type: BondOptionTypes

This enumerated type has the following values:

- EUROPEAN_CALL
- EUROPEAN_PUT
- AMERICAN_CALL
- AMERICAN_PUT

Class: BondEmbeddedOption

BondEmbeddedOption

Create a BondEmbeddedOption object with a maturity date, coupon and all of the bond inputs.

```
BondEmbeddedOption(issue_dt: Date,
                   maturity_dt: Date,  # Date
                   coupon: float,  # Annualised coupon - 0.03 = 3.00%
                   freq_type: FrequencyTypes,
                   dc_type: DayCountTypes,
                   call_dts: List[Date],
                   call_prices: List[float],
                   put_dts: List[Date],
                   put_prices: List[float]):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issue_dt	Date	-	-
maturity_dt	Date	Date	-
coupon	float	Annualised coupon - 0.03 = 3.00%	-
freq_type	FrequencyTypes	-	-
dc_type	DayCountTypes	-	-
call_dts	List[Date]	-	-
call_prices	List[float]	-	-
put_dts	List[Date]	-	-
put_prices	List[float]	-	-

value

Value the bond that settles on the specified date that can have both embedded call and put options. This is done using the specified model and a discount curve.

```
value(settle_dt: Date,
      discount_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
discount_curve	DiscountCurve	-	-
model	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

PLEASE ADD A FUNCTION DESCRIPTION

```
_print():
```

The function arguments are described in the following table.

7.4 bond_convertible

Class: BondConvertible

Class for convertible bonds. These bonds embed rights to call and put the bond in return for equity. Until then they are bullet bonds which means they have regular coupon payments of a known size that are paid on known dates plus a payment of par at maturity. As the options are price based, the decision to convert to equity depends on the stock price, the credit quality of the issuer and the level of interest rates.

BondConvertible

Create BondConvertible object by providing the bond Maturity date, coupon, frequency type, accrual convention type and then all of the details regarding the conversion option including the list of the call and put dates and the corresponding list of call and put prices.

```
BondConvertible(maturity_dt: Date,  # bond maturity date
                coupon: float,  # annual coupon
                freq_type: FrequencyTypes,  # coupon frequency type
                start_convert_dt: Date,  # conversion starts on this date
                conversion_ratio: float,  # num shares per face of notional
                call_dts: List[Date],  # list of call dates
                call_prices: List[float],  # list of call prices
                put_dts: List[Date],  # list of put dates
                put_prices: List[float],  # list of put prices
                dc_type: DayCountTypes,  # day count type for accrued
                cal_type: CalendarTypes = CalendarTypes.WEEKEND):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
maturity_dt	Date	bond maturity date	-
coupon	float	annual coupon	-
freq_type	FrequencyTypes	coupon frequency type	-
start_convert_dt	Date	conversion starts on this date	-
conversion_ratio	float	num shares per face of notional	-
call_dts	List[Date]	list of call dates	-
call_prices	List[float]	list of call prices	-
put_dts	List[Date]	list of put dates	-
put_prices	List[float]	list of put prices	-
dc_type	DayCountTypes	day count type for accrued	-
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND

_calculate_cpn_dts

Determine the convertible bond cash flow payment dates.

```
_calculate_cpn_dts(settle_dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-

value

A binomial tree valuation model for a convertible bond that captures the embedded equity option due to the existence of a conversion option which can be invoked after a specific date. The model allows the user to enter a schedule of dividend payment dates but the size of the payments must be in yield terms i.e. a known percentage of currently unknown future stock price is paid. Not a fixed amount. A fixed yield. Following this payment the stock is assumed to drop by the size of the dividend payment. The model also captures the stock dependent credit risk of the cash flows in which the bond price can default at any time with a hazard rate implied by the credit spread and an associated recovery rate. This is the model proposed by Hull (OFODS 6th edition,.page 522). The model captures both the issuer's call schedule which is assumed to apply on a list of dates provided by the user, along with a call price. It also captures the embedded owner's put schedule of prices.

```
value(settle_dt: Date,
      stock_price: float,
      stock_volatility: float,
      dividend_dts: List[Date],
      dividend_yields: List[float],
      discount_curve: DiscountCurve,
      credit_spread: float,
      recovery_rate: float = 0.40,
      num_steps_per_year: int = 100):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
stock_price	float	-	-
stock_volatility	float	-	-
dividend_dts	List[Date]	-	-
dividend_yields	List[float]	-	-
discount_curve	DiscountCurve	-	-
credit_spread	float	-	-
recovery_rate	float	-	0.40
num_steps_per_year	int	-	100

accrued_days

Calculate number days from previous coupon date to settlement.

```
accrued_days(settle_dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-

accrued_interest

Calculate the amount of coupon that has accrued between the previous coupon date and the settlement date.

```
accrued_interest(settle_dt: Date,
                 face: (float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float or (float	-	-

current_yield

Calculate the current yield of the bond which is the coupon divided by the clean price (not the full price)

```
current_yield(clean_price: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
clean_price	float	-	-

__repr__

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_value_convertible

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_convertible(tmat,
                   face_amount,
                   cpn_times,
                   cpn_flows,
                   call_times,
                   call_prices,
                   put_times,
                   put_prices,
                   conv_ratio,
                   start_convert_time,
                   # Market inputs
                   stock_price,
                   df_times,
                   df_values,
                   dividend_times,
                   dividend_yields,
                   stock_volatility,
                   credit_spread,
                   recovery_rate,
                   # Tree details
                   num_steps_per_year):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
tmat	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
call_times	-	-	-
call_prices	-	-	-
put_times	-	-	-
put_prices	-	-	-
conv_ratio	-	-	-
start_convert_time	-	-	-
stock_price	-	-	-
df_times	-	-	-
df_values	-	-	-
dividend_times	-	-	-
dividend_yields	-	-	-
stock_volatility	-	-	-
credit_spread	-	-	-
recovery_rate	-	-	-
num_steps_per_year	-	-	-

print_tree

PLEASE ADD A FUNCTION DESCRIPTION

```
print_tree(array):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
array	-	-	-

7.5 bond_frn

Class: BondFRN

Class for managing floating rate notes that pay a floating index plus a quoted margin.

BondFRN

Create FinFloatingRateNote object given its maturity date, its quoted margin, coupon frequency, DAY COUNT TYPE. Face is the size of the position and par is the notional on which price is quoted.

```
BondFRN(issue_dt: Date,
        maturity_dt: Date,
        quoted_margin: float,  # Fixed spread paid on top of index
        freq_type: FrequencyTypes,
        dc_type: DayCountTypes,
        cal_type: CalendarTypes = CalendarTypes.WEEKEND):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issue_dt	Date	-	-
maturity_dt	Date	-	-
quoted_margin	float	Fixed spread paid on top of index	-
freq_type	FrequencyTypes	-	-
dc_type	DayCountTypes	-	-
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND

_calculate_cpn_dts

Determine the bond cashflow payment dates.

```
_calculate_cpn_dts():
```

The function arguments are described in the following table.

dirty_price_from_dm

Calculate the full price of the bond from its discount margin (DM) using standard model based on assumptions about future Ibor rates. The next Ibor payment which has reset is entered, so to is the current Ibor rate from settlement to the next coupon date (NCD). Finally there is the level of subsequent future Ibor payments and the discount margin.

```
dirty_price_from_dm(settle_dt: Date,
                    next_cpn: float,  # The total reset coupon on NCD
                    current_ibor: float,  # Ibor discount to NCD
                    future_ibor: float,  # Future constant Ibor rates
                    dm: float):  # Discount margin
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	The total reset coupon on NCD	-
current_ibor	float	Ibor discount to NCD	-
future_ibor	float	Future constant Ibor rates	-
dm	float	Discount margin	-

principal

Calculate the clean trade price of the bond based on the face amount from its discount margin and making assumptions about the future Ibor rates.

```
principal(settle_dt: Date,
          next_cpn: float,
          current_ibor: float,
          future_ibor: float,
          dm: float,
          face: float = 100.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	-	-
current_ibor	float	-	-
future_ibor	float	-	-
dm	float	-	-
face	float	-	100.0

dollar_duration

Calculate the risk or dP/dy of the bond by bumping. This is also known as the DV01 in Bloomberg.

```
dollar_duration(settle_dt: Date,
                next_cpn: float,
                current_ibor: float,
                future_ibor: float,
                dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	-	-
current_ibor	float	-	-
future_ibor	float	-	-
dm	float	-	-

dollar_credit_duration

Calculate the risk or dP/dy of the bond by bumping.

```
dollar_credit_duration(settle_dt: Date,
                       next_cpn: float,
                       current_ibor: float,
                       future_ibor: float,
                       dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	-	-
current_ibor	float	-	-
future_ibor	float	-	-
dm	float	-	-

macauley_duration

Calculate the Macauley duration of the FRN on a settlement date given its yield to maturity.

```
macauley_duration(settle_dt: Date,
                  next_cpn: float,
                  current_ibor: float,
                  future_ibor: float,
                  dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	-	-
current_ibor	float	-	-
future_ibor	float	-	-
dm	float	-	-

modified_duration

Calculate the modified duration of the bond on a settlement date using standard model based on assumptions about future Ibor rates. The next Ibor payment which has reset is entered, so to is the current Ibor rate from settlement to the next coupon date (NCD). Finally there is the level of subsequent future Ibor payments and the discount margin.

```
modified_duration(settle_dt: Date,
                  next_cpn: float,
                  current_ibor: float,
                  future_ibor: float,
                  dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	-	-
current_ibor	float	-	-
future_ibor	float	-	-
dm	float	-	-

modified_credit_duration

Calculate the modified duration of the bond on a settlement date using standard model based on assumptions about future Ibor rates. The next Ibor payment which has reset is entered, so to is the current Ibor rate from settlement to the next coupon date (NCD). Finally there is the level of subsequent future Ibor payments and the discount margin.

```
modified_credit_duration(settle_dt: Date,
                         next_cpn: float,
                         current_ibor: float,
                         future_ibor: float,
                         dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	-	-
current_ibor	float	-	-
future_ibor	float	-	-
dm	float	-	-

convexity_from_dm

Calculate the bond convexity from the discount margin (DM) using a standard model based on assumptions about future Ibor rates. The next Ibor payment which has reset is entered, so to is the current Ibor rate from settlement to the next coupon date (NCD). Finally there is the level of subsequent future Ibor payments and the discount margin.

```
convexity_from_dm(settle_dt: Date,
                  next_cpn: float,
                  current_ibor: float,
                  future_ibor: float,
                  dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	-	-
current_ibor	float	-	-
future_ibor	float	-	-
dm	float	-	-

clean_price_from_dm

Calculate the bond clean price from the discount margin using standard model based on assumptions about future Ibor rates. The next Ibor payment which has reset is entered, so to is the current Ibor rate from settlement to the next coupon date (NCD). Finally there is the level of subsequent future Ibor payments and the discount margin.

```
clean_price_from_dm(settle_dt: Date,
                    next_cpn: float,
                    current_ibor: float,
                    future_ibor: float,
                    dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	-	-
current_ibor	float	-	-
future_ibor	float	-	-
dm	float	-	-

discount_margin

Calculate the bond's yield to maturity by solving the price yield relationship using a one-dimensional root solver.

```
discount_margin(settle_dt: Date,
                next_cpn: float,
                current_ibor: float,
                future_ibor: float,
                clean_price: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	-	-
current_ibor	float	-	-
future_ibor	float	-	-
clean_price	float	-	-

accrued_interest

Calculate the amount of coupon that has accrued between the previous coupon date and the settlement date. Ex-dividend dates are not handled. Contact me if you need this functionality.

```
accrued_interest(settle_dt: Date,
                 next_cpn: float,
                 face: (float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
next_cpn	float	-	-
face	float or (float	-	-

print_payments

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
print_payments(settle_dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_f

Function used to do solve root search in DM calculation

```
_f(dm, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dm	-	-	-
*args	-	-	-

7.6 bond_future

Class: BondFuture

Class for managing futures contracts on government bonds that follows CME conventions and related analytics.

BondFuture

PLEASE ADD A FUNCTION DESCRIPTION

```
BondFuture(ticker_name: str,
           first_delivery_dt: Date,
           last_delivery_dt: Date,
           contract_size: int,
           cpn: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
ticker_name	str	-	-
first_delivery_dt	Date	-	-
last_delivery_dt	Date	-	-
contract_size	int	-	-
cpn	float	-	-

conversion_factor

Determine the conversion factor for a specific bond using CME convention. To do this we need to know the contract standard coupon and must round the bond maturity (starting its life on the first delivery date) to the nearest 3 month multiple and then calculate the bond clean price.

```
conversion_factor(bond: Bond):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bond	Bond	-	-

principal_invoice_price

The principal invoice price as defined by the CME.

```
principal_invoice_price(bond: Bond,
                        futures_price: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bond	Bond	-	-
futures_price	float	-	-

total_invoice_amount

PLEASE ADD A FUNCTION DESCRIPTION

```
total_invoice_amount(settle_dt: Date,
                     bond: Bond,
                     futures_price: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
bond	Bond	-	-
futures_price	float	-	-

cheapest_to_deliver

Determination of CTD as deliverable bond with lowest cost to buy versus what is received when the bond is delivered.

```
cheapest_to_deliver(bonds: list,
                    bond_clean_prices: list,
                    futures_price: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bonds	list	-	-
bond_clean_prices	list	-	-
futures_price	float	-	-

delivery_gain_loss

Determination of what is received when the bond is delivered.

```
delivery_gain_loss(bond: Bond,
                   bond_clean_price: float,
                   futures_price: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bond	Bond	-	-
bond_clean_price	float	-	-
futures_price	float	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

7.7 bond_market

Enumerated Type: BondMarkets

This enumerated type has the following values:

- AUSTRIA
- BELGIUM
- CYPRUS
- ESTONIA
- FINLAND
- FRANCE
- GERMANY
- GREECE
- IRELAND
- ITALY
- LATVIA
- LITHUANIA
- LUXEMBOURG
- MALTA
- NETHERLANDS
- PORTUGAL
- SLOVAKIA
- SLOVENIA
- SPAIN
- ESM
- EFSF
- BULGARIA
- CROATIA
- CZECH_REPUBLIC
- DENMARK
- HUNGARY
- POLAND
- ROMANIA
- SWEDEN
- JAPAN
- SWITZERLAND
- UNITED_KINGDOM
- UNITED_STATES
- AUSTRALIA
- NEW_ZEALAND
- NORWAY
- SOUTH_AFRICA

get_bond_market_conventions

Returns the day count convention for accrued interest, the frequency and the number of days from trade date to settlement date. This is for Treasury markets. And for secondary bond markets.

```
get_bond_market_conventions(country):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
country	-	-	-

7.8 bond_mortgage

Enumerated Type: BondMortgageTypes

This enumerated type has the following values:
- REPAYMENT
- INTEREST_ONLY

Class: BondMortgage

A mortgage is a vector of dates and flows generated in order to repay a fixed amount given a known interest rate. Payments are all the same amount but with a varying mixture of interest and repayment of principal.

BondMortgage

Create the mortgage using start and end dates and principal.

```
BondMortgage(start_dt: Date,
             end_dt: Date,
             principal: float,
             freq_type: FrequencyTypes = FrequencyTypes.MONTHLY,
             cal_type: CalendarTypes = CalendarTypes.WEEKEND,
             bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
             dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD,
             dc_type: DayCountTypes = DayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_dt	Date	-	-
end_dt	Date	-	-
principal	float	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.MONTHLY
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD
dc_type	DayCountTypes	-	DayCountTypes.ACT_360

repayment_amount

Determine monthly repayment amount based on current zero rate.

```
repayment_amount(zero_rate: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
zero_rate	float	-	-

generate_flows

Generate the bond flow amounts.

```
generate_flows(zero_rate: float,
               mortgage_type: BondMortgageTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
zero_rate	float	-	-
mortgage_type	BondMortgageTypes	-	-

print_leg

PLEASE ADD A FUNCTION DESCRIPTION

```
print_leg():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

7.9 bond_option

Enumerated Type: BondModelTypes

This enumerated type has the following values:

- BLACK
- HO_LEE
- HULL_WHITE
- BLACK_KARASINSKI

Class: BondOption()

Class for options on fixed coupon bonds. These are options to either buy or sell a bond on or before a specific future expiry date at a strike price that is set on trade date. A European option only allows the bond to be exercised into on a specific expiry date. An American option allows the option holder to exercise early, potentially allowing earlier coupons to be received.

BondOption

PLEASE ADD A FUNCTION DESCRIPTION

```
BondOption(bond: Bond,
           expiry_dt: Date,
           strike_price: float,
           option_type: OptionTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bond	Bond	-	-
expiry_dt	Date	-	-
strike_price	float	-	-
option_type	OptionTypes	-	-

value

Value a bond option (option on a bond) using a specified model which include the Hull-White, Black-Karasinski and Black-Derman-Toy model which are all implemented as short rate tree models.

```
value(value_dt: Date,
      discount_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
discount_curve	DiscountCurve	-	-
model	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

7.10 bond_portfolio

Class: BondPortfolio

BondPortfolio

XXX

```
BondPortfolio(bonds: (list),
              bondWeights: (list, np.ndarray)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bonds	list or (list	-	-
bondWeights	list or np.ndarray	-	-

_calculate_flows

Determine the bond cashflow payment amounts without principal

```
_calculate_flows():
```

The function arguments are described in the following table.

dollar_duration

Calculate the risk or dP/dy of the bond by bumping. This is also known as the DV01 in Bloomberg.

```
dollar_duration(settle_dt: Date,
                ytm: float,
                convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

macauley_duration

Calculate the Macauley duration of the bond on a settlement date given its yield to maturity.

```
macauley_duration(settle_dt: Date,
                  ytm: float,
                  convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

modified_duration

Calculate the modified duration of the bondon a settlement date given its yield to maturity.

```
modified_duration(settle_dt: Date,
                  ytm: float,
                  convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

convexity_from_ytm

Calculate the bond convexity from the yield to maturity. This function is vectorised with respect to the yield input.

```
convexity_from_ytm(settle_dt: Date,
                   ytm: float,
                   convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

clean_price_from_ytm

Calculate the bond clean price from the yield to maturity. This function is vectorised with respect to the yield input.

```
clean_price_from_ytm(settle_dt: Date,
                     ytm: float,
                     convention: YTMCalcType = YTMCalcType.UK_DMO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.UK_DMO

clean_price_from_discount_curve

Calculate the clean bond value using some discount curve to present-value the bond's cash flows back to the curve anchor date and not to the settlement date.

```
clean_price_from_discount_curve(settle_dt: Date,
                                discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

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

dirty_price_from_discount_curve

Calculate the bond price using a provided discount curve to PV the bond's cash flows to the settlement date. As such it is effectively a forward bond price if the settlement date is after the valuation date.

```
dirty_price_from_discount_curve(settle_dt: Date,
                                discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

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

current_yield

Calculate the current yield of the bond which is the coupon divided by the clean price (not the full price)

```
current_yield(clean_price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
clean_price	-	-	-

yield_to_maturity

Calculate the bond's yield to maturity by solving the price yield relationship using a one-dimensional root solver.

```
yield_to_maturity(settle_dt: Date,
                  clean_price: float,
                  convention: YTMCalcType = YTMCalcType.US_TREASURY):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
clean_price	float	-	-
convention	YTMCalcType	-	YTMCalcType.US_TREASURY

accrued_interest

PLEASE ADD A FUNCTION DESCRIPTION

```
accrued_interest(settle_dt: Date,
                 num_ex_dividend_days: int = 0,
                 cal_type: CalendarTypes = CalendarTypes.WEEKEND):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
num_ex_dividend_days	int	-	0
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND

print_payments

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
print_payments(settle_dt: Date,
               face: (float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float or (float	-	-

dirty_price_from_survival_curve

Calculate discounted present value of flows assuming default model. The survival curve treats the coupons as zero recovery payments while the recovery fraction of the par amount is paid at default. For the defaulting principal we discretise the time steps using the coupon payment times. A finer discretisation may handle the time value with more accuracy. I reduce any error by averaging period start and period end payment present values.

```
dirty_price_from_survival_curve(settle_dt: Date,
                                discount_curve: DiscountCurve,
                                survival_curve: DiscountCurve,
                                recovery_rate: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
discount_curve	DiscountCurve	-	-
survival_curve	DiscountCurve	-	-
recovery_rate	float	-	-

clean_price_from_survival_curve

Calculate clean price value of flows assuming default model. The survival curve treats the coupons as zero recovery payments while the recovery fraction of the par amount is paid at default.

```
clean_price_from_survival_curve(settle_dt: Date,
                                discount_curve: DiscountCurve,
                                survival_curve: DiscountCurve,
                                recovery_rate: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
discount_curve	DiscountCurve	-	-
survival_curve	DiscountCurve	-	-
recovery_rate	float	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

7.11 bond_zero

Class: BondZero

A zero coupon bond is a bond which doesnt pay any periodic payments. Instead, it is issued at a discount. The entire face value of the bond is paid out at maturity. It is issued as a deep discount bond.

There is a special convention for accrued interest in which

Accrued_interest = (par - issue price) * D

where D = (settle_dt - issue_dt)/(maturity_dt - issue_dt).

BondZero

Create BondZero object by providing the issue date, maturity Date, face amount and issue price.

```
BondZero(issue_dt: Date,
         maturity_dt: Date,
         issue_price: float          # Issue price usually discounted
         ):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issue_dt	Date	-	-
maturity_dt	Date	-	-
issue_price	float	Issue price usually discounted	-

dirty_price_from_ytm

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

```
dirty_price_from_ytm(settle_dt: Date,
                     ytm: float,
                     convention: YTMCalcType = YTMCalcType.ZERO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.ZERO

principal

Calculate the principal value of the bond based on the face amount from its discount margin and making assumptions about the future Ibor rates.

```
principal(settle_dt: Date,
          ytm: float,
          face: (float),
          convention: YTMCalcType = YTMCalcType.ZERO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
face	float or (float	-	-
convention	YTMCalcType	-	YTMCalcType.ZERO

dollar_duration

Calculate the risk or dP/dy of the bond by bumping. This is also known as the DV01 in Bloomberg.

```
dollar_duration(settle_dt: Date,
                ytm: float,
                convention: YTMCalcType = YTMCalcType.ZERO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.ZERO

macauley_duration

Calculate the Macauley duration of the bond on a settlement date given its yield to maturity.

```
macauley_duration(settle_dt: Date,
                  ytm: float,
                  convention: YTMCalcType = YTMCalcType.ZERO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.ZERO

modified_duration

Calculate the modified duration of the bondon a settlement date given its yield to maturity.

```
modified_duration(settle_dt: Date,
                  ytm: float,
                  convention: YTMCalcType = YTMCalcType.ZERO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.ZERO

convexity_from_ytm

Calculate the bond convexity from the yield to maturity. This function is vectorised with respect to the yield input.

```
convexity_from_ytm(settle_dt: Date,
                   ytm: float,
                   convention: YTMCalcType = YTMCalcType.ZERO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.ZERO

clean_price_from_ytm

Calculate the bond clean price from the yield to maturity. This function is vectorised with respect to the yield input.

```
clean_price_from_ytm(settle_dt: Date,
                     ytm: float,
                     convention: YTMCalcType = YTMCalcType.ZERO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.ZERO

clean_price_from_discount_curve

Calculate the clean bond value using some discount curve to present-value the bond's cash flows back to the curve anchor date and not to the settlement date.

```
clean_price_from_discount_curve(settle_dt: Date,
                                discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

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

dirty_price_from_discount_curve

Calculate the bond price using a provided discount curve to PV the bond's cash flows to the settlement date. As such it is effectively a forward bond price if the settlement date is after the valuation date.

```
dirty_price_from_discount_curve(settle_dt: Date,
                                discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

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

current_yield

Calculate the current yield of the bond which is the coupon divided by the clean price (not the full price). The coupon of a zero coupon bond is defined as: (par - issue_price) / tenor

```
current_yield(clean_price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
clean_price	-	-	-

yield_to_maturity

Calculate the bond's yield to maturity by solving the price yield relationship using a one-dimensional root solver.

```
yield_to_maturity(settle_dt: Date,
                  clean_price: float,
                  convention: YTMCalcType = YTMCalcType.ZERO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
clean_price	float	-	-
convention	YTMCalcType	-	YTMCalcType.ZERO

accrued_interest

Calculate the amount of coupon that has accrued between the previous coupon date and the settlement date. Note that for some day count schemes (such as 30E/360) this is not actually the number of days between the previous coupon payment date and settlement date. If the bond trades with ex-coupon dates then you need to supply the number of days before the coupon date the ex-coupon date is. You can specify the calendar to be used - NONE means only calendar days, WEEKEND is only weekends or you can specify a country calendar for business days.

```
accrued_interest(settle_dt: Date,
                 face: (float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float or (float	-	-

asset_swap_spread

Calculate the par asset swap spread of the bond. The discount curve is a Ibor curve that is passed in. This function is vectorised with respect to the clean price.

```
asset_swap_spread(settle_dt: Date,
    clean_price: float,
    discount_curve: DiscountCurve,
    swapFloatDayCountConventionType=DayCountTypes.ACT_360,
    swapFloatFrequencyType=FrequencyTypes.SEMI_ANNUAL,
    swapFloatCalendarType=CalendarTypes.WEEKEND,
    swapFloatBusDayAdjustRuleType=BusDayAdjustTypes.FOLLOWING,
    swapFloatDateGenRuleType=DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
clean_price	float	-	-
discount_curve	DiscountCurve	-	-
swapFloatDayCountConventionType	-	-	DayCountTypes.ACT_360
swapFloatFrequencyType	-	-	FrequencyTypes.SEMI_ANNUAL
swapFloatCalendarType	-	-	CalendarTypes.WEEKEND
swapFloatBusDayAdjustRuleType	-	-	BusDayAdjustTypes.FOLLOWING
swapFloatDateGenRuleType	-	-	DateGenRuleTypes.BACKWARD

dirty_price_from_oas

Calculate the full price of the bond from its OAS given the bond settlement date, a discount curve and the oas as a number.

```
dirty_price_from_oas(settle_dt: Date,
                     discount_curve: DiscountCurve,
                     oas: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
discount_curve	DiscountCurve	-	-
oas	float	-	-

option_adjusted_spread

Return OAS for bullet bond given settlement date, clean bond price and the discount relative to which the spread is to be computed.

```
option_adjusted_spread(settle_dt: Date,
                       clean_price: float,
                       discount_curve: DiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
clean_price	float	-	-
discount_curve	DiscountCurve	-	-

bond_payments

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
bond_payments(settle_dt: Date,
              face: (float)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float or (float	-	-

print_bond_payments

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
print_bond_payments(settle_dt: Date,
                    face: (float) = 100.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
face	float or (float	-	100.0

dirty_price_from_survival_curve

Calculate discounted present value of flows assuming default model. The survival curve treats the coupons as zero recovery payments while the recovery fraction of the par amount is paid at default. For the defaulting principal we discretize the time steps using the coupon payment times. A finer discretization may handle the time value with more accuracy. I reduce any error by averaging period start and period end payment present values.

```
dirty_price_from_survival_curve(settle_dt: Date,
                                discount_curve: DiscountCurve,
                                survival_curve: DiscountCurve,
                                recovery_rate: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
discount_curve	DiscountCurve	-	-
survival_curve	DiscountCurve	-	-
recovery_rate	float	-	-

clean_price_from_survival_curve

Calculate clean price value of flows assuming default model. The survival curve treats the coupons as zero recovery payments while the recovery fraction of the par amount is paid at default.

```
clean_price_from_survival_curve(settle_dt: Date,
                                discount_curve: DiscountCurve,
                                survival_curve: DiscountCurve,
                                recovery_rate: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
discount_curve	DiscountCurve	-	-
survival_curve	DiscountCurve	-	-
recovery_rate	float	-	-

calc_ror

TODO: TIDY THIS UP !!!!!!!!!!!!!!!!! Calculates the rate of total return (capital return and interest) given a BUY YTM and a SELL YTM of this bond. This function computes the full prices at buying and selling, plus the coupon payments during the period. It returns a tuple which includes a simple rate of return, a compounded IRR and the PnL.

```
calc_ror(begin_dt: Date,
         end_dt: Date,
         begin_ytm: float,
         end_ytm: float,
         convention: YTMCalcType = YTMCalcType.ZERO):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
begin_dt	Date	-	-
end_dt	Date	-	-
begin_ytm	float	-	-
end_ytm	float	-	-
convention	YTMCalcType	-	YTMCalcType.ZERO

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

_f

Function used to do root search in price to yield calculation.

```
_f(ytm, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
ytm	-	-	-
*args	-	-	-

_g

Function used to do root search in price to OAS calculation.

```
_g(oas, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
oas	-	-	-
*args	-	-	-

7.12 yield_curve

Class: BondYieldCurve()

Class to do fitting of the yield curve and to enable interpolation of yields. Because yields assume a flat term structure for each bond, this class does not allow discounting to be done and so does not inherit from FinDiscountCurve. It should only be used for visualisation and simple interpolation but not for full term-structure-consistent pricing.

BondYieldCurve

Fit the curve to a set of bond yields using the type of curve specified. Bounds can be provided if you wish to enforce lower and upper limits on the respective model parameters.

```
BondYieldCurve(settle_dt: Date,
               bonds: list,
               ylds: (np.ndarray, list),
               curveFit):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
bonds	list	-	-
ylds	np.ndarray or list	-	-
curveFit	-	-	-

interpolated_yield

PLEASE ADD A FUNCTION DESCRIPTION

```
interpolated_yield(maturity_dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
maturity_dt	Date	-	-

plot

Display yield curve.

```
plot(title):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
title	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

7.13 yield_curve_model

Class: FinCurveFitMethod()

class FinCurveFitMethod():

Class: CurveFitPolynomial()

class CurveFitPolynomial():

CurveFitPolynomial

PLEASE ADD A FUNCTION DESCRIPTION

```
CurveFitPolynomial(power=3):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
power	-	-	3

_interpolated_yield

PLEASE ADD A FUNCTION DESCRIPTION

```
_interpolated_yield(t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

Class: CurveFitNelsonSiegel()

class CurveFitNelsonSiegel():

CurveFitNelsonSiegel

Fairly permissive bounds. Only tau1 is 1-100

```
CurveFitNelsonSiegel(tau=None, bounds=[(-1, -1, -1, 0.5), (1, 1, 1, 100)]):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
tau	-	-	None
bounds	-	-	[(-1, -1, -1, 0.5), (1, 1, 1, 100)]

_interpolated_yield

PLEASE ADD A FUNCTION DESCRIPTION

```
_interpolated_yield(t, beta1=None, beta2=None,
                    beta3=None, tau=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
beta1	-	-	None
beta2	-	-	None
beta3	-	-	None
tau	-	-	None

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

Class: CurveFitNelsonSiegelSvensson()

class CurveFitNelsonSiegelSvensson():

CurveFitNelsonSiegelSvensson

Create object to store calibration and functional form of NSS parametric fit.

```
CurveFitNelsonSiegelSvensson(tau1=None, tau2=None,
                             bounds=[(0, -1, -1, -1, 0, 1), (1, 1, 1, 1, 10, 100)]):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
tau1	-	-	None
tau2	-	-	None
bounds	-	-	[(0, -1, -1, -1, 0, 1), (1, 1, 1, 1, 10, 100)]

_interpolated_yield

PLEASE ADD A FUNCTION DESCRIPTION

```
_interpolated_yield(t, beta1=None, beta2=None, beta3=None,
                    beta4=None, tau1=None, tau2=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
beta1	-	-	None
beta2	-	-	None
beta3	-	-	None
beta4	-	-	None
tau1	-	-	None
tau2	-	-	None

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

Class: CurveFitBSpline()

class CurveFitBSpline():

CurveFitBSpline

PLEASE ADD A FUNCTION DESCRIPTION

```
CurveFitBSpline(power=3, knots=[1, 3, 5, 10]):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
power	-	-	3
knots	-	-	[1, 3, 5, 10]

_interpolated_yield

PLEASE ADD A FUNCTION DESCRIPTION

```
_interpolated_yield(t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

7.14 zero_curve

Class: BondZeroCurve(DiscountCurve)

BondZeroCurve

Fit a discount curve to a set of bond yields using the type of curve specified.

```
BondZeroCurve(value_dt: Date,
              bonds: list,
              clean_prices: list,
              interp_type: InterpTypes = InterpTypes.FLAT_FWD_RATES):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
bonds	list	-	-
clean_prices	list	-	-
interp_type	InterpTypes	-	InterpTypes.FLAT_FWD_RATES

_bootstrap_zero_rates

PLEASE ADD A FUNCTION DESCRIPTION

```
_bootstrap_zero_rates():
```

The function arguments are described in the following table.

zero_rate

Calculate the zero rate to maturity date.

```
zero_rate(dt: Date,
          frequencyType: FrequencyTypes = FrequencyTypes.CONTINUOUS):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-
frequencyType	FrequencyTypes	-	FrequencyTypes.CONTINUOUS

df

PLEASE ADD A FUNCTION DESCRIPTION

```
df(dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-

survival_prob

PLEASE ADD A FUNCTION DESCRIPTION

```
survival_prob(dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-

fwd

Calculate the continuous forward rate at the forward date.

```
fwd(dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	Date	-	-

fwd_rate

Calculate the forward rate according to the specified day count convention.

```
fwd_rate(date1: Date,
         date2: Date,
         day_count_type: DayCountTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
date1	Date	-	-
date2	Date	-	-
day_count_type	DayCountTypes	-	-

plot

Display yield curve.

```
plot(title: str):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
title	str	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_f

PLEASE ADD A FUNCTION DESCRIPTION

```
_f(df, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
df	-	-	-
*args	-	-	-

7.15 __init__

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:

IborDeposit

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 Liboir term structure.

IborFRA

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.

IborIborSwap - 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.

IborOISwap - 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.

IborIborCcySwap - 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.

IborSingleCurve

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

IborCapFloor

IborSwaption

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

IborBermudanSwaption

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 bermudan_swaption

Class: IborBermudanSwaption

This is the class for the Bermudan-style swaption, an option to enter into a swap (payer or receiver of the fixed coupon), that starts in the future and with a fixed maturity, at a swap rate fixed today. This swaption can be exercised on any of the fixed coupon payment dates after the first exercise date.

IborBermudanSwaption

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.

```
IborBermudanSwaption(settle_dt: Date,
                     exercise_dt: Date,
                     maturity_dt: Date,
                     fixed_leg_type: SwapTypes,
                     exercise_type: FinExerciseTypes,
                     fixed_coupon: float,
                     fixed_freq_type: FrequencyTypes,
                     fixed_dc_type: DayCountTypes,
                     notional=ONE_MILLION,
                     float_freq_type=FrequencyTypes.QUARTERLY,
                     float_dc_type=DayCountTypes.THIRTY_E_360,
                     cal_type=CalendarTypes.WEEKEND,
                     bd_type=BusDayAdjustTypes.FOLLOWING,
                     dg_type=DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
exercise_dt	Date	-	-
maturity_dt	Date	-	-
fixed_leg_type	SwapTypes	-	-
exercise_type	FinExerciseTypes	-	-
fixed_coupon	float	-	-
fixed_freq_type	FrequencyTypes	-	-
fixed_dc_type	DayCountTypes	-	-
notional	-	-	ONE_MILLION
float_freq_type	-	-	FrequencyTypes.QUARTERLY
float_dc_type	-	-	DayCountTypes.THIRTY_E_360
cal_type	-	-	CalendarTypes.WEEKEND
bd_type	-	-	BusDayAdjustTypes.FOLLOWING
dg_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(value_dt,
      discount_curve,
      model):
```

The function arguments are described in the following table.

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

print_swaption_value

PLEASE ADD A FUNCTION DESCRIPTION

```
print_swaption_value():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

PLEASE ADD A FUNCTION DESCRIPTION

```
_print():
```

The function arguments are described in the following table.

8.2 callable_swap

8.3 dual_curve

Class: IborDualCurve(DiscountCurve)

Constructs an index curve as implied by the prices of Ibor deposits, FRAs and IRS. Discounting is assumed to be at a discount rate that is an input and usually derived from OIS rates.

IborDualCurve

Create an instance of a Ibor curve given a valuation date and a set of ibor deposits, ibor FRAs and ibor_swaps. 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(value_dt: Date,
              discount_curve: DiscountCurve,
              ibor_deposits: list,
              ibor_fras: list,
              ibor_swaps: list,
              interp_type: InterpTypes = InterpTypes.FLAT_FWD_RATES,
              check_refit: bool = False):  # Set to True to test it works
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
discount_curve	DiscountCurve	-	-
ibor_deposits	list	-	-
ibor_fras	list	-	-
ibor_swaps	list	-	-
interp_type	InterpTypes	-	InterpTypes.FLAT_FWD_RATES
check_refit	bool	Set to True to test it works	False

_build_curve

Build curve based on interpolation.

```
_build_curve():
```

The function arguments are described in the following table.

_validate_inputs

Validate the inputs for each of the Ibor products.

```
_validate_inputs(ibor_deposits,
                 ibor_fras,
```

```
                 ibor_swaps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
ibor_deposits	-	-	-
ibor_fras	-	-	-
ibor_swaps	-	-	-

_build_curve_using_1d_solver

Construct the discount curve using a bootstrap approach. This is the non-linear slower method that allows the user to choose a number of interpolation approaches between the swap rates and other rates. It involves the use of a solver.

```
_build_curve_using_1d_solver():
```

The function arguments are described in the following table.

_check_refits

Ensure that the Ibor curve refits the calibration instruments.

```
_check_refits(depoTol, fraTol, swapTol):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
depoTol	-	-	-
fraTol	-	-	-
swapTol	-	-	-

__repr__

Print out the details of the Ibor curve.

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_f

Root search objective function for swaps

```
_f(df, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
df	-	-	-
*args	-	-	-

_g

Root search objective function for swaps

```
_g(df, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
df	-	-	-
*args	-	-	-

8.4 ibor_basis_swap

Class: IborBasisSwap

Class for managing an Ibor-Ibor basis swap contract. This is a contract in which a floating leg with one LIBOR tenor is exchanged for a floating leg payment in a different LIBOR tenor. There is no exchange of par. The contract is entered into at zero initial cost. The contract lasts from an effective date to a specified maturity date.

The value of the contract is the NPV of the two coupon streams. Discounting is done on a supplied discount curve which can be different from the two index discount from which the implied index rates are extracted.

IborBasisSwap

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.

```
IborBasisSwap(effective_dt: Date,  # Date interest starts to accrue
              term_dt_or_tenor: (Date, str),  # Date contract ends
              leg1Type: SwapTypes,
              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,
              cal_type: CalendarTypes = CalendarTypes.WEEKEND,
              bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
              dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	Date interest starts to accrue	-
term_dt_or_tenor	Date or str	Date contract ends	-
leg1Type	SwapTypes	-	-
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
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_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(value_dt: 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
value_dt	Date	-	-
discount_curve	DiscountCurve	-	-
index_curveLeg1	DiscountCurve	-	None
index_curveLeg2	DiscountCurve	-	None
firstFixingRateLeg1	-	-	None
firstFixingRateLeg2	-	-	None

print_float_leg_1_pv

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

```
print_float_leg_1_pv():
```

The function arguments are described in the following table.

print_float_leg_2_pv

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

```
print_float_leg_2_pv():
```

The function arguments are described in the following table.

print_payments

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

```
print_payments():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

8.5 ibor_cap_floor

Enumerated Type: IborCapFloorModelTypes

This enumerated type has the following values:

- BLACK
- SHIFTED_BLACK
- SABR

Class: IborCapFloor()

Class for Caps and Floors. These are contracts which observe a Ibor reset L on a future start date and then make a payoff at the end of the Ibor period which is Max[L-K,0] for a cap and Max[K-L,0] for a floor. This is then day count adjusted for the Ibor period and then scaled by the contract notional to produce a valuation. A number of models can be selected from.

IborCapFloor

Initialise IborCapFloor object.

```
IborCapFloor(start_dt: Date,
             maturity_dt_or_tenor: (Date, str),
             option_type: FinCapFloorTypes,
             strike_rate: float,
             last_fixing: Optional[float] = None,
             freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
             dc_type: DayCountTypes = DayCountTypes.THIRTY_E_360_ISDA,
             notional: float = ONE_MILLION,
             cal_type: CalendarTypes = CalendarTypes.WEEKEND,
             bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
             dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_dt	Date	-	-
maturity_dt_or_tenor	Date or str	-	-
option_type	FinCapFloorTypes	-	-
strike_rate	float	-	-
last_fixing	Optional[float]	-	None
freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
dc_type	DayCountTypes	-	DayCountTypes.THIRTY_E_360_ISDA
notional	float	-	ONE_MILLION
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

_generate_dts

PLEASE ADD A FUNCTION DESCRIPTION

```
_generate_dts():
```

The function arguments are described in the following table.

value

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

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

The function arguments are described in the following table.

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

value_caplet_floor_let

Value the caplet or floorlet using a specific model.

```
value_caplet_floor_let(value_dt,
                       caplet_start_dt,
                       caplet_end_dt,
                       libor_curve,
                       model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
caplet_start_dt	-	-	-
caplet_end_dt	-	-	-
libor_curve	-	-	-
model	-	-	-

print_leg

Prints the cap floor payment amounts.

```
print_leg():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

PLEASE ADD A FUNCTION DESCRIPTION

```
_print():
```

The function arguments are described in the following table.

8.6 ibor_conventions

Class: IborConventions()

class IborConventions():

IborConventions

PLEASE ADD A FUNCTION DESCRIPTION

```
IborConventions(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”

8.7 ibor_deposit

Class: IborDeposit

An Ibor deposit is an agreement to borrow money interbank at the Ibor fixing rate starting on the start date and repaid on the maturity date with the interest amount calculated according to a day count convention and dates calculated according to a calendar and business day adjustment rule.

Care must be taken to calculate the correct start (settlement) date. Start with the trade (value) date which is typically today, we may need to add on a number of business days (spot days) to get to the settlement date. The maturity date is then calculated by adding on the deposit tenor/term to the settlement date and adjusting for weekends and holidays according to the calendar and adjustment type.

Note that for over-night (ON) depos the settlement date is today with maturity in one business day. For tomorrow-next (TN) depos the settlement is in one business day with maturity on the following business day. For later maturity deposits, settlement is usually in 1-3 business days. The number of days depends on the currency and jurisdiction of the deposit contract.

IborDeposit

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.

```
IborDeposit(start_dt: Date,  # When the interest starts to accrue
            maturity_dt_or_tenor: (Date, str),  # Repayment of interest
            deposit_rate: float,  # MM rate using simple interest
            dc_type: DayCountTypes,  # How year fraction is calculated
            notional: float = 100.0,  # Amount borrowed
            cal_type: CalendarTypes=CalendarTypes.WEEKEND,  # Maturity date
            bd_type: BusDayAdjustTypes=BusDayAdjustTypes.MODIFIED_FOLLOWING):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_dt	Date	When the interest starts to accrue	-
maturity_dt_or_tenor	Date or str	Repayment of interest	-
deposit_rate	float	MM rate using simple interest	-
dc_type	DayCountTypes	How year fraction is calculated	-
notional	float	Amount borrowed	100.0
cal_type	CalendarTypes	Maturity date	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.MODIFIED_FOLL

_maturity_df

Returns the maturity date discount factor that would allow the Libor curve to reprice the contractual market deposit rate. Note that this is a forward discount factor that starts on settlement date.

```
_maturity_df():
```

The function arguments are described in the following table.

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(value_dt: Date,
      libor_curve):
```

The function arguments are described in the following table.

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

print_payments

Print the date and size of the future repayment.

```
print_payments(value_dt: Date):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-

__repr__

Print the contractual details of the Libor deposit.

```
__repr__():
```

The function arguments are described in the following table.

_print

PLEASE ADD A FUNCTION DESCRIPTION

```
_print():
```

The function arguments are described in the following table.

8.8 ibor_fra

Class: IborFRA

Class for managing LIBOR forward rate agreements. A forward rate agreement is an agreement to exchange a fixed pre-agreed rate for a floating rate linked to LIBOR that is not known until some specified future fixing date. The FRA payment occurs on or soon after this date on the FRA settlement date. Typically the timing gap is two days.

A FRA is used to hedge a Ibor quality loan or lend of some agreed notional amount. This period starts on the settlement date of the FRA and ends on the maturity date of the FRA. For example a 1x4 FRA relates to a Ibor starting in 1 month for a loan period ending in 4 months. Hence it links to 3-month Ibor rate. The amount received by a payer of fixed rate at settlement is:

acc(1,2) * (Ibor(1,2) - FRA RATE) / (1 + acc(0,1) x Ibor(0,1))

So the value at time 0 is

acc(1,2) * (FWD Ibor(1,2) - FRA RATE) x df(0,2)

If the base date of the curve is before the value date then we forward adjust this amount to that value date. For simplicity I have assumed that the fixing date and the settlement date are the same date. This should be amended later.

The valuation below incorporates a dual curve approach.

IborFRA

Create a Forward Rate Agreement object.

```
IborFRA(start_dt: Date,  # The date the FRA starts to accrue
        # End of the Ibor rate period
        maturity_dt_or_tenor: (Date, str),
        fraRate: float,  # The fixed contractual FRA rate
        dc_type: DayCountTypes,  # For interest period
        notional: float = 100.0,
        payFixedRate: bool = True,  # True if the FRA rate is being paid
        cal_type: CalendarTypes = CalendarTypes.WEEKEND,
        bd_type: BusDayAdjustTypes = BusDayAdjustTypes.MODIFIED_FOLLOWING):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
start_dt	Date	The date the FRA starts to accrue	-
maturity_dt_or_tenor	Date or str	-	-
fraRate	float	The fixed contractual FRA rate	-
dc_type	DayCountTypes	For interest period	-
notional	float	-	100.0
payFixedRate	bool	True if the FRA rate is being paid	True
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.MODIFIED_FOLI

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(value_dt: Date,
      discount_curve: DiscountCurve,
      index_curve: DiscountCurve = None):
```

The function arguments are described in the following table.

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

maturity_df

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.

```
maturity_df(index_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
index_curve	-	-	-

print_payments

Determine the value of the Deposit given a Ibor curve.

```
print_payments(value_dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

PLEASE ADD A FUNCTION DESCRIPTION

```
_print():
```

The function arguments are described in the following table.

8.9 ibor_future

Class: IborFuture

IborFuture

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

```
IborFuture(today_dt: Date,
           future_number: int,  # The number of the future after today_dt
           futureTenor: str = "3M",  # '1M', '2M', '3M'
           dc_type: DayCountTypes = DayCountTypes.ACT_360,
           contract_size: float = ONE_MILLION):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
today_dt	Date	-	-
future_number	int	The number of the future after today_dt	-
futureTenor	str	'1M', '2M', '3M'	"3M"
dc_type	DayCountTypes	-	DayCountTypes.ACT_360
contract_size	float	-	ONE_MILLION

to_fra

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

```
to_fra(futures_price, convexity):
```

The function arguments are described in the following table.

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

futures_rate

Calculate implied futures rate from the futures price.

```
futures_rate(futures_price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
futures_price	-	-	-

fra_rate

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.

```
fra_rate(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(value__dt, volatility, mean_reversion):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value__dt	-	-	-
volatility	-	-	-
mean_reversion	-	-	-

__repr__

Print a list of the unadjusted coupon payment _dts used in analytic calculations for the bond.

```
__repr__():
```

The function arguments are described in the following table.

8.10 ibor_lmm_products

Class: IborLMMProducts()

IborLMMProducts

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.

```
IborLMMProducts(settle_dt: Date,
                maturity_dt: Date,
                float_freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
                float_dc_type: DayCountTypes = DayCountTypes.THIRTY_E_360,
                cal_type: CalendarTypes = CalendarTypes.WEEKEND,
                bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
                dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
maturity_dt	Date	-	-
float_freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
float_dc_type	DayCountTypes	-	DayCountTypes.THIRTY_E_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

simulate_1f

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

```
simulate_1f(discount_curve,
            vol_curve: IborCapVolCurve,
            num_paths: int = 1000,
            numeraireIndex: int = 0,
            use_sobol: bool = True,
            seed: int = 42):
```

The function arguments are described in the following table.

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

simulate_mf

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.

```
simulate_mf(discount_curve,
            numFactors: int,
            lambdas: np.ndarray,
            num_paths: int = 10000,
            numeraireIndex: int = 0,
            use_sobol: 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
use_sobol	bool	-	True
seed	int	-	42

simulate_nf

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.

```
simulate_nf(discount_curve,
            vol_curve: IborCapVolCurve,
            corr_matrix: np.ndarray,
            model_type: ModelLMMModelTypes,
            num_paths: int = 1000,
            numeraire_index: int = 0,
            use_sobol: bool = True,
            seed: int = 42):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
discount_curve	-	-	-
vol_curve	IborCapVolCurve	-	-
corr_matrix	np.ndarray	-	-
model_type	ModelLMMModelTypes	-	-
num_paths	int	-	1000
numeraire_index	int	-	0
use_sobol	bool	-	True
seed	int	-	42

value_swaption

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.

```
value_swaption(settle_dt: Date,
               exercise_dt: Date,
               maturity_dt: Date,
               swaptionType: SwapTypes,
               fixed_coupon: float,
               fixed_freq_type: FrequencyTypes,
               fixed_dc_type: DayCountTypes,
               notional: float = ONE_MILLION,
               float_freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
               float_dc_type: DayCountTypes = DayCountTypes.THIRTY_E_360,
               cal_type: CalendarTypes = CalendarTypes.WEEKEND,
               bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
               dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
exercise_dt	Date	-	-
maturity_dt	Date	-	-
swaptionType	SwapTypes	-	-
fixed_coupon	float	-	-
fixed_freq_type	FrequencyTypes	-	-
fixed_dc_type	DayCountTypes	-	-
notional	float	-	ONE_MILLION
float_freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
float_dc_type	DayCountTypes	-	DayCountTypes.THIRTY_E_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value_cap_floor

Value a cap or floor in the LMM.

```
value_cap_floor(settle_dt: Date,
                maturity_dt: Date,
                cap_floor_type: FinCapFloorTypes,
                cap_floor_rate: float,
                freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
                dc_type: DayCountTypes = DayCountTypes.ACT_360,
                notional: float = ONE_MILLION,
                cal_type: CalendarTypes = CalendarTypes.WEEKEND,
                bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
                dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
maturity_dt	Date	-	-
cap_floor_type	FinCapFloorTypes	-	-
cap_floor_rate	float	-	-
freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
dc_type	DayCountTypes	-	DayCountTypes.ACT_360
notional	float	-	ONE_MILLION
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

__repr__

Function to allow us to print the LMM Products details.

```
__repr__():
```

The function arguments are described in the following table.

_print

Alternative print method.

```
_print():
```

The function arguments are described in the following table.

8.11 ibor_single_curve

Class: IborSingleCurve(DiscountCurve)

Constructs one discount and index curve as implied by prices of Ibor deposits, FRAs and IRS. Discounting is assumed to be at Libor and the value of the floating leg (including a notional) is assumed to be par. This approach has been overtaken since 2008 as OIS discounting has become the agreed discounting approach for ISDA derivatives. This curve method is therefore intended for those happy to assume simple Libor discounting.

The curve date is the date on which we are performing the valuation based on the information available on the curve date. Typically it is the date on which an amount of 1 unit paid has a present value of 1. This class inherits from FinDiscountCurve and so it has all of the methods that that class has.

There are two main curve-building approaches:

1) The first uses a bootstrap that interpolates swap rates linearly for coupon dates that fall between the swap maturity dates. With this, we can solve for the discount factors iteratively without need of a solver. This will give us a set of discount factors on the grid dates that refit the market exactly. However, when extracting discount factors, we will then assume flat forward rates between these coupon dates. There is no contradiction as it is as though we had been quoted a swap curve with all of the market swap rates, and with an additional set as though the market quoted swap rates at a higher frequency than the market.

2) The second uses a bootstrap that uses only the swap rates provided but which also assumes that forwards are flat between these swap maturity dates. This approach is non-linear and so requires a solver. Consequently it is slower. Its advantage is that we can switch interpolation schemes to provide a smoother or other functional curve shape which may have a more economically justifiable shape. However the root search makes it slower.

IborSingleCurve

Create an instance of a FinIbor curve given a valuation date and a set of ibor deposits, ibor FRAs and ibor_swaps. 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(value_dt: Date,  # This is the trade date (not T+2)
                ibor_deposits: list,
                ibor_fras: list,
                ibor_swaps: list,
                interp_type: InterpTypes = InterpTypes.FLAT_FWD_RATES,
                check_refit: bool = False):  # Set to True to test it works
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	This is the trade date (not T+2)	-
ibor_deposits	list	-	-
ibor_fras	list	-	-
ibor_swaps	list	-	-
interp_type	InterpTypes	-	InterpTypes.FLAT_FWD_RATES
check_refit	bool	Set to True to test it works	False

_build_curve

Build curve based on interpolation.

```
_build_curve():
```

The function arguments are described in the following table.

_validate_inputs

Validate the inputs for each of the Ibor products.

```
_validate_inputs(ibor_deposits,
                 ibor_fras,
                 ibor_swaps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
ibor_deposits	-	-	-
ibor_fras	-	-	-
ibor_swaps	-	-	-

_build_curve_using_1d_solver

Construct the discount curve using a bootstrap approach. This is the non-linear slower method that allows the user to choose a number of interpolation approaches between the swap rates and other rates. It involves the use of a solver.

```
_build_curve_using_1d_solver():
```

The function arguments are described in the following table.

_build_curve_using_quadratic_minimiser

Construct the discount curve using a minimisation approach. This is the This enables a more complex interpolation scheme.

```
_build_curve_using_quadratic_minimiser():
```

The function arguments are described in the following table.

_build_curve_linear_swap_rate_interpolation

Construct the discount curve using a bootstrap approach. This is the linear swap rate method that is fast and exact as it does not require the use of a solver. It is also market standard.

```
_build_curve_linear_swap_rate_interpolation():
```

The function arguments are described in the following table.

_check_refits

Ensure that the Ibor curve refits the calibration instruments.

```
_check_refits(depoTol, fraTol, swapTol):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
depoTol	-	-	-
fraTol	-	-	-
swapTol	-	-	-

__repr__

Print out the details of the Ibor curve.

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_f

Root search objective function for IRS

```
_f(df, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
df	-	-	-
*args	-	-	-

_g

Root search objective function for FRAs

```
_g(df, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
df	-	-	-
*args	-	-	-

_cost_function

Root search objective function for swaps

```
_cost_function(dfs, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dfs	-	-	-
*args	-	-	-

8.12 ibor_swap

Class: IborSwap

Class for managing a standard Fixed vs IBOR swap. This is a contract in which a fixed payment leg is exchanged for a series of floating rates payments linked to some IBOR index rate. There is no exchange of principal. The contract is entered into at zero initial cost. The contract lasts from a start date to a specified maturity date.

The floating rate is not known fully until the end of the preceding payment period. It is set in advance and paid in arrears.

The value of the contract is the NPV of the two coupon streams. Discounting is done on a supplied discount curve which is separate from the curve from which the implied index rates are extracted.

IborSwap

Create an interest rate 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.

```
IborSwap(effective_dt: Date,  # Date interest starts to accrue
         term_dt_or_tenor: (Date, str),  # Date contract ends
         fixed_leg_type: SwapTypes,
         fixed_coupon: float,  # Fixed coupon (annualised)
         fixed_freq_type: FrequencyTypes,
         fixed_dc_type: DayCountTypes,
         notional: float = ONE_MILLION,
         float_spread: float = 0.0,
         float_freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
         float_dc_type: DayCountTypes = DayCountTypes.THIRTY_E_360,
         cal_type: CalendarTypes = CalendarTypes.WEEKEND,
         bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
         dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	Date interest starts to accrue	-
term_dt_or_tenor	Date or str	Date contract ends	-
fixed_leg_type	SwapTypes	-	-
fixed_coupon	float	Fixed coupon (annualised)	-
fixed_freq_type	FrequencyTypes	-	-
fixed_dc_type	DayCountTypes	-	-
notional	float	-	ONE_MILLION
float_spread	float	-	0.0
float_freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
float_dc_type	DayCountTypes	-	DayCountTypes.THIRTY_E_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD

value

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

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

The function arguments are described in the following table.

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

pv01

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

```
pv01(value_dt, discount_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
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. The swap rate can also be calculated in a dual curve approach but in this case the first fixing on the floating leg is needed.*

```
swap_rate(value_dt: Date,
          discount_curve: DiscountCurve,
          index_curve: DiscountCurve = None,
          first_fixing: float = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
discount_curve	DiscountCurve	-	-
index_curve	DiscountCurve	-	None
first_fixing	float	-	None

cash_settled_pv01

Calculate the forward value of an annuity of a forward starting swap using a single flat discount rate equal to the swap rate. This is used in the pricing of a cash-settled swaption in the IborSwaption class. This method does not affect the standard valuation methods.

```
cash_settled_pv01(value_dt,
                  flat_swap_rate,
                  freq_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
flat_swap_rate	-	-	-
freq_type	-	-	-

print_fixed_leg_pv

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

```
print_fixed_leg_pv():
```

The function arguments are described in the following table.

print_float_leg_pv

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

```
print_float_leg_pv():
```

The function arguments are described in the following table.

print_payments

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

```
print_payments():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

8.13 ibor_swaption

Class: IborSwaption()

This is the class for the European-style swaption, an option to enter into a swap (payer or receiver of the fixed coupon), that starts in the future and with a fixed maturity, at a swap rate fixed today.

IborSwaption

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 IborBermudanSwaption class.

```
IborSwaption(settle_dt: Date,
             exercise_dt: Date,
             maturity_dt: Date,
             fixed_leg_type: SwapTypes,
             fixed_coupon: float,
             fixed_freq_type: FrequencyTypes,
             fixed_dc_type: DayCountTypes,
             notional: float = ONE_MILLION,
             float_freq_type: FrequencyTypes = FrequencyTypes.QUARTERLY,
             float_dc_type: DayCountTypes = DayCountTypes.THIRTY_E_360,
             cal_type: CalendarTypes = CalendarTypes.WEEKEND,
             bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
             dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settle_dt	Date	-	-
exercise_dt	Date	-	-
maturity_dt	Date	-	-
fixed_leg_type	SwapTypes	-	-
fixed_coupon	float	-	-
fixed_freq_type	FrequencyTypes	-	-
fixed_dc_type	DayCountTypes	-	-
notional	float	-	ONE_MILLION
float_freq_type	FrequencyTypes	-	FrequencyTypes.QUARTERLY
float_dc_type	DayCountTypes	-	DayCountTypes.THIRTY_E_360
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_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, SABR, SABRShifted, FinModelHW, FinModelBK and FinModelBDT. The last two involved a tree-based valuation.

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

The function arguments are described in the following table.

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

cash_settled_value

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.

```
cash_settled_value(value_dt: Date,
                   discount_curve,
                   swap_rate: float,
                   model):
```

The function arguments are described in the following table.

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

print_swap_fixed_leg

PLEASE ADD A FUNCTION DESCRIPTION

```
print_swap_fixed_leg():
```

The function arguments are described in the following table.

print_swap_float_leg

PLEASE ADD A FUNCTION DESCRIPTION

```
print_swap_float_leg():
```

The function arguments are described in the following table.

__repr__

Function to allow us to print the swaption details.

```
__repr__():
```

The function arguments are described in the following table.

_print

Alternative print method.

```
_print():
```

The function arguments are described in the following table.

8.14 ois

Enumerated Type: FinCompoundingTypes

This enumerated type has the following values:

- COMPOUNDED
- OVERNIGHT_COMPOUNDED_ANNUAL_RATE
- AVERAGED
- AVERAGED_DAILY

Class: OIS

Class for managing overnight index rate swaps (OIS) and Fed Fund swaps. This is a contract in which a fixed payment leg is exchanged for a payment which pays the rolled-up overnight index rate (OIR). There is no exchange of par. The contract is entered into at zero initial cost.

NOTE: This class is almost identical to IborSwap but will possibly deviate as distinctions between the two become clear to me. If not they will be converged (or inherited) to avoid duplication.

The contract lasts from a start date to a specified maturity date. The fixed coupon is the OIS fixed rate for the corresponding tenor which is set at contract initiation.

The floating rate is not known fully until the end of each payment period. Its calculated at the contract maturity and is based on daily observations of the overnight index rate which are compounded according to a specific convention. Hence the OIS floating rate is determined by the history of the OIS rates.

In its simplest form, there is just one fixed rate payment and one floating rate payment at contract maturity. However when the contract becomes longer than one year the floating and fixed payments become periodic, usually with annual exchanges of cash.

The value of the contract is the NPV of the two coupon streams. Discounting is done on the OIS curve which is itself implied by the term structure of market OIS rates.

OIS

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.

```
OIS(effective_dt: Date,  # Date interest starts to accrue
    term_dt_or_tenor: (Date, str),  # Date contract ends
    fixed_leg_type: SwapTypes,
    fixed_coupon: float,  # Fixed coupon (annualised)
    fixed_freq_type: FrequencyTypes,
    fixed_dc_type: DayCountTypes,
    notional: float = ONE_MILLION,
    payment_lag: int = 0,  # Number of days after period payment occurs
    float_spread: float = 0.0,
    float_freq_type: FrequencyTypes = FrequencyTypes.ANNUAL,
    float_dc_type: DayCountTypes = DayCountTypes.THIRTY_E_360,
    cal_type: CalendarTypes = CalendarTypes.WEEKEND,
```

```
    bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
    dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	Date interest starts to accrue	-
term_dt_or_tenor	Date or str	Date contract ends	-
fixed_leg_type	SwapTypes	-	-
fixed_coupon	float	Fixed coupon (annualised)	-
fixed_freq_type	FrequencyTypes	-	-
fixed_dc_type	DayCountTypes	-	-
notional	float	-	ONE_MILLION
payment_lag	int	Number of days after period payment occurs	0
float_spread	float	-	0.0
float_freq_type	FrequencyTypes	-	FrequencyTypes.ANNUAL
float_dc_type	DayCountTypes	-	DayCountTypes.THIRTY_E_
cal_type	CalendarTypes	-	CalendarTypes.WEEKENI
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOW
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWA

value

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

```
value(value_dt: Date,
      ois_curve: DiscountCurve,
      first_fixing_rate=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
ois_curve	DiscountCurve	-	-
first_fixing_rate	-	-	None

pv01

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

```
pv01(value_dt, discount_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
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(value_dt, ois_curve, first_fixing_rate=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
ois_curve	-	-	-
first_fixing_rate	-	-	None

print_fixed_leg_pv

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

```
print_fixed_leg_pv():
```

The function arguments are described in the following table.

print_float_leg_pv

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

```
print_float_leg_pv():
```

The function arguments are described in the following table.

print_payments

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

```
print_payments():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

8.15 ois_basis_swap

Class: OISBasisSwap

Class for managing an Ibor-OIS basis swap contract. This is a contract in which a floating leg with one LIBOR tenor is exchanged for a floating leg payment of an overnight index swap. There is no exchange of par. The contract is entered into at zero initial cost. The contract lasts from a start date to a specified maturity date.

The value of the contract is the NPV of the two coupon streams. Discounting is done on a supplied discount curve which is separate from the discount from which the implied index rates are extracted.

OISBasisSwap

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.

```
OISBasisSwap(effective_dt: Date,  # Date interest starts to accrue
             term_dt_or_tenor: (Date, str),  # Date contract ends
             iborType: SwapTypes,
             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,
             cal_type: CalendarTypes = CalendarTypes.WEEKEND,
             bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
             dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	Date interest starts to accrue	-
term_dt_or_tenor	Date or str	Date contract ends	-
iborType	SwapTypes	-	-
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
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_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(value_dt: 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
value_dt	Date	-	-
discount_curve	DiscountCurve	-	-
indexIborCurve	DiscountCurve	-	None
indexOISCurve	DiscountCurve	-	None
firstFixingRateLeg1	-	-	None
firstFixingRateLeg2	-	-	None

print_payments

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

```
print_payments():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

8.16 ois_curve

Class: OISCurve(DiscountCurve)

Constructs a discount curve as implied by the prices of Overnight Index Rate swaps. The curve date is the date on which we are performing the valuation based on the information available on the curve date. Typically it is the date on which an amount of 1 unit paid has a present value of 1. This class inherits from FinDiscountCurve and so it has all of the methods that that class has.

The construction of the curve is assumed to depend on just the OIS curve, i.e. it does not include information from Ibor-OIS basis swaps. For this reason I call it a one-curve.

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(value_dt: Date,
         ois_deposits: list,
         ois_fras: list,
         ois_swaps: list,
         interp_type: InterpTypes = InterpTypes.FLAT_FWD_RATES,
         check_refit: bool = False):  # Set to True to test it works
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
ois_deposits	list	-	-
ois_fras	list	-	-
ois_swaps	list	-	-
interp_type	InterpTypes	-	InterpTypes.FLAT_FWD_RATES
check_refit	bool	Set to True to test it works	False

_build_curve

Build curve based on interpolation.

```
_build_curve():
```

The function arguments are described in the following table.

_validate_inputs

Validate the inputs for each of the Libor products.

```
_validate_inputs(oisDeposits,
                 oisFRAs,
                 oisSwaps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
oisDeposits	-	-	-
oisFRAs	-	-	-
oisSwaps	-	-	-

_build_curve_using_1d_solver

Construct the discount curve using a bootstrap approach. This is the non-linear slower method that allows the user to choose a number of interpolation approaches between the swap rates and other rates. It involves the use of a solver.

```
_build_curve_using_1d_solver():
```

The function arguments are described in the following table.

_build_curve_linear_swap_rate_interpolation

Construct the discount curve using a bootstrap approach. This is the linear swap rate method that is fast and exact as it does not require the use of a solver. It is also market standard.

```
_build_curve_linear_swap_rate_interpolation():
```

The function arguments are described in the following table.

_check_refits

Ensure that the Libor curve refits the calibration instruments.

```
_check_refits(depoTol, fraTol, swapTol):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
depoTol	-	-	-
fraTol	-	-	-
swapTol	-	-	-

__repr__

Print out the details of the Libor curve.

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

_fois

Extract the implied overnight index rate assuming it is flat over period in question.

```
_fois(oir, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
oir	-	-	-
*args	-	-	-

_f

Root search objective function for OIS

```
_f(df, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
df	-	-	-
*args	-	-	-

_g

Root search objective function for swaps

```
_g(df, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
df	-	-	-
*args	-	-	-

8.17 swap_fixed_leg

Class: SwapFixedLeg

Class for managing the fixed leg of a swap. A fixed leg is a leg with a sequence of flows calculated according to an ISDA schedule and with a coupon that is fixed over the life of the swap.

SwapFixedLeg

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.

```
SwapFixedLeg(effective_dt: Date,  # Date interest starts to accrue
             end_dt: (Date, str),  # Date contract ends
             leg_type: SwapTypes,
             coupon: (float),
             freq_type: FrequencyTypes,
             dc_type: DayCountTypes,
             notional: float = ONE_MILLION,
             principal: float = 0.0,
             payment_lag: int = 0,
             cal_type: CalendarTypes = CalendarTypes.WEEKEND,
             bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
             dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD,
             end_of_month: bool = False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	Date interest starts to accrue	-
end_dt	Date or str	Date contract ends	-
leg_type	SwapTypes	-	-
coupon	float or (float	-	-
freq_type	FrequencyTypes	-	-
dc_type	DayCountTypes	-	-
notional	float	-	ONE_MILLION
principal	float	-	0.0
payment_lag	int	-	0
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD
end_of_month	bool	-	False

generate_payments

These are generated immediately as they are for the entire life of the swap. Given a valuation date we can determine which cash flows are in the future and value the swap The schedule allows for a specified lag in *the payment date Nothing is paid on the swap effective date and so the first payment date is the first actual payment date.*

```
generate_payments():
```

The function arguments are described in the following table.

value

PLEASE ADD A FUNCTION DESCRIPTION

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

The function arguments are described in the following table.

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

print_payments

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

```
print_payments():
```

The function arguments are described in the following table.

print_valuation

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

```
print_valuation():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

8.18 swap_float_leg

Class: SwapFloatLeg

Class for managing the floating leg of a swap. A float leg consists of a sequence of flows calculated according to an ISDA schedule and with a coupon determined by an index curve which changes over life of the swap.

SwapFloatLeg

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.

```
SwapFloatLeg(effective_dt: Date,  # Date interest starts to accrue
             end_dt: (Date, str),  # Date contract ends
             leg_type: SwapTypes,
             spread: (float),
             freq_type: FrequencyTypes,
             dc_type: DayCountTypes,
             notional: float = ONE_MILLION,
             principal: float = 0.0,
             payment_lag: int = 0,
             cal_type: CalendarTypes = CalendarTypes.WEEKEND,
             bd_type: BusDayAdjustTypes = BusDayAdjustTypes.FOLLOWING,
             dg_type: DateGenRuleTypes = DateGenRuleTypes.BACKWARD,
             end_of_month: bool = False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	Date interest starts to accrue	-
end_dt	Date or str	Date contract ends	-
leg_type	SwapTypes	-	-
spread	float or (float	-	-
freq_type	FrequencyTypes	-	-
dc_type	DayCountTypes	-	-
notional	float	-	ONE_MILLION
principal	float	-	0.0
payment_lag	int	-	0
cal_type	CalendarTypes	-	CalendarTypes.WEEKEND
bd_type	BusDayAdjustTypes	-	BusDayAdjustTypes.FOLLOWING
dg_type	DateGenRuleTypes	-	DateGenRuleTypes.BACKWARD
end_of_month	bool	-	False

generate_payment_dts

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

```
generate_payment_dts():
```

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(value_dt: 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
value_dt	Date	This should be the settlement date	-
discount_curve	DiscountCurve	-	-
index_curve	DiscountCurve	-	-
firstFixingRate	float	-	None

print_payments

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

```
print_payments():
```

The function arguments are described in the following table.

print_valuation

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

```
print_valuation():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

8.19 __init__

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

9.1 fx_barrier_option

Enumerated Type: FinFXBarrierTypes

This enumerated type has the following values:

- DOWN_AND_OUT_CALL
- DOWN_AND_IN_CALL
- UP_AND_OUT_CALL
- UP_AND_IN_CALL
- UP_AND_OUT_PUT
- UP_AND_IN_PUT
- DOWN_AND_OUT_PUT
- DOWN_AND_IN_PUT

Class: FXBarrierOption(FXOption)

class FXBarrierOption(FXOption):

FXBarrierOption

Create FX Barrier option product. This is an option that cancels if the FX rate crosses a barrier during the life of the option.

```
FXBarrierOption(expiry_dt: Date,
                strike_fx_rate: float,  # 1 unit of foreign in domestic
                currency_pair: str,  # FORDOM
                option_type: FinFXBarrierTypes,
                barrier_level: float,
                num_obs_per_year: int,
                notional: float,
                notional_currency: str):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
strike_fx_rate	float	1 unit of foreign in domestic	-
currency_pair	str	FORDOM	-
option_type	FinFXBarrierTypes	-	-
barrier_level	float	-	-
num_obs_per_year	int	-	-
notional	float	-	-
notional_currency	str	-	-

value

Value FX Barrier Option using Black-Scholes model with closed-form analytical models.

```
value(value_dt,
      spot_fx_rate,
      dom_discount_curve,
      for_discount_curve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	-	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-
model	-	-	-

value_mc

Value the FX Barrier Option using Monte Carlo.

```
value_mc(value_dt,
         spot_fx_rate,
         dom_interest_rate,
         process_type,
         model_params,
         num_ann_steps=552,
         num_paths=5000,
         seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	-	-
dom_interest_rate	-	-	-
process_type	-	-	-
model_params	-	-	-
num_ann_steps	-	-	552
num_paths	-	-	5000
seed	-	-	4242

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Print a list of the unadjusted coupon payment dates used in analytic calculations for the bond.

```
_print():
```

The function arguments are described in the following table.

9.2 fx_digital_option

Class: FXDigitalOption

class FXDigitalOption:

FXDigitalOption

Create the FX Digital Option object. Inputs include expiry date, strike, currency pair, option type (call or put), notional and the currency of the notional. And adjustment for spot days is enabled. All currency rates must be entered in the price in domestic currency of one unit of foreign. And the currency pair should be in the form FORDOM where FOR is the foreign currency pair currency code and DOM is the same for the domestic currency.

```
FXDigitalOption(expiry_dt: Date,
                strike_fx_rate: (float, np.ndarray),
                currency_pair: str,  # FORDOM
                option_type: (OptionTypes, list),
                notional: float,
                prem_currency: str,
                spot_days: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
strike_fx_rate	float or np.ndarray	-	-
currency_pair	str	FORDOM	-
option_type	OptionTypes or list	-	-
notional	float	-	-
prem_currency	str	-	-
spot_days	int	-	0

value

Valuation of a digital option using Black-Scholes model. This allows for 4 cases - first upper barriers that when crossed pay out cash (calls) and lower barriers than when crossed from above cause a cash payout (puts) PLUS the fact that the cash payment can be in domestic or foreign currency.

```
value(value_dt,
      spot_fx_rate,  # 1 unit of foreign in domestic
      dom_discount_curve,
      for_discount_curve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	1 unit of foreign in domestic	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-
model	-	-	-

9.3 fx_double_digital_option

Class: FXDoubleDigitalOption

class FXDoubleDigitalOption:

FXDoubleDigitalOption

Create the FX Double Digital Option object. Inputs include expiry date, upper strike, lower strike, currency pair, option type notional and the currency of the notional. An adjustment for spot days is enabled. All currency rates must be entered in the price in domestic currency of one unit of foreign. And the currency pair should be in the form FORDOM where FOR is the foreign currency pair currency code and DOM is the same for the domestic currency.

```
FXDoubleDigitalOption(expiry_dt: Date,
                      upper_strike: (float, np.ndarray),
                      lower_strike: (float, np.ndarray),
                      currency_pair: str,  # FORDOM
                      notional: float,
                      prem_currency: str,
                      spot_days: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
upper_strike	float or np.ndarray	-	-
lower_strike	float or np.ndarray	-	-
currency_pair	str	FORDOM	-
notional	float	-	-
prem_currency	str	-	-
spot_days	int	-	0

value

Valuation of a double digital option using Black-Scholes model. The option pays out the notional in the premium currency if the fx rate is between the upper and lower strike at maturity. The valuation is equivalent to the valuation of the difference of the value of two digital puts, one with the upper and the other with the lower strike

```
value(value_dt,
      spot_fx_rate,  # 1 unit of foreign in domestic
      dom_discount_curve,
      for_discount_curve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	1 unit of foreign in domestic	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-
model	-	-	-

9.4 fx_fixed_lookback_option

Class: FXFixedLookbackOption

FXFixedLookbackOption

Create option with expiry date, option type and the option strike

```
FXFixedLookbackOption(expiry_dt: Date,
                      option_type: OptionTypes,
                      optionStrike: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
option_type	OptionTypes	-	-
optionStrike	float	-	-

value

Value FX Fixed Lookback Option using Black Scholes model and analytical formulae.

```
value(value_dt: Date,
      stock_price: float,
      dom_discount_curve: DiscountCurve,
      for_discount_curve: DiscountCurve,
      volatility: float,
      stock_min_max: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
dom_discount_curve	DiscountCurve	-	-
for_discount_curve	DiscountCurve	-	-
volatility	float	-	-
stock_min_max	float	-	-

value_mc

Value FX Fixed Lookback option using Monte Carlo.

```
value_mc(value_dt: Date,
         spot_fx_rate: float,  # FORDOM
         domestic_curve: DiscountCurve,
         foreign_curve: DiscountCurve,
         volatility: float,
```

```
        spot_fx_rateMinMax: float,
        num_paths: int = 10000,
        num_steps_per_year: int = 252,
        seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
spot_fx_rate	float	FORDOM	-
domestic_curve	DiscountCurve	-	-
foreign_curve	DiscountCurve	-	-
volatility	float	-	-
spot_fx_rateMinMax	float	-	-
num_paths	int	-	10000
num_steps_per_year	int	-	252
seed	int	-	4242

9.5 fx_float_lookback_option

Class: FXFloatLookbackOption(FXOption)

This is an FX option in which the strike of the option is not fixed but is set at expiry to equal the minimum fx rate in the case of a call or the maximum fx rate in the case of a put.

FXFloatLookbackOption

Create the FX Float Look Back Option by specifying the expiry date and the option type.

```
FXFloatLookbackOption(expiry_dt: Date,
                      option_type: OptionTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
option_type	OptionTypes	-	-

value

Valuation of the Floating Lookback option using Black-Scholes using the formulae derived by Goldman, Sosin and Gatto (1979).

```
value(value_dt: Date,
      stock_price: float,
      domestic_curve: DiscountCurve,
      foreign_curve: DiscountCurve,
      volatility: float,
      stock_min_max: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
domestic_curve	DiscountCurve	-	-
foreign_curve	DiscountCurve	-	-
volatility	float	-	-
stock_min_max	float	-	-

value_mc

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc(value_dt,
         stock_price,
         domestic_curve,
```

```
            foreign_curve,
            volatility,
            stock_min_max,
            num_paths=10000,
            num_steps_per_year=252,
            seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	-	-
domestic_curve	-	-	-
foreign_curve	-	-	-
volatility	-	-	-
stock_min_max	-	-	-
num_paths	-	-	10000
num_steps_per_year	-	-	252
seed	-	-	4242

9.6 fx_forward

Class: FXForward

FXForward

Creates a FinFXForward which allows the owner to buy the FOR against the DOM currency at the strike_fx_rate and to pay it in the notional currency.

```
FXForward(expiry_dt: Date,
          strike_fx_rate: float,  # PRICE OF 1 UNIT OF FOREIGN IN DOM CCY
          currency_pair: str,  # FOR DOM
          notional: float,
          notional_currency: str,  # must be FOR or DOM
          spot_days: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
strike_fx_rate	float	PRICE OF 1 UNIT OF FOREIGN IN DOM CCY	-
currency_pair	str	FOR DOM	-
notional	float	-	-
notional_currency	str	must be FOR or DOM	-
spot_days	int	-	0

value

Calculate the value of an FX forward contract where the current FX rate is the spot_fx_rate.

```
value(value_dt,
      spot_fx_rate,  # 1 unit of foreign in domestic
      dom_discount_curve,
      for_discount_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	1 unit of foreign in domestic	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-

forward

Calculate the FX Forward rate that makes the value of the FX contract equal to zero.

```
forward(value_dt,
        spot_fx_rate,  # 1 unit of foreign in domestic
```

```
        dom_discount_curve,
        for_discount_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	1 unit of foreign in domestic	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

9.7 fx_mkt_conventions

Enumerated Type: FinFXATMMethod

This enumerated type has the following values:
- SPOT
- FWD
- FWD_DELTA_NEUTRAL
- FWD_DELTA_NEUTRAL_PREM_ADJ

Enumerated Type: FinFXDeltaMethod

This enumerated type has the following values:
- SPOT_DELTA
- FORWARD_DELTA
- SPOT_DELTA_PREM_ADJ
- FORWARD_DELTA_PREM_ADJ

Class: FinFXRate()

class FinFXRate():

FinFXRate

PLEASE ADD A FUNCTION DESCRIPTION

```
FinFXRate(ccy1,
          ccy2,
          rate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
ccy1	-	-	-
ccy2	-	-	-
rate	-	-	-

9.8 fx_one_touch_option

Class: FXOneTouchOption(FXOption)

A FinFXOneTouchOption is an option in which the buyer receives one unit of currency if the FX rate touches a barrier at any time before the option expiry date and zero otherwise. The single barrier payoff must define whether the option pays or cancels if the barrier is touched and also when the payment is made (at hit time or option expiry). All of these variants are members of the FinTouchOptionTypes type.

FXOneTouchOption

Create the one touch option by defining its expiry date and the barrier level and a payment size if it is a cash
.

```
FXOneTouchOption(expiry_dt: Date,
                 option_type: TouchOptionTypes,
                 barrier_rate: float,
                 payment_size: float = 1.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
option_type	TouchOptionTypes	-	-
barrier_rate	float	-	-
payment_size	float	-	1.0

value

FX One-Touch Option valuation using the Black-Scholes model assuming a continuous (American) barrier from value date to expiry. Handles both cash-or-nothing and asset-or-nothing options.

```
value(value_dt: Date,
      spot_fx_rate: (float, np.ndarray),
      dom_discount_curve: DiscountCurve,
      for_discount_curve: DiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
spot_fx_rate	float or np.ndarray	-	-
dom_discount_curve	DiscountCurve	-	-
for_discount_curve	DiscountCurve	-	-
model	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

value_mc

Touch Option valuation using the Black-Scholes model and Monte Carlo simulation. Accuracy is not great when compared to the analytical result as we only observe the barrier a finite number of times. The convergence is slow.

```
value_mc(value_dt: Date,
         stock_price: float,
         domCurve: DiscountCurve,
         forCurve: DiscountCurve,
         model,
         num_paths: int = 10000,
         num_steps_per_year: int = 252,
         seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	Date	-	-
stock_price	float	-	-
domCurve	DiscountCurve	-	-
forCurve	DiscountCurve	-	-
model	-	-	-
num_paths	int	-	10000
num_steps_per_year	int	-	252
seed	int	-	4242

_barrier_pay_one_at_hit_pv_down

Pay $1 if the stock crosses the barrier H from above. PV payment.

```
_barrier_pay_one_at_hit_pv_down(s, H, r, dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
H	-	-	-
r	-	-	-
dt	-	-	-

_barrier_pay_one_at_hit_pv_up

Pay $1 if the stock crosses the barrier H from below. PV payment.

```
_barrier_pay_one_at_hit_pv_up(s, H, r, dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
H	-	-	-
r	-	-	-
dt	-	-	-

_barrier_pay_asset_at_expiry_down_out

Pay $1 if the stock crosses the barrier H from above. PV payment.

```
_barrier_pay_asset_at_expiry_down_out(s, H):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
H	-	-	-

_barrier_pay_asset_at_expiry_up_out

Pay $1 if the stock crosses the barrier H from below. PV payment.

```
_barrier_pay_asset_at_expiry_up_out(s, H):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
H	-	-	-

9.9 fx_option

Class: FXOption

delta

Calculate the option delta (FX rate sensitivity) by adding on a small bump and calculating the change in the option price.

```
delta(value_dt,
      spot_fx_rate,
      discount_curve,
      dividend_curve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
model	-	-	-

gamma

Calculate the option gamma (delta sensitivity) by adding on a small bump and calculating the change in the option delta.

```
gamma(value_dt, spot_fx_rate, discount_curve, dividend_curve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
model	-	-	-

vega

Calculate the option vega (volatility sensitivity) by adding on a small bump and calculating the change in the option price.

```
vega(value_dt, spot_fx_rate, discount_curve, dividend_curve, model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
model	-	-	-

theta

Calculate the option theta (calendar time sensitivity) by moving forward one day and calculating the change in the option price.

```
theta(value_dt, spot_fx_rate, discount_curve, dividend_curve, model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
model	-	-	-

rho

Calculate the option rho (interest rate sensitivity) by perturbing the discount curve and revaluing.

```
rho(value_dt, spot_fx_rate, discount_curve, dividend_curve, model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
model	-	-	-

9.10 fx_rainbow_option

Enumerated Type: FXRainbowOptionTypes

This enumerated type has the following values:

- CALL_ON_MAXIMUM
- PUT_ON_MAXIMUM
- CALL_ON_MINIMUM
- PUT_ON_MINIMUM
- CALL_ON_NTH
- PUT_ON_NTH

Class: FXRainbowOption(EquityOption)

class FXRainbowOption(EquityOption):

FXRainbowOption

PLEASE ADD A FUNCTION DESCRIPTION

```
FXRainbowOption(expiry_dt: Date,
                payoff_type: FXRainbowOptionTypes,
                payoff_params: List[float],
                num_assets: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
payoff_type	FXRainbowOptionTypes	-	-
payoff_params	List[float]	-	-
num_assets	int	-	-

validate

PLEASE ADD A FUNCTION DESCRIPTION

```
validate(stock_prices,
         dividend_yields,
         volatilities,
         betas):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stock_prices	-	-	-
dividend_yields	-	-	-
volatilities	-	-	-
betas	-	-	-

validate_payoff

PLEASE ADD A FUNCTION DESCRIPTION

```
validate_payoff(payoff_type, payoff_params, num_assets):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
payoff_type	-	-	-
payoff_params	-	-	-
num_assets	-	-	-

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(value_dt,
      stock_prices,
      dom_discount_curve,
      for_discount_curve,
      volatilities,
      betas):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_prices	-	-	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-
volatilities	-	-	-
betas	-	-	-

value_mc

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc(value_dt,
         expiry_dt,
         stock_prices,
         discount_curve,
         dividend_yields,
         volatilities,
         betas,
         num_paths=10000,
         seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
expiry_dt	-	-	-
stock_prices	-	-	-
discount_curve	-	-	-
dividend_yields	-	-	-
volatilities	-	-	-
betas	-	-	-
num_paths	-	-	10000
seed	-	-	4242

payoff_value

PLEASE ADD A FUNCTION DESCRIPTION

```
payoff_value(s, payoff_typeValue, payoff_params):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
payoff_typeValue	-	-	-
payoff_params	-	-	-

value_mc_fast

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc_fast(t,
              stock_prices,
              discount_curve,
              dividend_yields,
              volatilities,
              betas,
              num_assets,
              payoff_type,
              payoff_params,
              num_paths=10000,
              seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
stock_prices	-	-	-
discount_curve	-	-	-
dividend_yields	-	-	-
volatilities	-	-	-
betas	-	-	-
num_assets	-	-	-
payoff_type	-	-	-
payoff_params	-	-	-
num_paths	-	-	10000
seed	-	-	4242

9.11 fx_vanilla_option

Class: FXVanillaOption()

This is a class for an FX Option trade. It permits the user to calculate the price of an FX Option trade which can be expressed in a number of ways depending on the investor or hedgers currency. It aslo allows the calculation of the options delta in a number of forms as well as the various Greek risk sensitivies.

FXVanillaOption

Create the FX Vanilla Option object. Inputs include expiry date, strike, currency pair, option type (call or put), notional and the currency of the notional. And adjustment for spot days is enabled. All currency rates must be entered in the price in domestic currency of one unit of foreign. And the currency pair should be in the form FORDOM where FOR is the foreign currency pair currency code and DOM is the same for the domestic currency.

```
FXVanillaOption(expiry_dt: Date,
                # 1 unit of foreign in domestic
                strike_fx_rate: (float, np.ndarray),
                currency_pair: str,  # FORDOM
                option_type: (OptionTypes, list),
                notional: float,
                prem_currency: str,
                spot_days: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiry_dt	Date	-	-
strike_fx_rate	float or np.ndarray	-	-
currency_pair	str	FORDOM	-
option_type	OptionTypes or list	-	-
notional	float	-	-
prem_currency	str	-	-
spot_days	int	-	0

value

This function calculates the value of the option using a specified model with the resulting value being in domestic i.e. ccy2 terms. Recall that Domestic = CCY2 and Foreign = CCY1 and FX rate is in price in domestic of one unit of foreign currency.

```
value(value_dt,
      spot_fx_rate,  # 1 unit of foreign in domestic
      dom_discount_curve,
      for_discount_curve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	1 unit of foreign in domestic	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-
model	-	-	-

delta_bump

Calculation of the FX option delta by bumping the spot FX rate by 1 cent of its value. This gives the FX spot delta. For speed we prefer to use the analytical calculation of the derivative given below.

```
delta_bump(value_dt,
           spot_fx_rate,
           ccy1DiscountCurve,
           ccy2DiscountCurve,
           model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	-	-
ccy1DiscountCurve	-	-	-
ccy2DiscountCurve	-	-	-
model	-	-	-

delta

Calculation of the FX Option delta. There are several definitions of delta and so we are required to return a dictionary of values. The definitions can be found on Page 44 of Foreign Exchange Option Pricing by Iain Clark, published by Wiley Finance.

```
delta(value_dt,
      spot_fx_rate,
      dom_discount_curve,
      for_discount_curve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	-	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-
model	-	-	-

fast_delta

Calculation of the FX Option delta. Used in the determination of the volatility surface. Avoids discount curve interpolation so it should be slightly faster than the full calculation of delta.

```
fast_delta(t,
           s,
           rd,
           rf,
           vol):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
s	-	-	-
rd	-	-	-
rf	-	-	-
vol	-	-	-

gamma

This function calculates the FX Option Gamma using the spot delta.

```
gamma(value_dt,
      spot_fx_rate,  # value of a unit of foreign in domestic currency
      dom_discount_curve,
      for_discount_curve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	value of a unit of foreign in domestic currency	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-
model	-	-	-

vega

This function calculates the FX Option Vega using the spot delta.

```
vega(value_dt,
     spot_fx_rate,  # value of a unit of foreign in domestic currency
     dom_discount_curve,
     for_discount_curve,
     model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	value of a unit of foreign in domestic currency	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-
model	-	-	-

theta

This function calculates the time decay of the FX option.

```
theta(value_dt,
      spot_fx_rate,  # value of a unit of foreign in domestic currency
      dom_discount_curve,
      for_discount_curve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	value of a unit of foreign in domestic currency	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-
model	-	-	-

implied_volatility

This function determines the implied volatility of an FX option given a price and the other option details. It uses a one-dimensional Newton root search algorith to determine the implied volatility.

```
implied_volatility(value_dt,
                   stock_price,
                   discount_curve,
                   dividend_curve,
                   price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	-	-
discount_curve	-	-	-
dividend_curve	-	-	-
price	-	-	-

value_mc

Calculate the value of an FX Option using Monte Carlo methods. This function can be used to validate the risk measures calculated above or used as the starting code for a model exotic FX product that cannot be priced analytically. This function uses Numpy vectorisation for speed of execution.

```
value_mc(value_dt,
         spot_fx_rate,
         dom_discount_curve,
         for_discount_curve,
         model,
         num_paths=10000,
         seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
spot_fx_rate	-	-	-
dom_discount_curve	-	-	-
for_discount_curve	-	-	-
model	-	-	-
num_paths	-	-	10000
seed	-	-	4242

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_print

Simple print function for backward compatibility.

```
_print():
```

The function arguments are described in the following table.

f

PLEASE ADD A FUNCTION DESCRIPTION

```
f(volatility, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	-	-	-
*args	-	-	-

fvega

PLEASE ADD A FUNCTION DESCRIPTION

```
fvega(volatility, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	-	-	-
*args	-	-	-

fast_delta

Calculation of the FX Option delta. Used in the determination of the volatility surface. Avoids discount curve interpolation so it should be slightly faster than the full calculation of delta.

```
fast_delta(s, t, k, rd, rf, vol, deltaTypeValue, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
rd	-	-	-
rf	-	-	-
vol	-	-	-
deltaTypeValue	-	-	-
option_type_value	-	-	-

9.12 fx_variance_swap

Class: FinFXVarianceSwap

FinFXVarianceSwap

Create variance swap contract.

```
FinFXVarianceSwap(effective_dt: Date,
                  maturity_dt_or_tenor: [Date, str],
                  strike_variance: float,
                  notional: float = ONE_MILLION,
                  pay_strike_flag: bool = True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effective_dt	Date	-	-
maturity_dt_or_tenor	[Date,str]	-	-
strike_variance	float	-	-
notional	float	-	ONE_MILLION
pay_strike_flag	bool	-	True

value

Calculate the value of the variance swap based on the realised volatility to the valuation date, the forward looking implied volatility to the maturity date using the libor discount curve.

```
value(value_dt,
      realisedVar,
      fair_strikeVar,
      libor_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
realisedVar	-	-	-
fair_strikeVar	-	-	-
libor_curve	-	-	-

fair_strike_approx

This is an approximation of the fair strike variance by Demeterfi et al. (1999) which assumes that sigma(K) = sigma(F) - b(K-F)/F where F is the forward stock price and sigma(F) is the ATM forward vol.

```
fair_strike_approx(value_dt,
                   fwdStockPrice,
                   strikes,
                   volatilities):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
fwdStockPrice	-	-	-
strikes	-	-	-
volatilities	-	-	-

fair_strike

Calculate the implied variance according to the volatility surface using a static replication methodology with a specially weighted portfolio of put and call options across a range of strikes using the approximate method set out by Demeterfi et al. 1999.

```
fair_strike(value_dt,
            stock_price,
            dividend_curve,
            volatility_curve,
            num_call_options,
            num_put_options,
            strike_spacing,
            discount_curve,
            use_forward=True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
stock_price	-	-	-
dividend_curve	-	-	-
volatility_curve	-	-	-
num_call_options	-	-	-
num_put_options	-	-	-
strike_spacing	-	-	-
discount_curve	-	-	-
use_forward	-	-	True

f

PLEASE ADD A FUNCTION DESCRIPTION

```
f(x): return (2.0/tmat)*((x-sstar)/sstar-np.log(x/sstar))
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x return (2.0/tmat)*((x-sstar)/sstar-np.log(x/sstar))	-	-	-

realised_variance

Calculate the realised variance according to market standard calculations which can either use log or percentage returns.

```
realised_variance(closePrices, useLogs=True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
closePrices	-	-	-
useLogs	-	-	True

print_strikes

PLEASE ADD A FUNCTION DESCRIPTION

```
print_strikes():
```

The function arguments are described in the following table.

9.13 __init__

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 stochastics 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:

- black is Black's model for pricing forward starting contracts (in the forward measure) assuming the forward is lognormally distributed.
- black_shifted is Black's model for pricing forward starting contracts (in the forward measure) assuming the forward plus a shift is lognormally distributed. CHECK
- bachelier prices options assuming the underlying evolves according to a Gaussian (normal) process.
- sabr 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.
- sabr_shifted 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

- heston_model
- process_simulator

Interest Rate Models

Equilibrium Rate Models

There are two main short rate models.

- CIR 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.
- Vasicek 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

- BKTree 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.
- HWTree 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

- GaussianCopula1F is a Gaussian copula one-factor model. This class includes functions that calculate the portfolio loss distribution. This is numerical but deterministic.
- GaussianCopulaLHP is a Gaussian copula one-factor model in the limit that the number of credits tends to infinity. This is an asymptotic analytical solution.
- GaussianCopula is a Gaussian copula model which is multifactor model. It has a Monte-Carlo implementation.
- LossDbnBuilder calculates the loss distribution.
- MertonFirm is a model of the firm as proposed by Merton (1974).

FX Models

10.1 bachelier

Class: Bachelier()

Bacheliers Model which prices call and put options in the forward measure assuming the underlying rate follows a normal process.

Bachelier

Create FinModel black using parameters.

```
Bachelier(volatility):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	-	-	-

value

Price a call or put option using Bachelier's model.

```
value(forward_rate,    # Forward rate F
      strike_rate,     # Strike Rate K
      time_to_expiry,  # Time to Expiry (years)
      df,              # Discount Factor to expiry date
      call_or_put):    # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forward_rate	-	Forward rate F	-
strike_rate	-	Strike Rate K	-
time_to_expiry	-	Time to Expiry (years)	-
df	-	Discount Factor to expiry date	-
call_or_put	-	Call or put	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

10.2 bdt_tree

Class: BDTTree()

class BDTTree():

BDTTree

*Constructs the Black-Derman-Toy rate model in the case when the volatility is assumed to be constant. The short rate process simplifies and is given by d(log(r)) = theta(t) * dt + sigma * dW. Althopugh*

```
BDTTree(sigma: float,
        num_time_steps: int = 100):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	float	-	-
num_time_steps	int	-	100

build_tree

PLEASE ADD A FUNCTION DESCRIPTION

```
build_tree(treeMat, df_times, df_values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
treeMat	-	-	-
df_times	-	-	-
df_values	-	-	-

bond_option

Value a bond option that can have European or American exercise using the Black-Derman-Toy model. The model uses a binomial tree.

```
bond_option(t_exp, strike_price, face_amount,
            cpn_times, cpn_flows, exercise_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_type	-	-	-

bermudan_swaption

Swaption that can be exercised on specific dates over the exercise period. Due to non-analytical bond price we need to extend tree out to bond maturity and take into account cash flows through time.

```
bermudan_swaption(t_exp, tmat, strike, face_amount,
                  cpn_times, cpn_flows, exercise_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_type	-	-	-

callable_puttable_bond_tree

Option that can be exercised at any time over the exercise period. Due to non-analytical bond price we need to extend tree out to bond maturity and take into account cash flows through time.

```
callable_puttable_bond_tree(cpn_times, cpn_flows,
                            call_times, call_prices,
                            put_times, put_prices,
                            face_amount):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cpn_times	-	-	-
cpn_flows	-	-	-
call_times	-	-	-
call_prices	-	-	-
put_times	-	-	-
put_prices	-	-	-
face_amount	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

option_exercise_types_to_int

PLEASE ADD A FUNCTION DESCRIPTION

```
option_exercise_types_to_int(option_exercise_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
option_exercise_type	-	-	-

f

PLEASE ADD A FUNCTION DESCRIPTION

```
f(x0, m, q_matrix, rt, df_end, dt, sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x0	-	-	-
m	-	-	-
q_matrix	-	-	-
rt	-	-	-
df_end	-	-	-
dt	-	-	-
sigma	-	-	-

search_root

PLEASE ADD A FUNCTION DESCRIPTION

```
search_root(x0, m, q_matrix, rt, df_end, dt, sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x0	-	-	-
m	-	-	-
q_matrix	-	-	-
rt	-	-	-
df_end	-	-	-
dt	-	-	-
sigma	-	-	-

bermudan_swaption_tree_fast

Option to enter into a swap that can be exercised on coupon payment dates after the start of the exercise period. Due to non-analytical bond price we need to extend tree out to bond maturity and take into account cash flows through time.

```
bermudan_swaption_tree_fast(t_exp, tmat,
                            strike_price,
                            face_amount,
                            cpn_times,
                            cpn_flows,
                            exercise_type_int,
                            _df_times,
                            _df_values,
                            _tree_times,
                            _Q, _rt, _dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_type_int	-	-	-
_df_times	-	-	-
_df_values	-	-	-
_tree_times	-	-	-
_Q	-	-	-
_rt	-	-	-
_dt	-	-	-

american_bond_option_tree_fast

Option to buy or sell bond at a specified strike price that can be exercised over the exercise period depending on choice of exercise type. Due to non-analytical bond price we need to extend tree out to bond maturity and take into account cash flows through time.

```
american_bond_option_tree_fast(t_exp, tmat,
                               strike_price, face_amount,
                               cpn_times, cpn_flows,
                               exercise_type_int,
                               _df_times, _df_values,
                               _tree_times, _Q,
                               _rt, _dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_type_int	-	-	-
_df_times	-	-	-
_df_values	-	-	-
_tree_times	-	-	-
_Q	-	-	-
_rt	-	-	-
_dt	-	-	-

callable_puttable_bond_tree_fast

Value a bond with embedded put and call options that can be exercised at any time over the specified list of put and call dates. Due to non-analytical bond price we need to extend tree out to bond maturity and take into account cash flows through time.

```
callable_puttable_bond_tree_fast(cpn_times, cpn_flows,
                                 call_times, call_prices,
                                 put_times, put_prices, face_amount,
                                 _sigma, _a, _q_matrix,  # IS SIGMA USED ?
                                 _pu, _pm, _pd, _rt, _dt, _tree_times,
                                 _df_times, _df_values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cpn_times	-	-	-
cpn_flows	-	-	-
call_times	-	-	-
call_prices	-	-	-
put_times	-	-	-
put_prices	-	-	-
face_amount	-	-	-
_sigma	-	IS SIGMA USED ?	-
_a	-	IS SIGMA USED ?	-
_q_matrix	-	IS SIGMA USED ?	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_tree_times	-	-	-
_df_times	-	-	-
_df_values	-	-	-

build_tree_fast

PLEASE ADD A FUNCTION DESCRIPTION

```
build_tree_fast(sigma, tree_times, num_time_steps, discount_factors):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
tree_times	-	-	-
num_time_steps	-	-	-
discount_factors	-	-	-

10.3 bk_tree

Class: BKTree()

class BKTree():

BKTree

*Constructs the Black Karasinski rate model. The speed of mean reversion a and volatility are passed in. The short rate process is given by d(log(r)) = (theta(t) - a*log(r)) * dt + sigma * dW*

```
BKTree(sigma: float,
       a: float,
       num_time_steps: int = 100):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	float	-	-
a	float	-	-
num_time_steps	int	-	100

build_tree

PLEASE ADD A FUNCTION DESCRIPTION

```
build_tree(tmat, df_times, df_values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
tmat	-	-	-
df_times	-	-	-
df_values	-	-	-

bond_option

Value a bond option that has European or American exercise using the Black-Karasinski model. The model uses a trinomial tree.

```
bond_option(t_exp, strike_price, face_amount,
            cpn_times, cpn_flows, exercise_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_type	-	-	-

bermudan_swaption

Swaption that can be exercised on specific dates over the exercise period. Due to non-analytical bond price we need to extend tree out to bond maturity and take into account cash flows through time.

```
bermudan_swaption(t_exp, tmat, strike_price, face_amount,
                  cpn_times, cpn_flows, exercise_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_type	-	-	-

callable_puttable_bond_tree

Option that can be exercised at any time over the exercise period. Due to non-analytical bond price we need to extend tree out to bond maturity and take into account cash flows through time.

```
callable_puttable_bond_tree(cpn_times, cpn_flows,
                            call_times, call_prices,
                            put_times, put_prices,
                            face):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cpn_times	-	-	-
cpn_flows	-	-	-
call_times	-	-	-
call_prices	-	-	-
put_times	-	-	-
put_prices	-	-	-
face	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

option_exercise_types_to_int

PLEASE ADD A FUNCTION DESCRIPTION

```
option_exercise_types_to_int(optionExerciseType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
optionExerciseType	-	-	-

f

PLEASE ADD A FUNCTION DESCRIPTION

```
f(alpha, nm, Q, P, dX, dt, N):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
nm	-	-	-
Q	-	-	-
P	-	-	-
dX	-	-	-
dt	-	-	-
N	-	-	-

fprime

PLEASE ADD A FUNCTION DESCRIPTION

```
fprime(alpha, nm, Q, P, dX, dt, N):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
nm	-	-	-
Q	-	-	-
P	-	-	-
dX	-	-	-
dt	-	-	-
N	-	-	-

search_root

PLEASE ADD A FUNCTION DESCRIPTION

```
search_root(x0, nm, Q, P, dX, dt, N):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x0	-	-	-
nm	-	-	-
Q	-	-	-
P	-	-	-
dX	-	-	-
dt	-	-	-
N	-	-	-

search_root_deriv

PLEASE ADD A FUNCTION DESCRIPTION

```
search_root_deriv(x0, nm, Q, P, dX, dt, N):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x0	-	-	-
nm	-	-	-
Q	-	-	-
P	-	-	-
dX	-	-	-
dt	-	-	-
N	-	-	-

bermudan_swaption_tree_fast

Option to enter into a swap that can be exercised on coupon payment dates after the start of the exercise *period. Due to multiple exercise times we need to extend tree out to bond maturity and take into account cash flows through time.*

```
bermudan_swaption_tree_fast(t_exp, tmat,
                            strike_price, face_amount,
                            cpn_times, cpn_flows,
                            exercise_typeInt,
                            _df_times, _df_values,
                            _tree_times, _Q,
                            _pu, _pm, _pd,
                            _rt, _dt, _a):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_typeInt	-	-	-
_df_times	-	-	-
_df_values	-	-	-
_tree_times	-	-	-
_Q	-	-	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_a	-	-	-

american_bond_option_tree_fast

Option that can be exercised at any time over the exercise period. Due to non-analytical bond price we need to extend tree out to bond maturity and take into account cash flows through time.

```
american_bond_option_tree_fast(t_exp, tmat,
                               strike_price, face_amount,
                               cpn_times, cpn_flows,
                               exercise_typeInt,
                               _df_times, _df_values,
                               _tree_times, _Q,
                               _pu, _pm, _pd,
                               _rt,
                               _dt, _a):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_typeInt	-	-	-
_df_times	-	-	-
_df_values	-	-	-
_tree_times	-	-	-
_Q	-	-	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_a	-	-	-

callable_puttable_bond_tree_fast

Value a bond with embedded put and call options that can be exercised at any time over the specified list of put and call dates. Due to non-analytical bond price we need to extend tree out to bond maturity and take into account cash flows through time.

```
callable_puttable_bond_tree_fast(cpn_times, cpn_flows,
                                 call_times, call_prices,
                                 put_times, put_prices, face_amount,
                                 _sigma, _a, _Q,  # IS SIGMA USED ?
                                 _pu, _pm, _pd, _rt, _dt, _tree_times,
                                 _df_times, _df_values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cpn_times	-	-	-
cpn_flows	-	-	-
call_times	-	-	-
call_prices	-	-	-
put_times	-	-	-
put_prices	-	-	-
face_amount	-	-	-
_sigma	-	IS SIGMA USED ?	-
_a	-	IS SIGMA USED ?	-
_Q	-	IS SIGMA USED ?	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_tree_times	-	-	-
_df_times	-	-	-
_df_values	-	-	-

build_tree_fast

Calibrate the tree to a term structure of interest rates.

```
build_tree_fast(a, sigma, tree_times, num_time_steps, discount_factors):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
sigma	-	-	-
tree_times	-	-	-
num_time_steps	-	-	-
discount_factors	-	-	-

10.4 black

Enumerated Type: BlackTypes

This enumerated type has the following values:

- ANALYTICAL
- CRR_TREE

Class: Black()

Blacks Model which prices call and put options in the forward measure according to the Black-Scholes equation.

Black

Create FinModel black using parameters.

```
Black(volatility, implementation_type=BlackTypes.ANALYTICAL,
      num_steps=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	-	-	-
implementation_type	-	-	BlackTypes.ANALYTICAL
num_steps	-	-	0

value

Price a derivative using Black's model which values in the forward measure following a change of measure.

```
value(forward_rate,    # Forward rate F
      strike_rate,     # Strike Rate K
      time_to_expiry,  # Time to Expiry (years)
      df,  # df RFR to expiry date
      option_type):    # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forward_rate	-	Forward rate F	-
strike_rate	-	Strike Rate K	-
time_to_expiry	-	Time to Expiry (years)	-
df	-	df RFR to expiry date	-
option_type	-	Call or put	-

delta

Calculate delta using Black's model which values in the forward measure following a change of measure.

```
delta(forward_rate,   # Forward rate F
      strike_rate,    # Strike Rate K
      time_to_expiry,  # Time to Expiry (years)
      df,  # RFR to expiry date
      option_type):    # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forward_rate	-	Forward rate F	-
strike_rate	-	Strike Rate K	-
time_to_expiry	-	Time to Expiry (years)	-
df	-	RFR to expiry date	-
option_type	-	Call or put	-

gamma

Calculate gamma using Black's model which values in the forward measure following a change of measure.

```
gamma(forward_rate,   # Forward rate F
      strike_rate,    # Strike Rate K
      time_to_expiry,  # Time to Expiry (years)
      df,  # RFR to expiry date
      option_type):    # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forward_rate	-	Forward rate F	-
strike_rate	-	Strike Rate K	-
time_to_expiry	-	Time to Expiry (years)	-
df	-	RFR to expiry date	-
option_type	-	Call or put	-

theta

Calculate theta using Black's model which values in the forward measure following a change of measure.

```
theta(forward_rate,   # Forward rate F
      strike_rate,    # Strike Rate K
      time_to_expiry,  # Time to Expiry (years)
      df,  # Discount Factor to expiry date
      option_type):    # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forward_rate	-	Forward rate F	-
strike_rate	-	Strike Rate K	-
time_to_expiry	-	Time to Expiry (years)	-
df	-	Discount Factor to expiry date	-
option_type	-	Call or put	-

vega

Calculate vega using Black's model which values in the forward measure following a change of measure.

```
vega(forward_rate,    # Forward rate F
     strike_rate,     # Strike Rate K
     time_to_expiry,  # Time to Expiry (years)
     df,  # df RFR to expiry date
     option_type):    # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forward_rate	-	Forward rate F	-
strike_rate	-	Strike Rate K	-
time_to_expiry	-	Time to Expiry (years)	-
df	-	df RFR to expiry date	-
option_type	-	Call or put	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_f_european

Function to determine ststar for pricing European options on future contracts.

```
_f_european(sigma, args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
args	-	-	-

_f_european_vega

Function to calculate the Vega of European options on future contracts.

```
_f_european_vega(sigma, args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
args	-	-	-

_f_american

Function to determine ststar for pricing American options on future contracts.

```
_f_american(sigma, args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
args	-	-	-

_f_american_vega

Function to calculate the Vega of American options on future contracts.

```
_f_american_vega(sigma, args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
args	-	-	-

_estimate_vol_from_price

PLEASE ADD A FUNCTION DESCRIPTION

```
_estimate_vol_from_price(fwd, t, k, european_option_type, european_price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fwd	-	-	-
t	-	-	-
k	-	-	-
european_option_type	-	-	-
european_price	-	-	-

black_value

Price a derivative using Black model.

```
black_value(fwd, t, k, r, v, option_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fwd	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
v	-	-	-
option_type	-	-	-

black_delta

Return delta of a derivative using Black model.

```
black_delta(fwd, t, k, r, v, option_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fwd	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
v	-	-	-
option_type	-	-	-

black_gamma

Return gamma of a derivative using Black model.

```
black_gamma(fwd, t, k, r, v, option_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fwd	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
v	-	-	-
option_type	-	-	-

black_vega

Return vega of a derivative using Black model.

```
black_vega(fwd, t, k, r, v, option_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fwd	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
v	-	-	-
option_type	-	-	-

black_theta

Return theta of a derivative using Black model.

```
black_theta(fwd, t, k, r, v, option_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fwd	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
v	-	-	-
option_type	-	-	-

calculate_d1_d2

PLEASE ADD A FUNCTION DESCRIPTION

```
calculate_d1_d2(f, t, k, v):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
f	-	-	-
t	-	-	-
k	-	-	-
v	-	-	-

implied_volatility

Calculate the Black implied volatility of a European/American options on futures contracts using Newton with a fallback to bisection.

```
implied_volatility(fwd, t, r, k, price, option_type, debug_print=True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fwd	-	-	-
t	-	-	-
r	-	-	-
k	-	-	-
price	-	-	-
option_type	-	-	-
debug_print	-	-	True

10.5 black_scholes

Enumerated Type: BlackScholesTypes

This enumerated type has the following values:

- DEFAULT
- ANALYTICAL
- CRR_TREE
- BARONE_ADESI
- LSMC
- Bjerksund_Stensland
- FINITE_DIFFERENCE
- PSOR

Class: BlackScholes(Model)

class BlackScholes(Model):

BlackScholes

PLEASE ADD A FUNCTION DESCRIPTION

```
BlackScholes(volatility: (float, np.ndarray),
             bs_type: BlackScholesTypes = BlackScholesTypes.DEFAULT,
             num_steps_per_year=52,
             num_paths=10000,
             seed=42,
             use_sobol=False,
             params=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	float or np.ndarray	-	-
bs_type	BlackScholesTypes	-	BlackScholesTypes.DEFAULT
num_steps_per_year	-	-	52
num_paths	-	-	10000
seed	-	-	42
use_sobol	-	-	False
params	-	-	None

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(spot_price: float,
      time_to_expiry: float,
      strike_price: float,
      risk_free_rate: float,
```

```
    dividend_rate: float,
    option_type: OptionTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spot_price	float	-	-
time_to_expiry	float	-	-
strike_price	float	-	-
risk_free_rate	float	-	-
dividend_rate	float	-	-
option_type	OptionTypes	-	-

10.6 black_scholes_analytic

bs_value

Price a derivative using Black-Scholes model.

```
bs_value(s, t, k, r, q, v, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
option_type_value	-	-	-

bs_delta

Price a derivative using Black-Scholes model.

```
bs_delta(s, t, k, r, q, v, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
option_type_value	-	-	-

bs_gamma

Price a derivative using Black-Scholes model.

```
bs_gamma(s, t, k, r, q, v, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
option_type_value	-	-	-

bs_vega

Price a derivative using Black-Scholes model.

```
bs_vega(s, t, k, r, q, v, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
option_type_value	-	-	-

bs_theta

Price a derivative using Black-Scholes model.

```
bs_theta(s, t, k, r, q, v, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
option_type_value	-	-	-

bs_rho

Price a derivative using Black-Scholes model.

```
bs_rho(s, t, k, r, q, v, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
option_type_value	-	-	-

bs_vanna

Price a derivative using Black-Scholes model.

```
bs_vanna(s, t, k, r, q, v, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
option_type_value	-	-	-

_f

PLEASE ADD A FUNCTION DESCRIPTION

```
_f(sigma, args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
args	-	-	-

_fvega

PLEASE ADD A FUNCTION DESCRIPTION

```
_fvega(sigma, args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
args	-	-	-

bs_intrinsic

Calculate the Black-Scholes implied volatility of a European vanilla option using Newton with a fallback to bisection.

```
bs_intrinsic(s, t, k, r, q, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
option_type_value	-	-	-

bs_implied_volatility

Calculate the Black-Scholes implied volatility of a European vanilla option using Newton with a fallback to bisection.

```
bs_implied_volatility(s, t, k, r, q, price, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
price	-	-	-
option_type_value	-	-	-

_fcall

Function to determine ststar for pricing American call options.

```
_fcall(si, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
si	-	-	-
*args	-	-	-

_fput

Function to determine sstar for pricing American put options.

```
_fput(si, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
si	-	-	-
*args	-	-	-

baw_value

American Option Pricing Approximation using the Barone-Adesi-Whaley approximation for the Black Scholes Model

```
baw_value(s, t, k, r, q, v, phi):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
phi	-	-	-

bjerksund_stensland_value

Price American Option using the Bjerksund-Stensland approximation (1993) for the Black Scholes Model

```
bjerksund_stensland_value(s, t, k, r, q, v, option_type_value):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
k	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
option_type_value	-	-	-

10.7 black_scholes_mc

_value_mc_nonumba_nonumpy

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_mc_nonumba_nonumpy(s, t, K, option_type, r, q, v,
                          num_paths, seed, use_sobol):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
K	-	-	-
option_type	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
num_paths	-	-	-
seed	-	-	-
use_sobol	-	-	-

_value_mc_numpy_only

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_mc_numpy_only(s, t, K, option_type, r, q, v, num_paths,
                     seed, use_sobol):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
K	-	-	-
option_type	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
num_paths	-	-	-
seed	-	-	-
use_sobol	-	-	-

_value_mc_numpy_numba

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_mc_numpy_numba(s, t, K, option_type, r, q, v, num_paths, seed,
                      use_sobol):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
K	-	-	-
option_type	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
num_paths	-	-	-
seed	-	-	-
use_sobol	-	-	-

_value_mc_numba_only

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_mc_numba_only(s, t, K, option_type, r, q, v, num_paths, seed,
                     use_sobol):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
K	-	-	-
option_type	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
num_paths	-	-	-
seed	-	-	-
use_sobol	-	-	-

_value_mc_numba_parallel

PLEASE ADD A FUNCTION DESCRIPTION

```
_value_mc_numba_parallel(s, t, K, option_type, r, q, v, num_paths, seed,
                         use_sobol):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
t	-	-	-
K	-	-	-
option_type	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
num_paths	-	-	-
seed	-	-	-
use_sobol	-	-	-

10.8 black_scholes_mc_tests

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	-	-	-

10.9 black_shifted

Class: BlackShifted()

Blacks Model which prices call and put options in the forward measure according to the Black-Scholes equation. This model also allows the distribution to be shifted to the negative in order to allow for negative interest rates.

BlackShifted

Create FinModel black using parameters.

```
BlackShifted(volatility, shift, implementation=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	-	-	-
shift	-	-	-
implementation	-	-	0

value

Price a derivative using Black's model which values in the forward measure following a change of measure. The sign of the shift is the same as Matlab.

```
value(forward_rate,    # Forward rate
      strike_rate,     # Strike Rate
      time_to_expiry,   # time to expiry in years
      df,             # Discount Factor to expiry date
      call_or_put):     # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forward_rate	-	Forward rate	-
strike_rate	-	Strike Rate	-
time_to_expiry	-	time to expiry in years	-
df	-	Discount Factor to expiry date	-
call_or_put	-	Call or put	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

10.10 bond_analytics

10.11 cir_mc

Enumerated Type: CIRNumericalScheme

This enumerated type has the following values:

- EULER
- LOGNORMAL
- MILSTEIN
- KAHLJACKEL
- EXACT

Class: CIR_MC()

class CIR_MC():

CIR_MC

PLEASE ADD A FUNCTION DESCRIPTION

```
CIR_MC(a, b, sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
b	-	-	-
sigma	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

meanr

Mean value of a CIR process after time t

```
meanr(r0, a, b, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
t	-	-	-

variancer

Variance of a CIR process after time t

```
variancer(r0, a, b, sigma, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
sigma	-	-	-
t	-	-	-

zero_price

Price of a zero coupon bond in CIR model.

```
zero_price(r0, a, b, sigma, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
sigma	-	-	-
t	-	-	-

draw

Draw a next rate from the CIR model in Monte Carlo.

```
draw(rt, a, b, sigma, dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rt	-	-	-
a	-	-	-
b	-	-	-
sigma	-	-	-
dt	-	-	-

rate_path_mc

Generate a path of CIR rates using a number of numerical schemes.

```
rate_path_mc(r0, a, b, sigma, t, dt, seed, scheme):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
sigma	-	-	-
t	-	-	-
dt	-	-	-
seed	-	-	-
scheme	-	-	-

zero_price_mc

Determine the CIR zero price using Monte Carlo.

```
zero_price_mc(r0, a, b, sigma, t, dt, num_paths, seed, scheme):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
sigma	-	-	-
t	-	-	-
dt	-	-	-
num_paths	-	-	-
seed	-	-	-
scheme	-	-	-

10.12 equity_barrier_models

value_bs

This values a single option. Because of its structure it cannot easily be vectorised which is why it has been wrapped. # number of observations per year

```
value_bs(t, k, h, s, r, q, v, option_type, nobs):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
k	-	-	-
h	-	-	-
s	-	-	-
r	-	-	-
q	-	-	-
v	-	-	-
option_type	-	-	-
nobs	-	-	-

10.13 equity_crr_tree

crr_tree_val

Value an American option using a Binomial Treee

```
crr_tree_val(stock_price,
             interest_rate,  # continuously compounded
             dividend_rate,  # continuously compounded
             volatility,  # Black scholes volatility
             num_steps_per_year,
             time_to_expiry,
             option_type,
             strike_price,
             isEven):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stock_price	-	-	-
interest_rate	-	continuously compounded	-
dividend_rate	-	continuously compounded	-
volatility	-	Black scholes volatility	-
num_steps_per_year	-	-	-
time_to_expiry	-	-	-
option_type	-	-	-
strike_price	-	-	-
isEven	-	-	-

crr_tree_val_avg

Calculate the average values off the tree using an even and an odd number of time steps.

```
crr_tree_val_avg(stock_price,
                 interest_rate,  # continuously compounded
                 dividend_rate,  # continuously compounded
                 volatility,  # Black scholes volatility
                 num_steps_per_year,
                 time_to_expiry,
                 option_type,
                 strike_price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stock_price	-	-	-
interest_rate	-	continuously compounded	-
dividend_rate	-	continuously compounded	-
volatility	-	Black scholes volatility	-
num_steps_per_year	-	-	-
time_to_expiry	-	-	-
option_type	-	-	-
strike_price	-	-	-

10.14 equity_lsmc

Enumerated Type: FIT_TYPES

This enumerated type has the following values:

- HERMITE_E
- LAGUERRE
- HERMITE
- LEGENDRE
- CHEBYCHEV
- POLYNOMIAL

equity_lsmc

PLEASE ADD A FUNCTION DESCRIPTION

```
equity_lsmc(spot_price,
            risk_free_rate,
            dividend_yield,
            sigma,
            num_paths,
            num_steps_per_year,
            time_to_expiry,
            option_type_value,
            strike_price,
            poly_degree,
            fit_type_value,
            use_sobol,
            seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spot_price	-	-	-
risk_free_rate	-	-	-
dividend_yield	-	-	-
sigma	-	-	-
num_paths	-	-	-
num_steps_per_year	-	-	-
time_to_expiry	-	-	-
option_type_value	-	-	-
strike_price	-	-	-
poly_degree	-	-	-
fit_type_value	-	-	-
use_sobol	-	-	-
seed	-	-	-

10.15 finite_difference

dx

PLEASE ADD A FUNCTION DESCRIPTION

```
dx(x, wind=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
wind	-	-	0

dxx

PLEASE ADD A FUNCTION DESCRIPTION

```
dxx(x):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-

calculate_fd_matrix

1d finite difference solution for pdes of the form 0 = dV/dt + A V A = -risk_free_rate + mu d/dx + 1/2 var d2/dx2 using the theta scheme [1-theta dt A] V(t) = [1 + (1-theta) dt A] V(t+dt)

```
calculate_fd_matrix(x, r, mu, var, dt, theta, wind=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
r	-	-	-
mu	-	-	-
var	-	-	-
dt	-	-	-
theta	-	-	-
wind	-	-	0

fd_roll_backwards

PLEASE ADD A FUNCTION DESCRIPTION

```
fd_roll_backwards(res, theta, Ai=None, Ae=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
res	-	-	-
theta	-	-	-
Ai	-	-	None
Ae	-	-	None

fd_roll_forwards

PLEASE ADD A FUNCTION DESCRIPTION

```
fd_roll_forwards(res, theta, Ai=None, Ae=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
res	-	-	-
theta	-	-	-
Ai	-	-	None
Ae	-	-	None

smooth_digital

PLEASE ADD A FUNCTION DESCRIPTION

```
smooth_digital(xl, xu, strike):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
xl	-	-	-
xu	-	-	-
strike	-	-	-

digital

PLEASE ADD A FUNCTION DESCRIPTION

```
digital(x, strike):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
strike	-	-	-

smooth_call

PLEASE ADD A FUNCTION DESCRIPTION

```
smooth_call(xl, xu, strike):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
xl	-	-	-
xu	-	-	-
strike	-	-	-

option_payoff

PLEASE ADD A FUNCTION DESCRIPTION

```
option_payoff(s, strike, smooth, dig, option_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
strike	-	-	-
smooth	-	-	-
dig	-	-	-
option_type	-	-	-

black_scholes_fd

PLEASE ADD A FUNCTION DESCRIPTION

```
black_scholes_fd(spot_price, volatility, time_to_expiry,
                 strike_price, risk_free_rate,
                 dividend_yield, option_type,
                 num_time_steps=None, num_samples=2000,
                 num_std=5, theta=0.5, wind=0,
                 digital=False,
                 smooth=False, update=False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spot_price	-	-	-
volatility	-	-	-
time_to_expiry	-	-	-
strike_price	-	-	-
risk_free_rate	-	-	-
dividend_yield	-	-	-
option_type	-	-	-
num_time_steps	-	-	None
num_samples	-	-	2000
num_std	-	-	5
theta	-	-	0.5
wind	-	-	0
digital	-	-	False
smooth	-	-	False
update	-	-	False

10.16 finite_difference_PSOR

black_scholes_fd_PSOR

Solve Black-Scholes equation using projected successive over-relaxtion. Parameters: acc: Keep iterating until this accuracy is achieved d_omega: Larger numbers lead to bigger changes in omega with each iteration max_iter: Maximum number of iterations in PSOR step. Set to 0 to allow any number of iterations.

```
black_scholes_fd_PSOR(spot_price, volatility, time_to_expiry,
                      strike_price, risk_free_rate, dividend_yield,
                      option_type, num_time_steps=None, num_samples=2000,
                      num_std=5, theta=0.5, wind=0, digital=False,
                      smooth=False, acc=1e-13, d_omega=5e-5, max_iter=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spot_price	-	-	-
volatility	-	-	-
time_to_expiry	-	-	-
strike_price	-	-	-
risk_free_rate	-	-	-
dividend_yield	-	-	-
option_type	-	-	-
num_time_steps	-	-	None
num_samples	-	-	2000
num_std	-	-	5
theta	-	-	0.5
wind	-	-	0
digital	-	-	False
smooth	-	-	False
acc	-	-	1e-13
d_omega	-	-	5e-5
max_iter	-	-	0

PSOR_roll_backwards

PLEASE ADD A FUNCTION DESCRIPTION

```
PSOR_roll_backwards(Ae, Ai, res_k, omega, acc=1e-13, max_iter=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
Ae	-	-	-
Ai	-	-	-
res_k	-	-	-
omega	-	-	-
acc	-	-	1e-13
max_iter	-	-	0

PSOR

Projected Successive Over Relaxation - Parameters: Ai (np.array): Implicit finite difference matrix omega (float): Number between 1 and 2 weighted average of previous and current iteration initial_value (np.array): Vector produced by previous iteration z_ip1 (matrix): Matrix for updating to next timestep max_iter (int): Maximum number of iterations before raising an error # (max_iter=0 means no maximum) acc (float): Accuracy. The maximum acceptable square difference # between two iterations Returns: res_k (np.array): Vector for next time step nloops: Number of iterations required to achieve required accuracy

```
PSOR(Ai, omega, initial_value, z_ip1, max_iter=0, acc=1e-13):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
Ai	-	-	-
omega	-	-	-
initial_value	-	-	-
z_ip1	-	-	-
max_iter	-	-	0
acc	-	-	1e-13

10.17 gauss_copula

default_times_gc

Generate a matrix of default times by credit and trial using a Gaussian copula model using a full rank correlation matrix.

```
default_times_gc(issuer_curves,
                 correlation_matrix,
                 num_trials,
                 seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issuer_curves	-	-	-
correlation_matrix	-	-	-
num_trials	-	-	-
seed	-	-	-

10.18 gauss_copula_lhp

tr_surv_prob_lhp

Get the approximated tranche survival probability of a portfolio of credits in the one-factor GC model using the large portfolio limit which assumes a homogenous portfolio with an infinite number of credits. This approach is very fast but not so as accurate as the adjusted binomial.

```
tr_surv_prob_lhp(k1,
                 k2,
                 num_credits,
                 survival_probs,
                 recovery_rates,
                 beta):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-
num_credits	-	-	-
survival_probs	-	-	-
recovery_rates	-	-	-
beta	-	-	-

portfolio_cdf_lhp

PLEASE ADD A FUNCTION DESCRIPTION

```
portfolio_cdf_lhp(k, num_credits, qvector, recovery_rates,
                  beta, num_points):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
num_credits	-	-	-
qvector	-	-	-
recovery_rates	-	-	-
beta	-	-	-
num_points	-	-	-

exp_min_lk

PLEASE ADD A FUNCTION DESCRIPTION

```
exp_min_lk(k, p, r, n, beta):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
p	-	-	-
r	-	-	-
n	-	-	-
beta	-	-	-

lhp_density

PLEASE ADD A FUNCTION DESCRIPTION

```
lhp_density(k, p, r, beta):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
p	-	-	-
r	-	-	-
beta	-	-	-

lhp_analytical_density_base_corr

PLEASE ADD A FUNCTION DESCRIPTION

```
lhp_analytical_density_base_corr(k, p, r, beta, dbeta_dk):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
p	-	-	-
r	-	-	-
beta	-	-	-
dbeta_dk	-	-	-

lhp_analytical_density

PLEASE ADD A FUNCTION DESCRIPTION

```
lhp_analytical_density(k, p, r, beta):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
p	-	-	-
r	-	-	-
beta	-	-	-

exp_min_lk

PLEASE ADD A FUNCTION DESCRIPTION

```
exp_min_lk(k, p, r, n, beta):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
p	-	-	-
r	-	-	-
n	-	-	-
beta	-	-	-

prob_l_greater_than_k

PLEASE ADD A FUNCTION DESCRIPTION

```
prob_l_greater_than_k(K, P, R, beta):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-
P	-	-	-
R	-	-	-
beta	-	-	-

10.19 gauss_copula_lhplus

Class: LHPlusModel()

Large Homogenous Portfolio model with extra asset. Used for approximating full Gaussian copula.

LHPlusModel

PLEASE ADD A FUNCTION DESCRIPTION

```
LHPlusModel(P, R, H, beta, P0, R0, H0, beta0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
P	-	-	-
R	-	-	-
H	-	-	-
beta	-	-	-
P0	-	-	-
R0	-	-	-
H0	-	-	-
beta0	-	-	-

prob_loss_gt_k

Returns P(L¿K) where L is the portfolio loss given by model.

```
prob_loss_gt_k(K):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-

exp_min_lk_integral

PLEASE ADD A FUNCTION DESCRIPTION

```
exp_min_lk_integral(K, dK):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-
dK	-	-	-

exp_min_lk

PLEASE ADD A FUNCTION DESCRIPTION

```
exp_min_lk(K):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-

exp_min_lk2

PLEASE ADD A FUNCTION DESCRIPTION

```
exp_min_lk2(K):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-

tranche_survival_prob

PLEASE ADD A FUNCTION DESCRIPTION

```
tranche_survival_prob(k1, k2):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-

10.20 gauss_copula_onefactor

loss_dbn_recursion_gcd

Full construction of the loss distribution of a portfolio of credits where losses have been calculate as number of units based on the GCD.

```
loss_dbn_recursion_gcd(num_credits,
                       default_probs,
                       loss_units,
                       beta_vector,
                       num_integration_steps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_credits	-	-	-
default_probs	-	-	-
loss_units	-	-	-
beta_vector	-	-	-
num_integration_steps	-	-	-

homog_basket_loss_dbn

Calculate the loss distribution of a CDS default basket where the portfolio is equally weighted and the losses in the portfolio are homo- geneous i.e. the credits have the same recovery rates.

```
homog_basket_loss_dbn(survival_probs,
                      recovery_rates,
                      beta_vector,
                      num_integration_steps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
survival_probs	-	-	-
recovery_rates	-	-	-
beta_vector	-	-	-
num_integration_steps	-	-	-

tranche_surv_prob_recursion

Get the tranche survival probability of a portfolio of credits in the one-factor GC model using a full recursion calculation of the loss distribution and survival probabilities to some time horizon.

```
tranche_surv_prob_recursion(k1,
                            k2,
                            num_credits,
                            survival_probs,
```

```
                    recovery_rates,
                    beta_vector,
                    num_integration_steps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-
num_credits	-	-	-
survival_probs	-	-	-
recovery_rates	-	-	-
beta_vector	-	-	-
num_integration_steps	-	-	-

gauss_approx_tranche_loss

PLEASE ADD A FUNCTION DESCRIPTION

```
gauss_approx_tranche_loss(k1, k2, mu, sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-
mu	-	-	-
sigma	-	-	-

tranch_surv_prob_gaussian

Get the approximated tranche survival probability of a portfolio of credits in the one-factor GC model using a Gaussian fit of the conditional loss distribution and survival probabilities to some time horizon. Note that the losses in this fit are allowed to be negative.

```
tranch_surv_prob_gaussian(k1,
                          k2,
                          num_credits,
                          survival_probs,
                          recovery_rates,
                          beta_vector,
                          num_integration_steps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-
num_credits	-	-	-
survival_probs	-	-	-
recovery_rates	-	-	-
beta_vector	-	-	-
num_integration_steps	-	-	-

loss_dbn_hetero_adj_binomial

Get the portfolio loss distribution using the adjusted binomial approximation to the conditional loss distribution.

```
loss_dbn_hetero_adj_binomial(num_credits,
                             default_probs,
                             loss_ratio,
                             beta_vector,
                             num_integration_steps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_credits	-	-	-
default_probs	-	-	-
loss_ratio	-	-	-
beta_vector	-	-	-
num_integration_steps	-	-	-

tranche_surv_prob_adj_binomial

Get the approximated tranche survival probability of a portfolio of credits in the one-factor GC model using the adjusted binomial fit of the conditional loss distribution and survival probabilities to some time horizon. This approach is both fast and highly accurate.

```
tranche_surv_prob_adj_binomial(k1,
                               k2,
                               num_credits,
                               survival_probs,
                               recovery_rates,
                               beta_vector,
                               num_integration_steps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-
num_credits	-	-	-
survival_probs	-	-	-
recovery_rates	-	-	-
beta_vector	-	-	-
num_integration_steps	-	-	-

10.21 gbm_process_simulator

Class: FinGBMProcess()

class FinGBMProcess():

get_paths

Get a matrix of simulated GBM asset values by path and time step. Inputs are the number of paths and time steps, the time horizon and the initial asset value, volatility and random number seed.

```
get_paths(num_paths: int,
          num_time_steps: int,
          t: float,
          mu: float,
          stock_price: float,
          volatility: float,
          seed: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_paths	int	-	-
num_time_steps	int	-	-
t	float	-	-
mu	float	-	-
stock_price	float	-	-
volatility	float	-	-
seed	int	-	-

get_paths_assets

Get a matrix of simulated GBM asset values by asset, path and time step. Inputs are the number of assets, paths and time steps, the time- horizon and the initial asset values, volatilities and betas.

```
get_paths_assets(num_assets,
                 num_paths,
                 num_time_steps,
                 t,
                 mus,
                 stock_prices,
                 volatilities,
                 corr_matrix,
                 seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_assets	-	-	-
num_paths	-	-	-
num_time_steps	-	-	-
t	-	-	-
mus	-	-	-
stock_prices	-	-	-
volatilities	-	-	-
corr_matrix	-	-	-
seed	-	-	-

get_paths

Get the simulated GBM process for a single asset with many paths and time steps. Inputs include the number of time steps, paths, the drift mu, stock price, volatility and a seed.

```
get_paths(num_paths,
          num_time_steps,
          t,
          mu,
          stock_price,
          volatility,
          seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_paths	-	-	-
num_time_steps	-	-	-
t	-	-	-
mu	-	-	-
stock_price	-	-	-
volatility	-	-	-
seed	-	-	-

get_paths_assets

Get the simulated GBM process for a number of assets and paths and num time steps. Inputs include the number of assets, paths, the vector of mus, stock prices, volatilities, a correlation matrix and a seed.

```
get_paths_assets(num_assets,
                 num_paths,
                 num_time_steps,
                 t,
                 mus,
                 stock_prices,
                 volatilities,
                 corr_matrix,
                 seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_assets	-	-	-
num_paths	-	-	-
num_time_steps	-	-	-
t	-	-	-
mus	-	-	-
stock_prices	-	-	-
volatilities	-	-	-
corr_matrix	-	-	-
seed	-	-	-

get_assets

Get the simulated GBM process for a number of assets and paths for one time step. Inputs include the number of assets, paths, the vector of mus, stock prices, volatilities, a correlation matrix and a seed.

```
get_assets(num_assets,
           num_paths,
           t,
           mus,
           stock_prices,
           volatilities,
           corr_matrix,
           seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_assets	-	-	-
num_paths	-	-	-
t	-	-	-
mus	-	-	-
stock_prices	-	-	-
volatilities	-	-	-
corr_matrix	-	-	-
seed	-	-	-

10.22 heston

Enumerated Type: HestonNumericalScheme

This enumerated type has the following values:

- EULER
- EULERLOG
- QUADEXP

Class: Heston()

class Heston():

Heston

PLEASE ADD A FUNCTION DESCRIPTION

```
Heston(v0, kappa, theta, sigma, rho):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
v0	-	-	-
kappa	-	-	-
theta	-	-	-
sigma	-	-	-
rho	-	-	-

value_mc

PLEASE ADD A FUNCTION DESCRIPTION

```
value_mc(value_dt,
         option,
         stock_price,
         interest_rate,
         dividend_yield,
         num_paths,
         num_steps_per_year,
         seed,
         scheme=HestonNumericalScheme.EULERLOG):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
option	-	-	-
stock_price	-	-	-
interest_rate	-	-	-
dividend_yield	-	-	-
num_paths	-	-	-
num_steps_per_year	-	-	-
seed	-	-	-
scheme	-	-	HestonNumericalScheme.EULERLOG

value_lewis

PLEASE ADD A FUNCTION DESCRIPTION

```
value_lewis(value_dt,
            option,
            stock_price,
            interest_rate,
            dividend_yield):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
option	-	-	-
stock_price	-	-	-
interest_rate	-	-	-
dividend_yield	-	-	-

phi

PLEASE ADD A FUNCTION DESCRIPTION

```
phi(k_in,):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k_in	-	-	-

phi_transform

PLEASE ADD A FUNCTION DESCRIPTION

```
phi_transform(x):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-

integrand

PLEASE ADD A FUNCTION DESCRIPTION

```
integrand(k): return 2.0 * np.real(np.exp(-1j * k * x)
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value

value_lewis_rouah

PLEASE ADD A FUNCTION DESCRIPTION

```
value_lewis_rouah(value_dt,
                  option,
                  stock_price,
                  interest_rate,
                  dividend_yield):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
option	-	-	-
stock_price	-	-	-
interest_rate	-	-	-
dividend_yield	-	-	-

f

PLEASE ADD A FUNCTION DESCRIPTION

```
f(k_in):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k_in	-	-	-

value_weber

PLEASE ADD A FUNCTION DESCRIPTION

```
value_weber(value_dt,
            option,
            stock_price,
            interest_rate,
            dividend_yield):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
option	-	-	-
stock_price	-	-	-
interest_rate	-	-	-
dividend_yield	-	-	-

f

PLEASE ADD A FUNCTION DESCRIPTION

```
f(s, b):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
b	-	-	-

integrand

PLEASE ADD A FUNCTION DESCRIPTION

```
integrand(u):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
u	-	-	-

value_gatheral

PLEASE ADD A FUNCTION DESCRIPTION

```
value_gatheral(value_dt,
               option,
```

```
                  stock_price,
                  interest_rate,
                  dividend_yield):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
value_dt	-	-	-
option	-	-	-
stock_price	-	-	-
interest_rate	-	-	-
dividend_yield	-	-	-

ff

PLEASE ADD A FUNCTION DESCRIPTION

```
ff(j):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
j	-	-	-

integrand

PLEASE ADD A FUNCTION DESCRIPTION

```
integrand(u):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
u	-	-	-

get_paths

PLEASE ADD A FUNCTION DESCRIPTION

```
get_paths(s0, r, q, v0, kappa, theta, sigma, rho, t, dt, num_paths,
            seed, scheme):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s0	-	-	-
r	-	-	-
q	-	-	-
v0	-	-	-
kappa	-	-	-
theta	-	-	-
sigma	-	-	-
rho	-	-	-
t	-	-	-
dt	-	-	-
num_paths	-	-	-
seed	-	-	-
scheme	-	-	-

10.23 hw_tree

Enumerated Type: FinHWEuropeanCalcType

This enumerated type has the following values:

- JAMSHIDIAN
- EXPIRY_ONLY
- EXPIRY_TREE

Class: HWTree()

class HWTree():

HWTree

*Constructs the Hull-White rate model. The speed of mean reversion a and volatility are passed in. The short rate process is given by dr = (theta(t) - ar) * dt + sigma * dW. The model will switch to use Jamshidian's approach where possible unless the useJamshidian flag is set to false in which case it uses the trinomial Tree.*

```
HWTree(sigma,
       a,
       num_time_steps=100,
       europeanCalcType=FinHWEuropeanCalcType.EXPIRY_TREE):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
a	-	-	-
num_time_steps	-	-	100
europeanCalcType	-	-	EXPIRY_TREE

option_on_zcb

Price an option on a zero coupon bond using analytical solution of Hull-White model. User provides bond face and option strike and expiry date and maturity date.

```
option_on_zcb(t_exp, tmat,
              strike, face_amount,
              df_times, df_values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike	-	-	-
face_amount	-	-	-
df_times	-	-	-
df_values	-	-	-

european_bond_option_jamshidian

Valuation of a European bond option using the Jamshidian deconstruction of the bond into a strip of zero coupon bonds with the short rate that would make the bond option be at the money forward.

```
european_bond_option_jamshidian(t_exp,
                                strike_price,
                                face,
                                cpn_times,
                                cpn_amounts,
                                df_times,
                                df_values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
strike_price	-	-	-
face	-	-	-
cpn_times	-	-	-
cpn_amounts	-	-	-
df_times	-	-	-
df_values	-	-	-

european_bond_option_expiry_only

Price a European option on a coupon-paying bond using a tree to generate short rates at the expiry date and then to use the analytical solution of zero coupon bond prices in the HW model to calculate the corresponding bond price. User provides bond object and option details.

```
european_bond_option_expiry_only(t_exp,
                                 strike_price,
                                 face_amount,
                                 cpn_times,
                                 cpn_amounts):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_amounts	-	-	-

option_on_zero_coupon_bond_tree

Price an option on a zero coupon bond using a HW trinomial tree. The discount curve was already supplied to the tree build.

```
option_on_zero_coupon_bond_tree(t_exp,
                                tmat,
                                strike_price,
                                face_amount):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike_price	-	-	-
face_amount	-	-	-

bermudan_swaption

Swaption that can be exercised on specific dates over the exercise period. Due to non-analytical bond price we need to extend tree out to bond maturity and take into account cash flows through time.

```
bermudan_swaption(t_exp, tmat, strike, face,
                  cpn_times, cpn_flows, exercise_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike	-	-	-
face	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_type	-	-	-

bond_option

Value a bond option that can have European or American exercise. This is done using a trinomial tree that *we extend out to bond maturity. For European bond options, Jamshidian's model is faster and is used instead i.e. not this function.*

```
bond_option(t_exp, strike_price, face_amount,
            cpn_times, cpn_flows, exercise_type):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_type	-	-	-

callable_puttable_bond_tree

PLEASE ADD A FUNCTION DESCRIPTION

```
callable_puttable_bond_tree(cpn_times,
                            cpn_flows,
                            call_times,
                            call_prices,
                            put_times,
                            put_prices,
                            face_amount):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cpn_times	-	-	-
cpn_flows	-	-	-
call_times	-	-	-
call_prices	-	-	-
put_times	-	-	-
put_prices	-	-	-
face_amount	-	-	-

df_tree

Discount factor as seen from now to time tmat as long as the time is on the tree grid.

```
df_tree(tmat):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
tmat	-	-	-

build_tree

Build the trinomial tree.

```
build_tree(treeMat, df_times, df_values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
treeMat	-	-	-
df_times	-	-	-
df_values	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

option_exercise_types_to_int

PLEASE ADD A FUNCTION DESCRIPTION

```
option_exercise_types_to_int(optionExerciseType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
optionExerciseType	-	-	-

p_fast

Forward discount factor as seen at some time t which may be in the future for payment at time T where Rt is the delta-period short rate seen at time t and pt is the discount factor to time t, ptd is the one period discount factor to time t+dt and pT is the discount factor from now until the payment of the 1 dollar of the discount factor.

```
p_fast(t, T, Rt, delta, pt, ptd, pT, _sigma, _a):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
T	-	-	-
Rt	-	-	-
delta	-	-	-
pt	-	-	-
ptd	-	-	-
pT	-	-	-
_sigma	-	-	-
_a	-	-	-

build_tree_fast

Fast tree construction using Numba.

```
build_tree_fast(a, sigma, tree_times, num_time_steps, discount_factors):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
sigma	-	-	-
tree_times	-	-	-
num_time_steps	-	-	-
discount_factors	-	-	-

american_bond_option_tree_fast

Value an option on a bond with coupons that can have European or American exercise. Some minor issues to do with handling coupons on the option expiry date need to be solved.

```
american_bond_option_tree_fast(t_exp,
                               strike_price,
                               face_amount,
                               cpn_times,
                               cpn_amounts,
                               exercise_typeInt,
                               _sigma,
                               _a,
                               _Q,
                               _pu, _pm, _pd,
                               _rt,
                               _dt,
                               _tree_times,
                               _df_times, _df_values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_amounts	-	-	-
exercise_typeInt	-	-	-
_sigma	-	-	-
_a	-	-	-
_Q	-	-	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_tree_times	-	-	-
_df_times	-	-	-
_df_values	-	-	-

bermudan_swaption_tree_fast

Option to enter into a swap that can be exercised on coupon payment dates after the start of the exercise period. Due to multiple exercise times we need to extend tree out to bond maturity and take into account cash flows through time.

```
bermudan_swaption_tree_fast(t_exp, tmat, strike_price, face_amount,
                            cpn_times, cpn_flows,
                            exercise_typeInt,
                            _df_times, _df_values,
                            _tree_times, _Q, _pu, _pm, _pd, _rt, _dt, _a):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike_price	-	-	-
face_amount	-	-	-
cpn_times	-	-	-
cpn_flows	-	-	-
exercise_typeInt	-	-	-
_df_times	-	-	-
_df_values	-	-	-
_tree_times	-	-	-
_Q	-	-	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_a	-	-	-

callable_puttable_bond_tree_fast

Value an option on a bond with coupons that can have European or American exercise. Some minor issues to do with handling coupons on the option expiry date need to be solved.

```
callable_puttable_bond_tree_fast(cpn_times, cpn_flows,
                                 call_times, call_prices,
                                 put_times, put_prices, face,
                                 _sigma, _a, _Q,  # IS SIGMA USED ?
                                 _pu, _pm, _pd, _rt, _dt, _tree_times,
                                 _df_times, _df_values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cpn_times	-	-	-
cpn_flows	-	-	-
call_times	-	-	-
call_prices	-	-	-
put_times	-	-	-
put_prices	-	-	-
face	-	-	-
_sigma	-	IS SIGMA USED ?	-
_a	-	IS SIGMA USED ?	-
_Q	-	IS SIGMA USED ?	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_tree_times	-	-	-
_df_times	-	-	-
_df_values	-	-	-

fwd_dirty_bond_price

Price a coupon bearing bond on the option expiry date and return the difference from a strike price. This is used in a root search to find the future expiry time short rate that makes the bond price equal to the option strike price. It is a key step in the Jamshidian bond decomposition approach. The strike is a clean price.

```
fwd_dirty_bond_price(rt, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rt	-	-	-
*args	-	-	-

10.24 lmm_mc

Enumerated Type: ModelLMMModelTypes

This enumerated type has the following values:
- LMM_ONE_FACTOR
- LMM_HW_M_FACTOR
- LMM_FULL_N_FACTOR

lmm_print_forwards

Helper function to display the simulated Ibor rates.

```
lmm_print_forwards(fwds):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fwds	-	-	-

lmm_swaption_vol_approx

Implements Rebonato's approximation for the swap rate volatility to be used when pricing a swaption that expires in period a for a swap maturing at the end of period b taking into account the forward volatility term structure (zetas) and the forward-forward correlation matrix rho..

```
lmm_swaption_vol_approx(a, b, fwd0, taus, zetas, rho):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
b	-	-	-
fwd0	-	-	-
taus	-	-	-
zetas	-	-	-
rho	-	-	-

lmm_sim_swaption_vol

Calculates the swap rate volatility using the forwards generated in the simulation to see how it compares to Rebonatto estimate.

```
lmm_sim_swaption_vol(a, b, fwd0, fwds, taus):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
b	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-

lmm_fwd_fwd_correlation

Extract forward forward correlation matrix at some future time index from the simulated forward rates and return the matrix.

```
lmm_fwd_fwd_correlation(numForwards, num_paths, i_time, fwds):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numForwards	-	-	-
num_paths	-	-	-
i_time	-	-	-
fwds	-	-	-

lmm_price_caps_black

Price a strip of capfloorlets using Black's model using the time grid of the LMM model. The prices can be compared with the LMM model prices.

```
lmm_price_caps_black(fwd0, volCaplet, p, K, taus):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fwd0	-	-	-
volCaplet	-	-	-
p	-	-	-
K	-	-	-
taus	-	-	-

sub_matrix

Returns a submatrix of correlation matrix at later time step in the LMM simulation which is then used to generate correlated Gaussian RVs.

```
sub_matrix(t, N):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
N	-	-	-

cholesky_np

Numba-compliant wrapper around Numpy cholesky function.

```
cholesky_np(rho):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rho	-	-	-

lmm_simulate_fwds_nf

Full N-Factor Arbitrage-free simulation of forward Ibor discount in the spot measure given an initial forward curve, volatility term structure and full rank correlation structure. Cholesky decomposition is used to extract the factor weights. The number of forwards at time 0 is given. The 3D matrix of forward rates by path, time and forward point is returned. WARNING: NEED TO CHECK THAT CORRECT VOLATILITY IS BEING USED (OFF BY ONE BUG NEEDS TO BE RULED OUT)

```
lmm_simulate_fwds_nf(num_fwds, num_paths, fwd0, zetas, correl, taus, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_fwds	-	-	-
num_paths	-	-	-
fwd0	-	-	-
zetas	-	-	-
correl	-	-	-
taus	-	-	-
seed	-	-	-

lmm_simulate_fwds_1f

One factor Arbitrage-free simulation of forward Ibor discount in the spot measure following Hull Page 768. Given an initial forward curve, volatility term structure. The 3D matrix of forward rates by path, time and forward point is returned. This function is kept mainly for its simplicity and speed. NB: The Gamma volatility has an initial entry of zero. This differs from Hull's indexing by one and so is why I do not subtract 1 from the index as Hull does in his equation 32.14. The Number of Forwards is the number of points on the initial curve to the trade maturity date. But be careful: a cap that matures in 10 years with quarterly caplets has 40 forwards BUT the last forward to reset occurs at 9.75 years. You should not simulate beyond this time. If you give the model 10 years as in the Hull examples, you need to simulate 41 (or in this case 11) forwards as the final cap or ratchet has its reset in 10 years.

```
lmm_simulate_fwds_1f(num_fwds, num_paths, numeraireIndex, fwd0, gammas,
                     taus, use_sobol, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_fwds	-	-	-
num_paths	-	-	-
numeraireIndex	-	-	-
fwd0	-	-	-
gammas	-	-	-
taus	-	-	-
use_sobol	-	-	-
seed	-	-	-

lmm_simulate_fwds_mf

Multi-Factor Arbitrage-free simulation of forward Ibor discount in the spot measure following Hull Page 768. Given an initial forward curve, volatility factor term structure. The 3D matrix of forward rates by path, time and forward point is returned.

```
lmm_simulate_fwds_mf(num_fwds, numFactors, num_paths, numeraireIndex,
                     fwd0, lambdas, taus, use_sobol, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_fwds	-	-	-
numFactors	-	-	-
num_paths	-	-	-
numeraireIndex	-	-	-
fwd0	-	-	-
lambdas	-	-	-
taus	-	-	-
use_sobol	-	-	-
seed	-	-	-

lmm_cap_flr_pricer

Function to price a strip of cap or floorlets in accordance with the simulated forward curve dynamics.

```
lmm_cap_flr_pricer(num_fwds, num_paths, K, fwd0, fwds, taus, isCap):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_fwds	-	-	-
num_paths	-	-	-
K	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-
isCap	-	-	-

lmm_swap_pricer

Function to reprice a basic swap using the simulated forward Ibors.

```
lmm_swap_pricer(cpn, num_periods, num_paths, fwd0, fwds, taus):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cpn	-	-	-
num_periods	-	-	-
num_paths	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-

lmm_swaption_pricer

Function to price a European swaption using the simulated forward discount.

```
lmm_swaption_pricer(strike, a, b, num_paths, fwd0, fwds, taus, isPayer):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
strike	-	-	-
a	-	-	-
b	-	-	-
num_paths	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-
isPayer	-	-	-

lmm_ratchet_caplet_pricer

Price a ratchet using the simulated Ibor rates.

```
lmm_ratchet_caplet_pricer(spd, num_periods, num_paths, fwd0, fwds, taus):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spd	-	-	-
num_periods	-	-	-
num_paths	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-

lmm_flexi_cap_pricer

Price a flexicap using the simulated Ibor rates.

```
lmm_flexi_cap_pricer(maxCaplets, K, num_periods, num_paths,
                     fwd0, fwds, taus):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
maxCaplets	-	-	-
K	-	-	-
num_periods	-	-	-
num_paths	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-

lmm_sticky_caplet_pricer

Price a sticky cap using the simulated Ibor rates.

```
lmm_sticky_caplet_pricer(spread, num_periods, num_paths, fwd0, fwds, taus):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spread	-	-	-
num_periods	-	-	-
num_paths	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-

10.25 loss_dbn_builder

indep_loss_dbn_heterogeneous_adj_binomial

PLEASE ADD A FUNCTION DESCRIPTION

```
indep_loss_dbn_heterogeneous_adj_binomial(num_credits,
                                          cond_probs,
                                          loss_ratio):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_credits	-	-	-
cond_probs	-	-	-
loss_ratio	-	-	-

portfolio_gcd

PLEASE ADD A FUNCTION DESCRIPTION

```
portfolio_gcd(actual_losses):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
actual_losses	-	-	-

indep_loss_dbn_recursion_gcd

PLEASE ADD A FUNCTION DESCRIPTION

```
indep_loss_dbn_recursion_gcd(num_credits,
                             cond_default_probs,
                             loss_units):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_credits	-	-	-
cond_default_probs	-	-	-
loss_units	-	-	-

10.26 merton_firm

Class: MertonFirm()

Implementation of the Merton Firm Value Model according to the original formulation by Merton with the inputs being the asset value of the firm, the liabilities (bond face), the time to maturity in years, the risk-free rate, the asset growth rate and the asset value volatility.

MertonFirm

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

```
MertonFirm(assetValue: (float, list, np.ndarray),
           bondFace: (float, list, np.ndarray),
           timeToMaturity: (float, list, np.ndarray),
           risk_free_rate: (float, list, np.ndarray),
           assetGrowthRate: (float, list, np.ndarray),
           assetVolatility: (float, list, np.ndarray)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
assetValue	float or list,np.ndarray	-	-
bondFace	float or list,np.ndarray	-	-
timeToMaturity	float or list,np.ndarray	-	-
risk_free_rate	float or list,np.ndarray	-	-
assetGrowthRate	float or list,np.ndarray	-	-
assetVolatility	float or list,np.ndarray	-	-

leverage

Calculate the leverage.

```
leverage():
```

The function arguments are described in the following table.

asset_value

Calculate the asset value.

```
asset_value():
```

The function arguments are described in the following table.

debt_face_value

Calculate the asset value.

```
debt_face_value():
```

The function arguments are described in the following table.

equity_vol

Calculate the equity volatility.

```
equity_vol():
```

The function arguments are described in the following table.

equity_value

Calculate the equity value.

```
equity_value():
```

The function arguments are described in the following table.

debt_value

Calculate the debt value

```
debt_value():
```

The function arguments are described in the following table.

credit_spread

Calculate the credit spread from the debt value.

```
credit_spread():
```

The function arguments are described in the following table.

prob_default

Calculate the default probability. This is not risk-neutral so it uses the real world drift rather than the risk-free rate.

```
prob_default():
```

The function arguments are described in the following table.

dist_default

Calculate the distance to default. This is not risk-neutral so it uses the real world drift rather than the risk-free rate.

```
dist_default():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

10.27 merton_firm_mkt

Class: MertonFirmMkt(MertonFirm)

Market Extension of the Merton Firm Model according to the original formulation by Merton with the inputs being the equity value of the firm, the liabilities (bond face), the time to maturity in years, the risk-free rate, the asset growth rate and the equity volatility. The asset value and asset volatility are computed internally by solving two non-linear simultaneous equations.

MertonFirmMkt

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

```
MertonFirmMkt(equity_value: (float, list, np.ndarray),
              bondFace: (float, list, np.ndarray),
              timeToMaturity: (float, list, np.ndarray),
              risk_free_rate: (float, list, np.ndarray),
              assetGrowthRate: (float, list, np.ndarray),
              equity_volatility: (float, list, np.ndarray)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
equity_value	float or list,np.ndarray	-	-
bondFace	float or list,np.ndarray	-	-
timeToMaturity	float or list,np.ndarray	-	-
risk_free_rate	float or list,np.ndarray	-	-
assetGrowthRate	float or list,np.ndarray	-	-
equity_volatility	float or list,np.ndarray	-	-

_solve_for_asset_value_and_vol

PLEASE ADD A FUNCTION DESCRIPTION

```
_solve_for_asset_value_and_vol():
```

The function arguments are described in the following table.

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_fobj

Find value of asset value and vol that fit equity value and vol

```
_fobj(x, *args):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
*args	-	-	-

10.28 model

Class: Model()

class Model():

Model

PLEASE ADD A FUNCTION DESCRIPTION

```
Model():
```

The function arguments are described in the following table.

10.29 option_implied_dbn

option_implied_dbn

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

```
option_implied_dbn(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	-	-	-

10.30 process_simulator

Enumerated Type: ProcessTypes

This enumerated type has the following values:

- GBM
- CIR
- HESTON
- VASICEK
- CEV
- JUMP_DIFFUSION

Enumerated Type: FinHestonNumericalScheme

This enumerated type has the following values:

- EULER
- EULERLOG
- QUADEXP

Enumerated Type: FinGBMNumericalScheme

This enumerated type has the following values:

- NORMAL
- ANTITHETIC

Enumerated Type: FinVasicekNumericalScheme

This enumerated type has the following values:

- NORMAL
- ANTITHETIC

Enumerated Type: CIRNumericalScheme

This enumerated type has the following values:

- EULER
- LOGNORMAL
- MILSTEIN
- KAHLJACKEL
- EXACT

Class: FinProcessSimulator()

class FinProcessSimulator():

FinProcessSimulator

PLEASE ADD A FUNCTION DESCRIPTION

```
FinProcessSimulator():
```

The function arguments are described in the following table.

get_process

PLEASE ADD A FUNCTION DESCRIPTION

```
get_process(process_type,
    t,
    model_params,
    numAnnSteps,
    num_paths,
    seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
process_type	-	-	-
t	-	-	-
model_params	-	-	-
numAnnSteps	-	-	-
num_paths	-	-	-
seed	-	-	-

get_heston_paths

PLEASE ADD A FUNCTION DESCRIPTION

```
get_heston_paths(num_paths,
                 numAnnSteps,
                 t,
                 drift,
                 s0,
                 v0,
                 kappa,
                 theta,
                 sigma,
                 rho,
                 scheme,
                 seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_paths	-	-	-
numAnnSteps	-	-	-
t	-	-	-
drift	-	-	-
s0	-	-	-
v0	-	-	-
kappa	-	-	-
theta	-	-	-
sigma	-	-	-
rho	-	-	-
scheme	-	-	-
seed	-	-	-

get_gbm_paths

PLEASE ADD A FUNCTION DESCRIPTION

```
get_gbm_paths(num_paths, numAnnSteps, t, mu, stock_price, sigma, scheme, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_paths	-	-	-
numAnnSteps	-	-	-
t	-	-	-
mu	-	-	-
stock_price	-	-	-
sigma	-	-	-
scheme	-	-	-
seed	-	-	-

get_vasicek_paths

PLEASE ADD A FUNCTION DESCRIPTION

```
get_vasicek_paths(num_paths,
                  numAnnSteps,
                  t,
                  r0,
                  kappa,
                  theta,
                  sigma,
                  scheme,
                  seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_paths	-	-	-
numAnnSteps	-	-	-
t	-	-	-
r0	-	-	-
kappa	-	-	-
theta	-	-	-
sigma	-	-	-
scheme	-	-	-
seed	-	-	-

get_cir_paths

PLEASE ADD A FUNCTION DESCRIPTION

```
get_cir_paths(num_paths,
              numAnnSteps,
              t,
              r0,
              kappa,
              theta,
              sigma,
              scheme,
              seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_paths	-	-	-
numAnnSteps	-	-	-
t	-	-	-
r0	-	-	-
kappa	-	-	-
theta	-	-	-
sigma	-	-	-
scheme	-	-	-
seed	-	-	-

10.31 rates_ho_lee

Class: ModelRatesHoLee

class ModelRatesHoLee:

ModelRatesHoLee

*Construct Ho-Lee model using single parameter of volatility. The dynamical equation is dr = theta(t) dt + sigma * dW. Any no-arbitrage fitting is done within functions below.*

```
ModelRatesHoLee(sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-

zcb

PLEASE ADD A FUNCTION DESCRIPTION

```
zcb(rt1, t1, t2, discount_curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rt1	-	-	-
t1	-	-	-
t2	-	-	-
discount_curve	-	-	-

option_on_zcb

Price an option on a zero coupon bond using analytical solution of Hull-White model. User provides bond face and option strike and expiry date and maturity date.

```
option_on_zcb(t_exp, tmat,
              strike_price, face_amount,
              df_times, df_values):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t_exp	-	-	-
tmat	-	-	-
strike_price	-	-	-
face_amount	-	-	-
df_times	-	-	-
df_values	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

p_fast

Forward discount factor as seen at some time t which may be in the future for payment at time T where Rt is the delta-period short rate seen at time t and pt is the discount factor to time t, ptd is the one period discount factor to time t+dt and pT is the discount factor from now until the payment of the 1 dollar of the discount factor.

```
p_fast(t, T, Rt, delta, pt, ptd, pT, _sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
T	-	-	-
Rt	-	-	-
delta	-	-	-
pt	-	-	-
ptd	-	-	-
pT	-	-	-
_sigma	-	-	-

10.32 sabr

Class: SABR()

SABR - Stochastic alpha beta rho model by Hagan et al. which is a stochastic volatility model where alpha controls the implied volatility, beta is the exponent on the the underlying assets process so beta = 0 is normal and beta = 1 is lognormal, rho is the correlation between the underlying and the volatility process.

SABR

Create SABR with all of the model parameters. We will also provide functions below to assist with the calibration of the value of alpha.

```
SABR(alpha, beta, rho, nu):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
beta	-	-	-
rho	-	-	-
nu	-	-	-

black_vol

Black volatility from SABR model using Hagan et al. approx.

```
black_vol(f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
f	-	-	-
k	-	-	-
t	-	-	-

black_vol_with_alpha

PLEASE ADD A FUNCTION DESCRIPTION

```
black_vol_with_alpha(alpha, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

value

Price an option using Black's model which values in the forward measure following a change of measure.

```
value(forward_rate,    # Forward rate
      strike_rate,     # Strike Rate
      time_to_expiry,   # time to expiry in years
      df,             # Discount Factor to expiry date
      call_or_put):     # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forward_rate	-	Forward rate	-
strike_rate	-	Strike Rate	-
time_to_expiry	-	time to expiry in years	-
df	-	Discount Factor to expiry date	-
call_or_put	-	Call or put	-

set_alpha_from_black_vol

Estimate the value of the alpha coefficient of the SABR model by solving for the value of alpha that makes the SABR black vol equal to the input black vol. This uses a numerical 1D solver.

```
set_alpha_from_black_vol(blackVol,
                         forward,
                         strike,
                         time_to_expiry):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
blackVol	-	-	-
forward	-	-	-
strike	-	-	-
time_to_expiry	-	-	-

fn

PLEASE ADD A FUNCTION DESCRIPTION

```
fn(x): return np.sqrt(
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value

set_alpha_from_atm_black_vol

We solve cubic equation for the unknown variable alpha for the special ATM case of the strike equalling the forward following Hagan and al. equation (3.3). We take the smallest real root as the preferred solution. This is useful for calibrating the model when beta has been chosen.

```
set_alpha_from_atm_black_vol(blackVol, atmStrike, time_to_expiry):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
blackVol	-	-	-
atmStrike	-	-	-
time_to_expiry	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_x

Return function x used in Hagan's 2002 SABR lognormal vol expansion.

```
_x(rho, z):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rho	-	-	-
z	-	-	-

vol_function_sabr

Black volatility implied by SABR model.

```
vol_function_sabr(params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

vol_function_sabr_beta_one

This is the SABR function with the exponent beta set equal to 1 so only 3 parameters are free. The first parameter is alpha, then nu and the third parameter is rho. Check the order as it is not the same as main SABR fn

```
vol_function_sabr_beta_one(params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

vol_function_sabr_beta_half

Black volatility implied by SABR model.

```
vol_function_sabr_beta_half(params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

10.33 sabr_shifted

Class: SABRShifted()

SABR - Shifted Stochastic alpha beta rho model by Hagan et al. is a stochastic volatility model where alpha controls the implied volatility, beta is the exponent on the the underlying assets process so beta = 0 is normal and beta = 1 is lognormal, rho is the correlation between the underlying and the volatility process. The shift allows negative rates.

SABRShifted

Create SABRShifted with all of the model parameters. We also provide functions below to assist with the calibration of the value of alpha.

```
SABRShifted(alpha, beta, rho, nu, shift):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
beta	-	-	-
rho	-	-	-
nu	-	-	-
shift	-	-	-

black_vol

Black volatility from SABR model using Hagan et al. approx.

```
black_vol(f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
f	-	-	-
k	-	-	-
t	-	-	-

black_vol_with_alpha

PLEASE ADD A FUNCTION DESCRIPTION

```
black_vol_with_alpha(alpha, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

value

Price an option using Black's model which values in the forward measure following a change of measure.

```
value(forward_rate,    # Forward rate F
      strike_rate,     # Strike Rate K
      time_to_expiry,  # Time to Expiry (years)
      df,             # Discount Factor to expiry date
      call_or_put):    # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forward_rate	-	Forward rate F	-
strike_rate	-	Strike Rate K	-
time_to_expiry	-	Time to Expiry (years)	-
df	-	Discount Factor to expiry date	-
call_or_put	-	Call or put	-

set_alpha_from_black_vol

Estimate the value of the alpha coefficient of the SABR model by solving for the value of alpha that makes the SABR black vol equal to the input black vol. This uses a numerical 1D solver.

```
set_alpha_from_black_vol(blackVol,
                         forward,
                         strike,
                         time_to_expiry):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
blackVol	-	-	-
forward	-	-	-
strike	-	-	-
time_to_expiry	-	-	-

fn

PLEASE ADD A FUNCTION DESCRIPTION

```
fn(x): return np.sqrt(
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value

set_alpha_from_atm_black_vol

We solve cubic equation for the unknown variable alpha for the special ATM case of the strike equalling the forward following Hagan and al. equation (3.3). We take the smallest real root as the preferred solution. This is useful for calibrating the model when beta has been chosen.

```
set_alpha_from_atm_black_vol(blackVol, atmStrike,
                             time_to_expiry):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
blackVol	-	-	-
atmStrike	-	-	-
time_to_expiry	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

_x

Return function x used in Hagan's 2002 SABR lognormal vol expansion.

```
_x(rho, z):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rho	-	-	-
z	-	-	-

vol_function_shifted_sabr

Black volatility implied by SABR model.

```
vol_function_shifted_sabr(params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

10.34 sobol

get_gaussian_sobol

Sobol Gaussian quasi random points generator based on graycode order. The generated points follow a normal distribution.

```
get_gaussian_sobol(num_points, dimension):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_points	-	-	-
dimension	-	-	-

get_uniform_sobol

Sobol uniform quasi random points generator based on graycode order. This function returns a 2D Numpy array of values where the number of rows is the number of draws and the number of columns is the number of dimensions of the random values. Each dimension has the same number of random draws. Each column of random numbers is ordered so as not to correlate, i.e be independent from any other column.

```
get_uniform_sobol(num_points, dimension):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
num_points	-	-	-
dimension	-	-	-

10.35 student_t_copula

Class: StudentTCopula()

class StudentTCopula():

default_times

PLEASE ADD A FUNCTION DESCRIPTION

```
default_times(issuer_curves,
              correlation_matrix,
              degreesOfFreedom,
              num_trials,
              seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issuer_curves	-	-	-
correlation_matrix	-	-	-
degreesOfFreedom	-	-	-
num_trials	-	-	-
seed	-	-	-

10.36 vasicek_mc

Class: ModelRatesVasicek()

class ModelRatesVasicek():

ModelRatesVasicek

PLEASE ADD A FUNCTION DESCRIPTION

```
ModelRatesVasicek(a, b, sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
b	-	-	-
sigma	-	-	-

__repr__

PLEASE ADD A FUNCTION DESCRIPTION

```
__repr__():
```

The function arguments are described in the following table.

meanr

PLEASE ADD A FUNCTION DESCRIPTION

```
meanr(r0, a, b, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
t	-	-	-

variancer

PLEASE ADD A FUNCTION DESCRIPTION

```
variancer(a, b, sigma, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
b	-	-	-
sigma	-	-	-
t	-	-	-

zero_price

PLEASE ADD A FUNCTION DESCRIPTION

```
zero_price(r0, a, b, sigma, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
sigma	-	-	-
t	-	-	-

rate_path_mc

PLEASE ADD A FUNCTION DESCRIPTION

```
rate_path_mc(r0, a, b, sigma, t, dt, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
sigma	-	-	-
t	-	-	-
dt	-	-	-
seed	-	-	-

zero_price_mc

PLEASE ADD A FUNCTION DESCRIPTION

```
zero_price_mc(r0, a, b, sigma, t, dt, num_paths, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
sigma	-	-	-
t	-	-	-
dt	-	-	-
num_paths	-	-	-
seed	-	-	-

10.37 volatility_fns

Enumerated Type: VolFuncTypes

This enumerated type has the following values:

- CLARK
- SABR
- SABR_BETA_ONE
- SABR_BETA_HALF
- BBG
- CLARK5
- SVI
- SSVI

vol_function_clark

Volatility Function in book by Iain Clark generalised to allow for higher than quadratic power. Care needs to be taken to avoid overfitting. The exact reference is Clark Page 59.

```
vol_function_clark(params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

vol_function_bloomberg

*Volatility Function similar to the one used by Bloomberg. It is a quadratic function in the spot delta of the option. It can therefore go negative so it requires a good initial guess when performing the fitting to avoid this happening. The first parameter is the quadratic coefficient i.e. sigma(K) = a * D * D + b * D + c where a = params[0], b = params[1], c = params[2] and D is the spot delta.*

```
vol_function_bloomberg(params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

vol_function_svi

Volatility Function proposed by Gatheral in 2004. Increasing a results in a vertical translation of the smile in the positive direction. Increasing b decreases the angle between the put and call wing, i.e. tightens the smile. Increasing rho results in a counter-clockwise rotation of the smile. Increasing m results in a horizontal translation of the smile in the positive direction. Increasing sigma reduces the at-the-money curvature of the smile.

```
vol_function_svi(params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

phi_ssvi

PLEASE ADD A FUNCTION DESCRIPTION

```
phi_ssvi(theta, gamma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
theta	-	-	-
gamma	-	-	-

ssvi

This is the total variance w = sigma(t) x sigma(t) (0,t) x t

```
ssvi(x, gamma, sigma, rho, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
gamma	-	-	-
sigma	-	-	-
rho	-	-	-
t	-	-	-

ssvi1

PLEASE ADD A FUNCTION DESCRIPTION

```
ssvi1(x, gamma, sigma, rho, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
gamma	-	-	-
sigma	-	-	-
rho	-	-	-
t	-	-	-

ssvi2

PLEASE ADD A FUNCTION DESCRIPTION

```
ssvi2(x, gamma, sigma, rho, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
gamma	-	-	-
sigma	-	-	-
rho	-	-	-
t	-	-	-

ssvit

PLEASE ADD A FUNCTION DESCRIPTION

```
ssvit(x, gamma, sigma, rho, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
gamma	-	-	-
sigma	-	-	-
rho	-	-	-
t	-	-	-

g

PLEASE ADD A FUNCTION DESCRIPTION

```
g(x, gamma, sigma, rho, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
gamma	-	-	-
sigma	-	-	-
rho	-	-	-
t	-	-	-

dminus

PLEASE ADD A FUNCTION DESCRIPTION

```
dminus(x, gamma, sigma, rho, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
gamma	-	-	-
sigma	-	-	-
rho	-	-	-
t	-	-	-

density_ssvi

PLEASE ADD A FUNCTION DESCRIPTION

```
density_ssvi(x, gamma, sigma, rho, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
gamma	-	-	-
sigma	-	-	-
rho	-	-	-
t	-	-	-

ssvi_local_varg

PLEASE ADD A FUNCTION DESCRIPTION

```
ssvi_local_varg(x, gamma, sigma, rho, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-
gamma	-	-	-
sigma	-	-	-
rho	-	-	-
t	-	-	-

vol_function_ssvi

Volatility Function proposed by Gatheral in 2004.

```
vol_function_ssvi(params, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
params	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

10.38 __init__