FinancePy 0.185

Dominic O'Kane

November 14, 2020

Contents

Chapter 1

Introduction to FinancePy

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'

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 wishing to learn derivative pricing and Python.
- Professors wishing to teach derivative pricing and Python.
- 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 a price.
- Portfolio managers wishing to check prices or calculate risk measures.
- Fund managers wanting to value a portfolio or examine a trading strategy.

- Structurers or financial engineers seeking to examine the pricing of a derivative structure.

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

1. To make the code as simple as possible so that those with a basic Python fluency can understand and check the code.

2. To keep all the code in Python so users can look through the code to the lowest level.

3. To offset the performance impact of (2) by leveraging Numba to make the code as fast as possible without resorting to Cython.

4. To make the design product-based rather than model-based so someone wanting to price a specific product can easily find that without having to worry too much about the model – just use the default – unless they want to. For most products, a Monte-Carlo implementation has been provided both as a reference for testing and as a way to better understand how the product functions in terms of payments, their timings and conditions.

5. To make the library as complete as possible so a user can find all their required finance-related functionality in one place. This is better for the user as they only have to learn one interface.

6. To avoid complex designs. Limited inheritance unless it allows for significant code reuse. Some code duplication is OK, at least temporarily.

7. To have good documentation and easy-to-follow examples.

8. To make it easy for interested parties to contribute.

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

Important Note:

- IF YOU HAVE ANY PRICING OR RISK EXAMPLES YOU WOULD LIKE REPLICATED, SEND SCREENSHOTS OF ALL THE UNDERLYING DATA, MODEL DETAILS AND VALUATION.

- IF THERE IS A PRODUCT YOU WOULD LIKE TO HAVE ADDED, SEND ME THE REQUEST.

- IF THERE IS FUNCTIONALITY YOU WOULD LIKE ADDED, SEND ME A REQUEST.

Contact me at quant@financepy.com.

The Library Design

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

- Finutils - These are utility functions used to assist you with modelling a security. These include dates (FinDate), calendars, schedule generation, some finance-related mathematics functions and some helper functions.

- Market - These are modules that capture the market information used to value a security. These include interest rate and credit curves, volatility surfaces and prices.
- Models - These are the low-level models used to value derivative securities ranging from Black-Scholes to complex stochastic volatility models.
- Products - These are the actual securities and range from Government bonds to Bermudan swaptions.

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

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

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

Author

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

Dependencies

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

Changelog

See the changelog for a detailed history of changes.

Contributions

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

- You should use CamelCase i.e. variables of the form optionPrice.
- Comments are required for every class and function and they should be clear.
- At least one test case must be provided for every function.
- Follow the style of the code as currently written. This may change over time but please use the current style as your guide.

Disclaimer

Chapter 2

financepy.finutils

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

- FinDate is a class for handling dates in a financial setting. Special functions are included for computing IMM dates and CDS dates and moving dates forward by tenors.
- FinCalendar is a class for determining which dates are not business dates in a specific region or country.
- FinDayCount is a class for determining accrued interest in bonds and also accrual factors in ISDA swap-like contracts.
- FinError is a class which handles errors in the calculations done within FinancePy
- FinFrequency takes in a frequency type and then returns the number of payments per year
- FinGlobalVariables holds the value of constants used across the whole of FinancePy
- FinHelperFunctions is a set of helpful functions that can be used in a number of places
- FinMath is a set of mathematical functions specific to finance which have been optimised for speed using Numba
- FinSobol is the implementation of Sobol quasi-random number generator. It has been speeded up using Numba.
- FinRateConverter converts rates for one compounding frequency to rates for a different 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 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 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 FinAmount

Class: FinAmount(object)

FinAmount

Create FinAmount object.

```
FinAmount(notional: float = ONE_MILLION,
          currencyType: FinCurrencyTypes = FinCurrencyTypes.NONE):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
notional	float	-	ONE_MILLION
currencyType	FinCurrencyTypes	-	NONE

amount

return self._notional

```
amount():
```

The function arguments are described in the following table.

2.2 FinCalendar

Enumerated Type: FinBusDayAdjustTypes

This enumerated type has the following values:

- NONE
- FOLLOWING
- MODIFIED_FOLLOWING
- PRECEDING
- MODIFIED_PRECEDING

Enumerated Type: FinCalendarTypes

This enumerated type has the following values:

- TARGET
- US
- UK
- WEEKEND
- JAPAN
- NONE

Enumerated Type: FinDateGenRuleTypes

This enumerated type has the following values:

- FORWARD
- BACKWARD

Class: FinCalendar(object)

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.

FinCalendar

Create a calendar based on a specified calendar type.

```
FinCalendar(calendarType: FinCalendarTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
calendarType	FinCalendarTypes	-	-

adjust

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

```
adjust(dt: FinDate,
       busDayConventionType: FinBusDayAdjustTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-
busDayConventionType	FinBusDayAdjustTypes	-	-

addBusinessDays

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

```
addBusinessDays(startDate: FinDate,
                numDays: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startDate	FinDate	-	-
numDays	int	-	-

isBusinessDay

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

```
isBusinessDay(dt: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-

getHolidayList

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

```
getHolidayList(year: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
year	float	-	-

easterMonday

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.

```
easterMonday(year: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
year	float	-	-

2.3 FinCurrency

Enumerated Type: FinCurrencyTypes

This enumerated type has the following values:

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

2.4 FinDate

Enumerated Type: FinDateFormatTypes

This enumerated type has the following values:

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

Class: FinDate()

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

FinDate

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. Example Input: start_date = FinDate('1-1-2018', 'start_date = FinDate(1, 1, 2018)

```
FinDate(*args):  # Year number which must be between 1900 and 2100
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
*args	-	Year number which must be between 1900 and 2100	-

isWeekend

returns True if the date falls on a weekend.

```
isWeekend():
```

The function arguments are described in the following table.

addDays

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

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

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numDays	int	-	1

addWeekDays

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

```
addWeekDays(numDays: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numDays	int	-	-

addMonths

Returns a new date that is mm months after the FinDate. 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.

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

The function arguments are described in the following table.

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

addYears

Returns a new date that is yy years after the FinDate. 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.

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

The function arguments are described in the following table.

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

nextCDSDate

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

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

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
mm	int	-	0

thirdWednesdayOfMonth

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

```
thirdWednesdayOfMonth(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	-

nextIMMDate

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.

```
nextIMMDate():
```

The function arguments are described in the following table.

addTenor

Return the date following the FinDate 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.

```
addTenor(tenor: str):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
tenor	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	-	-	-

setDateFormatType

global gDateFormatType

```
setDateFormatType(formatType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
formatType	-	-	-

isLeapYear

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

```
isLeapYear(y: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
y	int	-	-

parse_date

dt_obj = datetime.datetime.strptime(dateStr, dateFormat)

```
parse_date(dateStr, dateFormat):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dateStr	-	-	-
dateFormat	-	-	-

calculateList

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.

```
calculateList():
```

The function arguments are described in the following table.

dateIndex

*idx = (y-gStartYear) * 12 * 31 + (m-1) * 31 + (d-1)*

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

The function arguments are described in the following table.

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

dateFromIndex

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

```
dateFromIndex(idx):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
idx	-	-	-

weekDay

weekday = (dayCount+5)

```
weekDay(dayCount):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dayCount	-	-	-

dailyWorkingDaySchedule

Returns a list of working dates between startDate and endDate. This function should be replaced by dateRange once addTenor allows for working days.

```
dailyWorkingDaySchedule(self,
                        startDate: FinDate,
                        endDate: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
self	-	-	-
startDate	FinDate	-	-
endDate	FinDate	-	-

datediff

Calculate the number of days between two Findates.

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

The function arguments are described in the following table.

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

fromDatetime

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

```
fromDatetime(dt: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-

daysInMonth

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

```
daysInMonth(m, y):
```

The function arguments are described in the following table.

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

dateRange

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

```
dateRange(startDate: FinDate,
          endDate: FinDate,
          tenor: str = "1D"):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startDate	FinDate	-	-
endDate	FinDate	-	-
tenor	str	-	”1D”

testType

global gDateFormatType

```
testType():
```

The function arguments are described in the following table.

2.5 FinDayCount

Enumerated Type: FinDayCountTypes

This enumerated type has the following values:

- 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

Class: FinDayCount(object)

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

FinDayCount

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

```
FinDayCount(dccType: FinDayCountTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dccType	FinDayCountTypes	-	-

yearFrac

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. 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

```
yearFrac(dt1: FinDate,     # Start of coupon period
         dt2: FinDate,     # Settlement (for bonds) or period end(swaps)
         dt3: FinDate = None,    # End of coupon period for accrued
         freqType: FinFrequencyTypes = FinFrequencyTypes.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	FinDate	Start of coupon period	-
dt2	FinDate	Settlement (for bonds) or period end(swaps)	-
dt3	FinDate	End of coupon period for accrued	None
freqType	FinFrequencyTypes	-	ANNUAL
isTerminationDate	bool	Is dt2 a termination date	False

isLastDayOfFeb

 # Return true if we are on the last day of February

```
isLastDayOfFeb(dt: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-

2.6 FinError

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	-	-

func_name

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

```
func_name():
```

The function arguments are described in the following table.

suppressErrors

 # print(sys.tracebacklimit)

```
suppressErrors():
```

The function arguments are described in the following table.

2.7 FinFrequency

Enumerated Type: FinFrequencyTypes

This enumerated type has the following values:

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

FinFrequency

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

```
FinFrequency(freqType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
freqType	-	-	-

2.8 FinGlobalTypes

Enumerated Type: FinOptionTypes

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: FinCapFloorTypes

This enumerated type has the following values:

- CAP
- FLOOR

Enumerated Type: FinSwapTypes

This enumerated type has the following values:

- PAY
- RECEIVE

Enumerated Type: FinExerciseTypes

This enumerated type has the following values:

- EUROPEAN
- BERMUDAN
- AMERICAN

2.9 FinGlobalVariables

2.10 FinHelperFunctions

gridIndex

PLEASE ADD A FUNCTION DESCRIPTION

```
gridIndex(t, gridTimes):
```

The function arguments are described in the following table.

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

betaVectorToCorrMatrix

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

```
betaVectorToCorrMatrix(betas):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
betas	-	-	-

pv01Times

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.

```
pv01Times(t: float,
          f: float):
```

The function arguments are described in the following table.

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

timesFromDates

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.

```
timesFromDates(dt: FinDate,
               valuationDate: FinDate,
               dayCountType: FinDayCountTypes = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-
valuationDate	FinDate	-	-
dayCountType	FinDayCountTypes	-	None

checkVectorDifferences

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

```
checkVectorDifferences(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

checkDate

Check that input d is a FinDate.

```
checkDate(d: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
d	FinDate	-	-

dump

Get a list of all of the attributes of a class (not built in ones)

```
dump(obj):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
obj	-	-	-

printTree

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

```
printTree(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

inputTime

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 FinDate 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.

```
inputTime(dt: FinDate,
          curve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-
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	-	-

normaliseWeights

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

```
normaliseWeights(wtVector: np.ndarray):
```

The function arguments are described in the following table.

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

labelToString

Format label/value pairs for a unified formatting.

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

The function arguments are described in the following table.

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

tableToString

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

```
tableToString(header: str,
              valueTable,
              floatPrecision="10.7f"):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
header	str	-	-
valueTable	-	-	-
floatPrecision	-	-	”10.7f”

toUsableType

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

```
toUsableType(t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-

uniformToDefaultTime

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

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

The function arguments are described in the following table.

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

accruedTree

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

```
accruedTree(gridTimes: np.ndarray,
            gridFlows: np.ndarray,
            face: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
gridTimes	np.ndarray	-	-
gridFlows	np.ndarray	-	-
face	float	-	-

checkArgumentTypes

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.

```
checkArgumentTypes(func, values):
```

The function arguments are described in the following table.

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

2.11 FinMath

accruedInterpolator

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

```
accruedInterpolator(tset: float,  # Settlement time in years
                    couponTimes: np.ndarray,
                    couponAmounts: np.ndarray):
```

The function arguments are described in the following table.

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

isLeapYear

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

```
isLeapYear(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	-	-

testMonotonicity

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

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

The function arguments are described in the following table.

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

testRange

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

```
testRange(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	-	-

pairGCD

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.

```
pairGCD(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	-	-

normcdf_fast

Fast Normal CDF function based on XXX

```
normcdf_fast(x: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	float	-	-

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	-	-

normcdf

This is the Normal CDF function which forks to one of three of the implemented approximations. This is based on the choice of the fast flag variable. A value of 1 is the fast routine, 2 is the slow and 3 is the even slower integration scheme.

```
normcdf(x: float,
        fastFlag: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	float	-	-
fastFlag	int	-	-

N

This is the shortcut to the default Normal CDF function and currently is hardcoded to the fastest of the implemented routines. This is the most widely used way to access the Normal CDF.

```
N(x: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	float	-	-

phi3

Bivariate Normal CDF function to upper limits $b1$ and $b2$ which uses integration to perform the innermost 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

return phi2(a, b, c)

```
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	-	-	-

corrMatrixGenerator

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

```
corrMatrixGenerator(rho, n):
```

The function arguments are described in the following table.

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

2.12 FinSchedule

Class: FinSchedule(object)

A Schedule is a vector of dates generated according to ISDA standard rules which starts on the next date after the start date and runs up to an end 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).

FinSchedule

Create FinSchedule object which calculates a sequence of dates in line with market convention for fixed income products.

```
FinSchedule(effectiveDate: FinDate,   # Also known as the start date
            terminationDate: FinDate,  # Also known as the termination date
            freqType: FinFrequencyTypes = FinFrequencyTypes.ANNUAL,
            calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
            busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
            dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD,
            adjustTerminationDate:bool = False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effectiveDate	FinDate	Also known as the start date	-
terminationDate	FinDate	Also known as the termination date	-
freqType	FinFrequencyTypes	-	ANNUAL
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD
adjustTerminationDate	bool	-	False

scheduleDates

Returns a list of the schedule of FinDates.

```
scheduleDates():
```

The function arguments are described in the following table.

2.13 FinStatistics

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	-	-

Chapter 3

financepy.market.curves

Curves

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

1. Best fit yield curves fitting to bond prices which are used for interpolation. A range of curve shapes from polynomials to B-Splines is available.

2. Discount curves that can be used to present value a future cash flow. These differ from best fits curves in that they exactly refit the prices of bonds or CDS. The different discount curves are created by calibrating to different instruments. They also differ in terms of the term structure shapes they can have. Different shapes have different impacts in terms of locality on risk management performed using these different curves. There is often a trade-off between smoothness and locality.

Best Fit Bond Curves

The first category are FinBondYieldCurves.

FinBondYieldCurve

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

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

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

FinCurveFitMethod

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

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

Discount Curves

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

FinDiscountCurve

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

FinDiscountCurveFlat

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

FinDiscountCurveNS

Implementation of the Nelson-Siegel curve parametrisation.

FinDiscountCurveNSS

Implementation of the Nelson-Siegel-Svensson curve parametrisation.

FinDiscountCurveZeros

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

FinInterpolate

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

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

2. PIECEWISE LOG LINEAR - This assumes that the log of the discount factor is interpolated linearly. The log of a discount factor to time T is T 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 FinDiscountCurve

Class: FinDiscountCurve()

This is a base discount curve which has an internal representation of a vector of times and discount factors and an interpolation scheme for interpolating between these fixed points.

FinDiscountCurve

Create the discount curve from a vector of times and discount factors with an anchor date and specify an interpolation scheme. As we are explicity 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.

```
FinDiscountCurve(valuationDate: FinDate,
                 dfDates: list,
                 dfValues: np.ndarray,
                 interpType: FinInterpTypes = FinInterpTypes.FLAT_FWD_RATES):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
dfDates	list	-	-
dfValues	np.ndarray	-	-
interpType	FinInterpTypes	-	FLAT_FWD_RATES

zeroRate

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.

```
zeroRate(dts: (list, FinDate),
         freqType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS,
         dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dts	list or FinDate	-	-
freqType	FinFrequencyTypes	-	CONTINUOUS
dayCountType	FinDayCountTypes	-	ACT_360

swapRate

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.

```
swapRate(startDate: FinDate,
         maturityDate: (list, FinDate),
         freqType=FinFrequencyTypes.ANNUAL,
         dayCountType: FinDayCountTypes = FinDayCountTypes.THIRTY_E_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startDate	FinDate	-	-
maturityDate	list or FinDate	-	-
freqType	-	-	ANNUAL
dayCountType	FinDayCountTypes	-	THIRTY_E_360

df

Function to calculate a discount factor from a date or a vector of dates.

```
df(dt: (list, FinDate)):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	list or FinDate	-	-

survProb

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 analagous to a discount factor.

```
survProb(dt: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-

fwd

Calculate the continuously compounded forward rate at the forward FinDate 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: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dts	FinDate	-	-

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(bumpSize: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bumpSize	float	-	-

fwdRate

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.

```
fwdRate(startDate: (list, FinDate),
        dateOrTenor: (FinDate, str),
        dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startDate	list or FinDate	-	-
dateOrTenor	FinDate or str	-	-
dayCountType	FinDayCountTypes	-	ACT_360

3.2 FinDiscountCurveFlat

Class: FinDiscountCurveFlat(FinDiscountCurve)

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.

FinDiscountCurveFlat

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.

```
FinDiscountCurveFlat(valuationDate: FinDate,
                     flatRate: (float, np.ndarray),
                     freqType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS,
                     dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
flatRate	float or np.ndarray	-	-
freqType	FinFrequencyTypes	-	CONTINUOUS
dayCountType	FinDayCountTypes	-	ACT_ACT_ISDA

bump

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

```
bump(bumpSize: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bumpSize	float	-	-

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: (FinDate, list)):
```

The function arguments are described in the following table.

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

3.3 FinDiscountCurveNS

Class: FinDiscountCurveNS(FinDiscountCurve)

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.

FinDiscountCurveNS

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.

```
FinDiscountCurveNS(valuationDate: FinDate,
                   beta0: float,
                   beta1: float,
                   beta2: float,
                   tau: float,
                   freqType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS,
                   dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
beta0	float	-	-
beta1	float	-	-
beta2	float	-	-
tau	float	-	-
freqType	FinFrequencyTypes	-	CONTINUOUS
dayCountType	FinDayCountTypes	-	ACT_ACT_ISDA

zeroRate

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.

```
zeroRate(dates: (list, FinDate),
         freqType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS,
         dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dates	list or FinDate	-	-
freqType	FinFrequencyTypes	-	CONTINUOUS
dayCountType	FinDayCountTypes	-	ACT_360

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: (FinDate, list)):
```

The function arguments are described in the following table.

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

3.4 FinDiscountCurveNSS

Class: FinDiscountCurveNSS(FinDiscountCurve)

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.

FinDiscountCurveNSS

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.

```
FinDiscountCurveNSS(valuationDate: FinDate,
                    beta0: float,
                    beta1: float,
                    beta2: float,
                    beta3: float,
                    tau1: float,
                    tau2: float,
                    freqType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS,
                    dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
beta0	float	-	-
beta1	float	-	-
beta2	float	-	-
beta3	float	-	-
tau1	float	-	-
tau2	float	-	-
freqType	FinFrequencyTypes	-	CONTINUOUS
dayCountType	FinDayCountTypes	-	ACT_ACT_ISDA

zeroRate

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.

```
zeroRate(dates: (list, FinDate),
         freqType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS,
         dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dates	list or FinDate	-	-
freqType	FinFrequencyTypes	-	CONTINUOUS
dayCountType	FinDayCountTypes	-	ACT_360

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: (FinDate, list)):
```

The function arguments are described in the following table.

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

3.5 FinDiscountCurvePoly

Class: FinDiscountCurvePoly(FinDiscountCurve)

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.

FinDiscountCurvePoly

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

```
FinDiscountCurvePoly(valuationDate: FinDate,
                     coefficients: (list, np.ndarray),
                     freqType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS,
                     dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
coefficients	list or np.ndarray	-	-
freqType	FinFrequencyTypes	-	CONTINUOUS
dayCountType	FinDayCountTypes	-	ACT_ACT_ISDA

zeroRate

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.

```
zeroRate(dts: (list, FinDate),
         freqType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS,
         dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dts	list or FinDate	-	-
freqType	FinFrequencyTypes	-	CONTINUOUS
dayCountType	FinDayCountTypes	-	ACT_360

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, FinDate)):
```

The function arguments are described in the following table.

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

3.6 FinDiscountCurvePWF

Class: FinDiscountCurvePWF(FinDiscountCurve)

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.

FinDiscountCurvePWF

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

```
FinDiscountCurvePWF(valuationDate: FinDate,
                    zeroDates: list,
                    zeroRates: (list, np.ndarray),
                    freqType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS,
                    dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
zeroDates	list	-	-
zeroRates	list or np.ndarray	-	-
freqType	FinFrequencyTypes	-	CONTINUOUS
dayCountType	FinDayCountTypes	-	ACT_ACT_ISDA

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: (FinDate, list)):
```

The function arguments are described in the following table.

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

3.7 FinDiscountCurvePWL

Class: FinDiscountCurvePWL(FinDiscountCurve)

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.

FinDiscountCurvePWL

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

```
FinDiscountCurvePWL(valuationDate: FinDate,
                    zeroDates: list,
                    zeroRates: (list, np.ndarray),
                    freqType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS,
                    dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_ACT_ISDA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
zeroDates	list	-	-
zeroRates	list or np.ndarray	-	-
freqType	FinFrequencyTypes	-	CONTINUOUS
dayCountType	FinDayCountTypes	-	ACT_ACT_ISDA

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: (FinDate, list)):
```

The function arguments are described in the following table.

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

3.8 FinDiscountCurveZeros

Class: FinDiscountCurveZeros(FinDiscountCurve)

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.

FinDiscountCurveZeros

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.

```
FinDiscountCurveZeros(valuationDate: FinDate,
                      zeroDates: list,
                      zeroRates: (list, np.ndarray),
                      freqType: FinFrequencyTypes = FinFrequencyTypes.ANNUAL,
                      dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_ACT_ISDA,
                      interpType: FinInterpTypes = FinInterpTypes.FLAT_FWD_RATES):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
zeroDates	list	-	-
zeroRates	list or np.ndarray	-	-
freqType	FinFrequencyTypes	-	ANNUAL
dayCountType	FinDayCountTypes	-	ACT_ACT_ISDA
interpType	FinInterpTypes	-	FLAT_FWD_RATES

3.9 FinInterpolator

Enumerated Type: FinInterpTypes

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: FinInterpolator()

class FinInterpolator():

FinInterpolator

PLEASE ADD A FUNCTION DESCRIPTION

```
FinInterpolator(interpolatorType: FinInterpTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
interpolatorType	FinInterpTypes	-	-

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 FinInterpTypes. 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 FinInterpTypes. 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	-

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:

FinEquityVolCurve

Equity volatility as a function of option strike. This is usually a skew shape.

FinFXVolSurface

FX volatility as a function of option expiry and strike. This class constructs the surface from the ATM volatility and 25 delta strangles and risk reversals and does so for multiple expiry dates.

FinIborCapFloorVol

Libor cap/floor volatility as a function of option expiry (cap/floor start date). Takes in cap (flat) volatility and boostraps 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 FinEquityVolCurve

Class: FinEquityVolCurve()

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.

FinEquityVolCurve

PLEASE ADD A FUNCTION DESCRIPTION

```
FinEquityVolCurve(curveDate,
                  expiryDate,
                  strikes,
                  volatilities,
                  polynomial=3):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
curveDate	-	-	-
expiryDate	-	-	-
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	-	-	-

calculatePDF

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

```
calculatePDF():
```

The function arguments are described in the following table.

4.2 FinFXVolSurface

Class: FinFXVolSurface()

FinFXVolSurface

PLEASE ADD A FUNCTION DESCRIPTION

```
FinFXVolSurface(valueDate,
                 spotFXRate,
                 currencyPair,
                 notionalCurrency,
                 domDiscountCurve,
                 forDiscountCurve,
                 tenors,
                 atmVols,
                 mktStrangle25DeltaVols,
                 riskReversal25DeltaVols,
                 atmMethod=FinFXATMMethod.FWD_DELTA_NEUTRAL,
                 deltaMethod=FinFXDeltaMethod.SPOT_DELTA):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	-	-
currencyPair	-	-	-
notionalCurrency	-	-	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
tenors	-	-	-
atmVols	-	-	-
mktStrangle25DeltaVols	-	-	-
riskReversal25DeltaVols	-	-	-
atmMethod	-	-	FWD_DELTA_NEUTRAL
deltaMethod	-	-	SPOT_DELTA

volFunction

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

```
volFunction(K, tenorIndex):
```

The function arguments are described in the following table.

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

buildVolSurface

PLEASE ADD A FUNCTION DESCRIPTION

```
buildVolSurface():
```

The function arguments are described in the following table.

solveForSmileStrike

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.

```
solveForSmileStrike(vanillaOption,
                    deltaTarget,
                    tenorIndex):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
vanillaOption	-	-	-
deltaTarget	-	-	-
tenorIndex	-	-	-

checkCalibration

PLEASE ADD A FUNCTION DESCRIPTION

```
checkCalibration():
```

The function arguments are described in the following table.

plotVolCurves

PLEASE ADD A FUNCTION DESCRIPTION

```
plotVolCurves():
```

The function arguments are described in the following table.

obj

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

```
obj(cvec, *args):
```

The function arguments are described in the following table.

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

deltaFit

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

The function arguments are described in the following table.

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

4.3 FinIborCapVolCurve

Class: FinIborCapVolCurve()

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.

FinIborCapVolCurve

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.

```
FinIborCapVolCurve(curveDate,  # Valuation date for cap volatility
                   capMaturityDates,  # curve date + maturity dates for caps
                   capSigmas,  # Flat cap volatility for cap maturity dates
                   dayCountType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
curveDate	-	Valuation date for cap volatility	-
capMaturityDates	-	curve date + maturity dates for caps	-
capSigmas	-	Flat cap volatility for cap maturity dates	-
dayCountType	-	-	-

generateCapletVols

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.

```
generateCapletVols():
```

The function arguments are described in the following table.

capletVol

```
capletVol(dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-

capVol

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.

```
capVol(dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-

4.4 FinIborCapVolCurveFn

Class: FinIborCapVolCurveFn()

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

FinIborCapVolCurveFn

PLEASE ADD A FUNCTION DESCRIPTION

```
FinIborCapVolCurveFn(curveDate,
                     a,
                     b,
                     c,
                     d):
```

The function arguments are described in the following table.

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

capFloorletVol

Return the caplet volatility.

```
capFloorletVol(dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-

Chapter 5

financepy.products.equity

Equity Products

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

FinEquityVanillaOption

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

FinEquityAmericanOption

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

FinEquityAsianOption

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

FinEquityBasketOption

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

FinEquityCompoundOption

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.

FinEquityDigitalOption

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.

FinEquityFixedLookbackOption

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.

FinEquityFloatLookbackOption

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. "'

FinEquityBarrierOption

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.

FinEquityRainbowOption

TBD

FinEquityVarianceSwap

TBD

Products that have not yet been implemented include:

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

5.1 FinEquityAmericanOption

Class: FinEquityAmericanOption(FinEquityOption)

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

FinEquityAmericanOption

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.

```
FinEquityAmericanOption(expiryDate: FinDate,
                        strikePrice: float,
                        optionType: FinOptionTypes,
                        numOptions: float = 1.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
strikePrice	float	-	-
optionType	FinOptionTypes	-	-
numOptions	float	-	1.0

value

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

```
value(valueDate: FinDate,
      stockPrice: (np.ndarray, float),
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model: FinModel):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	np.ndarray or float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	FinModel	-	-

5.2 FinEquityAsianOption

Enumerated Type: FinAsianOptionValuationMethods

This enumerated type has the following values:

- GEOMETRIC
- TURNBULL_WAKEMAN
- CURRAN

Class: FinEquityAsianOption()

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.

FinEquityAsianOption

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

```
FinEquityAsianOption(startAveragingDate: FinDate,
                     expiryDate: FinDate,
                     strikePrice: float,
                     optionType: FinOptionTypes,
                     numberOfObservations: int = 100):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startAveragingDate	FinDate	-	-
expiryDate	FinDate	-	-
strikePrice	float	-	-
optionType	FinOptionTypes	-	-
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 FinAsianOptionValuationMethods. 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(valueDate: FinDate,
```

```
        stockPrice: float,
        discountCurve: FinDiscountCurve,
        dividendYield: float,
        model,
        method: FinAsianOptionValuationMethods,
        accruedAverage: float = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-
method	FinAsianOptionValuationMethods	-	-
accruedAverage	float	-	None

valueMC

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.

```
valueMC(valueDate: FinDate,
        stockPrice: float,
        discountCurve: FinDiscountCurve,
        dividendYield: float,
        model,
        numPaths: int,
        seed: int,
        accruedAverage: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-
numPaths	int	-	-
seed	int	-	-
accruedAverage	float	-	-

5.3 FinEquityBarrierOption

Enumerated Type: FinEquityBarrierTypes

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: FinEquityBarrierOption(FinEquityOption)

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 FinEquityBarrierTypes.

FinEquityBarrierOption

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

```
FinEquityBarrierOption(expiryDate: FinDate,
                       strikePrice: float,
                       optionType: FinEquityBarrierTypes,
                       barrierLevel: float,
                       numObservationsPerYear: int = 252,
                       notional: float = 1.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
strikePrice	float	-	-
optionType	FinEquityBarrierTypes	-	-
barrierLevel	float	-	-
numObservationsPerYear	int	-	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(valueDate: FinDate,
```

```
stockPrice: (float, np.ndarray),
discountCurve: FinDiscountCurve,
dividendYield: float,
model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float or np.ndarray	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

valueMC

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.

```
valueMC(valueDate: FinDate,
        stockPrice: float,
        discountCurve: FinDiscountCurve,
        processType,
        modelParams,
        numAnnObs: int = 252,
        numPaths: int = 10000,
        seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
processType	-	-	-
modelParams	-	-	-
numAnnObs	int	-	252
numPaths	int	-	10000
seed	int	-	4242

5.4 FinEquityBasketOption

Class: FinEquityBasketOption()

A FinEquityBasketOption 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.

FinEquityBasketOption

Define the FinEquityBasket 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.

```
FinEquityBasketOption(expiryDate: FinDate,
                      strikePrice: float,
                      optionType: FinOptionTypes,
                      numAssets: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
strikePrice	float	-	-
optionType	FinOptionTypes	-	-
numAssets	int	-	-

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(valueDate: FinDate,
      stockPrices: np.ndarray,
      discountCurve: FinDiscountCurve,
      dividendYields: np.ndarray,
      volatilities: np.ndarray,
      correlations: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrices	np.ndarray	-	-
discountCurve	FinDiscountCurve	-	-
dividendYields	np.ndarray	-	-
volatilities	np.ndarray	-	-
correlations	np.ndarray	-	-

valueMC

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 numPaths and seed are pre-set to default values but can be overwritten.

```
valueMC(valueDate: FinDate,
        stockPrices: np.ndarray,
        discountCurve: FinDiscountCurve,
        dividendYields: np.ndarray,
        volatilities: np.ndarray,
        corrMatrix: np.ndarray,
        numPaths:int = 10000,
        seed:int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrices	np.ndarray	-	-
discountCurve	FinDiscountCurve	-	-
dividendYields	np.ndarray	-	-
volatilities	np.ndarray	-	-
corrMatrix	np.ndarray	-	-
numPaths	int	-	10000
seed	int	-	4242

5.5 FinEquityBinomialTree

Enumerated Type: FinEquityTreePayoffTypes

This enumerated type has the following values:

- FWD_CONTRACT
- VANILLA_OPTION
- DIGITAL_OPTION
- POWER_CONTRACT
- POWER_OPTION
- LOG_CONTRACT
- LOG_OPTION

Enumerated Type: FinEquityTreeExerciseTypes

This enumerated type has the following values:

- EUROPEAN
- AMERICAN

Class: FinEquityBinomialTree()

class FinEquityBinomialTree():

FinEquityBinomialTree

pass

```
FinEquityBinomialTree():
```

The function arguments are described in the following table.

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(stockPrice,
      discountCurve,
      dividendYield,
      volatility,
      numSteps,
      valueDate,
      payoff,
      expiryDate,
      payoffType,
      exerciseType,
      payoffParams):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stockPrice	-	-	-
discountCurve	-	-	-
dividendYield	-	-	-
volatility	-	-	-
numSteps	-	-	-
valueDate	-	-	-
payoff	-	-	-
expiryDate	-	-	-
payoffType	-	-	-
exerciseType	-	-	-
payoffParams	-	-	-

5.6 FinEquityChooserOption

Class: FinEquityChooserOption(FinEquityOption)

A FinEquityChooserOption 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.

FinEquityChooserOption

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

```
FinEquityChooserOption(chooseDate: FinDate,
                       callExpiryDate: FinDate,
                       putExpiryDate: FinDate,
                       callStrikePrice: float,
                       putStrikePrice: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
chooseDate	FinDate	-	-
callExpiryDate	FinDate	-	-
putExpiryDate	FinDate	-	-
callStrikePrice	float	-	-
putStrikePrice	float	-	-

value

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

```
value(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

valueMC

Value the complex chooser option Monte Carlo.

```
valueMC(valueDate: FinDate,
        stockPrice: float,
        discountCurve: FinDiscountCurve,
        dividendYield: float,
        model,
        numPaths: int = 10000,
        seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-
numPaths	int	-	10000
seed	int	-	4242

5.7 FinEquityCliquetOption

Class: FinEquityCliquetOption(FinEquityOption)

A FinEquityCliquetOption 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.

FinEquityCliquetOption

Create the FinEquityCliquetOption 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.

```
FinEquityCliquetOption(startDate: FinDate,
                       finalExpiryDate: FinDate,
                       optionType: FinOptionTypes,
                       freqType: FinFrequencyTypes,
                       dayCountType: FinDayCountTypes = FinDayCountTypes.THIRTY_E_360,
                       calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
                       busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.
                           FOLLOWING,
                       dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.
                           BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startDate	FinDate	-	-
finalExpiryDate	FinDate	-	-
optionType	FinOptionTypes	-	-
freqType	FinFrequencyTypes	-	-
dayCountType	FinDayCountTypes	-	THIRTY_E_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

value

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

```
value(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model:FinModel):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	FinModel	-	-

printFlows

numOptions = len(self._v_options)

```
printFlows():
```

The function arguments are described in the following table.

5.8 FinEquityCompoundOption

Class: FinEquityCompoundOption(FinEquityOption)

A FinEquityCompoundOption 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.

FinEquityCompoundOption

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

```
FinEquityCompoundOption(cExpiryDate: FinDate,  # Compound Option expiry date
                        cOptionType: FinOptionTypes,  # Compound option type
                        cStrikePrice: float,  # Compound option strike
                        uExpiryDate: FinDate,  # Underlying option expiry date
                        uOptionType: FinOptionTypes,  # 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	FinDate	Compound Option expiry date	-
cOptionType	FinOptionTypes	Compound option type	-
cStrikePrice	float	Compound option strike	-
uExpiryDate	FinDate	Underlying option expiry date	-
uOptionType	FinOptionTypes	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(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model,
      numSteps: int = 200):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-
numSteps	int	-	200

5.9 FinEquityDigitalOption

Enumerated Type: FinDigitalOptionTypes

This enumerated type has the following values:

- CASH_OR_NOTHING
- ASSET_OR_NOTHING

Class: FinEquityDigitalOption(FinEquityOption)

A FinEquityDigitalOption 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 FinEquityOneTouchOption class.

FinEquityDigitalOption

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.

```
FinEquityDigitalOption(expiryDate: FinDate,
                       barrierPrice: float,
                       optionType: FinOptionTypes,
                       underlyingType: FinDigitalOptionTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
barrierPrice	float	-	-
optionType	FinOptionTypes	-	-
underlyingType	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(valueDate: FinDate,
      stockPrice: (float, np.ndarray),
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float or np.ndarray	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

valueMC

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.

```
valueMC(valueDate: FinDate,
        stockPrice: float,
        discountCurve: FinDiscountCurve,
        dividendYield: float,
        model,
        numPaths: int = 10000,
        seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-
numPaths	int	-	10000
seed	int	-	4242

5.10 FinEquityFixedLookbackOption

Class: FinEquityFixedLookbackOption(FinEquityOption)

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.

FinEquityFixedLookbackOption

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

```
FinEquityFixedLookbackOption(expiryDate: FinDate,
                             optionType: FinOptionTypes,
                             strikePrice: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
optionType	FinOptionTypes	-	-
strikePrice	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(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      volatility: float,
      stockMinMax: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
volatility	float	-	-
stockMinMax	float	-	-

valueMC

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

```
valueMC(valueDate: FinDate,
        stockPrice: float,
        discountCurve: FinDiscountCurve,
        dividendYield: float,
        volatility: float,
        stockMinMax: float,
        numPaths: int = 10000,
        numStepsPerYear: int = 252,
        seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
volatility	float	-	-
stockMinMax	float	-	-
numPaths	int	-	10000
numStepsPerYear	int	-	252
seed	int	-	4242

5.11 FinEquityFloatLookbackOption

Class: FinEquityFloatLookbackOption(FinEquityOption)

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.

FinEquityFloatLookbackOption

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.

```
FinEquityFloatLookbackOption(expiryDate: FinDate,
                             optionType: FinOptionTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
optionType	FinOptionTypes	-	-

value

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

```
value(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      volatility: float,
      stockMinMax: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
volatility	float	-	-
stockMinMax	float	-	-

valueMC

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

```
valueMC(valueDate: FinDate,
        stockPrice: float,
        discountCurve: FinDiscountCurve,
        dividendYield: float,
        volatility: float,
        stockMinMax: float,
        numPaths: int = 10000,
        numStepsPerYear: int = 252,
        seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
volatility	float	-	-
stockMinMax	float	-	-
numPaths	int	-	10000
numStepsPerYear	int	-	252
seed	int	-	4242

5.12 FinEquityModelTypes

Class: FinEquityModel(object)

FinEquityModel

pass

```
FinEquityModel():
```

The function arguments are described in the following table.

Class: FinEquityModelHeston(FinEquityModel)

class FinEquityModelHeston(FinEquityModel):

FinEquityModelHeston

self._parentType = FinEquityModel

```
FinEquityModelHeston(volatility, meanReversion):
```

The function arguments are described in the following table.

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

5.13 FinEquityOneTouchOption

Enumerated Type: FinTouchOptionPayoffTypes

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

Class: FinEquityOneTouchOption(FinEquityOption)

A FinEquityOneTouchOption 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.

FinEquityOneTouchOption

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

```
FinEquityOneTouchOption(expiryDate: FinDate,
                        optionType: FinTouchOptionPayoffTypes,
                        barrierPrice: float,
                        paymentSize: float = 1.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
optionType	FinTouchOptionPayoffTypes	-	-
barrierPrice	float	-	-
paymentSize	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(valueDate: FinDate,
      stockPrice: (float, np.ndarray),
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float or np.ndarray	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

valueMC

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.

```
valueMC(valueDate: FinDate,
        stockPrice: float,
        discountCurve: FinDiscountCurve,
        dividendYield: float,
        model,
        numPaths: int = 10000,
        numStepsPerYear: int = 252,
        seed: int = 4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-
numPaths	int	-	10000
numStepsPerYear	int	-	252
seed	int	-	4242

5.14 FinEquityOption

Enumerated Type: FinEquityOptionModelTypes

This enumerated type has the following values:

- BLACKSCHOLES
- ANOTHER

Class: FinEquityOption(object)

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

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

delta

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

```
delta(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

gamma

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

```
gamma(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

vega

Calculation of option vega by perturbing vol and revaluation.

```
vega(valueDate: FinDate,
     stockPrice: float,
     discountCurve: FinDiscountCurve,
     dividendYield: float,
     model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

theta

Calculation of option theta by perturbing value date and revaluation.

```
theta(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

rho

Calculation of option rho by perturbing interest rate and revaluation.

```
rho(valueDate: FinDate,
    stockPrice: float,
    discountCurve: FinDiscountCurve,
    dividendYield: float,
    model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

5.15 FinEquityRainbowOption

Enumerated Type: FinEquityRainbowOptionTypes

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: FinEquityRainbowOption(FinEquityOption)

class FinEquityRainbowOption(FinEquityOption):

FinEquityRainbowOption

PLEASE ADD A FUNCTION DESCRIPTION

```
FinEquityRainbowOption(expiryDate: FinDate,
                       payoffType: FinEquityRainbowOptionTypes,
                       payoffParams: List[float],
                       numAssets: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
payoffType	FinEquityRainbowOptionTypes	-	-
payoffParams	List[float]	-	-
numAssets	int	-	-

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(valueDate: FinDate,
      stockPrices: np.ndarray,
      discountCurve: FinDiscountCurve,
      dividendYields: np.ndarray,
      volatilities: np.ndarray,
      corrMatrix: np.ndarray):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrices	np.ndarray	-	-
discountCurve	FinDiscountCurve	-	-
dividendYields	np.ndarray	-	-
volatilities	np.ndarray	-	-
corrMatrix	np.ndarray	-	-

valueMC

PLEASE ADD A FUNCTION DESCRIPTION

```
valueMC(valueDate,
        stockPrices,
        discountCurve,
        dividendYields,
        volatilities,
        corrMatrix,
        numPaths=10000,
        seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
stockPrices	-	-	-
discountCurve	-	-	-
dividendYields	-	-	-
volatilities	-	-	-
corrMatrix	-	-	-
numPaths	-	-	10000
seed	-	-	4242

payoffValue

PLEASE ADD A FUNCTION DESCRIPTION

```
payoffValue(s, payoffTypeValue, payoffParams):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
payoffTypeValue	-	-	-
payoffParams	-	-	-

valueMCFast

PLEASE ADD A FUNCTION DESCRIPTION

```
valueMCFast(t,
            stockPrices,
            discountCurve,
            dividendYields,
            volatilities,
            betas,
            numAssets,
            payoffType,
            payoffParams,
            numPaths=10000,
            seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
stockPrices	-	-	-
discountCurve	-	-	-
dividendYields	-	-	-
volatilities	-	-	-
betas	-	-	-
numAssets	-	-	-
payoffType	-	-	-
payoffParams	-	-	-
numPaths	-	-	10000
seed	-	-	4242

5.16 FinEquityVanillaOption

Class: FinEquityVanillaOption()

Class for managing plain vanilla European calls and puts on equities. For American calls and puts see the FinEquityAmericanOption class.

FinEquityVanillaOption

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

```
FinEquityVanillaOption(expiryDate: FinDate,
                       strikePrice: (float, np.ndarray),
                       optionType: FinOptionTypes,
                       numOptions: float = 1.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
strikePrice	float or np.ndarray	-	-
optionType	FinOptionTypes	-	-
numOptions	float	-	1.0

value

Equity Vanilla Option valuation using Black-Scholes model.

```
value(valueDate: FinDate,
      stockPrice: (np.ndarray, float),
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model: FinModel):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	np.ndarray or float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	FinModel	-	-

delta

Calculate the analytical delta of a European vanilla option.

```
delta(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	-	-	-

gamma

Calculate the analytical gamma of a European vanilla option.

```
gamma(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model:FinModel):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	FinModel	-	-

vega

Calculate the analytical vega of a European vanilla option.

```
vega(valueDate: FinDate,
     stockPrice: float,
     discountCurve: FinDiscountCurve,
     dividendYield: float,
     model:FinModel):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	FinModel	-	-

theta

Calculate the analytical theta of a European vanilla option.

```
theta(valueDate: FinDate,
      stockPrice: float,
      discountCurve: FinDiscountCurve,
      dividendYield: float,
      model:FinModel):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	FinModel	-	-

rho

Calculate the analytical rho of a European vanilla option.

```
rho(valueDate: FinDate,
    stockPrice: float,
    discountCurve: FinDiscountCurve,
    dividendYield: float,
    model:FinModel):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	FinModel	-	-

impliedVolatility

Calculate the implied volatility of a European vanilla option.

```
impliedVolatility(valueDate: FinDate,
                  stockPrice: (float, list, np.ndarray),
                  discountCurve: FinDiscountCurve,
                  dividendYield: float,
                  price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float or list,np.ndarray	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
price	-	-	-

valueMC

Value European style call or put option using Monte Carlo. This is mainly for educational purposes. Sobol numbers can be used.

```
valueMC(valueDate: FinDate,
        stockPrice: float,
        discountCurve: FinDiscountCurve,
        dividendYield: float,
        model:FinModel,
        numPaths: int = 10000,
        seed: int = 4242,
        useSobol: bool = False):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
dividendYield	float	-	-
model	FinModel	-	-
numPaths	int	-	10000
seed	int	-	4242
useSobol	bool	-	False

5.17 FinEquityVarianceSwap

Class: FinEquityVarianceSwap(object)

FinEquityVarianceSwap

Create variance swap contract.

```
FinEquityVarianceSwap(startDate: FinDate,
                      maturityDateOrTenor: (FinDate, str),
                      strikeVariance: float,
                      notional: float = ONE_MILLION,
                      payStrikeFlag: bool = True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startDate	FinDate	-	-
maturityDateOrTenor	FinDate or str	-	-
strikeVariance	float	-	-
notional	float	-	ONE_MILLION
payStrikeFlag	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(valuationDate,
      realisedVar,
      fairStrikeVar,
      liborCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
realisedVar	-	-	-
fairStrikeVar	-	-	-
liborCurve	-	-	-

fairStrikeApprox

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.

```
fairStrikeApprox(valuationDate,
                 fwdStockPrice,
                 strikes,
                 volatilities):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
fwdStockPrice	-	-	-
strikes	-	-	-
volatilities	-	-	-

fairStrike

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.

```
fairStrike(valuationDate,
           stockPrice,
           dividendYield,
           volatilityCurve,
           numCallOptions,
           numPutOptions,
           strikeSpacing,
           discountCurve,
           useForward=True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
stockPrice	-	-	-
dividendYield	-	-	-
volatilityCurve	-	-	-
numCallOptions	-	-	-
numPutOptions	-	-	-
strikeSpacing	-	-	-
discountCurve	-	-	-
useForward	-	-	True

f

PLEASE ADD A FUNCTION DESCRIPTION

```
f(x): return (2.0/tmat)*((x-sstar)/sstar-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-log(x/sstar))	-	-	-

realisedVariance

Calculate the realised variance according to market standard calculations which can either use log or percentage returns.

```
realisedVariance(closePrices, useLogs=True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
closePrices	-	-	-
useLogs	-	-	True

printWeights

Print the list of puts and calls used to replicate the static replication component of the variance swap hedge.

```
printWeights():
```

The function arguments are described in the following table.

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:

- FinCDS is a credit default swap contract. It includes schedule generation, contract valuation and risk-management functionality.
- FinCDSBasket is a credit default basket such as a first-to-default basket. The class includes valuation according to the Gaussian copula.
- FinCDSIndexOption is an option on an index of CDS such as CDX or iTraxx. A full valuation model is included.
- FinCDSOption 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.
- FinCDSTranche is a synthetic CDO tranche. This is a financial derivative which takes a loss if the total loss on the portfolio exceeds a lower threshold K1 and which is wiped out if it exceeds a higher threshold K2. The value depends on the default correlation between the assets in the portfolio of credits. This also includes a valuation model based on the Gaussian copula model.

FinCDSCurve

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

6.1 FinCDS

Class: FinCDS(object)

A class which manages a Credit Default Swap. It performs schedule generation and the valuation and risk management of CDS.

FinCDS

Create a CDS from the step-in date, maturity date and coupon

```
FinCDS(stepInDate: FinDate,  # Date protection starts
       maturityDateOrTenor: (FinDate, str),  # FinDate or tenor
       runningCoupon: float,  # Annualised coupon on premium fee leg
       notional: float = ONE_MILLION,
       longProtection: bool = True,
       freqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
       dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360,
       calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
       busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
       dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stepInDate	FinDate	Date protection starts	-
maturityDateOrTenor	FinDate or str	FinDate or tenor	-
runningCoupon	float	Annualised coupon on premium fee leg	-
notional	float	-	ONE_MILLION
longProtection	bool	-	True
freqType	FinFrequencyTypes	-	QUARTERLY
dayCountType	FinDayCountTypes	-	ACT_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

value

Valuation of a CDS contract on a specific valuation date given an issuer curve and a contract recovery rate.

```
value(valuationDate,
      issuerCurve,
      contractRecovery=standardRecovery,
      pv01Method=0,
      prot_method=0,
      numStepsPerYear=25):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurve	-	-	-
contractRecovery	-	-	standardRecovery
pv01Method	-	-	0
prot_method	-	-	0
numStepsPerYear	-	-	25

creditDV01

Calculation of the change in the value of the CDS contract for a one basis point change in the level of the CDS curve.

```
creditDV01(valuationDate,
           issuerCurve,
           contractRecovery=standardRecovery,
           pv01Method=0,
           prot_method=0,
           numStepsPerYear=25):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurve	-	-	-
contractRecovery	-	-	standardRecovery
pv01Method	-	-	0
prot_method	-	-	0
numStepsPerYear	-	-	25

interestDV01

Calculation of the interest DV01 based on a simple bump of the discount factors and reconstruction of the CDS curve.

```
interestDV01(valuationDate: FinDate,
             issuerCurve,
             contractRecovery=standardRecovery,
             pv01Method: int = 0,
             prot_method: int = 0,
             numStepsPerYear: int = 25):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
issuerCurve	-	-	-
contractRecovery	-	-	standardRecovery
pv01Method	int	-	0
prot_method	int	-	0
numStepsPerYear	int	-	25

cashSettlementAmount

Value of the contract on the settlement date including accrued interest.

```
cashSettlementAmount(valuationDate,
                     settlementDate,
                     issuerCurve,
                     contractRecovery=standardRecovery,
                     pv01Method=0,
                     prot_method=0,
                     numStepsPerYear=25):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
settlementDate	-	-	-
issuerCurve	-	-	-
contractRecovery	-	-	standardRecovery
pv01Method	-	-	0
prot_method	-	-	0
numStepsPerYear	-	-	25

cleanPrice

Value of the CDS contract excluding accrued interest.

```
cleanPrice(valuationDate,
           issuerCurve,
           contractRecovery=standardRecovery,
           pv01Method=0,
           prot_method=0,
           numStepsPerYear=52):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurve	-	-	-
contractRecovery	-	-	standardRecovery
pv01Method	-	-	0
prot_method	-	-	0
numStepsPerYear	-	-	52

riskyPV01_OLD

RiskyPV01 of the contract using the OLD method.

```
riskyPV01_OLD(valuationDate,
              issuerCurve,
              pv01Method=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurve	-	-	-
pv01Method	-	-	0

accruedDays

Number of days between the previous coupon and the currrent step in date.

```
accruedDays():
```

The function arguments are described in the following table.

accruedInterest

Calculate the amount of accrued interest that has accrued from the previous coupon date (PCD) to the stepInDate of the CDS contract.

```
accruedInterest():
```

The function arguments are described in the following table.

protectionLegPV

Calculates the protection leg PV of the CDS by calling into the fast NUMBA code that has been defined above.

```
protectionLegPV(valuationDate,
                issuerCurve,
                contractRecovery=standardRecovery,
                numStepsPerYear=25,
                protMethod=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurve	-	-	-
contractRecovery	-	-	standardRecovery
numStepsPerYear	-	-	25
protMethod	-	-	0

riskyPV01

The riskyPV01 is the present value of a risky one dollar paid on the premium leg of a CDS contract.

```
riskyPV01(valuationDate,
          issuerCurve,
          pv01Method=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurve	-	-	-
pv01Method	-	-	0

premiumLegPV

Value of the premium leg of a CDS.

```
premiumLegPV(valuationDate,
             issuerCurve,
             pv01Method=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurve	-	-	-
pv01Method	-	-	0

parSpread

Breakeven CDS coupon that would make the value of the CDS contract equal to zero.

```
parSpread(valuationDate,
          issuerCurve,
          contractRecovery=standardRecovery,
          numStepsPerYear=25,
          pv01Method=0,
          protMethod=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurve	-	-	-
contractRecovery	-	-	standardRecovery
numStepsPerYear	-	-	25
pv01Method	-	-	0
protMethod	-	-	0

valueFastApprox

Implementation of fast valuation of the CDS contract using an accurate approximation that avoids curve building.

```
valueFastApprox(valuationDate,
                flatContinuousInterestRate,
                flatCDSCurveSpread,
                curveRecovery=standardRecovery,
                contractRecovery=standardRecovery):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
flatContinuousInterestRate	-	-	-
flatCDSCurveSpread	-	-	-
curveRecovery	-	-	standardRecovery
contractRecovery	-	-	standardRecovery

printFlows

PLEASE ADD A FUNCTION DESCRIPTION

```
printFlows(issuerCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issuerCurve	-	-	-

6.2 FinCDSBasket

Class: FinCDSBasket(object)

FinCDSBasket

PLEASE ADD A FUNCTION DESCRIPTION

```
FinCDSBasket(stepInDate: FinDate,
             maturityDate: FinDate,
             notional: float = ONE_MILLION,
             runningCoupon: float = 0.0,
             longProtection: bool = True,
             freqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
             dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360,
             calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
             busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
             dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stepInDate	FinDate	-	-
maturityDate	FinDate	-	-
notional	float	-	ONE_MILLION
runningCoupon	float	-	0.0
longProtection	bool	-	True
freqType	FinFrequencyTypes	-	QUARTERLY
dayCountType	FinDayCountTypes	-	ACT_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

valueLegs_MC

Value the legs of the default basket using Monte Carlo. The default times are an input so this valuation is not model dependent.

```
valueLegs_MC(valuationDate,
             nToDefault,
             defaultTimes,
             issuerCurves,
             liborCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
nToDefault	-	-	-
defaultTimes	-	-	-
issuerCurves	-	-	-
liborCurve	-	-	-

valueGaussian_MC

Value the default basket using a Gaussian copula model. This depends on the issuer curves and correlation matrix.

```
valueGaussian_MC(valuationDate,
                 nToDefault,
                 issuerCurves,
                 correlationMatrix,
                 liborCurve,
                 numTrials,
                 seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
nToDefault	-	-	-
issuerCurves	-	-	-
correlationMatrix	-	-	-
liborCurve	-	-	-
numTrials	-	-	-
seed	-	-	-

valueStudentT_MC

Value the default basket using the Student-T copula.

```
valueStudentT_MC(valuationDate,
                 nToDefault,
                 issuerCurves,
                 correlationMatrix,
                 degreesOfFreedom,
                 liborCurve,
                 numTrials,
                 seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
nToDefault	-	-	-
issuerCurves	-	-	-
correlationMatrix	-	-	-
degreesOfFreedom	-	-	-
liborCurve	-	-	-
numTrials	-	-	-
seed	-	-	-

value1FGaussian_Homo

Value default basket using 1 factor Gaussian copula and analytical approach which is only exact when all recovery rates are the same.

```
value1FGaussian_Homo(valuationDate,
                     nToDefault,
                     issuerCurves,
                     betaVector,
                     liborCurve,
                     numPoints=50):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
nToDefault	-	-	-
issuerCurves	-	-	-
betaVector	-	-	-
liborCurve	-	-	-
numPoints	-	-	50

6.3 FinCDSCurve

Class: FinCDSCurve()

Generate a survival probability curve implied by the value of CDS contracts given a Ibor curve and an assumed recovery rate. A scheme for the interpolation of the survival probabilities is also required.

FinCDSCurve

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.

```
FinCDSCurve(valuationDate: FinDate,
            cdsContracts: list,
            liborCurve,
            recoveryRate: float = 0.40,
            useCache: bool = False,
            interpolationMethod: FinInterpTypes = FinInterpTypes.FLAT_FWD_RATES):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
cdsContracts	list	-	-
liborCurve	-	-	-
recoveryRate	float	-	0.40
useCache	bool	-	False
interpolationMethod	FinInterpTypes	-	FLAT_FWD_RATES

survProb

Extract the survival probability to date dt. This function supports vectorisation.

```
survProb(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	-	-	-

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	-	-	-

fwdRate

Calculate the forward rate according between dates date1 and date2 according to the specified day count convention.

```
fwdRate(date1, date2, dayCountType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
date1	-	-	-
date2	-	-	-
dayCountType	-	-	-

zeroRate

Calculate the zero rate to date dt in the chosen compounding frequency where -1 is continuous is the default.

```
zeroRate(dt,
         freqType=FinFrequencyTypes.CONTINUOUS):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	-	-	-
freqType	-	-	CONTINUOUS

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 FinCDSIndexOption

Class: FinCDSIndexOption(object)

Class to manage the pricing and risk management of an option to enter into a CDS index. Different pricing algorithms are presented.

FinCDSIndexOption

Initialisation of the class object. Note that a large number of the

```
FinCDSIndexOption(expiryDate: FinDate,
                  maturityDate: FinDate,
                  indexCoupon: float,
                  strikeCoupon: float,
                  notional: float = ONE_MILLION,
                  longProtection: bool = True,
                  freqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
                  dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360,
                  calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
                  busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.
                      FOLLOWING,
                  dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
maturityDate	FinDate	-	-
indexCoupon	float	-	-
strikeCoupon	float	-	-
notional	float	-	ONE_MILLION
longProtection	bool	-	True
freqType	FinFrequencyTypes	-	QUARTERLY
dayCountType	FinDayCountTypes	-	ACT_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

valueAdjustedBlack

This approach uses two adjustments to Black's option pricing model to value an option on a CDS index.

```
valueAdjustedBlack(valuationDate,
                   indexCurve,
                   indexRecovery,
                   liborCurve,
                   sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
indexCurve	-	-	-
indexRecovery	-	-	-
liborCurve	-	-	-
sigma	-	-	-

valueAnderson

This function values a CDS index option following approach by Anderson (2006). This ensures that a no-arbitrage relationship between the consituent 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.

```
valueAnderson(valuationDate,
              issuerCurves,
              indexRecovery,
              sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurves	-	-	-
indexRecovery	-	-	-
sigma	-	-	-

6.5 FinCDSIndexPortfolio

Class: FinCDSIndexPortfolio()

This class manages the calculations associated with an equally weighted portfolio of CDS contracts with the same maturity date.

FinCDSIndexPortfolio

Create FinCDSIndexPortfolio object. Note that all of the inputs have a default value which reflects the CDS market standard.

```
FinCDSIndexPortfolio(freqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
                     dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360,
                     calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
                     busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.
                         FOLLOWING,
                     dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.
                         BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
freqType	FinFrequencyTypes	-	QUARTERLY
dayCountType	FinDayCountTypes	-	ACT_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

intrinsicRPV01

Calculation of the risky PV01 of the CDS porfolio by taking the average of the risky PV01s of each contract.

```
intrinsicRPV01(valuationDate,
               stepInDate,
               maturityDate,
               issuerCurves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
stepInDate	-	-	-
maturityDate	-	-	-
issuerCurves	-	-	-

intrinsicProtectionLegPV

Calculation of intrinsic protection leg value of the CDS porfolio by taking the average sum the protection legs of each contract.

```
intrinsicProtectionLegPV(valuationDate,
                         stepInDate,
                         maturityDate,
                         issuerCurves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
stepInDate	-	-	-
maturityDate	-	-	-
issuerCurves	-	-	-

intrinsicSpread

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.

```
intrinsicSpread(valuationDate,
                stepInDate,
                maturityDate,
                issuerCurves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
stepInDate	-	-	-
maturityDate	-	-	-
issuerCurves	-	-	-

averageSpread

Calculates the average par CDS spread of the CDS portfolio.

```
averageSpread(valuationDate,
              stepInDate,
              maturityDate,
              issuerCurves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
stepInDate	-	-	-
maturityDate	-	-	-
issuerCurves	-	-	-

totalSpread

Calculates the total CDS spread of the CDS portfolio by summing over all of the issuers and adding the spread with no weights.

```
totalSpread(valuationDate,
            stepInDate,
            maturityDate,
            issuerCurves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
stepInDate	-	-	-
maturityDate	-	-	-
issuerCurves	-	-	-

minSpread

Calculates the minimum par CDS spread across all of the issuers in the CDS portfolio.

```
minSpread(valuationDate,
          stepInDate,
          maturityDate,
          issuerCurves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
stepInDate	-	-	-
maturityDate	-	-	-
issuerCurves	-	-	-

maxSpread

Calculates the maximum par CDS spread across all of the issuers in the CDS portfolio.

```
maxSpread(valuationDate,
          stepInDate,
          maturityDate,
          issuerCurves):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
stepInDate	-	-	-
maturityDate	-	-	-
issuerCurves	-	-	-

spreadAdjustIntrinsic

Adjust individual CDS curves 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

```
spreadAdjustIntrinsic(valuationDate,
                      issuerCurves,
                      indexCoupons,
                      indexUpfronts,
                      indexMaturityDates,
                      indexRecoveryRate,
                      tolerance=1e-6):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurves	-	-	-
indexCoupons	-	-	-
indexUpfronts	-	-	-
indexMaturityDates	-	-	-
indexRecoveryRate	-	-	-
tolerance	-	-	1e-6

hazardRateAdjustIntrinsic

Adjust individual CDS curves 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.

```
hazardRateAdjustIntrinsic(valuationDate,
                          issuerCurves,
                          indexCoupons,
                          indexUpfronts,
                          indexMaturityDates,
                          indexRecoveryRate,
                          tolerance=1e-6,
                          maxIterations=100):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurves	-	-	-
indexCoupons	-	-	-
indexUpfronts	-	-	-
indexMaturityDates	-	-	-
indexRecoveryRate	-	-	-
tolerance	-	-	1e-6
maxIterations	-	-	100

6.6 FinCDSOption

Class: FinCDSOption()

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.

FinCDSOption

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.

```
FinCDSOption(expiryDate: FinDate,
             maturityDate: FinDate,
             strikeCoupon: float,
             notional: float = ONE_MILLION,
             longProtection: bool = True,
             knockoutFlag: bool = True,
             freqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
             dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360,
             calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
             busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
             dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
maturityDate	FinDate	-	-
strikeCoupon	float	-	-
notional	float	-	ONE_MILLION
longProtection	bool	-	True
knockoutFlag	bool	-	True
freqType	FinFrequencyTypes	-	QUARTERLY
dayCountType	FinDayCountTypes	-	ACT_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	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(valuationDate,
      issuerCurve,
```

```
      volatility):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurve	-	-	-
volatility	-	-	-

impliedVolatility

Calculate the implied CDS option volatility from a price.

```
impliedVolatility(valuationDate,
                  issuerCurve,
                  optionValue):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurve	-	-	-
optionValue	-	-	-

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 FinCDSTranche

Enumerated Type: FinLossDistributionBuilder

This enumerated type has the following values:

- RECURSION
- ADJUSTED_BINOMIAL
- GAUSSIAN
- LHP

Class: FinCDSTranche(object)

class FinCDSTranche(object):

FinCDSTranche

PLEASE ADD A FUNCTION DESCRIPTION

```
FinCDSTranche(stepInDate: FinDate,
              maturityDate: FinDate,
              k1: float,
              k2: float,
              notional: float = ONE_MILLION,
              runningCoupon: float = 0.0,
              longProtection: bool = True,
              freqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
              dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360,
              calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
              busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
              dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stepInDate	FinDate	-	-
maturityDate	FinDate	-	-
k1	float	-	-
k2	float	-	-
notional	float	-	ONE_MILLION
runningCoupon	float	-	0.0
longProtection	bool	-	True
freqType	FinFrequencyTypes	-	QUARTERLY
dayCountType	FinDayCountTypes	-	ACT_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

valueBC

PLEASE ADD A FUNCTION DESCRIPTION

```
valueBC(valuationDate,
        issuerCurves,
        upfront,
        runningCoupon,
        corr1,
        corr2,
        numPoints=50,
        model=FinLossDistributionBuilder.RECURSION):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
issuerCurves	-	-	-
upfront	-	-	-
runningCoupon	-	-	-
corr1	-	-	-
corr2	-	-	-
numPoints	-	-	50
model	-	-	RECURSION

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:

- FinAnnuity is a stream of cashflows that is generated and can be priced.
- FinBond 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.
- FinBondCallable is a bond that has an embedded call and put option. A number of rate models pricing functions have been included to allow such bonds to be priced and risk-managed.
- FinBondFuture 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.
- FinBondMarket is a database of country-specific bond market conventions that can be referenced. These include settlement days and accrued interest conventions.
- FinBondOption 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.
- FinConvertibleBond 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.
- FinFloatingNote enables the pricing and risk-management of a bond with floating rate coupons. Discount margin calculations are provided.
- FinMortgage generates the periodic cashflows for an interest-only and a repayment mortgage.

Conventions

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

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

Bond Curves

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

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

FinBondYieldCurve

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 FinBond

Enumerated Type: FinYTMCalcType

This enumerated type has the following values:

- UK_DMO
- US_STREET
- US_TREASURY

Class: FinBond(object)

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

FinBond

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

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

The function arguments are described in the following table.

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

fullPriceFromYTM

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

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

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
ytm	float	-	-
convention	FinYTMCalcType	-	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(settlementDate: FinDate,
          y: float,
          convention: FinYTMCalcType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
y	float	-	-
convention	FinYTMCalcType	-	-

dollarDuration

Calculate the risk or dP/dy of the bond by bumping. This is also known as the DV01 in Bloomberg.

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

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
ytm	float	-	-
convention	FinYTMCalcType	-	UK_DMO

macauleyDuration

Calculate the Macauley duration of the bond on a settlement date given its yield to maturity.

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

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
ytm	float	-	-
convention	FinYTMCalcType	-	UK_DMO

modifiedDuration

Calculate the modified duration of the bondon a settlement date given its yield to maturity.

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

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
ytm	float	-	-
convention	FinYTMCalcType	-	UK_DMO

convexityFromYTM

Calculate the bond convexity from the yield to maturity. This function is vectorised with respect to the yield input.

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

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
ytm	float	-	-
convention	FinYTMCalcType	-	UK_DMO

cleanPriceFromYTM

Calculate the bond clean price from the yield to maturity. This function is vectorised with respect to the yield input.

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

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
ytm	float	-	-
convention	FinYTMCalcType	-	UK_DMO

cleanPriceFromDiscountCurve

Calculate the clean bond value using some discount curve to present-value the bond's cashflows back to the curve anchor date and not to the settlement date.

```
cleanPriceFromDiscountCurve(settlementDate: FinDate,
                            discountCurve: FinDiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-

fullPriceFromDiscountCurve

Calculate the bond price using a provided discount curve to PV the bond's cashflows to the settlement date. As such it is effectively a forward bond price if the settlement date is after the valuation date.

```
fullPriceFromDiscountCurve(settlementDate: FinDate,
                           discountCurve: FinDiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-

currentYield

Calculate the current yield of the bond which is the coupon divided by the clean price (not the full price)

```
currentYield(cleanPrice):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cleanPrice	-	-	-

yieldToMaturity

Calculate the bond's yield to maturity by solving the price yield relationship using a one-dimensional root solver.

```
yieldToMaturity(settlementDate: FinDate,
                cleanPrice: float,
                convention: FinYTMCalcType = FinYTMCalcType.US_TREASURY):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
cleanPrice	float	-	-
convention	FinYTMCalcType	-	US_TREASURY

calcAccruedInterest

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.

```
calcAccruedInterest(settlementDate: FinDate,
                    numExDividendDays: int = 0,
                    calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
numExDividendDays	int	-	0
calendarType	FinCalendarTypes	-	WEEKEND

assetSwapSpread

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.

```
assetSwapSpread(settlementDate: FinDate,
    cleanPrice: float,
    discountCurve: FinDiscountCurve,
    swapFloatDayCountConventionType=FinDayCountTypes.ACT_360,
    swapFloatFrequencyType=FinFrequencyTypes.SEMI_ANNUAL,
    swapFloatCalendarType=FinCalendarTypes.WEEKEND,
    swapFloatBusDayAdjustRuleType=FinBusDayAdjustTypes.FOLLOWING,
    swapFloatDateGenRuleType=FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
cleanPrice	float	-	-
discountCurve	FinDiscountCurve	-	-
swapFloatDayCountConventionType	-	-	ACT_360
swapFloatFrequencyType	-	-	SEMI_ANNUAL
swapFloatCalendarType	-	-	WEEKEND
swapFloatBusDayAdjustRuleType	-	-	FOLLOWING
swapFloatDateGenRuleType	-	-	BACKWARD

fullPriceFromOAS

Calculate the full price of the bond from its OAS given the bond settlement date, a discount curve and the oas as a number.

```
fullPriceFromOAS(settlementDate: FinDate,
                 discountCurve: FinDiscountCurve,
                 oas: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-
oas	float	-	-

optionAdjustedSpread

Return OAS for bullet bond given settlement date, clean bond price and the discount relative to which the spread is to be computed.

```
optionAdjustedSpread(settlementDate: FinDate,
                     cleanPrice: float,
                     discountCurve: FinDiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
cleanPrice	float	-	-
discountCurve	FinDiscountCurve	-	-

printFlows

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

```
printFlows(settlementDate: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-

cleanPriceFromSurvivalCurve

Calculate discounted present value of flows assuming default model. The survival curve treats This has not been completed.

```
cleanPriceFromSurvivalCurve(discountCurve: FinDiscountCurve,
                            survivalCurve: FinDiscountCurve,
                            recoveryRate: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
discountCurve	FinDiscountCurve	-	-
survivalCurve	FinDiscountCurve	-	-
recoveryRate	float	-	-

7.2 FinBondAnnuity

Class: FinBondAnnuity(object)

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.

FinBondAnnuity

PLEASE ADD A FUNCTION DESCRIPTION

```
FinBondAnnuity(maturityDate: FinDate,
               coupon: float,
               freqType: FinFrequencyTypes,
               calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
               busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
               dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD,
               dayCountConventionType: FinDayCountTypes = FinDayCountTypes.ACT_360,
               face: float = 100.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
maturityDate	FinDate	-	-
coupon	float	-	-
freqType	FinFrequencyTypes	-	-
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD
dayCountConventionType	FinDayCountTypes	-	ACT_360
face	float	-	100.0

cleanPriceFromDiscountCurve

Calculate the bond price using some discount curve to present-value the bond's cashflows.

```
cleanPriceFromDiscountCurve(settlementDate: FinDate,
                            discountCurve: FinDiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-

fullPriceFromDiscountCurve

Calculate the bond price using some discount curve to present-value the bond's cashflows.

```
fullPriceFromDiscountCurve(settlementDate: FinDate,
                           discountCurve: FinDiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-

calculateFlowDatesPayments

PLEASE ADD A FUNCTION DESCRIPTION

```
calculateFlowDatesPayments(settlementDate: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-

calcAccruedInterest

Calculate the amount of coupon that has accrued between the previous coupon date and the settlement date.

```
calcAccruedInterest(settlementDate: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-

printFlows

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

```
printFlows(settlementDate: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-

7.3 FinBondConvertible

Class: FinBondConvertible(object)

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.

FinBondConvertible

Create FinBondConvertible 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.

```
FinBondConvertible(maturityDate: FinDate,  # bond maturity date
                   coupon: float,  # annual coupon
                   freqType: FinFrequencyTypes,  # coupon frequency type
                   startConvertDate: FinDate,  # conversion starts on this date
                   conversionRatio: float,  # num shares per face of notional
                   callDates: List[FinDate],  # list of call dates
                   callPrices: List[float],  # list of call prices
                   putDates: List[FinDate],  # list of put dates
                   putPrices: List[float],  # list of put prices
                   accrualType: FinDayCountTypes,  # day count type for accrued
                   faceAmount: float = 100.0):  # face amount
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
maturityDate	FinDate	bond maturity date	-
coupon	float	annual coupon	-
freqType	FinFrequencyTypes	coupon frequency type	-
startConvertDate	FinDate	conversion starts on this date	-
conversionRatio	float	num shares per face of notional	-
callDates	List[FinDate]	list of call dates	-
callPrices	List[float]	list of call prices	-
putDates	List[FinDate]	list of put dates	-
putPrices	List[float]	list of put prices	-
accrualType	FinDayCountTypes	day count type for accrued	-
faceAmount	float	face amount	100.0

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(settlementDate: FinDate,
      stockPrice: float,
      stockVolatility: float,
      dividendDates: List[FinDate],
      dividendYields: List[float],
      discountCurve: FinDiscountCurve,
      creditSpread: float,
      recoveryRate: float = 0.40,
      numStepsPerYear: int = 100):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
stockPrice	float	-	-
stockVolatility	float	-	-
dividendDates	List[FinDate]	-	-
dividendYields	List[float]	-	-
discountCurve	FinDiscountCurve	-	-
creditSpread	float	-	-
recoveryRate	float	-	0.40
numStepsPerYear	int	-	100

accruedDays

Calculate number days from previous coupon date to settlement.

```
accruedDays(settlementDate: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-

calcAccruedInterest

Calculate the amount of coupon that has accrued between the previous coupon date and the settlement date.

```
calcAccruedInterest(settlementDate: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-

currentYield

Calculate the current yield of the bond which is the coupon divided by the clean price (not the full price)

```
currentYield(cleanPrice: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cleanPrice	float	-	-

printTree

n1, n2 = array.shape

```
printTree(array):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
array	-	-	-

7.4 FinBondEmbeddedOption

Enumerated Type: FinBondModelTypes

This enumerated type has the following values:

- BLACK
- HO_LEE
- HULL_WHITE
- BLACK_KARASINSKI

Enumerated Type: FinBondOptionTypes

This enumerated type has the following values:

- EUROPEAN_CALL
- EUROPEAN_PUT
- AMERICAN_CALL
- AMERICAN_PUT

Class: FinBondEmbeddedOption(object)

FinBondEmbeddedOption

Create a FinBondEmbeddedOption object with a maturity date, coupon and all of the bond inputs.

```
FinBondEmbeddedOption(issueDate: FinDate,
                      maturityDate: FinDate,  # FinDate
                      coupon: float,  # Annualised coupon - 0.03 = 3.00%
                      freqType: FinFrequencyTypes,
                      accrualType: FinDayCountTypes,
                      callDates: List[FinDate],
                      callPrices: List[float],
                      putDates: List[FinDate],
                      putPrices: List[float],
                      faceAmount: float = 100.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issueDate	FinDate	-	-
maturityDate	FinDate	FinDate	-
coupon	float	Annualised coupon - 0.03 = 3.00%	-
freqType	FinFrequencyTypes	-	-
accrualType	FinDayCountTypes	-	-
callDates	List[FinDate]	-	-
callPrices	List[float]	-	-
putDates	List[FinDate]	-	-
putPrices	List[float]	-	-
faceAmount	float	-	100.0

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(settlementDate: FinDate,
      discountCurve: FinDiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-
model	-	-	-

7.5 FinBondFRN

Class: FinBondFRN(object)

Class for managing floating rate notes that pay a floating index plus a quoted margin.

FinBondFRN

Create FinFloatingRateNote object given its maturity date, its quoted margin, coupon frequency, accrual type. Face is the size of the position and par is the notional on which price is quoted.

```
FinBondFRN(issueDate: FinDate,
           maturityDate: FinDate,
           quotedMargin: float,    # Fixed spread paid on top of index
           freqType: FinFrequencyTypes,
           accrualType: FinDayCountTypes,
           faceAmount: float = 100.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issueDate	FinDate	-	-
maturityDate	FinDate	-	-
quotedMargin	float	Fixed spread paid on top of index	-
freqType	FinFrequencyTypes	-	-
accrualType	FinDayCountTypes	-	-
faceAmount	float	-	100.0

fullPriceFromDM

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.

```
fullPriceFromDM(settlementDate: FinDate,
                resetIbor: float,  # Next Ibor payment on NCD
                currentIbor: float,  # Ibor discount to NCD
                futureIbor: float,  # Future constant Ibor rates
                dm: float):   # Discount margin
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	Next Ibor payment on NCD	-
currentIbor	float	Ibor discount to NCD	-
futureIbor	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(settlementDate: FinDate,
          resetIbor: float,
          currentIbor: float,
          futureIbor: float,
          dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	-	-
currentIbor	float	-	-
futureIbor	float	-	-
dm	float	-	-

dollarDuration

Calculate the risk or dP/dy of the bond by bumping. This is also known as the DV01 in Bloomberg.

```
dollarDuration(settlementDate: FinDate,
               resetIbor: float,
               currentIbor: float,
               futureIbor: float,
               dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	-	-
currentIbor	float	-	-
futureIbor	float	-	-
dm	float	-	-

dollarCreditDuration

Calculate the risk or dP/dy of the bond by bumping.

```
dollarCreditDuration(settlementDate: FinDate,
                     resetIbor: float,
                     currentIbor: float,
                     futureIbor: float,
                     dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	-	-
currentIbor	float	-	-
futureIbor	float	-	-
dm	float	-	-

macauleyRateDuration

Calculate the Macauley duration of the FRN on a settlement date given its yield to maturity.

```
macauleyRateDuration(settlementDate: FinDate,
                     resetIbor: float,
                     currentIbor: float,
                     futureIbor: float,
                     dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	-	-
currentIbor	float	-	-
futureIbor	float	-	-
dm	float	-	-

modifiedRateDuration

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.

```
modifiedRateDuration(settlementDate: FinDate,
                     resetIbor: float,
                     currentIbor: float,
                     futureIbor: float,
                     dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	-	-
currentIbor	float	-	-
futureIbor	float	-	-
dm	float	-	-

modifiedCreditDuration

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.

```
modifiedCreditDuration(settlementDate: FinDate,
                       resetIbor: float,
                       currentIbor: float,
                       futureIbor: float,
                       dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	-	-
currentIbor	float	-	-
futureIbor	float	-	-
dm	float	-	-

convexityFromDM

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.

```
convexityFromDM(settlementDate: FinDate,
                resetIbor: float,
                currentIbor: float,
                futureIbor: float,
                dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	-	-
currentIbor	float	-	-
futureIbor	float	-	-
dm	float	-	-

cleanPriceFromDM

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.*

```
cleanPriceFromDM(settlementDate: FinDate,
                 resetIbor: float,
                 currentIbor: float,
                 futureIbor: float,
                 dm: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	-	-
currentIbor	float	-	-
futureIbor	float	-	-
dm	float	-	-

discountMargin

Calculate the bond's yield to maturity by solving the price yield relationship using a one-dimensional root solver.

```
discountMargin(settlementDate: FinDate,
               resetIbor: float,
               currentIbor: float,
               futureIbor: float,
               cleanPrice: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	-	-
currentIbor	float	-	-
futureIbor	float	-	-
cleanPrice	float	-	-

calcAccruedInterest

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.

```
calcAccruedInterest(settlementDate: FinDate,
                    resetIbor: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
resetIbor	float	-	-

printFlows

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

```
printFlows(settlementDate: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-

7.6 FinBondFuture

Class: FinBondFuture(object)

Class for managing futures contracts on government bonds that follows CME conventions and related analytics.

FinBondFuture

PLEASE ADD A FUNCTION DESCRIPTION

```
FinBondFuture(tickerName: str,
              firstDeliveryDate: FinDate,
              lastDeliveryDate: FinDate,
              contractSize: int,
              coupon: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
tickerName	str	-	-
firstDeliveryDate	FinDate	-	-
lastDeliveryDate	FinDate	-	-
contractSize	int	-	-
coupon	float	-	-

conversionFactor

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.

```
conversionFactor(bond: FinBond):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bond	FinBond	-	-

principalInvoicePrice

```
principalInvoicePrice(bond: FinBond,
                      futuresPrice: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bond	FinBond	-	-
futuresPrice	float	-	-

totalInvoiceAmount

' The total invoice amount paid to take delivery of bond. '

```
totalInvoiceAmount(settlementDate: FinDate,
                   bond: FinBond,
                   futuresPrice: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
bond	FinBond	-	-
futuresPrice	float	-	-

cheapestToDeliver

Determination of CTD as deliverable bond with lowest cost to buy versus what is received when the bond is delivered.

```
cheapestToDeliver(bonds: list,
                  bondCleanPrices: list,
                  futuresPrice: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bonds	list	-	-
bondCleanPrices	list	-	-
futuresPrice	float	-	-

deliveryGainLoss

Determination of what is received when the bond is delivered.

```
deliveryGainLoss(bond: FinBond,
                 bondCleanPrice: float,
                 futuresPrice: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bond	FinBond	-	-
bondCleanPrice	float	-	-
futuresPrice	float	-	-

7.7 FinBondMarket

Enumerated Type: FinBondMarkets

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

getTreasuryBondMarketConventions

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.*

```
getTreasuryBondMarketConventions(country):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
country	-	-	-

7.8 FinBondMortgage

Enumerated Type: FinBondMortgageTypes

This enumerated type has the following values:

- REPAYMENT
- INTEREST_ONLY

Class: FinBondMortgage(object)

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.

FinBondMortgage

Create the mortgage using start and end dates and principal.

```
FinBondMortgage(startDate: FinDate,
                endDate: FinDate,
                principal: float,
                freqType: FinFrequencyTypes = FinFrequencyTypes.MONTHLY,
                calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
                busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING
                    ,
                dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD,
                dayCountConventionType: FinDayCountTypes = FinDayCountTypes.ACT_360):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startDate	FinDate	-	-
endDate	FinDate	-	-
principal	float	-	-
freqType	FinFrequencyTypes	-	MONTHLY
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD
dayCountConventionType	FinDayCountTypes	-	ACT_360

repaymentAmount

Determine monthly repayment amount based on current zero rate.

```
repaymentAmount(zeroRate: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
zeroRate	float	-	-

generateFlows

Generate the bond flow amounts.

```
generateFlows(zeroRate: float,
              mortgageType: FinBondMortgageTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
zeroRate	float	-	-
mortgageType	FinBondMortgageTypes	-	-

printLeg

print("START DATE:", self._startDate)

```
printLeg():
```

The function arguments are described in the following table.

7.9 FinBondOption

Enumerated Type: FinBondModelTypes

This enumerated type has the following values:
- BLACK
- HO_LEE
- HULL_WHITE
- BLACK_KARASINSKI

Class: FinBondOption()

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.

FinBondOption

PLEASE ADD A FUNCTION DESCRIPTION

```
FinBondOption(bond: FinBond,
              expiryDate: FinDate,
              strikePrice: float,
              faceAmount: float,
              optionType: FinOptionTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
bond	FinBond	-	-
expiryDate	FinDate	-	-
strikePrice	float	-	-
faceAmount	float	-	-
optionType	FinOptionTypes	-	-

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(valuationDate: FinDate,
      discountCurve: FinDiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-
model	-	-	-

7.10 FinBondYieldCurve

Class: FinBondYieldCurve()

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.

FinBondYieldCurve

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.

```
FinBondYieldCurve(settlementDate: FinDate,
                  bonds: list,
                  ylds: (np.ndarray, list),
                  curveFit):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
bonds	list	-	-
ylds	np.ndarray or list	-	-
curveFit	-	-	-

interpolatedYield

PLEASE ADD A FUNCTION DESCRIPTION

```
interpolatedYield(maturityDate: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
maturityDate	FinDate	-	-

plot

Display yield curve.

```
plot(title):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
title	-	-	-

7.11 FinBondYieldCurveModel

Class: FinCurveFitMethod()

class FinCurveFitMethod():

Class: FinCurveFitPolynomial()

class FinCurveFitPolynomial():

FinCurveFitPolynomial

self._parentType = FinCurveFitMethod

```
FinCurveFitPolynomial(power=3):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
power	-	-	3

Class: FinCurveFitNelsonSiegel()

class FinCurveFitNelsonSiegel():

FinCurveFitNelsonSiegel

Fairly permissive bounds. Only tau1 is 1-100

```
FinCurveFitNelsonSiegel(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)]

Class: FinCurveFitNelsonSiegelSvensson()

class FinCurveFitNelsonSiegelSvensson():

FinCurveFitNelsonSiegelSvensson

Create object to store calibration and functional form of NSS parametric fit.

```
FinCurveFitNelsonSiegelSvensson(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)]

Class: FinCurveFitBSpline()

class FinCurveFitBSpline():

FinCurveFitBSpline

self._parentType = FinCurveFitMethod

```
FinCurveFitBSpline(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]

7.12 FinBondZeroCurve

Class: FinBondZeroCurve(FinDiscountCurve)

FinBondZeroCurve

Fit a discount curve to a set of bond yields using the type of curve specified.

```
FinBondZeroCurve(valuationDate: FinDate,
                 bonds: list,
                 cleanPrices: list,
                 interpType: FinInterpTypes = FinInterpTypes.FLAT_FWD_RATES):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
bonds	list	-	-
cleanPrices	list	-	-
interpType	FinInterpTypes	-	FLAT_FWD_RATES

zeroRate

Calculate the zero rate to maturity date.

```
zeroRate(dt: FinDate,
         frequencyType: FinFrequencyTypes = FinFrequencyTypes.CONTINUOUS):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-
frequencyType	FinFrequencyTypes	-	CONTINUOUS

df

t = inputTime(dt, self)

```
df(dt: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-

survProb

t = inputTime(dt, self)

```
survProb(dt: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-

fwd

Calculate the continuous forward rate at the forward date.

```
fwd(dt: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
dt	FinDate	-	-

fwdRate

Calculate the forward rate according to the specified day count convention.

```
fwdRate(date1: FinDate,
        date2: FinDate,
        dayCountType: FinDayCountTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
date1	FinDate	-	-
date2	FinDate	-	-
dayCountType	FinDayCountTypes	-	-

plot

Display yield curve.

```
plot(title: str):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
title	str	-	-

Chapter 8

financepy.products.funding

Funding

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

FinIborDeposit

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

FinInterestRateFuture

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

FinIborFRA

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

Swaps

FinFixedIborSwap

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

FinFixedIborSwap - IN PROGRESS

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

FinIborIborSwap - IN PROGRESS

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

FinFixedOISwap - IN PROGRESS

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

FinIborOISwap - IN PROGRESS

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

Currency Swaps

FinFixedFixedCcySwap - IN PROGRESS

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

FinIborIborCcySwap - IN PROGRESS

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

FinOIS

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

FinOISCurve

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

FinIborSingleCurve

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

Options

FinIborCapFloor

FinIborSwaption

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

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

FinIborBermudanSwaption

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

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

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

8.1 FinFixedLeg

Class: FinFixedLeg(object)

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.

FinFixedLeg

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.

```
FinFixedLeg(effectiveDate: FinDate,  # Date interest starts to accrue
            endDate: (FinDate, str),  # Date contract ends
            legType: FinSwapTypes,
            coupon: (float),
            freqType: FinFrequencyTypes,
            dayCountType: FinDayCountTypes,
            notional: float = ONE_MILLION,
            principal: float = 0.0,
            paymentLag: int = 0,
            calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
            busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
            dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effectiveDate	FinDate	Date interest starts to accrue	-
endDate	FinDate or str	Date contract ends	-
legType	FinSwapTypes	-	-
coupon	float or (float	-	-
freqType	FinFrequencyTypes	-	-
dayCountType	FinDayCountTypes	-	-
notional	float	-	ONE_MILLION
principal	float	-	0.0
paymentLag	int	-	0
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

generatePayments

 # These are generated immediately as they are for the entire

```
generatePayments():
```

The function arguments are described in the following table.

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(valuationDate: FinDate,
      discountCurve: FinDiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-

printPayments

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

```
printPayments():
```

The function arguments are described in the following table.

printValuation

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

```
printValuation():
```

The function arguments are described in the following table.

8.2 FinFloatLeg

Class: FinFloatLeg(object)

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.

FinFloatLeg

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.

```
FinFloatLeg(effectiveDate: FinDate,  # Date interest starts to accrue
            endDate: (FinDate, str),  # Date contract ends
            legType: FinSwapTypes,
            spread: (float),
            freqType: FinFrequencyTypes,
            dayCountType: FinDayCountTypes,
            notional: float = ONE_MILLION,
            principal: float = 0.0,
            paymentLag: int= 0,
            calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
            busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
            dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effectiveDate	FinDate	Date interest starts to accrue	-
endDate	FinDate or str	Date contract ends	-
legType	FinSwapTypes	-	-
spread	float or (float	-	-
freqType	FinFrequencyTypes	-	-
dayCountType	FinDayCountTypes	-	-
notional	float	-	ONE_MILLION
principal	float	-	0.0
paymentLag	int	-	0
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

generatePaymentDates

PLEASE ADD A FUNCTION DESCRIPTION

```
generatePaymentDates():
```

The function arguments are described in the following table.

value

Value the floating leg with payments from an index curve and discounting based on a supplied discount curve. The valuation date can be the today date. In this case the price of the floating leg will not be par (assuming we added on a principal repayment). This is only the case if we set the valuation date to be the swap's actual settlement date.

```
value(valuationDate: FinDate,  # This should be the settlement date
      discountCurve: FinDiscountCurve,
      indexCurve: FinDiscountCurve,
      firstFixingRate: float=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	This should be the settlement date	-
discountCurve	FinDiscountCurve	-	-
indexCurve	FinDiscountCurve	-	-
firstFixingRate	float	-	None

printPayments

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

```
printPayments():
```

The function arguments are described in the following table.

printValuation

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

```
printValuation():
```

The function arguments are described in the following table.

8.3 FinIborBermudanSwaption

Class: FinIborBermudanSwaption(object)

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.

FinIborBermudanSwaption

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

```
FinIborBermudanSwaption(settlementDate: FinDate,
                        exerciseDate: FinDate,
                        maturityDate: FinDate,
                        fixedLegType: FinSwapTypes,
                        exerciseType: FinExerciseTypes,
                        fixedCoupon: float,
                        fixedFrequencyType: FinFrequencyTypes,
                        fixedDayCountType: FinDayCountTypes,
                        notional=ONE_MILLION,
                        floatFrequencyType=FinFrequencyTypes.QUARTERLY,
                        floatDayCountType=FinDayCountTypes.THIRTY_E_360,
                        calendarType=FinCalendarTypes.WEEKEND,
                        busDayAdjustType=FinBusDayAdjustTypes.FOLLOWING,
                        dateGenRuleType=FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
exerciseDate	FinDate	-	-
maturityDate	FinDate	-	-
fixedLegType	FinSwapTypes	-	-
exerciseType	FinExerciseTypes	-	-
fixedCoupon	float	-	-
fixedFrequencyType	FinFrequencyTypes	-	-
fixedDayCountType	FinDayCountTypes	-	-
notional	-	-	ONE_MILLION
floatFrequencyType	-	-	QUARTERLY
floatDayCountType	-	-	THIRTY_E_360
calendarType	-	-	WEEKEND
busDayAdjustType	-	-	FOLLOWING
dateGenRuleType	-	-	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(valuationDate,
      discountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
discountCurve	-	-	-
model	-	-	-

8.4 FinIborCallableSwap

8.5 FinIborCapFloor

Enumerated Type: FinIborCapFloorModelTypes

This enumerated type has the following values:

- BLACK
- SHIFTED_BLACK
- SABR

Class: FinIborCapFloor()

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.

FinIborCapFloor

Initialise FinIborCapFloor object.

```
FinIborCapFloor(startDate: FinDate,
                maturityDateOrTenor: (FinDate, str),
                optionType: FinCapFloorTypes,
                strikeRate: float,
                lastFixing: Optional[float] = None,
                freqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
                dayCountType: FinDayCountTypes = FinDayCountTypes.THIRTY_E_360_ISDA,
                notional: float = ONE_MILLION,
                calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
                busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING
                    ,
                dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startDate	FinDate	-	-
maturityDateOrTenor	FinDate or str	-	-
optionType	FinCapFloorTypes	-	-
strikeRate	float	-	-
lastFixing	Optional[float]	-	None
freqType	FinFrequencyTypes	-	QUARTERLY
dayCountType	FinDayCountTypes	-	THIRTY_E_360_ISDA
notional	float	-	ONE_MILLION
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

value

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

```
value(valuationDate, liborCurve, model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
liborCurve	-	-	-
model	-	-	-

valueCapletFloorLet

Value the caplet or floorlet using a specific model.

```
valueCapletFloorLet(valuationDate,
                    capletStartDate,
                    capletEndDate,
                    liborCurve,
                    model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
capletStartDate	-	-	-
capletEndDate	-	-	-
liborCurve	-	-	-
model	-	-	-

printLeg

Prints the cap floor payment amounts.

```
printLeg():
```

The function arguments are described in the following table.

8.6 FinIborConventions

Class: FinIborConventions()

class FinIborConventions():

FinIborConventions

PLEASE ADD A FUNCTION DESCRIPTION

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

The function arguments are described in the following table.

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

8.7 FinIborDeposit

Class: FinIborDeposit(object)

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.

FinIborDeposit

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.

```
FinIborDeposit(startDate: FinDate,  #  When the interest starts to accrue
               maturityDateOrTenor: (FinDate, str),  # Repayment of interest
               depositRate: float,   # MM rate using simple interest
               dayCountType: FinDayCountTypes,  # How year fraction is calculated
               notional: float = 100.0,   # Amount borrowed
               calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,  #  Holidays
                   for maturity date
               busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.
                   MODIFIED_FOLLOWING):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startDate	FinDate	When the interest starts to accrue	-
maturityDateOrTenor	FinDate or str	Repayment of interest	-
depositRate	float	MM rate using simple interest	-
dayCountType	FinDayCountTypes	How year fraction is calculated	-
notional	float	Amount borrowed	100.0
calendarType	FinCalendarTypes	Holidays for maturity date	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	MODIFIED_FOLLOWING

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(valuationDate: FinDate,
      liborCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
liborCurve	-	-	-

printFlows

Print the date and size of the future repayment.

```
printFlows(valuationDate: FinDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-

8.8 FinIborDualCurve

Class: FinIborDualCurve(FinDiscountCurve)

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.

FinIborDualCurve

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

```
FinIborDualCurve(valuationDate: FinDate,
                 discountCurve: FinDiscountCurve,
                 iborDeposits: list,
                 iborFRAs: list,
                 iborSwaps: list,
                 interpType: FinInterpTypes = FinInterpTypes.FLAT_FWD_RATES,
                 checkRefit: bool = False):  # Set to True to test it works
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-
iborDeposits	list	-	-
iborFRAs	list	-	-
iborSwaps	list	-	-
interpType	FinInterpTypes	-	FLAT_FWD_RATES
checkRefit	bool	Set to True to test it works	False

8.9 FinIborFRA

Class: FinIborFRA(object)

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.

FinIborFRA

Create a Forward Rate Agreeement object.

```
FinIborFRA(startDate: FinDate,  # The date the FRA starts to accrue
           maturityDateOrTenor: (FinDate, str),  # End of the Ibor rate period
           fraRate: float,  # The fixed contractual FRA rate
           dayCountType: FinDayCountTypes,  # For interest period
           notional: float = 100.0,
           payFixedRate: bool = True,  # True if the FRA rate is being paid
           calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
           busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.
               MODIFIED_FOLLOWING):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
startDate	FinDate	The date the FRA starts to accrue	-
maturityDateOrTenor	FinDate or str	End of the Ibor rate period	-
fraRate	float	The fixed contractual FRA rate	-
dayCountType	FinDayCountTypes	For interest period	-
notional	float	-	100.0
payFixedRate	bool	True if the FRA rate is being paid	True
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	MODIFIED_FOLLOWING

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 cashflows.

```
value(valuationDate: FinDate,
      discountCurve: FinDiscountCurve,
      indexCurve: FinDiscountCurve = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-
indexCurve	FinDiscountCurve	-	None

maturityDf

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

```
maturityDf(indexCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
indexCurve	-	-	-

printFlows

Determine the value of the Deposit given a Ibor curve.

```
printFlows(valuationDate):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-

8.10 FinIborFuture

Class: FinIborFuture(object)

FinIborFuture

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

```
FinIborFuture(todayDate: FinDate,
              futureNumber: int,  # The number of the future after todayDate
              futureTenor: str = "3M",  # '1M', '2M', '3M'
              accrualType: FinDayCountTypes = FinDayCountTypes.ACT_360,
              contractSize: float = ONE_MILLION):
```

The function arguments are described in the following table.

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

toFRA

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

```
toFRA(futuresPrice, convexity):
```

The function arguments are described in the following table.

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

futuresRate

Calculate implied futures rate from the futures price.

```
futuresRate(futuresPrice):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
futuresPrice	-	-	-

FRARate

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

```
FRARate(futuresPrice, convexity):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
futuresPrice	-	-	-
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(valuationDate, volatility, meanReversion):
```

The function arguments are described in the following table.

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

8.11 FinIborIborSwap

Class: FinIborBasisSwap(object)

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 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 curves from which the implied index rates are extracted.

FinIborBasisSwap

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.

```
FinIborBasisSwap(effectiveDate: FinDate,  # Date interest starts to accrue
                 terminationDateOrTenor: (FinDate, str),  # Date contract ends
                 leg1Type: FinSwapTypes,
                 leg1FreqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
                 leg1DayCountType: FinDayCountTypes  = FinDayCountTypes.THIRTY_E_360,
                 leg1Spread: float = 0.0,
                 leg2FreqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
                 leg2DayCountType: FinDayCountTypes = FinDayCountTypes.THIRTY_E_360,
                 leg2Spread: float = 0.0,
                 notional: float = ONE_MILLION,
                 calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
                 busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.
                    FOLLOWING,
                 dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effectiveDate	FinDate	Date interest starts to accrue	-
terminationDateOrTenor	FinDate or str	Date contract ends	-
leg1Type	FinSwapTypes	-	-
leg1FreqType	FinFrequencyTypes	-	QUARTERLY
leg1DayCountType	FinDayCountTypes	-	THIRTY_E_360
leg1Spread	float	-	0.0
leg2FreqType	FinFrequencyTypes	-	QUARTERLY
leg2DayCountType	FinDayCountTypes	-	THIRTY_E_360
leg2Spread	float	-	0.0
notional	float	-	ONE_MILLION
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	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(valuationDate: FinDate,
      discountCurve: FinDiscountCurve,
      indexCurveLeg1: FinDiscountCurve = None,
      indexCurveLeg2: FinDiscountCurve = None,
      firstFixingRateLeg1=None,
      firstFixingRateLeg2=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-
indexCurveLeg1	FinDiscountCurve	-	None
indexCurveLeg2	FinDiscountCurve	-	None
firstFixingRateLeg1	-	-	None
firstFixingRateLeg2	-	-	None

printFlows

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

```
printFlows():
```

The function arguments are described in the following table.

8.12 FinIborLMMProducts

Class: FinIborLMMProducts()

FinIborLMMProducts

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

```
FinIborLMMProducts(settlementDate: FinDate,
                   maturityDate: FinDate,
                   floatFrequencyType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
                   floatDayCountType: FinDayCountTypes = FinDayCountTypes.THIRTY_E_360,
                   calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
                   busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.
                       FOLLOWING,
                   dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD)
                       :
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
maturityDate	FinDate	-	-
floatFrequencyType	FinFrequencyTypes	-	QUARTERLY
floatDayCountType	FinDayCountTypes	-	THIRTY_E_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

simulate1F

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

```
simulate1F(discountCurve,
           volCurve: FinIborCapVolCurve,
           numPaths: int = 1000,
           numeraireIndex: int = 0,
           useSobol: bool = True,
           seed: int = 42):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
discountCurve	-	-	-
volCurve	FinIborCapVolCurve	-	-
numPaths	int	-	1000
numeraireIndex	int	-	0
useSobol	bool	-	True
seed	int	-	42

simulateMF

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

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

The function arguments are described in the following table.

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

simulateNF

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

```
simulateNF(discountCurve,
           volCurve: FinIborCapVolCurve,
           correlationMatrix: np.ndarray,
           modelType: FinRateModelLMMModelTypes,
           numPaths: int = 1000,
           numeraireIndex: int = 0,
           useSobol: bool = True,
           seed: int = 42):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
discountCurve	-	-	-
volCurve	FinIborCapVolCurve	-	-
correlationMatrix	np.ndarray	-	-
modelType	FinRateModelLMMModelTypes	-	-
numPaths	int	-	1000
numeraireIndex	int	-	0
useSobol	bool	-	True
seed	int	-	42

valueSwaption

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

```
valueSwaption(settlementDate: FinDate,
              exerciseDate: FinDate,
              maturityDate: FinDate,
              swaptionType: FinSwapTypes,
              fixedCoupon: float,
              fixedFrequencyType: FinFrequencyTypes,
              fixedDayCountType: FinDayCountTypes,
              notional: float = ONE_MILLION,
              floatFrequencyType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
              floatDayCountType: FinDayCountTypes = FinDayCountTypes.THIRTY_E_360,
              calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
              busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
              dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
exerciseDate	FinDate	-	-
maturityDate	FinDate	-	-
swaptionType	FinSwapTypes	-	-
fixedCoupon	float	-	-
fixedFrequencyType	FinFrequencyTypes	-	-
fixedDayCountType	FinDayCountTypes	-	-
notional	float	-	ONE_MILLION
floatFrequencyType	FinFrequencyTypes	-	QUARTERLY
floatDayCountType	FinDayCountTypes	-	THIRTY_E_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

valueCapFloor

Value a cap or floor in the LMM.

```
valueCapFloor(settlementDate: FinDate,
              maturityDate: FinDate,
              capFloorType: FinCapFloorTypes,
              capFloorRate: float,
              frequencyType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
              dayCountType: FinDayCountTypes = FinDayCountTypes.ACT_360,
              notional: float = ONE_MILLION,
              calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
              busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
              dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
maturityDate	FinDate	-	-
capFloorType	FinCapFloorTypes	-	-
capFloorRate	float	-	-
frequencyType	FinFrequencyTypes	-	QUARTERLY
dayCountType	FinDayCountTypes	-	ACT_360
notional	float	-	ONE_MILLION
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

8.13 FinIborOIS

Class: FinIborOIS(object)

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 curves from which the implied index rates are extracted.

FinIborOIS

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.

```
FinIborOIS(effectiveDate: FinDate,  # Date interest starts to accrue
           terminationDateOrTenor: (FinDate, str),  # Date contract ends
           iborType: FinSwapTypes,
           iborFreqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
           iborDayCountType: FinDayCountTypes  = FinDayCountTypes.THIRTY_E_360,
           iborSpread: float = 0.0,
           oisFreqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
           oisDayCountType: FinDayCountTypes = FinDayCountTypes.THIRTY_E_360,
           oisSpread: float = 0.0,
           oisPaymentLag: int = 0,
           notional: float = ONE_MILLION,
           calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
           busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
           dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effectiveDate	FinDate	Date interest starts to accrue	-
terminationDateOrTenor	FinDate or str	Date contract ends	-
iborType	FinSwapTypes	-	-
iborFreqType	FinFrequencyTypes	-	QUARTERLY
iborDayCountType	FinDayCountTypes	-	THIRTY_E_360
iborSpread	float	-	0.0
oisFreqType	FinFrequencyTypes	-	QUARTERLY
oisDayCountType	FinDayCountTypes	-	THIRTY_E_360
oisSpread	float	-	0.0
oisPaymentLag	int	-	0
notional	float	-	ONE_MILLION
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	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(valuationDate: FinDate,
      discountCurve: FinDiscountCurve,
      indexIborCurve: FinDiscountCurve = None,
      indexOISCurve: FinDiscountCurve = None,
      firstFixingRateLeg1=None,
      firstFixingRateLeg2=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-
indexIborCurve	FinDiscountCurve	-	None
indexOISCurve	FinDiscountCurve	-	None
firstFixingRateLeg1	-	-	None
firstFixingRateLeg2	-	-	None

printFlows

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

```
printFlows():
```

The function arguments are described in the following table.

8.14 FinIborSingleCurve

Class: FinIborSingleCurve(FinDiscountCurve)

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.

FinIborSingleCurve

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

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

The function arguments are described in the following table.

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

8.15 FinIborSwap

Class: FinIborSwap(object)

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.

FinIborSwap

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.

```
FinIborSwap(effectiveDate: FinDate,  # Date interest starts to accrue
            terminationDateOrTenor: (FinDate, str),  # Date contract ends
            fixedLegType: FinSwapTypes,
            fixedCoupon: float,  # Fixed coupon (annualised)
            fixedFreqType: FinFrequencyTypes,
            fixedDayCountType: FinDayCountTypes,
            notional: float = ONE_MILLION,
            floatSpread: float = 0.0,
            floatFreqType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
            floatDayCountType: FinDayCountTypes = FinDayCountTypes.THIRTY_E_360,
            calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
            busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
            dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effectiveDate	FinDate	Date interest starts to accrue	-
terminationDateOrTenor	FinDate or str	Date contract ends	-
fixedLegType	FinSwapTypes	-	-
fixedCoupon	float	Fixed coupon (annualised)	-
fixedFreqType	FinFrequencyTypes	-	-
fixedDayCountType	FinDayCountTypes	-	-
notional	float	-	ONE_MILLION
floatSpread	float	-	0.0
floatFreqType	FinFrequencyTypes	-	QUARTERLY
floatDayCountType	FinDayCountTypes	-	THIRTY_E_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

value

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

```
value(valuationDate: FinDate,
      discountCurve: FinDiscountCurve,
      indexCurve: FinDiscountCurve=None,
      firstFixingRate=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-
indexCurve	FinDiscountCurve	-	None
firstFixingRate	-	-	None

pv01

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

```
pv01(valuationDate, discountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
discountCurve	-	-	-

swapRate

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.*

```
swapRate(valuationDate:FinDate,
         discountCurve: FinDiscountCurve,
         indexCurve: FinDiscountCurve = None,
         firstFixing: float = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-
indexCurve	FinDiscountCurve	-	None
firstFixing	float	-	None

cashSettledPV01

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 FinIborSwaption class. This method does not affect the standard valuation methods.

```
cashSettledPV01(valuationDate,
                flatSwapRate,
                frequencyType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
flatSwapRate	-	-	-
frequencyType	-	-	-

printFlows

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

```
printFlows():
```

The function arguments are described in the following table.

8.16 FinIborSwaption

Class: FinIborSwaption()

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.

FinIborSwaption

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

```
FinIborSwaption(settlementDate: FinDate,
                exerciseDate: FinDate,
                maturityDate: FinDate,
                fixedLegType: FinSwapTypes,
                fixedCoupon: float,
                fixedFrequencyType: FinFrequencyTypes,
                fixedDayCountType: FinDayCountTypes,
                notional: float = ONE_MILLION,
                floatFrequencyType: FinFrequencyTypes = FinFrequencyTypes.QUARTERLY,
                floatDayCountType: FinDayCountTypes = FinDayCountTypes.THIRTY_E_360,
                calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
                busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING
                    ,
                dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
settlementDate	FinDate	-	-
exerciseDate	FinDate	-	-
maturityDate	FinDate	-	-
fixedLegType	FinSwapTypes	-	-
fixedCoupon	float	-	-
fixedFrequencyType	FinFrequencyTypes	-	-
fixedDayCountType	FinDayCountTypes	-	-
notional	float	-	ONE_MILLION
floatFrequencyType	FinFrequencyTypes	-	QUARTERLY
floatDayCountType	FinDayCountTypes	-	THIRTY_E_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

value

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

```
value(valuationDate,
      discountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
discountCurve	-	-	-
model	-	-	-

cashSettledValue

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

```
cashSettledValue(valuationDate: FinDate,
                 discountCurve,
                 swapRate: float,
                 model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
discountCurve	-	-	-
swapRate	float	-	-
model	-	-	-

printSwapFixedLeg

PLEASE ADD A FUNCTION DESCRIPTION

```
printSwapFixedLeg():
```

The function arguments are described in the following table.

printSwapFloatLeg

PLEASE ADD A FUNCTION DESCRIPTION

```
printSwapFloatLeg():
```

The function arguments are described in the following table.

8.17 FinOIS

Enumerated Type: FinCompoundingTypes

This enumerated type has the following values:

- COMPOUNDED
- OVERNIGHT_COMPOUNDED_ANNUAL_RATE
- AVERAGED
- AVERAGED_DAILY

Class: FinOIS(object)

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 FinIborSwap 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.

FinOIS

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.

```
FinOIS(effectiveDate: FinDate,  # Date interest starts to accrue
       terminationDateOrTenor: (FinDate, str),  # Date contract ends
       fixedLegType: FinSwapTypes,
       fixedCoupon: float,  # Fixed coupon (annualised)
       fixedFreqType: FinFrequencyTypes,
       fixedDayCountType: FinDayCountTypes,
       notional: float = ONE_MILLION,
       paymentLag: int = 0, # Number of days after period payment occurs
       floatSpread: float = 0.0,
       floatFreqType: FinFrequencyTypes = FinFrequencyTypes.ANNUAL,
       floatDayCountType: FinDayCountTypes = FinDayCountTypes.THIRTY_E_360,
       calendarType: FinCalendarTypes = FinCalendarTypes.WEEKEND,
```

```
        busDayAdjustType: FinBusDayAdjustTypes = FinBusDayAdjustTypes.FOLLOWING,
        dateGenRuleType: FinDateGenRuleTypes = FinDateGenRuleTypes.BACKWARD):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effectiveDate	FinDate	Date interest starts to accrue	-
terminationDateOrTenor	FinDate or str	Date contract ends	-
fixedLegType	FinSwapTypes	-	-
fixedCoupon	float	Fixed coupon (annualised)	-
fixedFreqType	FinFrequencyTypes	-	-
fixedDayCountType	FinDayCountTypes	-	-
notional	float	-	ONE_MILLION
paymentLag	int	Number of days after period payment occurs	0
floatSpread	float	-	0.0
floatFreqType	FinFrequencyTypes	-	ANNUAL
floatDayCountType	FinDayCountTypes	-	THIRTY_E_360
calendarType	FinCalendarTypes	-	WEEKEND
busDayAdjustType	FinBusDayAdjustTypes	-	FOLLOWING
dateGenRuleType	FinDateGenRuleTypes	-	BACKWARD

value

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

```
value(valuationDate: FinDate,
      discountCurve: FinDiscountCurve,
      indexCurve: FinDiscountCurve=None,
      firstFixingRate=None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
discountCurve	FinDiscountCurve	-	-
indexCurve	FinDiscountCurve	-	None
firstFixingRate	-	-	None

pv01

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

```
pv01(valuationDate, discountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
discountCurve	-	-	-

swapRate

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.

```
swapRate(valuationDate, discountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
discountCurve	-	-	-

cashSettledPV01

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 FinIborSwaption class. This method does not affect the standard valuation methods.

```
cashSettledPV01(valuationDate,
                flatSwapRate,
                freqType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
flatSwapRate	-	-	-
freqType	-	-	-

printFlows

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

```
printFlows():
```

The function arguments are described in the following table.

8.18 FinOISCurve

Class: FinOISCurve(FinDiscountCurve)

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.

FinOISCurve

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.

```
FinOISCurve(valuationDate: FinDate,
            oisDeposits: list,
            oisFRAs: list,
            oisSwaps: list,
            interpType: FinInterpTypes = FinInterpTypes.FLAT_FWD_RATES,
            checkRefit: bool = False):  # Set to True to test it works
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	FinDate	-	-
oisDeposits	list	-	-
oisFRAs	list	-	-
oisSwaps	list	-	-
interpType	FinInterpTypes	-	FLAT_FWD_RATES
checkRefit	bool	Set to True to test it works	False

Chapter 9

financepy.products.fx

FX Derivatives

Overview

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

FX Forwards

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

FX Vanilla Option

FX Option

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

FX Barrier Options

FX Basket Options

FX Digital Options

FX Fixed Lookback Option

FX Float Lookback Option

FX Rainbow Option

FX Variance Swap

9.1 FinFXBarrierOption

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: FinFXBarrierOption(FinFXOption)

class FinFXBarrierOption(FinFXOption):

FinFXBarrierOption

Create FX Barrier option product. This is an option that cancels if the FX rate crosses a barrier during the life of the option.

```
FinFXBarrierOption(expiryDate: FinDate,
                   strikeFXRate: float,  # 1 unit of foreign in domestic
                   currencyPair: str,    # FORDOM
                   optionType: FinFXBarrierTypes,
                   barrierLevel: float,
                   numObservationsPerYear: int,
                   notional: float,
                   notionalCurrency: str):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
strikeFXRate	float	1 unit of foreign in domestic	-
currencyPair	str	FORDOM	-
optionType	FinFXBarrierTypes	-	-
barrierLevel	float	-	-
numObservationsPerYear	int	-	-
notional	float	-	-
notionalCurrency	str	-	-

value

Value FX Barrier Option using Black-Scholes model with closed-form analytical models.

```
value(valueDate,
      spotFXRate,
      domDiscountCurve,
      forDiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	-	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
model	-	-	-

valueMC

Value the FX Barrier Option using Monte Carlo.

```
valueMC(valueDate,
        spotFXRate,
        domInterestRate,
        processType,
        modelParams,
        numAnnSteps=552,
        numPaths=5000,
        seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	-	-
domInterestRate	-	-	-
processType	-	-	-
modelParams	-	-	-
numAnnSteps	-	-	552
numPaths	-	-	5000
seed	-	-	4242

9.2 FinFXBasketOption

Class: FinFXBasketOption(FinFXOption)

Class to manage FX Basket Option which is an option on a portfolio of FX rates.

FinFXBasketOption

Create FX Basket Option with expiry date, strike price, option type, number of assets and notional.

```
FinFXBasketOption(expiryDate: FinDate,
                  strikePrice: float,
                  optionType: FinOptionTypes,
                  numAssets: int,
                  notional: float = 1.0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
strikePrice	float	-	-
optionType	FinOptionTypes	-	-
numAssets	int	-	-
notional	float	-	1.0

validate

Check that there is an input for each asset in the basket.

```
validate(stockPrices,
         dividendYields,
         volatilities,
         betas):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stockPrices	-	-	-
dividendYields	-	-	-
volatilities	-	-	-
betas	-	-	-

value

Value an FX Basket Option using Black-Scholes closed-form model which takes into account mean and variance of underlying.

```
value(valueDate,
      stockPrices,
```

```
        discountCurve,
        dividendYields,
        volatilities,
        betas):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
stockPrices	-	-	-
discountCurve	-	-	-
dividendYields	-	-	-
volatilities	-	-	-
betas	-	-	-

valueMC

Value the FX Basket Option using Monte Carlo.

```
valueMC(valueDate,
        stockPrices,
        domDiscountCurve,
        forDiscountCurve,
        volatilities,
        betas,
        numPaths=10000,
        seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
stockPrices	-	-	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
volatilities	-	-	-
betas	-	-	-
numPaths	-	-	10000
seed	-	-	4242

9.3 FinFXDigitalOption

Class: FinFXDigitalOption()

class FinFXDigitalOption():

FinFXDigitalOption

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.

```
FinFXDigitalOption(expiryDate: FinDate,
                   strikePrice: float,  # 1 unit of foreign in domestic
                   currencyPair: str,  # FORDOM
                   optionType: FinOptionTypes,
                   notional: float,
                   premCurrency: str):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
strikePrice	float	1 unit of foreign in domestic	-
currencyPair	str	FORDOM	-
optionType	FinOptionTypes	-	-
notional	float	-	-
premCurrency	str	-	-

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(valueDate,
      spotFXRate,  # 1 unit of foreign in domestic
      domDiscountCurve,
      forDiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	1 unit of foreign in domestic	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
model	-	-	-

9.4 FinFXFixedLookbackOption

Class: FinFXFixedLookbackOption()

FinFXFixedLookbackOption

Create option with expiry date, option type and the option strike

```
FinFXFixedLookbackOption(expiryDate: FinDate,
                         optionType: FinOptionTypes,
                         optionStrike: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
optionType	FinOptionTypes	-	-
optionStrike	float	-	-

value

Value FX Fixed Lookback Option using Black Scholes model and analytical formulae.

```
value(valueDate: FinDate,
      stockPrice: float,
      domesticCurve: FinDiscountCurve,
      foreignCurve: FinDiscountCurve,
      volatility: float,
      stockMinMax: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
domesticCurve	FinDiscountCurve	-	-
foreignCurve	FinDiscountCurve	-	-
volatility	float	-	-
stockMinMax	float	-	-

valueMC

Value FX Fixed Lookback option using Monte Carlo.

```
valueMC(valueDate: FinDate,
        spotFXRate: float,  # FORDOM
        domesticCurve: FinDiscountCurve,
        foreignCurve: FinDiscountCurve,
        volatility: float,
```

```
        spotFXRateMinMax: float,
        numPaths:int = 10000,
        numStepsPerYear: int =252,
        seed: int =4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
spotFXRate	float	FORDOM	-
domesticCurve	FinDiscountCurve	-	-
foreignCurve	FinDiscountCurve	-	-
volatility	float	-	-
spotFXRateMinMax	float	-	-
numPaths	int	-	10000
numStepsPerYear	int	-	252
seed	int	-	4242

9.5 FinFXFloatLookbackOption

Class: FinFXFloatLookbackOption(FinFXOption)

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.

FinFXFloatLookbackOption

Create the FloatLookbackOption by specifying the expiry date and the option type.

```
FinFXFloatLookbackOption(expiryDate: FinDate,
                         optionType: FinOptionTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
optionType	FinOptionTypes	-	-

value

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

```
value(valueDate: FinDate,
      stockPrice: float,
      domesticCurve: FinDiscountCurve,
      foreignCurve: FinDiscountCurve,
      volatility: float,
      stockMinMax: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	FinDate	-	-
stockPrice	float	-	-
domesticCurve	FinDiscountCurve	-	-
foreignCurve	FinDiscountCurve	-	-
volatility	float	-	-
stockMinMax	float	-	-

valueMC

PLEASE ADD A FUNCTION DESCRIPTION

```
valueMC(valueDate,
    stockPrice,
    domesticCurve,
```

```
    foreignCurve,
    volatility,
    stockMinMax,
    numPaths=10000,
    numStepsPerYear=252,
    seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
stockPrice	-	-	-
domesticCurve	-	-	-
foreignCurve	-	-	-
volatility	-	-	-
stockMinMax	-	-	-
numPaths	-	-	10000
numStepsPerYear	-	-	252
seed	-	-	4242

9.6 FinFXForward

Class: FinFXForward()

FinFXForward

Creates a FinFXForward which allows the owner to buy the FOR against the DOM currency at the strikeFXRate and to pay it in the notional currency.

```
FinFXForward(expiryDate: FinDate,
             strikeFXRate: float,  # PRICE OF 1 UNIT OF FOREIGN IN DOM CCY
             currencyPair: str,  # FORDOM
             notional: float,
             notionalCurrency: str,  # must be FOR or DOM
             spotDays: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
strikeFXRate	float	PRICE OF 1 UNIT OF FOREIGN IN DOM CCY	-
currencyPair	str	FORDOM	-
notional	float	-	-
notionalCurrency	str	must be FOR or DOM	-
spotDays	int	-	0

value

Calculate the value of an FX forward contract where the current FX rate is the spotFXRate.

```
value(valueDate,
      spotFXRate,  # 1 unit of foreign in domestic
      domDiscountCurve,
      forDiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	1 unit of foreign in domestic	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-

forward

Calculate the FX Forward rate that makes the value of the FX contract equal to zero.

```
forward(valueDate,
        spotFXRate,  # 1 unit of foreign in domestic
```

```
        domDiscountCurve,
        forDiscountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	1 unit of foreign in domestic	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-

9.7 FinFXMktConventions

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 FinFXModelTypes

Class: FinFXModel(object)

class FinFXModel(object):

FinFXModel

pass

```
FinFXModel():
```

The function arguments are described in the following table.

Class: FinFXModelBlackScholes(FinFXModel)

FinFXModelBlackScholes

Create Black Scholes FX model object which holds volatility.

```
FinFXModelBlackScholes(volatility):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	-	-	-

Class: FinFXModelHeston(FinFXModel)

FinFXModelHeston

Create Heston FX Model which takes in volatility and mean reversion.

```
FinFXModelHeston(volatility, meanReversion):
```

The function arguments are described in the following table.

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

Class: FinFXModelSABR(FinFXModel)

FinFXModelSABR

Create FX Model SABR which takes alpha, beta, rho, nu and volatility as parameters.

```
FinFXModelSABR(alpha, beta, rho, nu, volatility):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
beta	-	-	-
rho	-	-	-
nu	-	-	-
volatility	-	-	-

9.9 FinFXOption

Class: FinFXOption(object)

delta

Calculate the option delta (FX rate sensitivity) by adding on a small bump and calculating the change in the option price.

```
delta(valueDate, stockPrice, discountCurve,
      dividendYield, model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
stockPrice	-	-	-
discountCurve	-	-	-
dividendYield	-	-	-
model	-	-	-

gamma

Calculate the option gamma (delta sensitivity) by adding on a small bump and calculating the change in the option delta.

```
gamma(valueDate, stockPrice, discountCurve, dividendYield,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
stockPrice	-	-	-
discountCurve	-	-	-
dividendYield	-	-	-
model	-	-	-

vega

Calculate the option vega (volatility sensitivity) by adding on a small bump and calculating the change in the option price.

```
vega(valueDate, stockPrice, discountCurve, dividendYield, model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
stockPrice	-	-	-
discountCurve	-	-	-
dividendYield	-	-	-
model	-	-	-

theta

Calculate the option theta (calendar time sensitivity) by moving forward one day and calculating the change in the option price.

```
theta(valueDate, stockPrice, discountCurve, dividendYield, model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
stockPrice	-	-	-
discountCurve	-	-	-
dividendYield	-	-	-
model	-	-	-

rho

Calculate the option rho (interest rate sensitivity) by perturbing the discount curve and revaluing.

```
rho(valueDate, stockPrice, discountCurve, dividendYield, model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
stockPrice	-	-	-
discountCurve	-	-	-
dividendYield	-	-	-
model	-	-	-

9.10 FinFXRainbowOption

Enumerated Type: FinFXRainbowOptionTypes

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: FinRainbowOption(FinEquityOption)

class FinRainbowOption(FinEquityOption):

FinRainbowOption

PLEASE ADD A FUNCTION DESCRIPTION

```
FinRainbowOption(expiryDate: FinDate,
                 payoffType: FinFXRainbowOptionTypes,
                 payoffParams: List[float],
                 numAssets: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
payoffType	FinFXRainbowOptionTypes	-	-
payoffParams	List[float]	-	-
numAssets	int	-	-

validate

PLEASE ADD A FUNCTION DESCRIPTION

```
validate(stockPrices,
         dividendYields,
         volatilities,
         betas):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stockPrices	-	-	-
dividendYields	-	-	-
volatilities	-	-	-
betas	-	-	-

validatePayoff

PLEASE ADD A FUNCTION DESCRIPTION

```
validatePayoff(payoffType, payoffParams, numAssets):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
payoffType	-	-	-
payoffParams	-	-	-
numAssets	-	-	-

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(valueDate,
      expiryDate,
      stockPrices,
      discountCurve,
      dividendYields,
      volatilities,
      betas):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
expiryDate	-	-	-
stockPrices	-	-	-
discountCurve	-	-	-
dividendYields	-	-	-
volatilities	-	-	-
betas	-	-	-

valueMC

PLEASE ADD A FUNCTION DESCRIPTION

```
valueMC(valueDate,
        expiryDate,
        stockPrices,
        discountCurve,
        dividendYields,
        volatilities,
        betas,
        numPaths=10000,
        seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
expiryDate	-	-	-
stockPrices	-	-	-
discountCurve	-	-	-
dividendYields	-	-	-
volatilities	-	-	-
betas	-	-	-
numPaths	-	-	10000
seed	-	-	4242

payoffValue

PLEASE ADD A FUNCTION DESCRIPTION

```
payoffValue(s, payoffTypeValue, payoffParams):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	-	-	-
payoffTypeValue	-	-	-
payoffParams	-	-	-

valueMCFast

PLEASE ADD A FUNCTION DESCRIPTION

```
valueMCFast(t,
            stockPrices,
            discountCurve,
            dividendYields,
            volatilities,
            betas,
            numAssets,
            payoffType,
            payoffParams,
            numPaths=10000,
            seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
t	-	-	-
stockPrices	-	-	-
discountCurve	-	-	-
dividendYields	-	-	-
volatilities	-	-	-
betas	-	-	-
numAssets	-	-	-
payoffType	-	-	-
payoffParams	-	-	-
numPaths	-	-	10000
seed	-	-	4242

9.11 FinFXVanillaOption

Class: FinFXVanillaOption()

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.

FinFXVanillaOption

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.

```
FinFXVanillaOption(expiryDate: FinDate,
                   strikeFXRate: float,  # 1 unit of foreign in domestic
                   currencyPair: str,  # FORDOM
                   optionType: FinOptionTypes,
                   notional: float,
                   premCurrency: str,
                   spotDays: int = 0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
expiryDate	FinDate	-	-
strikeFXRate	float	1 unit of foreign in domestic	-
currencyPair	str	FORDOM	-
optionType	FinOptionTypes	-	-
notional	float	-	-
premCurrency	str	-	-
spotDays	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(valueDate,
      spotFXRate,  # 1 unit of foreign in domestic
      domDiscountCurve,
      forDiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	1 unit of foreign in domestic	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
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(valueDate,
           spotFXRate,
           ccy1DiscountCurve,
           ccy2DiscountCurve,
           model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	-	-
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(valueDate,
      spotFXRate,
      domDiscountCurve,
      forDiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	-	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
model	-	-	-

gamma

This function calculates the FX Option Gamma using the spot delta.

```
gamma(valueDate,
      spotFXRate,  # value of a unit of foreign in domestic currency
      domDiscountCurve,
      forDiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	value of a unit of foreign in domestic currency	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
model	-	-	-

vega

This function calculates the FX Option Vega using the spot delta.

```
vega(valueDate,
     spotFXRate,  # value of a unit of foreign in domestic currency
     domDiscountCurve,
     forDiscountCurve,
     model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	value of a unit of foreign in domestic currency	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
model	-	-	-

theta

This function calculates the time decay of the FX option.

```
theta(valueDate,
      spotFXRate,  # value of a unit of foreign in domestic currency
      domDiscountCurve,
      forDiscountCurve,
      model):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	value of a unit of foreign in domestic currency	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
model	-	-	-

impliedVolatility

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.

```
impliedVolatility(valueDate,
                  stockPrice,
                  discountCurve,
                  dividendYield,
                  price):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
stockPrice	-	-	-
discountCurve	-	-	-
dividendYield	-	-	-
price	-	-	-

valueMC

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.

```
valueMC(valueDate,
        spotFXRate,
        domDiscountCurve,
        forDiscountCurve,
        model,
        numPaths=10000,
        seed=4242):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
spotFXRate	-	-	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
model	-	-	-
numPaths	-	-	10000
seed	-	-	4242

f

```
f(volatility, *args):
```

The function arguments are described in the following table.

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

fvega

```
fvega(volatility, *args):
```

The function arguments are described in the following table.

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

g

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

The function arguments are described in the following table.

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

solveForStrike

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 algorith to determine the strike that matches an input volatility.

```
solveForStrike(valueDate,
               vanillaOption,
               spotFXRate,
               domDiscountCurve,
               forDiscountCurve,
               delta,
               deltaType,
               volatility):
```

The function arguments are described in the following table.

Argument Name	**Type**	**Description**	**Default Value**
valueDate	-	-	-
vanillaOption	-	-	-
spotFXRate	-	-	-
domDiscountCurve	-	-	-
forDiscountCurve	-	-	-
delta	-	-	-
deltaType	-	-	-
volatility	-	-	-

9.12 FinFXVarianceSwap

Class: FinFXVarianceSwap(object)

FinFXVarianceSwap

Create variance swap contract.

```
FinFXVarianceSwap(effectiveDate: FinDate,
                  maturityDateOrTenor: [FinDate, str],
                  strikeVariance: float,
                  notional: float = ONE_MILLION,
                  payStrikeFlag: bool = True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
effectiveDate	FinDate	-	-
maturityDateOrTenor	[FinDate,str]	-	-
strikeVariance	float	-	-
notional	float	-	ONE_MILLION
payStrikeFlag	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(valuationDate,
      realisedVar,
      fairStrikeVar,
      liborCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
realisedVar	-	-	-
fairStrikeVar	-	-	-
liborCurve	-	-	-

fairStrikeApprox

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.

```
fairStrikeApprox(valuationDate,
                 fwdStockPrice,
                 strikes,
                 volatilities):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
fwdStockPrice	-	-	-
strikes	-	-	-
volatilities	-	-	-

fairStrike

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.

```
fairStrike(valuationDate,
           stockPrice,
           dividendYield,
           volatilityCurve,
           numCallOptions,
           numPutOptions,
           strikeSpacing,
           discountCurve,
           useForward=True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valuationDate	-	-	-
stockPrice	-	-	-
dividendYield	-	-	-
volatilityCurve	-	-	-
numCallOptions	-	-	-
numPutOptions	-	-	-
strikeSpacing	-	-	-
discountCurve	-	-	-
useForward	-	-	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))	-	-	-

realisedVariance

Calculate the realised variance according to market standard calculations which can either use log or percentage returns.

```
realisedVariance(closePrices, useLogs=True):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
closePrices	-	-	-
useLogs	-	-	True

printStrikes

PLEASE ADD A FUNCTION DESCRIPTION

```
printStrikes():
```

The function arguments are described in the following table.

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:

- FinModelBlack is Black's model for pricing forward starting contracts (in the forward measure) assuming the forward is lognormally distributed.

- FinModelBlackShifted is Black's model for pricing forward starting contracts (in the forward measure) assuming the forward plus a shift is lognormally distributed. CHECK

- FinModelBachelier prices options assuming the underlying evolves according to a Gaussian (normal) process.

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

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

The following asset-specific models have been implemented:

Equity Models

- FinHestonModel
- FinHestonModelProcess
- FinProcessSimulator

Interest Rate Models

Equilibrium Rate Models

There are two main short rate models.

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

Arbitrage Free Rate Models

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

Credit Models

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

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

FX Models

10.1 FinGBMProcess

Class: FinGBMProcess()

class FinGBMProcess():

getPaths

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.

```
getPaths(numPaths: int,
         numTimeSteps: int,
         t: float,
         mu: float,
         stockPrice: float,
         volatility: float,
         seed: int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numPaths	int	-	-
numTimeSteps	int	-	-
t	float	-	-
mu	float	-	-
stockPrice	float	-	-
volatility	float	-	-
seed	int	-	-

getPathsAssets

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.

```
getPathsAssets(numAssets,
               numPaths,
               numTimeSteps,
               t,
               mus,
               stockPrices,
               volatilities,
               corrMatrix,
               seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numAssets	-	-	-
numPaths	-	-	-
numTimeSteps	-	-	-
t	-	-	-
mus	-	-	-
stockPrices	-	-	-
volatilities	-	-	-
corrMatrix	-	-	-
seed	-	-	-

getPaths

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.

```
getPaths(numPaths,
         numTimeSteps,
         t,
         mu,
         stockPrice,
         volatility,
         seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numPaths	-	-	-
numTimeSteps	-	-	-
t	-	-	-
mu	-	-	-
stockPrice	-	-	-
volatility	-	-	-
seed	-	-	-

getPathsAssets

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.

```
getPathsAssets(numAssets,
               numPaths,
               numTimeSteps,
               t,
               mus,
               stockPrices,
               volatilities,
               corrMatrix,
               seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numAssets	-	-	-
numPaths	-	-	-
numTimeSteps	-	-	-
t	-	-	-
mus	-	-	-
stockPrices	-	-	-
volatilities	-	-	-
corrMatrix	-	-	-
seed	-	-	-

getAssets

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.

```
getAssets(numAssets,
          numPaths,
          t,
          mus,
          stockPrices,
          volatilities,
          corrMatrix,
          seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numAssets	-	-	-
numPaths	-	-	-
t	-	-	-
mus	-	-	-
stockPrices	-	-	-
volatilities	-	-	-
corrMatrix	-	-	-
seed	-	-	-

10.2 FinModel

Class: FinModel()

class FinModel():

FinModel

PLEASE ADD A FUNCTION DESCRIPTION

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

The function arguments are described in the following table.

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

10.3 FinModelBachelier

Class: FinModelBachelier()

Bacheliers Model which prices call and put options in the forward measure assuming the underlying rate follows a normal process.

FinModelBachelier

Create FinModel black using parameters.

```
FinModelBachelier(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(forwardRate,    # Forward rate F
      strikeRate,     # Strike Rate K
      timeToExpiry,   # Time to Expiry (years)
      df,             # Discount Factor to expiry date
      callOrPut):     # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forwardRate	-	Forward rate F	-
strikeRate	-	Strike Rate K	-
timeToExpiry	-	Time to Expiry (years)	-
df	-	Discount Factor to expiry date	-
callOrPut	-	Call or put	-

10.4 FinModelBlack

Class: FinModelBlack()

Blacks Model which prices call and put options in the forward measure according to the Black-Scholes equation.

FinModelBlack

Create FinModel black using parameters.

```
FinModelBlack(volatility, implementation=0):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	-	-	-
implementation	-	-	0

value

Price a derivative using Black's model which values in the forward measure following a change of measure.

```
value(forwardRate,    # Forward rate F
      strikeRate,     # Strike Rate K
      timeToExpiry,   # Time to Expiry (years)
      df,             # Discount Factor to expiry date
      callOrPut):     # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forwardRate	-	Forward rate F	-
strikeRate	-	Strike Rate K	-
timeToExpiry	-	Time to Expiry (years)	-
df	-	Discount Factor to expiry date	-
callOrPut	-	Call or put	-

10.5 FinModelBlackScholes

Enumerated Type: FinModelBlackScholesTypes

This enumerated type has the following values:

- ANALYTICAL
- CRR_TREE
- BARONE_ADESI

Class: FinModelBlackScholes(FinModel)

class FinModelBlackScholes(FinModel):

FinModelBlackScholes

PLEASE ADD A FUNCTION DESCRIPTION

```
FinModelBlackScholes(volatility: float,
                     implementationType: FinModelBlackScholesTypes =
                        FinModelBlackScholesTypes.ANALYTICAL,
                     parametersDict: dict = None):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
volatility	float	-	-
implementationType	FinModelBlackScholesTypes	-	ANALYTICAL
parametersDict	dict	-	None

value

PLEASE ADD A FUNCTION DESCRIPTION

```
value(spotPrice: float,
      timeToExpiry: float,
      strikePrice: float,
      riskFreeRate: float,
      dividendRate: float,
      optionType: FinOptionTypes):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spotPrice	float	-	-
timeToExpiry	float	-	-
strikePrice	float	-	-
riskFreeRate	float	-	-
dividendRate	float	-	-
optionType	FinOptionTypes	-	-

10.6 FinModelBlackScholesAnalytical

bsValue

Price a derivative using Black-Scholes model where phi is +1 for a call, and phi is -1 for a put.

```
bsValue(s:float,
        t:float,
        k:float,
        r:float,
        q:float,
        v:float,
        phi:int):   # +1 for call, -1 for put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
s	float	-	-
t	float	-	-
k	float	-	-
r	float	-	-
q	float	-	-
v	float	-	-
phi	int	+1 for call, -1 for put	-

bawValue

American Option Pricing Approximation using the Barone-Adesi-Whaley approximation for the Black Scholes Model

```
bawValue(stockPrice: float,
         timeToExpiry: float,
         strikePrice: float,
         riskFreeRate: float,
         dividendRate: float,
         volatility: float,
         phi:int):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stockPrice	float	-	-
timeToExpiry	float	-	-
strikePrice	float	-	-
riskFreeRate	float	-	-
dividendRate	float	-	-
volatility	float	-	-
phi	int	-	-

10.7 FinModelBlackShifted

Class: FinModelBlackShifted()

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.

FinModelBlackShifted

Create FinModel black using parameters.

```
FinModelBlackShifted(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(forwardRate,    # Forward rate
      strikeRate,     # Strike Rate
      timeToExpiry,   # time to expiry in years
      df,             # Discount Factor to expiry date
      callOrPut):     # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forwardRate	-	Forward rate	-
strikeRate	-	Strike Rate	-
timeToExpiry	-	time to expiry in years	-
df	-	Discount Factor to expiry date	-
callOrPut	-	Call or put	-

10.8 FinModelCRRTree

crrTreeVal

Value an American option using a Binomial Treee

```
crrTreeVal(stockPrice,
           ccInterestRate,  # continuously compounded
           ccDividendRate,  # continuously compounded
           volatility,  # Black scholes volatility
           numStepsPerYear,
           timeToExpiry,
           optionType,
           strikePrice,
           isEven):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stockPrice	-	-	-
ccInterestRate	-	continuously compounded	-
ccDividendRate	-	continuously compounded	-
volatility	-	Black scholes volatility	-
numStepsPerYear	-	-	-
timeToExpiry	-	-	-
optionType	-	-	-
strikePrice	-	-	-
isEven	-	-	-

crrTreeValAvg

Calculate the average values off the tree using an even and an odd number of time steps.

```
crrTreeValAvg(stockPrice,
              ccInterestRate,  # continuously compounded
              ccDividendRate,  # continuously compounded
              volatility,  # Black scholes volatility
              numStepsPerYear,
              timeToExpiry,
              optionType,
              strikePrice):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
stockPrice	-	-	-
ccInterestRate	-	continuously compounded	-
ccDividendRate	-	continuously compounded	-
volatility	-	Black scholes volatility	-
numStepsPerYear	-	-	-
timeToExpiry	-	-	-
optionType	-	-	-
strikePrice	-	-	-

10.9 FinModelGaussianCopula

defaultTimesGC

Generate a matrix of default times by credit and trial using a Gaussian copula model using a full rank correlation matrix.

```
defaultTimesGC(issuerCurves,
               correlationMatrix,
               numTrials,
               seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issuerCurves	-	-	-
correlationMatrix	-	-	-
numTrials	-	-	-
seed	-	-	-

10.10 FinModelGaussianCopula1F

lossDbnRecursionGCD

Full construction of the loss distribution of a portfolio of credits where losses have been calculate as number of units based on the GCD.

```
lossDbnRecursionGCD(numCredits,
                    defaultProbs,
                    lossUnits,
                    betaVector,
                    numIntegrationSteps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numCredits	-	-	-
defaultProbs	-	-	-
lossUnits	-	-	-
betaVector	-	-	-
numIntegrationSteps	-	-	-

homogeneousBasketLossDbn

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.

```
homogeneousBasketLossDbn(survivalProbabilities,
                         recoveryRates,
                         betaVector,
                         numIntegrationSteps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
survivalProbabilities	-	-	-
recoveryRates	-	-	-
betaVector	-	-	-
numIntegrationSteps	-	-	-

trSurvProbRecursion

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.

```
trSurvProbRecursion(k1,
                    k2,
                    numCredits,
                    survivalProbabilities,
```

```
                    recoveryRates,
                    betaVector,
                    numIntegrationSteps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-
numCredits	-	-	-
survivalProbabilities	-	-	-
recoveryRates	-	-	-
betaVector	-	-	-
numIntegrationSteps	-	-	-

gaussApproxTrancheLoss

PLEASE ADD A FUNCTION DESCRIPTION

```
gaussApproxTrancheLoss(k1, k2, mu, sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-
mu	-	-	-
sigma	-	-	-

trSurvProbGaussian

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.

```
trSurvProbGaussian(k1,
                   k2,
                   numCredits,
                   survivalProbabilities,
                   recoveryRates,
                   betaVector,
                   numIntegrationSteps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-
numCredits	-	-	-
survivalProbabilities	-	-	-
recoveryRates	-	-	-
betaVector	-	-	-
numIntegrationSteps	-	-	-

lossDbnHeterogeneousAdjBinomial

Get the portfolio loss distribution using the adjusted binomial approximation to the conditional loss distribution.

```
lossDbnHeterogeneousAdjBinomial(numCredits,
                                defaultProbs,
                                lossRatio,
                                betaVector,
                                numIntegrationSteps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numCredits	-	-	-
defaultProbs	-	-	-
lossRatio	-	-	-
betaVector	-	-	-
numIntegrationSteps	-	-	-

trSurvProbAdjBinomial

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.

```
trSurvProbAdjBinomial(k1,
                      k2,
                      numCredits,
                      survivalProbabilities,
                      recoveryRates,
                      betaVector,
                      numIntegrationSteps):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-
numCredits	-	-	-
survivalProbabilities	-	-	-
recoveryRates	-	-	-
betaVector	-	-	-
numIntegrationSteps	-	-	-

10.11 FinModelGaussianCopulaLHP

trSurvProbLHP

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.

```
trSurvProbLHP(k1,
              k2,
              numCredits,
              survivalProbabilities,
              recoveryRates,
              beta):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-
numCredits	-	-	-
survivalProbabilities	-	-	-
recoveryRates	-	-	-
beta	-	-	-

portfolioCDF_LHP

PLEASE ADD A FUNCTION DESCRIPTION

```
portfolioCDF_LHP(k, numCredits, qvector, recoveryRates, beta, numPoints):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
numCredits	-	-	-
qvector	-	-	-
recoveryRates	-	-	-
beta	-	-	-
numPoints	-	-	-

expMinLK

PLEASE ADD A FUNCTION DESCRIPTION

```
expMinLK(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	-	-	-

LHPDensity

PLEASE ADD A FUNCTION DESCRIPTION

```
LHPDensity(k, p, r, beta):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
p	-	-	-
r	-	-	-
beta	-	-	-

LHPAnalyticalDensityBaseCorr

PLEASE ADD A FUNCTION DESCRIPTION

```
LHPAnalyticalDensityBaseCorr(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	-	-	-

LHPAnalyticalDensity

PLEASE ADD A FUNCTION DESCRIPTION

```
LHPAnalyticalDensity(k, p, r, beta):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k	-	-	-
p	-	-	-
r	-	-	-
beta	-	-	-

ExpMinLK

PLEASE ADD A FUNCTION DESCRIPTION

```
ExpMinLK(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	-	-	-

probLGreaterThanK

c = normpdf(P)

```
probLGreaterThanK(K, P, R, beta):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-
P	-	-	-
R	-	-	-
beta	-	-	-

10.12 FinModelHeston

Enumerated Type: FinHestonNumericalScheme

This enumerated type has the following values:

- EULER
- EULERLOG
- QUADEXP

Class: FinModelHeston()

class FinModelHeston():

FinModelHeston

PLEASE ADD A FUNCTION DESCRIPTION

```
FinModelHeston(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(valueDate,
         option,
         stockPrice,
         interestRate,
         dividendYield,
         numPaths,
         numStepsPerYear,
         seed,
         scheme=FinHestonNumericalScheme.EULERLOG):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
option	-	-	-
stockPrice	-	-	-
interestRate	-	-	-
dividendYield	-	-	-
numPaths	-	-	-
numStepsPerYear	-	-	-
seed	-	-	-
scheme	-	-	EULERLOG

value_Lewis

PLEASE ADD A FUNCTION DESCRIPTION

```
value_Lewis(valueDate,
            option,
            stockPrice,
            interestRate,
            dividendYield):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
option	-	-	-
stockPrice	-	-	-
interestRate	-	-	-
dividendYield	-	-	-

phi

*k = k_in + 0.5 * 1j*

```
phi(k_in,):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k_in	-	-	-

phi_transform

*def integrand(k): return 2.0 * np.real(np.exp(-1j **

```
phi_transform(x):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x	-	-	-

integrand

*k * x) * phi(k)) / (k**2 + 1.0 / 4.0)*

```
integrand(k): return 2.0 * np.real(np.exp(-1j * \
```

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(valueDate,
                  option,
                  stockPrice,
                  interestRate,
                  dividendYield):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
option	-	-	-
stockPrice	-	-	-
interestRate	-	-	-
dividendYield	-	-	-

f

*k = k_in + 0.5 * 1j*

```
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(valueDate,
            option,
            stockPrice,
            interestRate,
            dividendYield):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
option	-	-	-
stockPrice	-	-	-
interestRate	-	-	-
dividendYield	-	-	-

F

def integrand(u):

```
F(s, b):
```

The function arguments are described in the following table.

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

integrand

*beta = b - 1j * rho * sigma * u*

```
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(valueDate,
               option,
```

```
                stockPrice,
                interestRate,
                dividendYield):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
valueDate	-	-	-
option	-	-	-
stockPrice	-	-	-
interestRate	-	-	-
dividendYield	-	-	-

FF

def integrand(u):

```
FF(j):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
j	-	-	-

integrand

*V = sigma * sigma*

```
integrand(u):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
u	-	-	-

getPaths

PLEASE ADD A FUNCTION DESCRIPTION

```
getPaths(s0, r, q, v0, kappa, theta, sigma, rho, t, dt, numPaths,
         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	-	-	-
numPaths	-	-	-
seed	-	-	-
scheme	-	-	-

10.13 FinModelLHPlus

Class: LHPlusModel()

Large Homogenous Portfolio model with extra asset. Used for approximating full Gaussian copula.

LHPlusModel

self._P = P

```
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	-	-	-

probLossGreaterThanK

Returns P(L¿K) where L is the portfolio loss given by model.

```
probLossGreaterThanK(K):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-

expMinLKIntegral

PLEASE ADD A FUNCTION DESCRIPTION

```
expMinLKIntegral(K, dK):
```

The function arguments are described in the following table.

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

expMinLK

PLEASE ADD A FUNCTION DESCRIPTION

```
expMinLK(K):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-

expMinLK2

PLEASE ADD A FUNCTION DESCRIPTION

```
expMinLK2(K):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
K	-	-	-

trancheSurvivalProbability

PLEASE ADD A FUNCTION DESCRIPTION

```
trancheSurvivalProbability(k1, k2):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
k1	-	-	-
k2	-	-	-

10.14 FinModelLossDbnBuilder

indepLossDbnHeterogeneousAdjBinomial

PLEASE ADD A FUNCTION DESCRIPTION

```
indepLossDbnHeterogeneousAdjBinomial(numCredits,
                                     condProbs,
                                     lossRatio):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numCredits	-	-	-
condProbs	-	-	-
lossRatio	-	-	-

portfolioGCD

PLEASE ADD A FUNCTION DESCRIPTION

```
portfolioGCD(actualLosses):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
actualLosses	-	-	-

indepLossDbnRecursionGCD

PLEASE ADD A FUNCTION DESCRIPTION

```
indepLossDbnRecursionGCD(numCredits,
                         condDefaultProbs,
                         lossUnits):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numCredits	-	-	-
condDefaultProbs	-	-	-
lossUnits	-	-	-

10.15 FinModelMertonCredit

Class: FinMertonCreditModel()

class FinMertonCreditModel():

FinMertonCreditModel

PLEASE ADD A FUNCTION DESCRIPTION

```
FinMertonCreditModel(assetValue: float,
                     bondFace: float,
                     timeToMaturity: float,
                     riskFreeRate: float,
                     assetGrowthRate: float,
                     volatility: float):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
assetValue	float	-	-
bondFace	float	-	-
timeToMaturity	float	-	-
riskFreeRate	float	-	-
assetGrowthRate	float	-	-
volatility	float	-	-

leverage

Calculate the leverage.

```
leverage():
```

The function arguments are described in the following table.

equityValue

Calculate the equity value.

```
equityValue():
```

The function arguments are described in the following table.

debtValue

Calculate the debt value

```
debtValue():
```

The function arguments are described in the following table.

creditSpread

Calculate the credit spread

```
creditSpread():
```

The function arguments are described in the following table.

probDefault

Calculate the default probability.

```
probDefault():
```

The function arguments are described in the following table.

10.16 FinModelRatesBDT

Class: FinModelRatesBDT()

class FinModelRatesBDT():

FinModelRatesBDT

*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*

```
FinModelRatesBDT(sigma, numTimeSteps=100):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
numTimeSteps	-	-	100

buildTree

PLEASE ADD A FUNCTION DESCRIPTION

```
buildTree(treeMat, dfTimes, dfValues):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
treeMat	-	-	-
dfTimes	-	-	-
dfValues	-	-	-

bondOption

Value a bond option that can have European or American exercise using the Black-Derman-Toy model. The model uses a binomial tree.

```
bondOption(texp, strikePrice, faceAmount,
           couponTimes, couponFlows, exerciseType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseType	-	-	-

bermudanSwaption

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.

```
bermudanSwaption(texp, tmat, strike, faceAmount,
                 couponTimes, couponFlows, exerciseType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strike	-	-	-
faceAmount	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseType	-	-	-

callablePuttableBond_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.

```
callablePuttableBond_Tree(couponTimes, couponFlows,
                          callTimes, callPrices,
                          putTimes, putPrices,
                          faceAmount):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
couponTimes	-	-	-
couponFlows	-	-	-
callTimes	-	-	-
callPrices	-	-	-
putTimes	-	-	-
putPrices	-	-	-
faceAmount	-	-	-

optionExerciseTypesToInt

PLEASE ADD A FUNCTION DESCRIPTION

```
optionExerciseTypesToInt(optionExerciseType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
optionExerciseType	-	-	-

f

PLEASE ADD A FUNCTION DESCRIPTION

```
f(x0, m, Q, rt, dfEnd, dt, sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x0	-	-	-
m	-	-	-
Q	-	-	-
rt	-	-	-
dfEnd	-	-	-
dt	-	-	-
sigma	-	-	-

searchRoot

PLEASE ADD A FUNCTION DESCRIPTION

```
searchRoot(x0, m, Q, rt, dfEnd, dt, sigma):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
x0	-	-	-
m	-	-	-
Q	-	-	-
rt	-	-	-
dfEnd	-	-	-
dt	-	-	-
sigma	-	-	-

bermudanSwaption_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.

```
bermudanSwaption_Tree_Fast(texp, tmat,
                           strikePrice, faceAmount,
                           couponTimes, couponFlows,
                           exerciseTypeInt,
                           _dfTimes, _dfValues,
                           _treeTimes,
                           _Q, _rt, _dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseTypeInt	-	-	-
_dfTimes	-	-	-
_dfValues	-	-	-
_treeTimes	-	-	-
_Q	-	-	-
_rt	-	-	-
_dt	-	-	-

americanBondOption_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.

```
americanBondOption_Tree_Fast(texp, tmat,
                             strikePrice, faceAmount,
                             couponTimes, couponFlows,
                             exerciseTypeInt,
                             _dfTimes, _dfValues,
                             _treeTimes, _Q,
                             _rt, _dt):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseTypeInt	-	-	-
_dfTimes	-	-	-
_dfValues	-	-	-
_treeTimes	-	-	-
_Q	-	-	-
_rt	-	-	-
_dt	-	-	-

callablePuttableBond_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.

```
callablePuttableBond_Tree_Fast(couponTimes, couponFlows,
                               callTimes, callPrices,
                               putTimes, putPrices, faceAmount,
                               _sigma, _a, _Q,  # IS SIGMA USED ?
                               _pu, _pm, _pd, _rt, _dt, _treeTimes,
                               _dfTimes, _dfValues):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
couponTimes	-	-	-
couponFlows	-	-	-
callTimes	-	-	-
callPrices	-	-	-
putTimes	-	-	-
putPrices	-	-	-
faceAmount	-	-	-
_sigma	-	IS SIGMA USED ?	-
_a	-	IS SIGMA USED ?	-
_Q	-	IS SIGMA USED ?	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_treeTimes	-	-	-
_dfTimes	-	-	-
_dfValues	-	-	-

buildTreeFast

 # Unlike the BK and HW Trinomial trees, this Tree is packed into the lower

```
buildTreeFast(sigma, treeTimes, numTimeSteps, discountFactors):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
treeTimes	-	-	-
numTimeSteps	-	-	-
discountFactors	-	-	-

10.17 FinModelRatesBK

Class: FinModelRatesBK()

class FinModelRatesBK():

FinModelRatesBK

*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*

```
FinModelRatesBK(sigma: float,
                a: float,
                numTimeSteps:int=100):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	float	-	-
a	float	-	-
numTimeSteps	int	-	100

buildTree

PLEASE ADD A FUNCTION DESCRIPTION

```
buildTree(tmat, dfTimes, dfValues):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
tmat	-	-	-
dfTimes	-	-	-
dfValues	-	-	-

bondOption

Value a bond option that has European or American exercise using the Black-Karasinski model. The model uses a trinomial tree.

```
bondOption(texp, strikePrice, faceAmount,
           couponTimes, couponFlows, exerciseType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseType	-	-	-

bermudanSwaption

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.

```
bermudanSwaption(texp, tmat, strikePrice, faceAmount,
                 couponTimes, couponFlows, exerciseType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseType	-	-	-

callablePuttableBond_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.

```
callablePuttableBond_Tree(couponTimes, couponFlows,
                          callTimes, callPrices,
                          putTimes, putPrices,
                          face):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
couponTimes	-	-	-
couponFlows	-	-	-
callTimes	-	-	-
callPrices	-	-	-
putTimes	-	-	-
putPrices	-	-	-
face	-	-	-

optionExerciseTypesToInt

PLEASE ADD A FUNCTION DESCRIPTION

```
optionExerciseTypesToInt(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	-	-	-

searchRoot

PLEASE ADD A FUNCTION DESCRIPTION

```
searchRoot(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	-	-	-

searchRootDeriv

PLEASE ADD A FUNCTION DESCRIPTION

```
searchRootDeriv(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	-	-	-

bermudanSwaption_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.

```
bermudanSwaption_Tree_Fast(texp, tmat,
                           strikePrice, faceAmount,
                           couponTimes, couponFlows,
                           exerciseTypeInt,
                           _dfTimes, _dfValues,
                           _treeTimes, _Q,
                           _pu, _pm, _pd,
                           _rt, _dt, _a):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseTypeInt	-	-	-
_dfTimes	-	-	-
_dfValues	-	-	-
_treeTimes	-	-	-
_Q	-	-	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_a	-	-	-

americanBondOption_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.

```
americanBondOption_Tree_Fast(texp, tmat,
                             strikePrice, faceAmount,
                             couponTimes, couponFlows,
                             exerciseTypeInt,
                             _dfTimes, _dfValues,
                             _treeTimes, _Q,
                             _pu, _pm, _pd,
                             _rt,
                             _dt, _a):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseTypeInt	-	-	-
_dfTimes	-	-	-
_dfValues	-	-	-
_treeTimes	-	-	-
_Q	-	-	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_a	-	-	-

callablePuttableBond_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.

```
callablePuttableBond_Tree_Fast(couponTimes, couponFlows,
                               callTimes, callPrices,
                               putTimes, putPrices, faceAmount,
                               _sigma, _a, _Q,  # IS SIGMA USED ?
                               _pu, _pm, _pd, _rt, _dt, _treeTimes,
                               _dfTimes, _dfValues):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
couponTimes	-	-	-
couponFlows	-	-	-
callTimes	-	-	-
callPrices	-	-	-
putTimes	-	-	-
putPrices	-	-	-
faceAmount	-	-	-
_sigma	-	IS SIGMA USED ?	-
_a	-	IS SIGMA USED ?	-
_Q	-	IS SIGMA USED ?	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_treeTimes	-	-	-
_dfTimes	-	-	-
_dfValues	-	-	-

buildTreeFast

PLEASE ADD A FUNCTION DESCRIPTION

```
buildTreeFast(a, sigma, treeTimes, numTimeSteps, discountFactors):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
sigma	-	-	-
treeTimes	-	-	-
numTimeSteps	-	-	-
discountFactors	-	-	-

10.18 FinModelRatesCIR

Enumerated Type: FinCIRNumericalScheme

This enumerated type has the following values:

- EULER
- LOGNORMAL
- MILSTEIN
- KAHLJACKEL
- EXACT

Class: FinModelRatesCIR()

class FinModelRatesCIR():

FinModelRatesCIR

self._a = a

```
FinModelRatesCIR(a, b, sigma):
```

The function arguments are described in the following table.

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

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	-	-	-

zeroPrice

Price of a zero coupon bond in CIR model.

```
zeroPrice(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	-	-	-

ratePath_MC

Generate a path of CIR rates using a number of numerical schemes.

```
ratePath_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	-	-	-

zeroPrice_MC

```
zeroPrice_MC(r0, a, b, sigma, t, dt, numPaths, seed, scheme):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
sigma	-	-	-
t	-	-	-
dt	-	-	-
numPaths	-	-	-
seed	-	-	-
scheme	-	-	-

10.19 FinModelRatesHL

Class: FinModelRatesHL()

class FinModelRatesHL():

FinModelRatesHL

*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.*

```
FinModelRatesHL(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, discountCurve):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rt1	-	-	-
t1	-	-	-
t2	-	-	-
discountCurve	-	-	-

optionOnZCB

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.

```
optionOnZCB(texp, tmat,
            strikePrice, faceAmount,
            dfTimes, dfValues):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
dfTimes	-	-	-
dfValues	-	-	-

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	-	-	-

10.20 FinModelRatesHW

Enumerated Type: FinHWEuropeanCalcType

This enumerated type has the following values:

- JAMSHIDIAN
- EXPIRY_ONLY
- EXPIRY_TREE

Class: FinModelRatesHW()

class FinModelRatesHW():

FinModelRatesHW

*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.*

```
FinModelRatesHW(sigma,
                a,
                numTimeSteps=100,
                europeanCalcType=FinHWEuropeanCalcType.EXPIRY_TREE):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
sigma	-	-	-
a	-	-	-
numTimeSteps	-	-	100
europeanCalcType	-	-	EXPIRY_TREE

optionOnZCB

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.

```
optionOnZCB(texp, tmat,
            strike, faceAmount,
            dfTimes, dfValues):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strike	-	-	-
faceAmount	-	-	-
dfTimes	-	-	-
dfValues	-	-	-

europeanBondOptionJamshidian

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.

```
europeanBondOptionJamshidian(texp,
                             strikePrice,
                             face,
                             cpnTimes,
                             cpnAmounts,
                             dfTimes,
                             dfValues):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
strikePrice	-	-	-
face	-	-	-
cpnTimes	-	-	-
cpnAmounts	-	-	-
dfTimes	-	-	-
dfValues	-	-	-

europeanBondOptionExpiryOnly

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.

```
europeanBondOptionExpiryOnly(texp,
                       strikePrice,
                       faceAmount,
                       cpnTimes,
                       cpnAmounts):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
cpnTimes	-	-	-
cpnAmounts	-	-	-

optionOnZeroCouponBond_Tree

Price an option on a zero coupon bond using a HW trinomial tree. The discount curve was already supplied to the tree build.

```
optionOnZeroCouponBond_Tree(texp,
                            tmat,
                            strikePrice,
                            faceAmount):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-

bermudanSwaption

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.

```
bermudanSwaption(texp, tmat, strike, face,
                 couponTimes, couponFlows, exerciseType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strike	-	-	-
face	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseType	-	-	-

bondOption

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.*

```
bondOption(texp, strikePrice, faceAmount,
           couponTimes, couponFlows, exerciseType):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseType	-	-	-

callablePuttableBond_Tree

```
callablePuttableBond_Tree(couponTimes,
                          couponFlows,
                          callTimes,
                          callPrices,
                          putTimes,
                          putPrices,
                          faceAmount):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
couponTimes	-	-	-
couponFlows	-	-	-
callTimes	-	-	-
callPrices	-	-	-
putTimes	-	-	-
putPrices	-	-	-
faceAmount	-	-	-

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	-	-	-

buildTree

Build the trinomial tree.

```
buildTree(treeMat, dfTimes, dfValues):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
treeMat	-	-	-
dfTimes	-	-	-
dfValues	-	-	-

optionExerciseTypesToInt

PLEASE ADD A FUNCTION DESCRIPTION

```
optionExerciseTypesToInt(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	-	-	-

buildTree_Fast

Fast tree construction using Numba.

```
buildTree_Fast(a, sigma, treeTimes, numTimeSteps, discountFactors):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
a	-	-	-
sigma	-	-	-
treeTimes	-	-	-
numTimeSteps	-	-	-
discountFactors	-	-	-

americanBondOption_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.

```
americanBondOption_Tree_Fast(texp,
                             strikePrice,
                             faceAmount,
                             couponTimes,
                             couponAmounts,
                             exerciseTypeInt,
                             _sigma,
                             _a,
                             _Q,
                             _pu, _pm, _pd,
                             _rt,
                             _dt,
                             _treeTimes,
                             _dfTimes, _dfValues):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
strikePrice	-	-	-
faceAmount	-	-	-
couponTimes	-	-	-
couponAmounts	-	-	-
exerciseTypeInt	-	-	-
_sigma	-	-	-
_a	-	-	-
_Q	-	-	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_treeTimes	-	-	-
_dfTimes	-	-	-
_dfValues	-	-	-

bermudanSwaption_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.

```
bermudanSwaption_Tree_Fast(texp, tmat, strikePrice, face,
                           couponTimes, couponFlows,
                           exerciseTypeInt,
                           _dfTimes, _dfValues,
                           _treeTimes, _Q, _pu, _pm, _pd, _rt, _dt, _a):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
texp	-	-	-
tmat	-	-	-
strikePrice	-	-	-
face	-	-	-
couponTimes	-	-	-
couponFlows	-	-	-
exerciseTypeInt	-	-	-
_dfTimes	-	-	-
_dfValues	-	-	-
_treeTimes	-	-	-
_Q	-	-	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_a	-	-	-

callablePuttableBond_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.

```
callablePuttableBond_Tree_Fast(couponTimes, couponFlows,
                               callTimes, callPrices,
                               putTimes, putPrices, face,
                               _sigma, _a, _Q,  # IS SIGMA USED ?
                               _pu, _pm, _pd, _rt, _dt, _treeTimes,
                               _dfTimes, _dfValues):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
couponTimes	-	-	-
couponFlows	-	-	-
callTimes	-	-	-
callPrices	-	-	-
putTimes	-	-	-
putPrices	-	-	-
face	-	-	-
_sigma	-	IS SIGMA USED ?	-
_a	-	IS SIGMA USED ?	-
_Q	-	IS SIGMA USED ?	-
_pu	-	-	-
_pm	-	-	-
_pd	-	-	-
_rt	-	-	-
_dt	-	-	-
_treeTimes	-	-	-
_dfTimes	-	-	-
_dfValues	-	-	-

fwdFullBondPrice

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.

```
fwdFullBondPrice(rt, *args):
```

The function arguments are described in the following table.

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

10.21 FinModelRatesLMM

Enumerated Type: FinRateModelLMMModelTypes

This enumerated type has the following values:

- LMM_ONE_FACTOR
- LMM_HW_M_FACTOR
- LMM_FULL_N_FACTOR

LMMPrintForwards

Helper function to display the simulated Ibor rates.

```
LMMPrintForwards(fwds):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
fwds	-	-	-

LMMSwaptionVolApprox

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..

```
LMMSwaptionVolApprox(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	-	-	-

LMMSimSwaptionVol

Calculates the swap rate volatility using the forwards generated in the simulation to see how it compares to Rebonatto estimate.

```
LMMSimSwaptionVol(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	-	-	-

LMMFwdFwdCorrelation

Extract forward forward correlation matrix at some future time index from the simulated forward rates and return the matrix.

```
LMMFwdFwdCorrelation(numForwards, numPaths, iTime, fwds):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numForwards	-	-	-
numPaths	-	-	-
iTime	-	-	-
fwds	-	-	-

LMMPriceCapsBlack

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.

```
LMMPriceCapsBlack(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	-	-	-

subMatrix

Returns a submatrix of correlation matrix at later time step in the LMM simulation which is then used to generate correlated Gaussian RVs.

```
subMatrix(t, N):
```

The function arguments are described in the following table.

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

CholeskyNP

Numba-compliant wrapper around Numpy cholesky function.

```
CholeskyNP(rho):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
rho	-	-	-

LMMSimulateFwdsNF

Full N-Factor Arbitrage-free simulation of forward Ibor curves 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)

```
LMMSimulateFwdsNF(numForwards, numPaths, fwd0, zetas, correl, taus, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numForwards	-	-	-
numPaths	-	-	-
fwd0	-	-	-
zetas	-	-	-
correl	-	-	-
taus	-	-	-
seed	-	-	-

LMMSimulateFwds1F

One factor Arbitrage-free simulation of forward Ibor curves 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.

```
LMMSimulateFwds1F(numForwards, numPaths, numeraireIndex, fwd0, gammas,
                  taus, useSobol, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numForwards	-	-	-
numPaths	-	-	-
numeraireIndex	-	-	-
fwd0	-	-	-
gammas	-	-	-
taus	-	-	-
useSobol	-	-	-
seed	-	-	-

LMMSimulateFwdsMF

Multi-Factor Arbitrage-free simulation of forward Ibor curves 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.

```
LMMSimulateFwdsMF(numForwards, numFactors, numPaths, numeraireIndex, fwd0,
                  lambdas, taus, useSobol, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numForwards	-	-	-
numFactors	-	-	-
numPaths	-	-	-
numeraireIndex	-	-	-
fwd0	-	-	-
lambdas	-	-	-
taus	-	-	-
useSobol	-	-	-
seed	-	-	-

LMMCapFlrPricer

Function to price a strip of cap or floorlets in accordance with the simulated forward curve dynamics.

```
LMMCapFlrPricer(numForwards, numPaths, K, fwd0, fwds, taus, isCap):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numForwards	-	-	-
numPaths	-	-	-
K	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-
isCap	-	-	-

LMMSwapPricer

Function to reprice a basic swap using the simulated forward Ibors.

```
LMMSwapPricer(cpn, numPeriods, numPaths, fwd0, fwds, taus):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
cpn	-	-	-
numPeriods	-	-	-
numPaths	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-

LMMSwaptionPricer

Function to price a European swaption using the simulated forward curves.

```
LMMSwaptionPricer(strike, a, b, numPaths, fwd0, fwds, taus, isPayer):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
strike	-	-	-
a	-	-	-
b	-	-	-
numPaths	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-
isPayer	-	-	-

LMMRatchetCapletPricer

Price a ratchet using the simulated Ibor rates.

```
LMMRatchetCapletPricer(spread, numPeriods, numPaths, fwd0, fwds, taus):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spread	-	-	-
numPeriods	-	-	-
numPaths	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-

LMMFlexiCapPricer

Price a flexicap using the simulated Ibor rates.

```
LMMFlexiCapPricer(maxCaplets, K, numPeriods, numPaths, fwd0, fwds, taus):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
maxCaplets	-	-	-
K	-	-	-
numPeriods	-	-	-
numPaths	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-

LMMStickyCapletPricer

Price a sticky cap using the simulated Ibor rates.

```
LMMStickyCapletPricer(spread, numPeriods, numPaths, fwd0, fwds, taus):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
spread	-	-	-
numPeriods	-	-	-
numPaths	-	-	-
fwd0	-	-	-
fwds	-	-	-
taus	-	-	-

10.22 FinModelRatesVasicek

Class: FinModelRatesVasicek()

class FinModelRatesVasicek():

FinModelRatesVasicek

self._a = a

```
FinModelRatesVasicek(a, b, sigma):
```

The function arguments are described in the following table.

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

meanr

*mr = r0 * exp(-a * t) + b * (1 - exp(-a * 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

*vr = sigma * sigma * (1.0 - exp(-2.0 * a * t)) / 2.0 / a*

```
variancer(a, b, sigma, t):
```

The function arguments are described in the following table.

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

zeroPrice

*B = (1.0 - exp(-a * t)) / a*

```
zeroPrice(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	-	-	-

ratePath_MC

PLEASE ADD A FUNCTION DESCRIPTION

```
ratePath_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	-	-	-

zeroPrice_MC

PLEASE ADD A FUNCTION DESCRIPTION

```
zeroPrice_MC(r0, a, b, sigma, t, dt, numPaths, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
r0	-	-	-
a	-	-	-
b	-	-	-
sigma	-	-	-
t	-	-	-
dt	-	-	-
numPaths	-	-	-
seed	-	-	-

10.23 FinModelSABR

Class: FinModelSABR()

FinModelSABR

Create FinModelSABR with model parameters.

```
FinModelSABR(alpha, beta, rho, nu):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
beta	-	-	-
rho	-	-	-
nu	-	-	-

blackVol

Black volatility from SABR model using Hagan et al. approx.

```
blackVol(f, k, t):
```

The function arguments are described in the following table.

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

value

Price an option using Black's model which values in the forward measure following a change of measure.

```
value(forwardRate,    # Forward rate
      strikeRate,     # Strike Rate
      timeToExpiry,   # time to expiry in years
      df,             # Discount Factor to expiry date
      callOrPut):     # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forwardRate	-	Forward rate	-
strikeRate	-	Strike Rate	-
timeToExpiry	-	time to expiry in years	-
df	-	Discount Factor to expiry date	-
callOrPut	-	Call or put	-

blackVolFromSABR

PLEASE ADD A FUNCTION DESCRIPTION

```
blackVolFromSABR(alpha, beta, rho, nu, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
beta	-	-	-
rho	-	-	-
nu	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

10.24 FinModelSABRShifted

Class: FinModelSABRShifted()

FinModelSABRShifted

self._alpha = alpha

```
FinModelSABRShifted(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	-	-	-

blackVol

Black volatility from SABR model using Hagan et al. approx.

```
blackVol(f, k, t):
```

The function arguments are described in the following table.

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

value

Price an option using Black's model which values in the forward measure following a change of measure.

```
value(forwardRate,    # Forward rate F
      strikeRate,     # Strike Rate K
      timeToExpiry,   # Time to Expiry (years)
      df,             # Discount Factor to expiry date
      callOrPut):     # Call or put
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
forwardRate	-	Forward rate F	-
strikeRate	-	Strike Rate K	-
timeToExpiry	-	Time to Expiry (years)	-
df	-	Discount Factor to expiry date	-
callOrPut	-	Call or put	-

blackVolFromShiftedSABR

PLEASE ADD A FUNCTION DESCRIPTION

```
blackVolFromShiftedSABR(alpha, beta, rho, nu, s, f, k, t):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
alpha	-	-	-
beta	-	-	-
rho	-	-	-
nu	-	-	-
s	-	-	-
f	-	-	-
k	-	-	-
t	-	-	-

10.25 FinModelStudentTCopula

Class: FinModelStudentTCopula()

class FinModelStudentTCopula():

defaultTimes

PLEASE ADD A FUNCTION DESCRIPTION

```
defaultTimes(issuerCurves,
             correlationMatrix,
             degreesOfFreedom,
             numTrials,
             seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
issuerCurves	-	-	-
correlationMatrix	-	-	-
degreesOfFreedom	-	-	-
numTrials	-	-	-
seed	-	-	-

10.26 FinProcessSimulator

Enumerated Type: FinProcessTypes

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: FinCIRNumericalScheme

This enumerated type has the following values:

- EULER
- LOGNORMAL
- MILSTEIN
- KAHLJACKEL
- EXACT

Class: FinProcessSimulator()

class FinProcessSimulator():

FinProcessSimulator

pass

```
FinProcessSimulator():
```

The function arguments are described in the following table.

getProcess

PLEASE ADD A FUNCTION DESCRIPTION

```
getProcess(processType,
    t,
    modelParams,
    numAnnSteps,
    numPaths,
    seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
processType	-	-	-
t	-	-	-
modelParams	-	-	-
numAnnSteps	-	-	-
numPaths	-	-	-
seed	-	-	-

getHestonPaths

PLEASE ADD A FUNCTION DESCRIPTION

```
getHestonPaths(numPaths,
               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
numPaths	-	-	-
numAnnSteps	-	-	-
t	-	-	-
drift	-	-	-
s0	-	-	-
v0	-	-	-
kappa	-	-	-
theta	-	-	-
sigma	-	-	-
rho	-	-	-
scheme	-	-	-
seed	-	-	-

getGBMPaths

PLEASE ADD A FUNCTION DESCRIPTION

```
getGBMPaths(numPaths, numAnnSteps, t, mu, stockPrice, sigma, scheme, seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numPaths	-	-	-
numAnnSteps	-	-	-
t	-	-	-
mu	-	-	-
stockPrice	-	-	-
sigma	-	-	-
scheme	-	-	-
seed	-	-	-

getVasicekPaths

PLEASE ADD A FUNCTION DESCRIPTION

```
getVasicekPaths(numPaths,
                numAnnSteps,
                t,
                r0,
                kappa,
                theta,
                sigma,
                scheme,
                seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numPaths	-	-	-
numAnnSteps	-	-	-
t	-	-	-
r0	-	-	-
kappa	-	-	-
theta	-	-	-
sigma	-	-	-
scheme	-	-	-
seed	-	-	-

getCIRPaths

PLEASE ADD A FUNCTION DESCRIPTION

```
getCIRPaths(numPaths,
            numAnnSteps,
            t,
            r0,
            kappa,
            theta,
            sigma,
            scheme,
            seed):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numPaths	-	-	-
numAnnSteps	-	-	-
t	-	-	-
r0	-	-	-
kappa	-	-	-
theta	-	-	-
sigma	-	-	-
scheme	-	-	-
seed	-	-	-

10.27 FinSobol

getGaussianSobol

Sobol Gaussian quasi random points generator based on graycode order. The generated points follow a normal distribution.

```
getGaussianSobol(numPoints, dimension):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numPoints	-	-	-
dimension	-	-	-

getUniformSobol

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.

```
getUniformSobol(numPoints, dimension):
```

The function arguments are described in the following table.

Argument Name	Type	Description	Default Value
numPoints	-	-	-
dimension	-	-	-