cred

Release 0.0.1

Jul 13, 2020

Overview:

CHAPTER 1

Table of Contents

1.1 cred Overview

1.1.1 What is cred?

cred is a python package for modeling **commercial real estate debt**. It is designed to enable quickly building and analyzing cash flows for common debt structures while still retaining the flexibility to easily customize debt terms and performance scenarios.

Homepage: https://github.com/jordanhitchcock/cred

Documentation: https://cred.readthedocs.io/en/latest/

Tutorials: Beginner's Guide to Building a Loan with cred

1.1.2 Installation

```
pip install cred
```

1.1.3 Quickstart

The main interface for creating cash flows for loans with regular interest periods is *PeriodicBorrowing*. The *FixedRateBorrowing* subclass is a convenient class representation of fixed rate debt. The example below builds a borrowing object for a 1 year loan with monthly interest payments at 5.0%.

```
from datetime import date

from cred import FixedRateBorrowing, Monthly

loan = FixedRateBorrowing(
    start_date=date(2020, 1, 1),
```

(continues on next page)

(continued from previous page)

```
    end_date=date(2021, 1, 1),
    freq=Monthly(months=1),
    initial_principal=10_000_000,
    coupon=0.05
)

loan.schedule()
```

Other convenience arguments than can be defined at initialization include the amortization method (default is interest only) and day count convention (default is actual / 360). See the documentation for additional detail.

Calling *borrowing.schedule()* returns a *pandas.DataFrame* with the scheduled cash flows:

```
index  start_date    end_date payment_date  bop_principal  interest_rate  interest_
↪payment  principal_payment       payment  eop_principal
0      2020-01-01  2020-02-01   2020-02-01       10000000           0.05      43055.
↪555556                  0  4.305556e+04       10000000
1      2020-02-01  2020-03-01   2020-03-01       10000000           0.05      40277.
↪777778                  0  4.027778e+04       10000000
2      2020-03-01  2020-04-01   2020-04-01       10000000           0.05      43055.
↪555556                  0  4.305556e+04       10000000
3      2020-04-01  2020-05-01   2020-05-01       10000000           0.05      41666.
↪666667                  0  4.166667e+04       10000000
4      2020-05-01  2020-06-01   2020-06-01       10000000           0.05      43055.
↪555556                  0  4.305556e+04       10000000
5      2020-06-01  2020-07-01   2020-07-01       10000000           0.05      41666.
↪666667                  0  4.166667e+04       10000000
6      2020-07-01  2020-08-01   2020-08-01       10000000           0.05      43055.
↪555556                  0  4.305556e+04       10000000
7      2020-08-01  2020-09-01   2020-09-01       10000000           0.05      43055.
↪555556                  0  4.305556e+04       10000000
8      2020-09-01  2020-10-01   2020-10-01       10000000           0.05      41666.
↪666667                  0  4.166667e+04       10000000
9      2020-10-01  2020-11-01   2020-11-01       10000000           0.05      43055.
↪555556                  0  4.305556e+04       10000000
10     2020-11-01  2020-12-01   2020-12-01       10000000           0.05      41666.
↪666667                  0  4.166667e+04       10000000
11     2020-12-01  2021-01-01   2021-01-01       10000000           0.05      43055.
↪555556           10000000  1.004306e+07              0
```

Floating Rate Example

Floating rate borrowing objects require subclassing *PeriodicBorrowing* and implementing the *interest_rate* method. The *interest_rate* method is called as part of the loop that calculates loan values for each period. It receives an *InterestPeriod* object which contains *index*, *start_date*, and *end_date* attributes for the period.

The example below generates random index resets between 1.5% and 2.0% and adds a 2.0% margin to calculate total interest.

```
import random
from cred import PeriodicBorrowing

class MyFloatingRateLoanType(PeriodicBorrowing):

    def interest_rate(self, period):
```

(continues on next page)

(continued from previous page)

```
        return random.uniform(0.015, 0.02) + 0.02

floating_loan = MyFloatingRateLoanType(
    start_date=date(2020, 1, 1),
    end_date=date(2021, 1, 1),
    freq=Monthly(months=1),
    initial_principal=10_000_000
)

floating_loan.schedule()
```

Output:

```
index  start_date    end_date payment_date  bop_principal  interest_rate  interest_
↪payment  principal_payment       payment  eop_principal
0      2020-01-01  2020-02-01   2020-02-01       10000000       0.039226      33778.
↪233665                  0  3.377823e+04       10000000
1      2020-02-01  2020-03-01   2020-03-01       10000000       0.036212      29170.
↪599256                  0  2.917060e+04       10000000
2      2020-03-01  2020-04-01   2020-04-01       10000000       0.039830      34298.
↪387753                  0  3.429839e+04       10000000
3      2020-04-01  2020-05-01   2020-05-01       10000000       0.037286      31072.
↪075651                  0  3.107208e+04       10000000
4      2020-05-01  2020-06-01   2020-06-01       10000000       0.038355      33027.
↪953727                  0  3.302795e+04       10000000
5      2020-06-01  2020-07-01   2020-07-01       10000000       0.036090      30074.
↪908731                  0  3.007491e+04       10000000
# ...
```

Custom implementations of other cash flow and data fields can similarly be modified by subclassing and overriding the applicable method.

Adding Custom Fields to the Borrowing Schedule

In addition to modifying current schedule columns, new fields can easily be added to the schedule as well. The example below adds two new columns:

- **NOI**: Net operating income for each month ($60,000 per month, growing monthly at an annual rate of 3.0%)
- **DSCR**: The debt service coverage ratio for each month based on a constant 6.44% debt service multiple (approximately the debt multiple for a 30 year amortizing loan with 5% interest)

set_period_values is the main method inside *schedule* that sets period values. Since the two new methods are called after the super class sets its period values, the new columns will be appended to the right side of the schedule.

```
class MyCustomLoanType(MyFloatingRateLoanType):

    def noi(self, period):
        return 60000 * (1 + 0.03 / 12 * period.index)

    def dscr(self, period):
        return period.noi / period.interest_payment

    def set_period_values(self, period):
        super().set_period_values(period)
        period.add_display_field(self.noi(period), 'noi')
```

(continues on next page)

(continued from previous page)

```
        period.add_display_field(self.dscr(period), 'dscr')

custom_loan = MyCustomLoanType(
    start_date=date(2020, 1, 1),
    end_date=date(2021, 1, 1),
    freq=Monthly(months=1),
    initial_principal=10_000_000
)

custom_loan.schedule()
```

Result (scroll all the way to the right):

```
index  start_date    end_date payment_date  bop_principal  interest_rate  interest_
↪payment  principal_payment       payment  eop_principal      noi      dscr
0      2020-01-01  2020-02-01   2020-02-01       10000000       0.036185      31159.
↪351494                  0  3.115935e+04       10000000  60000.0  1.925586
1      2020-02-01  2020-03-01   2020-03-01       10000000       0.035363      28486.
↪801992                  0  2.848680e+04       10000000  60150.0  2.111504
2      2020-03-01  2020-04-01   2020-04-01       10000000       0.035551      30613.
↪195658                  0  3.061320e+04       10000000  60300.0  1.969739
3      2020-04-01  2020-05-01   2020-05-01       10000000       0.037290      31075.
↪189753                  0  3.107519e+04       10000000  60450.0  1.945282
4      2020-05-01  2020-06-01   2020-06-01       10000000       0.037907      32642.
↪384490                  0  3.264238e+04       10000000  60600.0  1.856482
5      2020-06-01  2020-07-01   2020-07-01       10000000       0.037355      31129.
↪007229                  0  3.112901e+04       10000000  60750.0  1.951556
# ...
```

Accessing Period Values

In addition to accessing the entire loan schedule through the *schedule* method, values for individual periods can be accessed through the *borrowing.period* method. This method takes the zero-based index of the target period and returns the schedule values for the period as a dictionary.

self.period is the recommended way to recursively pull in values from previous periods when setting period values. For example, after the initial period the beginning-of-period principal (*bop_principal*) balance is equal to the previous period's ending value. The implementation for the *bop_principal* method is:

```
def bop_principal(self, period):
    if period.index == 0:
        return self.initial_principal
    return self.period(period.index - 1).eop_principal
```

Note: Always reference the current period with the *period* argument and not through *self.period* as doing so will cause infinite recursion problems.

Accessing values from previous periods provides a simple and intuitive way to implement recursive calculations, for example capitalizing interest expense for a construction loan.

Period Value Caching

Certain debt assumptions may change during project evaluation or may be unknown prior to building the cash flows. The clearest example is interest rates which change second by second.

In order to avoid accidentally using stale values, *Borrowing* objects do not store schedule values. They are recalculated any time *schedule* or *period* is called. This means that it is safe to update borrowing attributes, and any attribute changes will be reflected in subsequent calls.

Recalculating values for every period could hamper performance if many recursive look-ups exist, however the *schedule* method is smart and caches previous period values during execution of the method.

Additionally, borrowings have a context manager that will enable period caching on entry and purge cached values on exit.

1.2 Borrowing Classes

1.2.1 PeriodicBorrowing

class cred.**PeriodicBorrowing**(*start_date*, *end_date*, *freq*, *initial_principal*, *first_reg_start=None*, *year_frac=<function actual360>*, *calc_convention=<function unadjusted>*, *pmt_convention=<function unadjusted>*, *holiday_calendar=None*, *desc=None*, *prepayment=None*)

Abstract class for debt with regular, periodic principal and interest periods. Superclass for FixedRateBorrowing.

Parameters

- **start_date** (*datetime-like*) – Borrowing start date
- **end_date** (*datetime-like*) – Borrowing end date
- **freq** (Monthly, *dateutil.relativedelta.relativedelta*) – Interest period frequency. Using the *Monthly* offset is recommended to automatically recognize end of month roll dates appropriately.
- **initial_principal** – Initial principal amount of the borrowing
- **first_reg_start** (*datetime-like, optional(default=None)*) – Start date of first regular interest period. If *None* (default), will be the same as the *start_date*.
- **year_frac** (*function*) – Function that takes two dates and returns the year fraction between them. Bound to *Borrowing.year_frac*. Default function is *cred.interest.actual360*. Use *cred.interest.thrity360* for NASD 30 / 360 day count.
- **calc_convention** (*function, optional(default=cred.businessdays.unadjusted)*) – Business day adjustment method for interest calculation dates. Function that takes a date as its first argument and a list of holidays as its second argument and returns the adjusted date. See *cred.businessdays.following*, *preceding*, and *modified_following*. Assigned to 'adjust_calc_date'.
- **pmt_convention** (*function, optional(default=cred.businessdays.unadjusted)*) – Business day adjustment method for payment dates. Function that takes a date as its first argument and a list of holidays as its second argument and returns the adjusted date. See *cred.businessdays.following*, *preceding*, and *modified_following*. Assigned to 'adjust_pmt_date'.
- **holiday_calendar** (*pandas.tseries.holiday.AbstractHolidayCalendar, optional(default=None)*) – Payment holidays to use in adjusting payment dates. Defaults to None.

- **desc** (*int, str, optional(default=None)*) – Optional borrowing description.
- **prepayment** (BasePrepayment, *optional(default=None)*) – Optional BasePrepayment subclass that defines prepayment terms and calculates prepayment costs.

accrued_interest(*dt*, *include_dt=False*)
Returns the amount of interest accrued from the start of the interest period in which *dt* falls to *dt*. By default, calculates interest from and including the period start date to but excluding *dt*. Set *include_dt=True* to include interest accrued through and including *dt*.

bop_principal(*period*)
Returns the beginning of interest period principal balance for the *InterestPeriod* argument.

date_index(*dt*, *inc_period_end=False*)
Returns the index of the period that contains the date argument. Default is inclusive of period start dates and exclusive of end dates. Dates prior to the borrowing start date or after the latest of the borrowing end date or final payment date raise and index error.

Parameters

- **dt** (*datetime-like*) – Look-up date
- **inc_period_end** (*bool*) – Determines whether roll dates are included in the ending or starting period. Be default, end dates are considered the start date for the next period and are not included.

Returns

Return type int

date_period(*dt*, *inc_period_end=False*)
Returns the period that date falls inside. Default is inclusive of period start date and exclusive of end date. Dates prior to the borrowing start date or after the max(borrowing end date, final payment date) raise and index error.

Parameters

- **dt** (*datetime-like*) – Look-up date
- **inc_period_end** (*bool optional(default=False)*) – Determines whether roll dates are included in the ending or starting period. Be default, end dates are considered the start date for the next period and are not included.

Returns

Return type *InterestPeriod*

eop_principal(*period*)
Period beginning balance less the period principal payment.

interest_payment(*period*)
Calculates the interest payment for the period as the year fraction from the period's start date to end date (using the borrowing's *year_frac* method) times the period's *interest_rate* property times the period's *bop_principal* property.

Parameters **period** (InterestPeriod) – *PeriodicBorrowing* uses *InterestPeriod* class periods.

Returns

Return type float

outstanding_principal (*dt*, *include_dt=False*)

Returns the clean amount not including any accrued interest. The outstanding amount contemplates payment dates rather than period end dates, so the balance is not reduced by any amortization until the payment date occurs.

Returns None for dates prior to the start date, and returns the last period's beginning balance less principal payment for any date equal or greater than the final payment date.

See *repayment_amount* for total cost to repay including any prepayment premiums.

Parameters

- **dt** (*datetime-like*) – As-of date
- **include_dt** (*bool, optional(default=False)*) – Indicates whether to include principal payments due on *dt*

Returns

Return type float

payments (*first_dt=None*, *last_dt=None*, *pmt_dt=False*)

Returns a list of list of *(date, payment_amount)* for all payments from *first_dt* to *last_dt* inclusive. If *pmt_dt=False*, then dates will correspond to scheduled period end dates. *pmt_dt=True* will evaluate and return dates based on their adjusted payment dates.

Passing *first_dt* or *last_dt* as *None* will return payments from the first period or through the last period respectively. Passing both as *None* will return payments for the entire term of the borrowing.

Parameters

- **first_dt** (*datetime-like, optional(default=None)*) – First payment date, inclusive
- **last_dt** (*datetime-like, optional(default=None)*) – Last payment date, inclusive
- **pmt_dt** (*bool*) – Whether to evaluate dates based on scheduled period end dates or adjusted period payment dates

Returns

Return type list((date, float))

period_end_date (*i*)

Returns end period calculation end date for period with index *i*. Returns *None* for indexes greater than then number of interest periods in the loan.

period_payment (*period*)

Returns the sum of the period's interest and principal payments.

period_start_date (*i*)

Returns the calculation start date for period with index *i*.

pmt_date (*i*)

Returns the payment date for period with index *i*.

principal_payment (*period*)

Returns the principal payment for the period. By default this assumings interest only unless overridden by a subclass (e.g. FixedRateBorrowing).

repayment_amount (*dt*)

Required repayment amount including any prepayment premiums as defined by the *prepayment* object. See *borrowing.outstanding_principal* for clean balance.

schedule()
Returns the borrowing's cash flow schedule as a *pandas.DataFrame*.

set_period_values(*period*)
Called for each period after it is initialized to set the period's values. Calculating period values may rely on previously set values for that period (e.g. see interest payment), so order matters.

PeriodicBorrowing and its subclasses use *InterestPeriod* which should have a start date, end date, payment date, beginning period principal, interest payment, and principal payments.

unpaid_amount(*dt*, *interest=True*, *princ=True*, *include_dt=False*)
Returns the total unpaid interest and/or principal, if any. Specifically, returns the amount of interest to be paid on the payment date if *dt* is greater than or equal to the period end date but less than (or less than or equal to if *include_dt=True*) the period payment date and/or any scheduled principal payments if dt is between the period end date and payment date.

By default, does not include any payments due on *dt*.

Parameters

- **dt** (*datetime-like*) – Date of evaluation
- **interest** (*bool*) – Include unpaid interest if True
- **princ** (*bool*) – Include unpaid principal if True
- **include_dt** (*bool*) – Include amounts scheduled to be paid on *dt*

1.2.2 FixedRateBorrowing

class cred.**FixedRateBorrowing**(*start_date*, *end_date*, *freq*, *initial_principal*, *coupon*, *amort_periods=None*, *io_periods=0*, ***kwargs*)
PeriodicBorrowing subclass for fixed rate borrowings.

Parameters

- **start_date** (*datetime-like*) – Borrowing start date
- **end_date** (*datetime-like*) – Borrowing end date
- **freq** (*dateutil.relativedelta.relativedelta*) – Interest period frequency
- **initial_principal** – Initial principal amount of the borrowing
- **coupon** (*float*) – Coupon rate
- **amort_periods** (*int, object, optional(default=None)*) – If None (default), will be calculated as interest only.

  If *amort_periods* is a single number *n*, then will calculate principal payments based on a fully amortizing schedule over *n* periods of length *freq* with constant principal and interest payments (e.g. *360* where *freq=relativedelta(months=1)* will calculate 30 year amortization with constant monthly payments.

  If *amort_periods* is an object, it must implement *__getitem__* and must have length at least greater than or equal to the number of periods. Custom amortization schedules can be provided this way, for example using lists or *pandas.Series* objects with amortization amount for period i at index i. Note that custom amortizations schedules should include the balloon payment as well.

- **io_periods** (*int, optional(default=0)*) – If *amort_periods* is a number (i.e. amortization with constant principal and interest payments), then defines the leading number

of full interest only periods. Calculated from the *first_reg_start* date, so any leading stub periods are ignored.

- ****kwargs** – Keyword arguments passed to superclass (PeriodicBorrowing) initialization. Ex. *desc* for borrowing description, *year_frac* for day count convention, *pmt_convention* for business day adjustment, *first_reg_start*, etc.

principal_payment(*period*)
Returns the principal payment for the period. By default this assumings interest only unless overridden by a subclass (e.g. FixedRateBorrowing).

1.3 Date and Calendar

1.3.1 Date Offset

class cred.**Monthly**(*months=1*)
Monthly date offset that recognizes whether it is added to the last day of the month. If so, returns the last day of the corresponding month. Convenience offset for interest periods that may roll on the last day of the month.

Example: adding *Monthly()* to *date(2020, 6, 30)* returns *date(2020, 7, 31)*. The default object is a one month offset.

Parameters **months** (*int*) – Offset in months.

1.3.2 Business Day Calendars

cred.**FederalReserveHolidays**(*name=None*, *rules=None*)
U.S. Federal Reserve banking holidays. Holidays are thought to be accurate, but you should verify independently.

cred.**LondonBankHolidays**(*name=None*, *rules=None*)
London banking holidays. Holidays are thought to be accurate, but you should verify independently.

1.3.3 Date Adjustment Conventions

cred.**unadjusted**(*dt*, *holidays=None*)
Return unadjusted date. *calendar* parameter does not affect return value, provides consistency with other convention functions.

cred.**modified_following**(*dt*, *holidays*)
Return the next business day if *dt* is on a weekend or holiday in *holidays* unless the next business day is in the following calendar month, in which case returns the previous business day.

cred.**following**(*dt*, *holidays*)
Return the next business day if *dt* is on a weekend or a date in *holidays*.

cred.**preceding**(*dt*, *holidays*)
Return the previous business day if *dt* is on a weekend or a date in *holidays*.

1.4 Helper Functions

1.4.1 Actual / 360

cred.**actual360**(*dt1*, *dt2*)
Returns the fraction of a year between *dt1* and *dt2* on an actual / 360 day count basis.

1.4.2 30 / 360

cred.**thirty360**(*dt1*, *dt2*)
Returns the fraction of a year between *dt1* and *dt2* on 30 / 360 day count basis.

1.5 Period Classes

1.5.1 Period

class cred.period.**Period**(*i*)
Superclass for InterestPeriod.

Parameters **i** (*int*) – Zero-based period index (e.g. the fourth period will have index 3)

add_display_field(*value*, *name*)
Adds *value* as an attribute named *name* to the period marks it as a display field. Display field attributes are included in *period.schedule*.

Parameters

- **value** – Numerical value of attribute
- **name** (*str*) – Name of attribute

add_payment(*value*, *name*)
Adds *value* as an attribute of the period object named *name* and marks it as a payment. Payment attributes are included in *period.schedule*.

Parameters

- **value** – Numerical value of attribute
- **name** (*str*) – Name of attribute

get_payment()
Returns the sum of payment attributes.

schedule()
Returns the period schedule as a {name: value} dictionary.

1.5.2 InterestPeriod

class cred.period.**InterestPeriod**(*i*)
Period type used by PeriodicBorrowing and its subclasses.

add_bop_principal(*amt*, *name='bop_principal'*)
Adds the value as a period attribute and marks it as the beginning of period principal balance. Each period

can only have one beginning principal balance attribute. Also added as an attribute that should be included in the schedule.

add_end_date(*dt*, *name='end_date'*)
Add an attribute to the object and mark it as the period end date. There can only be one end date. Also marks the attribute as a field that should be added to the period schedule.

add_interest_pmt(*amt*, *name='interest_payment'*)
Adds the value as a period attribute and marks it as an interest payment. Also added as an attribute that should be included in the schedule.

add_pmt_date(*dt*, *name='payment_date'*)
Add an attribute to the object and mark it as the period payment date. There can only be one payment date. Also marks the attribute as a field that should be added to the period schedule.

add_principal_pmt(*amt*, *name='principal_payment'*)
Adds the value as a period attribute and marks it as a principal payment. Also added as an attribute that should be included in the schedule.

add_start_date(*dt*, *name='start_date'*)
Add an attribute to the object and mark it as the period start date. There can only be one start date. Also marks the attribute as a field that should be added to the period schedule.

get_bop_principal()
Beginning of period (BoP) principal amount

get_end_date()
Period end date

get_interest_pmt()
Returns the sum of attributes marked as interest payments

get_pmt_date()
Returns the sum of attributes marked as payments, interest payments, principal payments

get_principal_pmt()
Returns the sum of attributes marked as principal payments

get_start_date()
Period start date

1.6 Prepayment Classes

1.6.1 BasePrepayment

class cred.**BasePrepayment**

required_repayment(*borrowing*, *dt*)
Calculate the prepayment amount. Called by PeriodicBorrowings to calculate prepayment. Must be implemented by subclasses.

Parameters

- **borrowing** (PeriodicBorrowing) – Borrowing to use in calculating prepayment amount
- **dt** (*datetime*) – Date of repayment

Returns

Return type float

1.6.2 Defeasance

class cred.**Defeasance**(*df_func*, *open_dt_offset=None*, *dfz_to_open=False*, *period_breakage='full_period'*)

open_date(*borrowing*)
Open date for borrowing

required_repayment(*borrowing*, *dt*)
Return the total estimated cost of replacement collateral

1.6.3 OpenPrepayment

class cred.**OpenPrepayment**(*period_breakage='full_period'*)

net_accrued_interest(*borrowing*, *dt*)
Accrued but interest during the period in which *dt* falls. Calculates interest from and including the first day of the period, to but excluding *dt* net of any amount already paid.

required_repayment(*borrowing*, *dt*)
Return required repayment amount based on open prepayment.

unpaid_and_current_period_interest(*borrowing*, *dt*)
Unpaid interest including interest due on *dt* plus interest accruing through the end of the period in which *dt* falls.

unpaid_interest(*borrowing*, *dt*)
Unpaid interest, including interest due on *dt*.

1.6.4 StepDown

class cred.**StepDown**(*expiration_offsets*, *premiums*, *period_breakage='full_period'*)

expiration_dates(*borrowing*)
Premium expiration dates adjusted for payment date business days by applying the expiration offsets to the borrowing's first regular period start date.

ppmt_premium(*borrowing*, *dt*)
The prepayment premium for prepaying on *dt*.

premium_pct(*borrowing*, *dt*)
The premium at *dt* expressed as a percent in decimal form of the then outstanding balance.

required_repayment(*borrowing*, *dt*)
Required amount to prepay the borrowing at the given date.

1.6.5 SimpleYieldMaintenance

class cred.**SimpleYieldMaintenance**(*rate_func*, *margin=0.0*, *wal_rate=False*, *open_dt_offset=None*, *ym_to_open=False*, *min_penalty=None*, *period_breakage='full_period'*)

discount_factors(*borrowing*, *dt*)
Discount factors used to calculate yield maintenance.

discount_rate(*borrowing*, *dt*)
Discount rate for yield maintenance at date.

discount_rate_term(*borrowing*, *dt*)
Return the number of days used to calculate the term of the discount rate. If *wal_rate* is True, returns the weight avgerage number of remaining days. If *ym_to_open* is True, calculates (weighted avg) days to the open date rather than the maturity date.

open_date(*borrowing*)
Open date for borrowing.

remaining_pmts(*borrowing*, *dt*)
Principal and interest payments from *dt* to maturity, or the open date if *ym_to_open* is True. Based on period end dates. The last payment will include all outstanding principal and accrued interest. Return value is a dict of {pmt_date: pmt} pairs.

required_repayment(*borrowing*, *dt*)
Return required repayment amount based on open prepayment.

1.7 Changelog

1.7.1 0.1.0 (2020-07-12)

- Prepayment with built-in support for common defeasance, open, step-down, and YM structures
- Significantly faster adjustments of holidays
- Partial interest only for FixedRateBorrowings
- Calculation date business day adjustments independent of payment date adjustments

1.7.2 0.0.2 (2020-04-14)

- Payment date adjustment convention and business days for PeriodicBorrowings
- Handling of starting and ending stub periods
- Additional tests and other improvements

1.7.3 0.0.1 (2020-03-29)

- Initial release

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

O

P

R

S

T

U