Cashflow Documentation

Release 0.0.3

Author

May 20, 2018

Contents

Date: May 20, 2018 **Version**: 0.0.5

Binary Installers: https://pypi.org/project/cashflows

Source Repository: https://github.com/jdvelasq/cashflows

Documentation: http://cashflows.readthedocs.io/en/latest/

cashflows is an open source (distributed under the MIT license) and friendly-user package to do financial computations. The package was developed and tested in Python version 3.6. It can be installed from the command line using:

```
$ pip install cashflows
```

cashflows can be used interactively at the Python's command prompt, but a better experience is achieved when IPython or Jupyter's notebook are used, allows the user to fully document the analysis and draw conclusions. Due to the design of the package, it is easy to use cashflows with the tools available in the ecosystem of open source tools for data science.

cashflows is well suited for many financial computations and it can be used to:

- Realize compound interest rate calculations.
- Converts between nominal, effective and periodic interest rates.
- Realize bond valuation.
- Realize currency conversions.
- Realize constant dollar transformations.
- Analyze different types of loans.
- Compute assets depreciation.
- Realize cashflow analysis.

CHAPTER 1

Table of Contents

cashflows is organized in the following modules:

1.1 Time value of money models

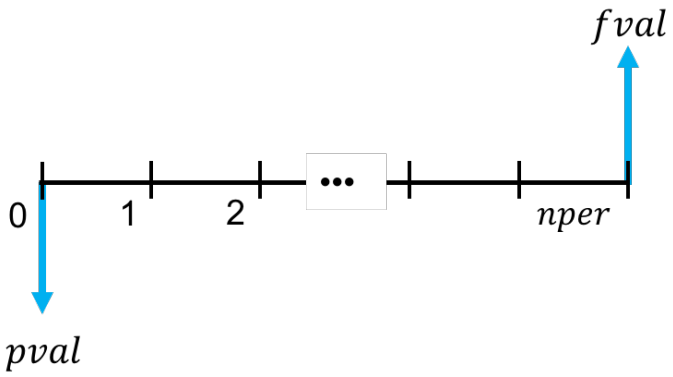
1.1.1 Overview

The functions in this module are used for certain compound interest calculations for a cashflow under the following restrictions:

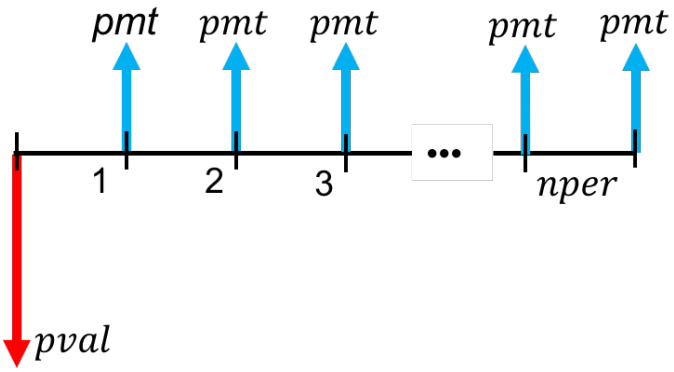
- Payment periods coincide with the compounding periods.
- Payments occur at regular intervals.
- Payments are a constant amount.
- Interest rate is the same over all analysis period.

For convenience of the user, this module implements the following simplified functions:

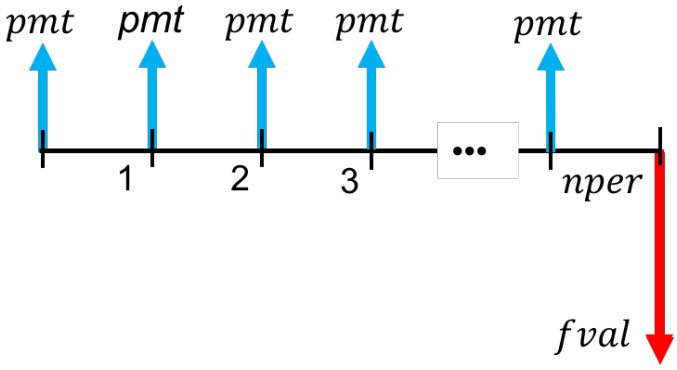
- `pvfv`: computes the missing value in the equation `fval = pval * (1 + rate) ** nper`, that represents the following cashflow.



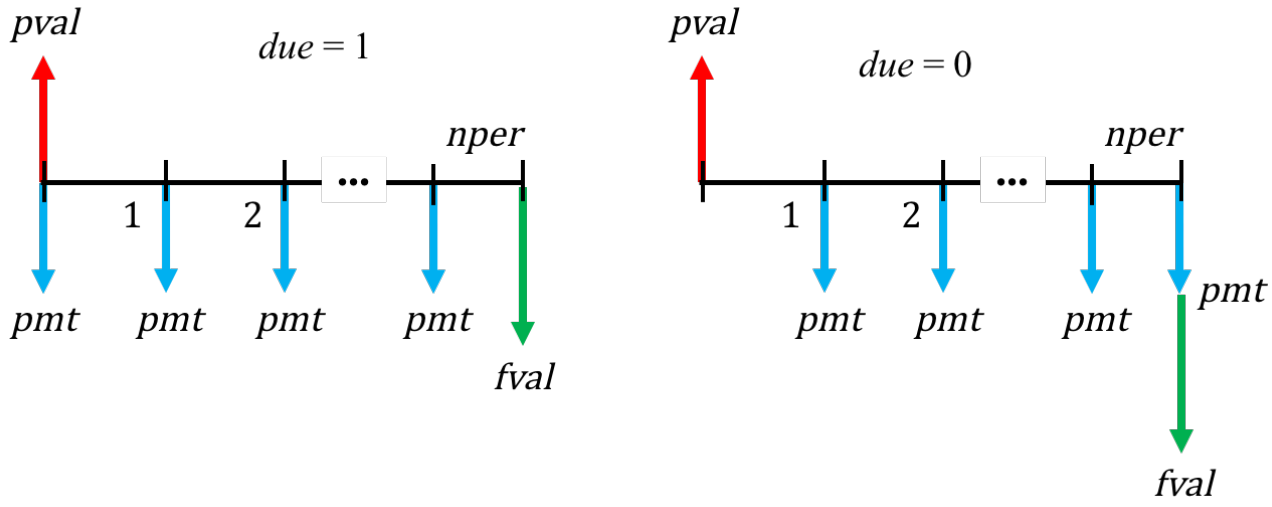
- `pvpmt`: computes the missing value (*pmt*, *pval*, *nper*, *nrate*) in a model relating a present value and a finite sequence of payments made at the end of the period (payments in arrears, or ordinary annuities), as indicated in the following cashflow diagram:



- `pmtfv`: computes the missing value (*pmt*, *fval*, *nper*, *nrate*) in a model relating a finite sequence of payments in advance (annuities due) and a future value, as indicated in the following diagram:



- `tvmm`: computes the missing value (*pmt*, *fval*, *pval, 'nper*, *nrate*) in a model relating a finite sequence of payments made at the beginning or at the end of the period, a present value, a future value, and an interest rate, as indicated in the following diagram:



In addition, the function `amortize` computes and returns the amortization schedule of a loan.

1.1.2 Functions in this module

`cashflows.tvmm.`**`amortize`**(*pval=None*, *fval=None*, *pmt=None*, *nrate=None*, *nper=None*, *due=0*, *pyr=1*, *noprint=True*)

Amortization schedule of a loan.

Parameters

- **`pval`** (*`float`*) – present value.
- **`fval`** (*`float`*) – Future value.
- **`pmt`** (*`float`*) – periodic payment per period.
- **`nrate`** (*`float`*) – nominal interest rate per year.
- **`nper`** (*`int`*) – total number of compounding periods.
- **`due`** (*`int`*) – When payments are due.
- **`pyr`** (*`int, list`*) – number of periods per year.
- **`noprint`** (*`bool`*) – prints enhanced output

Returns (principal, interest, payment, balance)

Return type A tuple

Details

The `amortize` function computes and returns the columns of a amortization schedule of a loan. The function returns the interest payment, the principal repayment, the periodic payment and the balance at the end of each period.

Examples

The amortization schedule for a loan of 100, at a interest rate of 10%, and a life of 5 periods, can be computed as:

```
>>> pmt = tvmm(pval=100, nrate=10, nper=5, fval=0)
>>> amortize(pval=100, nrate=10, nper=5, fval=0, noprint=False)
t      Beginning     Periodic     Interest    Principal        Final
       Principal      Payment      Payment    Repayment    Principal
          Amount       Amount                                 Amount
-------------------------------------------------------------------
0          100.00         0.00         0.00         0.00       100.00
1          100.00       -26.38        10.00       -16.38        83.62
2           83.62       -26.38         8.36       -18.02        65.60
3           65.60       -26.38         6.56       -19.82        45.78
4           45.78       -26.38         4.58       -21.80        23.98
5           23.98       -26.38         2.40       -23.98         0.00
```

```
>>> amortize(pval=-100, nrate=10, nper=5, fval=0, noprint=False)
t      Beginning     Periodic     Interest    Principal        Final
       Principal      Payment      Payment    Repayment    Principal
          Amount       Amount                                 Amount
-------------------------------------------------------------------
0         -100.00         0.00         0.00         0.00      -100.00
1         -100.00        26.38       -10.00        16.38       -83.62
2          -83.62        26.38        -8.36        18.02       -65.60
3          -65.60        26.38        -6.56        19.82       -45.78
4          -45.78        26.38        -4.58        21.80       -23.98
5          -23.98        26.38        -2.40        23.98        -0.00
```

In the next example, the argument `due` is used to indicate that the periodic payment occurs at the begining of period.

```
>>> amortize(pval=100, nrate=10, nper=5, fval=0, due=1, noprint=False)
t      Beginning     Periodic     Interest    Principal        Final
       Principal      Payment      Payment    Repayment    Principal
          Amount       Amount                                 Amount
-------------------------------------------------------------------
0          100.00       -23.98         0.00       -23.98        76.02
1           76.02       -23.98         7.60       -16.38        59.64
2           59.64       -23.98         5.96       -18.02        41.62
3           41.62       -23.98         4.16       -19.82        21.80
4           21.80       -23.98         2.18       -21.80         0.00
5            0.00         0.00         0.00         0.00         0.00
```

```
>>> amortize(pval=-100, nrate=10, nper=5, fval=0, due=1, noprint=False)
t      Beginning     Periodic     Interest    Principal        Final
       Principal      Payment      Payment    Repayment    Principal
          Amount       Amount                                 Amount
-------------------------------------------------------------------
0         -100.00        23.98         0.00        23.98       -76.02
1          -76.02        23.98        -7.60        16.38       -59.64
2          -59.64        23.98        -5.96        18.02       -41.62
3          -41.62        23.98        -4.16        19.82       -21.80
4          -21.80        23.98        -2.18        21.80        -0.00
5           -0.00         0.00        -0.00        -0.00        -0.00
```

The function returns a tuple with the columns of the amortization schedule.

```
>>> principal, interest, payment, balance = amortize(pval=100,
... nrate=10, nper=5, fval=0)
```

```
>>> principal
[0, -16.37..., -18.01..., -19.81..., -21.80..., -23.98...]
```

```
>>> interest
[0, 10.0, 8.36..., 6.56..., 4.57..., 2.39...]
```

```
>>> payment
[0, -26.37..., -26.37..., -26.37..., -26.37..., -26.37...]
```

```
>>> balance
[100, 83.62..., 65.60..., 45.78..., 23.98..., 1...]
```

In the following examples, the `sum` function is used to sum of different columns of the amortization schedule.

```
>>> principal, interest, payment, balance = amortize(pval=100,
... nrate=10, nper=5, pmt=pmt)
```

```
>>> sum(interest)
31.89...
```

```
>>> sum(principal)
-99.99...
```

```
>>> principal, interest, payment, balance = amortize(fval=0,
... nrate=10, nper=5, pmt=pmt)
```

```
>>> sum(interest)
31.89...
```

```
>>> sum(principal)
-99.99...
```

```
>>> principal, interest, payment, balance = amortize(pval=100,
... fval=0, nper=5, pmt=pmt)
```

```
>>> sum(interest)
31.89...
```

```
>>> sum(principal)
-99.99...
```

```
>>> amortize(pval=100, fval=0, nrate=10, pmt=pmt, noprint=False)
t      Beginning     Periodic     Interest    Principal        Final
       Principal      Payment      Payment    Repayment    Principal
          Amount       Amount                                 Amount
-------------------------------------------------------------------
0         100.00         0.00         0.00         0.00       100.00
1         100.00       -26.38        10.00       -16.38        83.62
2          83.62       -26.38         8.36       -18.02        65.60
3          65.60       -26.38         6.56       -19.82        45.78
4          45.78       -26.38         4.58       -21.80        23.98
5          23.98       -26.38         2.40       -23.98         0.00
```

```
>>> principal, interest, payment, balance = amortize(pval=100,
... fval=0, nrate=10, pmt=pmt)
```

```
>>> sum(interest)
31.89...
```

```
>>> sum(principal)
-99.99...
```

cashflows.tvmm.**pmtfv**(*pmt=None*, *fval=None*, *nrate=None*, *nper=None*, *pyr=1*, *noprint=True*)

Computes the missing argument (set to None) in a model relating the the future value, the periodic payment, the number of compounding periods and the nominal interest rate in a cashflow.

Parameters

- **pmt** (*float, list*) – Periodic payment.
- **fval** (*float, list*) – Future value.
- **nrate** (*float, list*) – Nominal rate per year.
- **nper** (*int, list*) – Number of compounding periods.
- **pyr** (*int, list*) – number of periods per year.
- **noprint** (*bool*) – prints enhanced output

Returns The value of the parameter set to None in the function call.

Details

The pmtfv function computes and returns the missing value (pmt, fval, nper, nrate) in a model relating a finite sequence of payments made at the beginning or at the end of each period, a future value, and a nominal interest rate. The time intervals between consecutive payments are assumed to be equial. For internal computations, the effective interest rate per period is calculated as nrate / pyr.

This function is used to simplify the call to the tvmm function. See the tvmm function for details.

cashflows.tvmm.**pvfv**(*pval=None*, *fval=None*, *nrate=None*, *nper=None*, *pyr=1*, *noprint=True*)

Computes the missing argument (set to None) in a model relating the present value, the future value, the number of compoundig periods and the nominal interest rate in a cashflow.

Parameters

- **pval** (*float, list*) – Present value.
- **fval** (*float, list*) – Future value.
- **nrate** (*float, list*) – Nominal interest rate per year.
- **nper** (*int, list*) – Number of compounding periods.
- **pyr** (*int, list*) – number of periods per year.
- **noprint** (*bool*) – prints enhanced output

Returns The value of the parameter set to None in the function call.

Details

The pvfv function computes and returns the missing value (fval, pval, nper, nrate) in a model relating these variables. The time intervals between consecutive payments are assumed to be equial. For internal computations, the effective interest rate per period is calculated as nrate / pyr.

This function is used to simplify the call to the tvmm function. See the tvmm function for details.

cashflows.tvmm.**pvpmt**(*pmt=None*, *pval=None*, *nrate=None*, *nper=None*, *pyr=1*, *noprint=True*)

Computes the missing argument (set to None) in a model relating the present value, the periodic payment, the number of compounding periods and the nominal interest rate in a cashflow.

Parameters

- **pmt** (*float, list*) – Periodic payment.
- **pval** (*float, list*) – Present value.
- **nrate** (*float, list*) – Nominal interest rate per year.
- **nper** (*int, list*) – Number of compounding periods.
- **pyr** (*int, list*) – number of periods per year.
- **noprint** (*bool*) – prints enhanced output

Returns The value of the parameter set to None in the function call.

Details

The pvpmt function computes and returns the missing value (pmt, pval, nper, nrate) in a model relating a finite sequence of payments made at the beginning or at the end of each period, a present value, and a nominal interest rate. The time intervals between consecutive payments are assumed to be equial. For internal computations, the effective interest rate per period is calculated as nrate / pyr.

This function is used to simplify the call to the tvmm function. See the tvmm function for details.

cashflows.tvmm.**tvmm**(*pval=None*, *fval=None*, *pmt=None*, *nrate=None*, *nper=None*, *due=0*, *pyr=1*, *noprint=True*)

Computes the missing argument (set to None) in a model relating the present value, the future value, the periodic payment, the number of compounding periods and the nominal interest rate in a cashflow.

Parameters

- **pval** (*float, list*) – Present value.
- **fval** (*float, list*) – Future value.
- **pmt** (*float, list*) – Periodic payment.
- **nrate** (*float, list*) – Nominal interest rate per year.
- **nper** (*int, list*) – Number of compounding periods.
- **due** (*int*) – When payments are due.
- **pyr** (*int, list*) – number of periods per year.
- **noprint** (*bool*) – prints enhanced output

Returns Argument set to None in the function call.

Details

The tvmmm function computes and returns the missing value (pmt, fval, pval, nper, nrate) in a model relating a finite sequence of payments made at the beginning or at the end of each period, a present value, a future value, and a nominal interest rate. The time intervals between consecutive payments are assumed to be equal. For internal computations, the effective interest rate per period is calculated as nrate / pyr.

Examples

This example shows how to find different values for a loan of 5000, with a monthly payment of 130 at the end of the month, a life of 48 periods, and a interest rate of 0.94 per month (equivalent to a nominal interest rate of 11.32%). For a loan, the future value is 0.

- Monthly payments: Note that the parameter pmt is set to None.

```
>>> tvmm(pval=5000, nrate=11.32, nper=48, fval=0, pyr=12)
-130.00...
```

When the parameter `noprint` is set to `False`, a user friendly table with the data computed by the function is print.

```
>>> tvmm(pval=5000, nrate=11.32, nper=48, fval=0, pyr=12, noprint=False)
Present Value: .......  5000.00
Future Value: ........     0.00
Payment: ..............  -130.01
Due: ..................      END
No. of Periods: ......    48.00
Compoundings per Year:    12
Nominal Rate: .......     11.32
Effective Rate: .....     11.93
Periodic Rate: ......      0.94
```

- Future value:

```
>>> tvmm(pval=5000, nrate=11.32, nper=48, pmt=pmt, fval=None, pyr=12)
-0.0...
```

- Present value:

```
>>> tvmm(nrate=11.32, nper=48, pmt=pmt, fval = 0.0, pval=None, pyr=12)
5000...
```

All the arguments support lists as inputs. When a argument is a list and the `noprint` is set to `False`, a table with the data is print.

```
>>> tvmm(pval=[5, 500, 5], nrate=11.32, nper=48, fval=0, pyr=12, noprint=False)
#   pval   fval    pmt   nper  nrate  erate  prate due
-------------------------------------------------------
0   5.00   0.00  -0.13  48.00  11.32  11.93   0.94 END
1 500.00   0.00  -0.13  48.00  11.32  11.93   0.94 END
2   5.00   0.00  -0.13  48.00  11.32  11.93   0.94 END
```

- Interest rate:

```
>>> tvmm(pval=5000, nper=48, pmt=pmt, fval = 0.0, pyr=12)
11.32...
```

- Number of periods:

```
>>> tvmm(pval=5000, nrate=11.32/12, pmt=pmt, fval=0.0)
48.0...
```

1.2 Representation of Cashflows and Interest Rates

1.2.1 Overview

The functions in this module allow the user to create generic cashflows and interest rates as *pandas.Series* objects under the following restrictions:

- Frequency of time series is restricted to the following values: `'A'`, `'BA'`, `'Q'`, `'BQ'`, `'M'`, `'BM'`, `'CBM'`, `'SM'`, `'6M'`, `'6BM'` and `'6CMB'`.
- Interest rates are represented as percentages (not as a fraction).
- Appropriate values must be supplied for the arguments used to create the timestamps of the time series.

Due to generic cashflows and interest rates are *pandas.Series* objects, all available functions for manipulating and transforming pandas time series can be used with this package.

The `cashflow` function returns a *pandas.Series* object that represents a generic cashflow. The user must supply two of the following arguments `start`, `end` and `periods` in order to create the corresponding timestamps for the time series. The generic cashflow is set to the value specified by the argument `const_value`. In addition, when the value of the argument `const_value` is a list, only is necessary to specify the `start` or the `end` dates.

For convenience of the user, point values of the time series can be changed using the argument `chgpts`. In this case, the value passed to this argument is a dictionary where the keys are valid dates and the values are the new values specified for the generic cashflow in these dates.

The `interest_rate` function returns a *pandas.Series* object in the same way as the `cashflow` function. The only difference is that the dictionary passed to the argument `chgpts` specifies change points in the time series, where the value of the interest rate changes for all points ahead.

1.2.2 Functions in this module

`cashflows.timeseries.`**`cashflow`**(*const_value=0*, *start=None*, *end=None*, *periods=None*, *freq='A'*, *chgpts=None*)

Returns a generic cashflow as a pandas.Series object.

Parameters

- **`const_value`** (*`number`*) – constant value for all time series.
- **`start`** (*`string`*) – Date as string using pandas convetion for dates.
- **`end`** (*`string`*) – Date as string using pandas convetion for dates.
- **`peridos`** (*`integer`*) – Length of the time seriesself.
- **`freq`** (*`string`*) – String indicating the period of time series. Valid values are `'A'`, `'BA'`, `'Q'`, `'BQ'`, `'M'`, `'BM'`, `'CBM'`, `'SM'`, `'6M'`, `'6BM'` and `'6CMB'`. See https://pandas.pydata.org/pandas-docs/stable/timeseries.html#timeseries-offset-aliases
- **`chgpts`** (*`dict`*) – Dictionary indicating point changes in the values of the time series.

Returns A pandas time series object.

Examples

A quarterly cashflow with a constant value 1.0 beginning in 2000Q1 can be expressed as:

```
>>> cashflow(const_value=1.0, start='2000Q1', periods=8, freq='Q')
2000Q1    1.0
2000Q2    1.0
2000Q3    1.0
2000Q4    1.0
2001Q1    1.0
2001Q2    1.0
2001Q3    1.0
2001Q4    1.0
Freq: Q-DEC, dtype: float64
```

In the following example, the cashflow function returns a time series object using a list for the `const_value` and a timestamp for the parameter `start`.

```
>>> cashflow(const_value=[10]*10, start='2000Q1', freq='Q')
2000Q1    10.0
2000Q2    10.0
2000Q3    10.0
2000Q4    10.0
2001Q1    10.0
2001Q2    10.0
2001Q3    10.0
2001Q4    10.0
2002Q1    10.0
2002Q2    10.0
Freq: Q-DEC, dtype: float64
```

The following example uses the operator `[]` to modify the value with index equal to 3.

```
>>> x = cashflow(const_value=[0, 1, 2, 3], start='2000Q1', freq='Q')
>>> x[3] = 10
>>> x
2000Q1     0.0
2000Q2     1.0
2000Q3     2.0
2000Q4    10.0
Freq: Q-DEC, dtype: float64
```

```
>>> x[3]
10.0
```

Indexes in the time series also can be specified using a valid timestamp.

```
>>> x['2000Q4'] = 0
>>> x
2000Q1    0.0
2000Q2    1.0
2000Q3    2.0
2000Q4    0.0
Freq: Q-DEC, dtype: float64
```

```
>>> x['2000Q3']
2.0
```

The following example uses the member function `cumsum()` for computing the cumulative sum of the original time series.

```
>>> cashflow(const_value=[0, 1, 2, 3, 4, 5], freq='Q', start='2000Q1').cumsum()
2000Q1     0.0
2000Q2     1.0
2000Q3     3.0
2000Q4     6.0
2001Q1    10.0
2001Q2    15.0
Freq: Q-DEC, dtype: float64
```

In the next examples, a change points are specified using a dictionary. The key can be a integer or a valid timestamp.

```
>>> cashflow(const_value=0, freq='Q', periods=6, start='2000Q1', chgpts={2:10})
2000Q1     0.0
2000Q2     0.0
2000Q3    10.0
2000Q4     0.0
2001Q1     0.0
2001Q2     0.0
Freq: Q-DEC, dtype: float64
```

```
>>> cashflow(const_value=0, freq='Q', periods=6, start='2000Q1', chgpts={'2000Q3
↪':10})
2000Q1     0.0
2000Q2     0.0
2000Q3    10.0
2000Q4     0.0
2001Q1     0.0
2001Q2     0.0
Freq: Q-DEC, dtype: float64
```

cashflows.timeseries.**interest_rate**(*const_value=0*, *start=None*, *end=None*, *periods=None*, *freq='A'*, *chgpts=None*)

Creates a time series object specified as a interest rate.

Parameters

- **const_value** (*number*) – constant value for all time series.
- **start** (*string*) – Date as string using pandas convetion for dates.
- **end** (*string*) – Date as string using pandas convetion for dates.
- **peridos** (*integer*) – Length of the time seriesself.
- **freq** (*string*) – String indicating the period of time series. Valid values are `'A'`, `'BA'`, `'Q'`, `'BQ'`, `'M'`, `'BM'`, `'CBM'`, `'SM'`, `'6M'`, `'6BM'` and `'6CMB'`. See https://pandas.pydata.org/pandas-docs/stable/timeseries.html#timeseries-offset-aliases
- **chgpts** (*dict*) – Dictionary indicating point changes in the values of the time series.

Returns A *pandas.Series* object.

Examples

In the following examples, the argument `chgpts` is used to specify chnages in the value of the interest rate. The keys in the dictionary can be integers or valid timestamps.

```
>>> chgpts = {'2000Q4':10}
>>> interest_rate(const_value=1, start='2000Q1', periods=8, freq='Q',␣
↪chgpts=chgpts)
```

(continues on next page)

(continued from previous page)

```
2000Q1     1.0
2000Q2     1.0
2000Q3     1.0
2000Q4    10.0
2001Q1    10.0
2001Q2    10.0
2001Q3    10.0
2001Q4    10.0
Freq: Q-DEC, dtype: float64
```

```
>>> chgpts = {'2000Q4':10, '2001Q2':20}
>>> interest_rate(const_value=1, start='2000Q1', periods=8, freq='Q',␣
↪chgpts=chgpts)
2000Q1     1.0
2000Q2     1.0
2000Q3     1.0
2000Q4    10.0
2001Q1    10.0
2001Q2    20.0
2001Q3    20.0
2001Q4    20.0
Freq: Q-DEC, dtype: float64
```

```
>>> chgpts = {3:10}
>>> interest_rate(const_value=1, start='2000Q1', periods=8, freq='Q',␣
↪chgpts=chgpts)
2000Q1     1.0
2000Q2     1.0
2000Q3     1.0
2000Q4    10.0
2001Q1    10.0
2001Q2    10.0
2001Q3    10.0
2001Q4    10.0
Freq: Q-DEC, dtype: float64
```

```
>>> chgpts = {3:10, 6:20}
>>> interest_rate(const_value=1, start='2000Q1', periods=8, freq='Q',␣
↪chgpts=chgpts)
2000Q1     1.0
2000Q2     1.0
2000Q3     1.0
2000Q4    10.0
2001Q1    10.0
2001Q2    10.0
2001Q3    20.0
2001Q4    20.0
Freq: Q-DEC, dtype: float64
```

The parameter `const_value` can be a list of numbers. In this case, only is necessary to specify the `start` or `end` arguments.

```
>>> interest_rate(const_value=[10]*12, start='2000-1', freq='M')
2000-01    10.0
2000-02    10.0
```

(continues on next page)

(continued from previous page)

```
2000-03    10.0
2000-04    10.0
2000-05    10.0
2000-06    10.0
2000-07    10.0
2000-08    10.0
2000-09    10.0
2000-10    10.0
2000-11    10.0
2000-12    10.0
Freq: M, dtype: float64
```

cashflows.timeseries.**period2pos**(*index*, *date*)

Returns the position (index) of a timestamp vector.

Parameters

- **index** (*list*) – timestamp vector.
- **date** (*string*) – date to search.

Returns position of date in index.

Return type position (int)

cashflows.timeseries.**textplot**(*cflo*)

Text plot of a generic cashflow.

Parameters **cflo** (*pandas.Series*) – Generic cashflow.

Returns None.

Example

```
>>> cflo = cashflow(const_value=[-10, 5, 0, 20] * 3, start='2000Q1', freq='Q')
>>> textplot(cflo)
time      value +------------------+------------------+
2000Q1   -10.00           **********
2000Q2     5.00                     *****
2000Q3     0.00                     *
2000Q4    20.00                     ********************
2001Q1   -10.00           **********
2001Q2     5.00                     *****
2001Q3     0.00                     *
2001Q4    20.00                     ********************
2002Q1   -10.00           **********
2002Q2     5.00                     *****
2002Q3     0.00                     *
2002Q4    20.00                     ********************
```

cashflows.timeseries.**verify_period_range**(*x*)

Verify if all time series in a list have the same timestamp.

Parameters **x** (*list*) – list of *pandas.Series*.

Returns None.

1.3 Interest rate transformations

1.3.1 Overview

Transformations between nominal, effective and periodic interest rates can be realized using **cashflows**. This module implements the following functions:

- `effrate`: computes the effective interest rate given the nominal interest rate or the periodic interest rate.
- `nomrate`: computes the nominal interest rate given the effective interest rate or the periodic interest rate.
- `perrate`: computes the periodic interest rate given the effective interest rate or the nominal interest rate.

In addition, it is possible to compute discount and compounidng factors.

- `to_discount_factor`: Returns a list of discount factors calculated as 1 / (1 + r)^(t - t0).
- `to_compound_factor`: Returns a list of compounding factors calculated as (1 + r)^(t - t0).

Finally, also it is possible to compute a fixed equivalent rate given interest rate changing over time using `equivalent_rate`.

1.3.2 Functions in this module

`cashflows.rate.`**`effrate`**(*nrate=None*, *prate=None*, *pyr=1*)

Computes the effective interest rate given the nominal interest rate or the periodic interest rate.

Parameters

- **`nrate`** (*`float, pandas.Series`*) – Nominal interest rate.
- **`prate`** (*`float, pandas.Series`*) – Periodic interest rate.
- **`pyr`** (*`int`*) – Number of compounding periods per year.

Returns Effective interest rate(float, pandas.Series).

Examples

In this example, the equivalent effective interest rate for a periodic monthly interest rate of 1% is computed.

```
>>> effrate(prate=1, pyr=12)
12.68...
```

This example shows hot to compute the effective interes rate equivalent to a nominal interest rate of 10% with montlhy compounding.

```
>>> effrate(nrate=10, pyr=12)
10.4713...
```

Also it is possible to use list for some arguments of the functions as it is shown bellow.

```
>>> effrate(prate=1, pyr=[3, 6, 12])
0     3.030100
1     6.152015
2    12.682503
dtype: float64
```

```
>>> effrate(nrate=10, pyr=[3, 6, 12])
0    10.337037
1    10.426042
2    10.471307
dtype: float64
```

```
>>> effrate(prate=[1, 2, 3], pyr=12)
0    12.682503
1    26.824179
2    42.576089
dtype: float64
```

```
>>> effrate(nrate=[10, 12, 14], pyr=12)
0    10.471307
1    12.682503
2    14.934203
dtype: float64
```

When a rate and the number of compounding periods (`pyr`) are vectors, they must have the same length. Computations are executed using the first rate with the first compounding and so on.

```
>>> effrate(nrate=[10, 12, 14], pyr=[3, 6, 12])
0    10.337037
1    12.616242
2    14.934203
dtype: float64
```

```
>>> effrate(prate=[1, 2, 3], pyr=[3, 6, 12])
0     3.030100
1    12.616242
2    42.576089
dtype: float64
```

Also it is possible to transform interest rate time series.

```
>>> nrate = interest_rate(const_value=12, start='2000-06', periods=12, freq='6M')
>>> prate = perrate(nrate=nrate)
>>> effrate(nrate = nrate)
2000-06    12.36
2000-12    12.36
2001-06    12.36
2001-12    12.36
2002-06    12.36
2002-12    12.36
2003-06    12.36
2003-12    12.36
2004-06    12.36
2004-12    12.36
2005-06    12.36
2005-12    12.36
Freq: 6M, dtype: float64
```

```
>>> effrate(prate = prate)
2000-06    12.36
2000-12    12.36
2001-06    12.36
```

(continues on next page)

(continued from previous page)

```
2001-12    12.36
2002-06    12.36
2002-12    12.36
2003-06    12.36
2003-12    12.36
2004-06    12.36
2004-12    12.36
2005-06    12.36
2005-12    12.36
Freq: 6M, dtype: float64
```

cashflows.rate.**equivalent_rate**(*nrate=None*, *erate=None*, *prate=None*)

Returns the equivalent interest rate over a time period.

Parameters

- **nrate** (*TimeSeries*) – Nominal interest rate per year.
- **erate** (*TimeSeries*) – Effective interest rate per year.
- **prate** (*TimeSeries*) – Periodic interest rate.

Returns float value.

Only one of the interest rate must be supplied for the computation.

Example

In this example, the equivalent rate for a periodic interest rate of 10% is computed.

```
>>> equivalent_rate(prate=interest_rate([10]*5, start='2000Q1', freq='Q'))
10.0...
```

cashflows.rate.**nomrate**(*erate=None*, *prate=None*, *pyr=1*)

Computes the nominal interest rate given the nominal interest rate or the periodic interest rate.

Parameters

- **erate** (*float, pandas.Series*) – Effective interest rate.
- **prate** (*float, pandas.Series*) – Periodic interest rate.
- **pyr** (*int*) – Number of compounding periods per year.

Returns Nominal interest rate(float, pandas.Series).

Examples

```
>>> nomrate(prate=1, pyr=12)
12.0
```

```
>>> nomrate(erate=10, pyr=[3, 6, 12])
0    9.684035
1    9.607121
2    9.568969
dtype: float64
```

```
>>> nomrate(erate=10, pyr=12)
9.5689...
```

```
>>> nomrate(prate=1, pyr=[3, 6, 12])
0     3.0
1     6.0
2    12.0
dtype: float64
```

```
>>> nomrate(erate=[10, 12, 14], pyr=12)
0     9.568969
1    11.386552
2    13.174622
dtype: float64
```

```
>>> nomrate(prate=[1, 2, 3], pyr=12)
0    12.0
1    24.0
2    36.0
dtype: float64
```

When a rate and the number of compounding periods (*pyr*) are vectors, they must have the same length. Computations are executed using the first rate with the first compounding and so on.

```
>>> nomrate(erate=[10, 12, 14], pyr=[3, 6, 12])
0     9.684035
1    11.440574
2    13.174622
dtype: float64
```

```
>>> nomrate(prate=[1, 2, 3], pyr=[3, 6, 12])
0     3.0
1    12.0
2    36.0
dtype: float64
```

```
>>> prate = interest_rate(const_value=6.00, start='2000-06', periods=12, freq='6M
↪')
>>> erate = effrate(prate=prate)
>>> nomrate(erate=erate)
2000-06    12.0
2000-12    12.0
2001-06    12.0
2001-12    12.0
2002-06    12.0
2002-12    12.0
2003-06    12.0
2003-12    12.0
2004-06    12.0
2004-12    12.0
2005-06    12.0
2005-12    12.0
Freq: 6M, dtype: float64
```

```
>>> nomrate(prate=prate)
2000-06    12.0
2000-12    12.0
2001-06    12.0
```

(continues on next page)

(continued from previous page)

```
2001-12    12.0
2002-06    12.0
2002-12    12.0
2003-06    12.0
2003-12    12.0
2004-06    12.0
2004-12    12.0
2005-06    12.0
2005-12    12.0
Freq: 6M, dtype: float64
```

cashflows.rate.**perrate**(*nrate=None*, *erate=None*, *pyr=1*)

Computes the periodic interest rate given the nominal interest rate or the effective interest rate.

Parameters

- **nrate** (*float, pandas.Series*) – Nominal interest rate.
- **erate** (*float, pandas.Series*) – Effective interest rate.
- **pyr** (*int*) – Number of compounding periods per year.

Returns Periodic interest rate(float, pandas.Series).

Examples

```
>>> perrate(nrate=10, pyr=12)
0.8333...
```

```
>>> perrate(erate=10, pyr=12)
0.7974...
```

```
>>> perrate(erate=10, pyr=[3, 6, 12])
0    3.228012
1    1.601187
2    0.797414
dtype: float64
```

```
>>> perrate(nrate=10, pyr=[3, 6, 12])
0    3.333333
1    1.666667
2    0.833333
dtype: float64
```

```
>>> perrate(erate=[10, 12, 14], pyr=12)
0    0.797414
1    0.948879
2    1.097885
dtype: float64
```

```
>>> perrate(nrate=[10, 12, 14], pyr=12)
0    0.833333
1    1.000000
2    1.166667
dtype: float64
```

When a rate and the number of compounding periods (`pyr`) are vectors, they must have the same length. Computations are executed using the first rate with the first compounding and so on.

```
>>> perrate(erate=[10, 12, 14], pyr=[3, 6, 12])
0    3.228012
1    1.906762
2    1.097885
dtype: float64
```

```
>>> perrate(nrate=[10, 12, 14], pyr=[3, 6, 12])
0    3.333333
1    2.000000
2    1.166667
dtype: float64
```

```
>>> nrate = interest_rate(const_value=12.0,  start='2000-06', periods=12, freq='6M
↪')
>>> erate = effrate(nrate=nrate)
>>> perrate(erate=erate)
2000-06    6.0
2000-12    6.0
2001-06    6.0
2001-12    6.0
2002-06    6.0
2002-12    6.0
2003-06    6.0
2003-12    6.0
2004-06    6.0
2004-12    6.0
2005-06    6.0
2005-12    6.0
Freq: 6M, dtype: float64
```

```
>>> perrate(nrate=nrate)
2000-06    6.0
2000-12    6.0
2001-06    6.0
2001-12    6.0
2002-06    6.0
2002-12    6.0
2003-06    6.0
2003-12    6.0
2004-06    6.0
2004-12    6.0
2005-06    6.0
2005-12    6.0
Freq: 6M, dtype: float64
```

cashflows.rate.**to_compound_factor**(*nrate=None*, *erate=None*, *prate=None*, *base_date=0*)

Returns a list of compounding factors calculated as (1 + r)^(t - t0).

Parameters

- **nrate** (*TimeSeries*) – Nominal interest rate per year.
- **nrate** – Effective interest rate per year.
- **prate** (*TimeSeries*) – Periodic interest rate.
- **base_date** (*int, tuple*) – basis time.

Returns Compound factor (list)

Example

In this example, a compound factor is computed for a interest rate expressed as nominal, periodic or effective interest rate.

```
>>> nrate = interest_rate(const_value=4, start='2000', periods=10, freq='Q')
>>> erate = effrate(nrate=nrate)
>>> prate = perrate(nrate=nrate)
>>> to_compound_factor(prate=prate, base_date=2)
[0.980..., 0.990..., 1.0, 1.01, 1.0201, 1.030..., 1.040..., 1.051..., 1.061..., 1.
↪072...]
```

```
>>> to_compound_factor(nrate=nrate, base_date=2)
[0.980..., 0.990..., 1.0, 1.01, 1.0201, 1.030..., 1.040..., 1.051..., 1.061..., 1.
↪072...]
```

```
>>> to_compound_factor(erate=erate, base_date=2)
[0.980..., 0.990..., 1.0, 1.01, 1.0201, 1.030..., 1.040..., 1.051..., 1.061..., 1.
↪072...]
```

cashflows.rate.**to_discount_factor**(*nrate=None*, *erate=None*, *prate=None*, *base_date=None*)

Returns a list of discount factors calculated as 1 / (1 + r)^(t - t0).

Parameters

- **nrate** (*pandas.Series*) – Nominal interest rate per year.
- **nrate** – Effective interest rate per year.
- **prate** (*pandas.Series*) – Periodic interest rate.
- **base_date** (*string*) – basis time.

Returns *pandas.Series* of float values.

Only one of the interest rates must be supplied for the computation.

Example

In this example, a discount factor is computed for a interest rate expressed as nominal, periodic or effective interest rate.

```
>>> nrate = interest_rate(const_value=4, periods=10, start='2016Q1', freq='Q')
>>> erate = effrate(nrate=nrate)
>>> prate = perrate(nrate=nrate)
>>> to_discount_factor(nrate=nrate, base_date='2016Q3')
[1.0201, 1.01, 1.0, 0.990..., 0.980..., 0.970..., 0.960..., 0.951..., 0.942..., 0.
↪932...]
```

```
>>> to_discount_factor(erate=erate, base_date='2016Q3')
[1.0201, 1.01, 1.0, 0.990..., 0.980..., 0.970..., 0.960..., 0.951..., 0.942..., 0.
↪932...]
```

```
>>> to_discount_factor(prate=prate, base_date='2016Q3')
[1.0201, 1.01, 1.0, 0.990..., 0.980..., 0.970..., 0.960..., 0.951..., 0.942..., 0.
↪932...]
```

1.4 After tax cashflow calculation

1.4.1 Overview

The function `after_tax_cashflow` returns a new cashflow object for which the values are taxed. The specified tax rate is only appled to positive values in the cashflow. Negative values are reemplazed by a zero value.

1.4.2 Functions in this module

`cashflows.taxing.`**`after_tax_cashflow`**(*cflo*, *tax_rate*)

Computes the after cashflow for a tax rate. Taxes are not computed for negative values in the cashflow.

Parameters

- **`cflo`** (*`pandas.Series`*) – generic cashflow.
- **`tax_rate`** (*`pandas.Series`*) – periodic income tax rate.

Returns Taxed values (*pandas.Series*)

Example*

```
>>> cflo = cashflow(const_value=[-50] + [100] * 4, start='2010', freq='A')
>>> tax_rate = interest_rate(const_value=[10] * 5, start='2010', freq='A')
>>> after_tax_cashflow(cflo=cflo, tax_rate=tax_rate)
2010     0.0
2011    10.0
2012    10.0
2013    10.0
2014    10.0
Freq: A-DEC, dtype: float64
```

1.5 Currency conversion

1.5.1 Overview

The function `currency_conversion` allows the user to convert an cashflow in a currency to the equivalent cashflow in other currency using the specified `exchange_rate`. In addition, it is possible to include the devaluation of the foreign exchange rate.

1.5.2 Functions in this module

`cashflows.currency.`**`currency_conversion`**(*cflo*, *exchange_rate=1*, *devaluation=None*, *base_date=0*)

Converts a cashflow of dollars to another currency.

Parameters

- **`cflo`** (*`pandas.Series`*) – Generic cashflow.
- **`exchange_rate`** (*`float`*) – Exchange rate at time *base_date*.
- **`devaluation`** (*`pandas.Series`*) – Devaluation rate per compounding period.
- **`base_date`** (*`int`*) – Time index for the *exchange_rate* in current dollars.

Returns A TimeSeries object.

Examples.

```
>>> cflo = cashflow(const_value=[100] * 5, start='2015', freq='A')
>>> devaluation = interest_rate(const_value=[5]*5, start='2015', freq='A')
>>> currency_conversion(cflo=cflo, exchange_rate=2)
2015    200.0
2016    200.0
2017    200.0
2018    200.0
2019    200.0
Freq: A-DEC, dtype: float64
```

```
>>> currency_conversion(cflo=cflo, exchange_rate=2, devaluation=devaluation)
2015    200.00000
2016    210.00000
2017    220.50000
2018    231.52500
2019    243.10125
Freq: A-DEC, dtype: float64
```

```
>>> currency_conversion(cflo=cflo, exchange_rate=2,
... devaluation=devaluation, base_date='2017')
2015    181.405896
2016    190.476190
2017    200.000000
2018    210.000000
2019    220.500000
Freq: A-DEC, dtype: float64
```

1.6 Constant dollar transformations

1.6.1 Overview

The function `const2curr` computes the equivalent generic cashflow in current dollars from a generic cashflow in constant dollars of the date given by `base_date`. `inflation` is the inflation rate per compounding period. `curr2const` computes the inverse transformation.

1.6.2 Functions in this module

`cashflows.inflation.`**`const2curr`**(*cflo*, *inflation*, *base_date=0*)

Converts a cashflow of constant dollars to current dollars of the time *base_date*.

Parameters

- **`cflo`** (*`pandas.Series`*) – Generic cashflow.
- **`inflation`** (*`pandas.Series`*) – Inflation rate per compounding period.
- **`base_date`** (*`int, str`*) – base date.

Returns A cashflow in current money (pandas.Series)

Examples.

```
>>> cflo=cashflow(const_value=[100] * 5, start='2000', freq='A')
>>> inflation=interest_rate(const_value=[10, 10, 20, 20, 20], start='2000', freq=
↪'A')
>>> const2curr(cflo=cflo, inflation=inflation)
2000    100.00
2001    110.00
2002    132.00
2003    158.40
2004    190.08
Freq: A-DEC, dtype: float64
```

```
>>> const2curr(cflo=cflo, inflation=inflation, base_date=0)
2000    100.00
2001    110.00
2002    132.00
2003    158.40
2004    190.08
Freq: A-DEC, dtype: float64
```

```
>>> const2curr(cflo=cflo, inflation=inflation, base_date='2000')
2000    100.00
2001    110.00
2002    132.00
2003    158.40
2004    190.08
Freq: A-DEC, dtype: float64
```

```
>>> const2curr(cflo=cflo, inflation=inflation, base_date=4)
2000     52.609428
2001     57.870370
2002     69.444444
2003     83.333333
2004    100.000000
Freq: A-DEC, dtype: float64
```

```
>>> const2curr(cflo=cflo, inflation=inflation, base_date='2004')
2000     52.609428
2001     57.870370
2002     69.444444
2003     83.333333
2004    100.000000
Freq: A-DEC, dtype: float64
```

cashflows.inflation.**curr2const**(*cflo*, *inflation*, *base_date=0*)

Converts a cashflow of current dollars to constant dollars of the date *base_date*.

Parameters

- **cflo** (*pandas.Series*) – Generic cashflow.
- **inflation_rate** (*float, pandas.Series*) – Inflation rate per compounding period.
- **base_date** (*int*) – base time..

Returns A cashflow in constant dollars

```
>>> cflo = cashflow(const_value=[100] * 5, start='2015', freq='A')
>>> inflation = interest_rate(const_value=[10, 10, 20, 20, 20], start='2015',␣
↪freq='A')
>>> curr2const(cflo=cflo, inflation=inflation)
2015    100.000000
2016     90.909091
2017     75.757576
2018     63.131313
2019     52.609428
Freq: A-DEC, dtype: float64
```

```
>>> curr2const(cflo=cflo, inflation=inflation, base_date=4)
2015    190.08
2016    172.80
2017    144.00
2018    120.00
2019    100.00
Freq: A-DEC, dtype: float64
```

```
>>> curr2const(cflo=cflo, inflation=inflation, base_date='2017')
2015    132.000000
2016    120.000000
2017    100.000000
2018     83.333333
2019     69.444444
Freq: A-DEC, dtype: float64
```

1.7 Analysis of cashflows

1.7.1 Overview

This module implements the following functions for financial analysis of cashflows:

- `timevalue`: computes the equivalent net value of a cashflow in a specified time moment.
- `net_uniform_series`: computes the periodic equivalent net value of a cashflow for a specified number of payments.
- `benefit_cost_ratio`: computes the benefit cost ratio of a cashflow using a periodic interest rate for discounting the cashflow.
- `irr`: calculates the periodic internal rate of return of a cashflow.
- `mirr`: calculates the periodic modified internal rate of return of a cashflow.
- `list_as_table`: prints a list as a table. This function is useful for comparing financial indicators for different alternatives.

1.7.2 Functions in this module

`cashflows.analysis.`**`benefit_cost_ratio`**(*cflo*, *prate*, *base_date=0*)

Computes a benefit cost ratio at time *base_date* of a discounted cashflow using the periodic interest rate *prate*.

Parameters

- **prate** (*float, pandas.Series*) – Periodic interest rate.
- **cflo** (*pandas.Series*) – Generic cashflow.
- **base_date** (*int, list*) – Time.

Returns Float or list of floats.

Examples.

```
>>> prate = interest_rate([2]*9, start='2000Q1', freq='Q')
>>> cflo = cashflow([-717.01] + [100]*8, start='2000Q1', freq='Q')
>>> benefit_cost_ratio(cflo, prate)
1.02...
```

```
>>> prate = interest_rate([12]*5, start='2000Q1', freq='Q')
>>> cflo = cashflow([-200] + [100]*4, start='2000Q1', freq='Q')
>>> benefit_cost_ratio(cflo, prate)
1.518...
```

```
>>> benefit_cost_ratio([cflo, cflo], prate)
0    1.518675
1    1.518675
dtype: float64
```

cashflows.analysis.**irr**(*cflo*)

Computes the internal rate of return of a generic cashflow as a periodic interest rate.

Parameters **cflo** (*pandas.Series*) – Generic cashflow.

Returns Float or list of floats.

Examples.

```
>>> cflo = cashflow([-200] + [100]*4, start='2000Q1', freq='Q')
>>> irr(cflo)
34.90...
```

```
>>> irr([cflo, cflo])
0    34.90...
1    34.90...
dtype: float64
```

cashflows.analysis.**mirr**(*cflo*, *finance_rate=0*, *reinvest_rate=0*)

Computes the modified internal rate of return of a generic cashflow as a periodic interest rate.

Parameters

- **cflo** (*pandas.Series*) – Generic cashflow.
- **finance_rate** (*float*) – Periodic interest rate applied to negative values of the cashflow.
- **reinvest_rate** (*float*) – Periodic interest rate applied to positive values of the cashflow.

Returns Float or list of floats.

Examples.

```
>>> cflo = cashflow([-200] + [100]*4, start='2000Q1', freq='Q')
>>> mirr(cflo)
18.92...
```

```
>>> mirr([cflo, cflo])
0    18.920712
1    18.920712
dtype: float64
```

cashflows.analysis.**net_uniform_series**(*cflo*, *prate*, *nper=1*)

Computes a net uniform series equivalent of a cashflow. This is, a fixed periodic payment during *nper* periods that is equivalent to the cashflow *cflo* at the periodic interest rate *prate*.

Parameters

- **cflo** (*pandas.Series*) – Generic cashflow.
- **prate** (*pandas.Series*) – Periodic interest rate.
- **nper** (*int, list*) – Number of equivalent payment periods.

Returns Float or list of floats.

Examples.

```
>>> prate = interest_rate([2]*9, start='2000Q1', freq='Q')
>>> cflo = cashflow([-732.54] + [100]*8, start='2000Q1', freq='Q')
>>> net_uniform_series(cflo, prate)
0.00...
```

```
>>> prate = interest_rate([12]*5, start='2000Q1', freq='Q')
>>> cflo = cashflow([-200] + [100]*4, start='2000Q1', freq='Q')
>>> net_uniform_series(cflo, prate)
116.18...
```

```
>>> net_uniform_series([cflo, cflo], prate)
0    116.183127
1    116.183127
dtype: float64
```

cashflows.analysis.**timevalue**(*cflo*, *prate*, *base_date=0*, *utility=None*)

Computes the equivalent net value of a generic cashflow at time *base_date* using the periodic interest rate *prate*. If *base_date* is 0, *timevalue* computes the net present value of the cashflow. If *base_date* is the index of the last element of *cflo*, this function computes the equivalent future value.

Parameters

- **cflo** (*pandas.Series, list of pandas.Series*) – Generic cashflow.
- **prate** (*pandas.Series*) – Periodic interest rate.
- **base_date** (*int, tuple*) – Time.
- **utility** (*function*) – Utility function.

Returns Float or list of floats.

Examples.

```
>>> cflo = cashflow([-732.54] + [100]*8, start='2000Q1', freq='Q')
>>> prate = interest_rate([2]*9, start='2000Q1', freq='Q')
>>> timevalue(cflo, prate)
0.00...
```

```
>>> prate = interest_rate([12]*5, start='2000Q1', freq='Q')
>>> cflo = cashflow([-200]+[100]*4, start='2000Q1', freq='Q')
>>> timevalue(cflo, prate)
103.73...
```

```
>>> timevalue(cflo, prate, 4)
163.22...
```

```
>>> prate = interest_rate([12]*5, start='2000Q1', freq='Q')
>>> cflo = cashflow([-200] + [100]*4, start='2000Q1', freq='Q')
>>> timevalue(cflo=cflo, prate=prate)
103.73...
```

```
>>> timevalue(cflo=[cflo, cflo], prate=prate)
0    103.734935
1    103.734935
dtype: float64
```

1.8 Bond Valuation

1.8.1 Overview

This module computes the present value or the yield-to-maturity of of the expected cashflow of a bond. Also,it is possible to make a sensibility analysis for different values for the yield-to-maturity and one present value of the bond.

1.8.2 Functions in this module

cashflows.bond.**bond**(*maturity_date=None*, *freq='A'*, *face_value=None*, *coupon_rate=None*, *coupon_value=None*, *num_coupons=None*, *value=None*, *ytm=None*)

Parameters

- **face_value** (*float, list*) – bond's value at maturity.
- **coupon_value** (*float*) – amount of money you receive periodically as the bond matures.
- **num_coupons** (*int, list*) – number of coupons before maturity.
- **ytm** (*float*) – yield to maturity.
- **coupon_rate** (*float, list*) – rate of the face value that defines the coupon value.

Returns Float or list of floats.

Examples:

```
>>> bond(face_value=1000, coupon_value=56, num_coupons=10, ytm=5.6)
1000.0...
```

```
>>> bond(face_value=1000, coupon_rate=5.6, num_coupons=10, value=1000)
5.6...
```

```
>>> bond(face_value=[1000, 1200, 1400], coupon_value=56, num_coupons=10,␣
↪value=1000)
   Coupon_Rate  Coupon_Value  Face_Value  Num_Coupons  Value      YTM
0     5.600000            56        1000           10   1000  5.60000
1     4.666667            56        1200           10   1000  7.04451
2     4.000000            56        1400           10   1000  8.31956
```

```
>>> bond(face_value=1000, coupon_value=[56., 57, 58], num_coupons=10, value=1000)
   Coupon_Rate  Coupon_Value  Face_Value  Num_Coupons  Value  YTM
0          5.6          56.0        1000           10   1000  5.6
1          5.7          57.0        1000           10   1000  5.7
2          5.8          58.0        1000           10   1000  5.8
```

```
>>> bond(face_value=1000, coupon_rate=[5.6, 5.7, 5.8], num_coupons=10, value=1000)
   Coupon_Rate  Coupon_Value  Face_Value  Num_Coupons  Value  YTM
0          5.6          56.0        1000           10   1000  5.6
1          5.7          57.0        1000           10   1000  5.7
2          5.8          58.0        1000           10   1000  5.8
```

```
>>> bond(face_value=1000, coupon_rate=5.6, num_coupons=[10, 20, 30], value=1000)
   Coupon_Rate  Coupon_Value  Face_Value  Num_Coupons  Value  YTM
0          5.6          56.0        1000           10   1000  5.6
1          5.6          56.0        1000           20   1000  5.6
2          5.6          56.0        1000           30   1000  5.6
```

```
>>> bond(face_value=1000, coupon_rate=5.6, num_coupons=10, value=[800, 900, 1000])
   Coupon_Rate  Coupon_Value  Face_Value  Num_Coupons  Value      YTM
0          5.6          56.0        1000           10    800  8.671484
1          5.6          56.0        1000           10    900  7.025450
2          5.6          56.0        1000           10   1000  5.600000
```

```
>>> bond(face_value=[1000, 1100, 1200], coupon_rate=5.6, num_coupons=10,␣
↪value=[800, 900, 1000])
   Coupon_Rate  Coupon_Value  Face_Value  Num_Coupons  Value       YTM
0     5.600000          56.0        1000           10    800  8.671484
1     5.600000          56.0        1000           10    900  7.025450
2     5.600000          56.0        1000           10   1000  5.600000
3     5.090909          56.0        1100           10    800  9.419301
4     5.090909          56.0        1100           10    900  7.772838
5     5.090909          56.0        1100           10   1000  6.346424
6     4.666667          56.0        1200           10    800 10.119360
7     4.666667          56.0        1200           10    900  8.472129
8     4.666667          56.0        1200           10   1000  7.044510
```

```
>>> bond(face_value=[1000, 1100, 1200], coupon_rate=[5.6, 5.7, 5.8], num_
↪coupons=10, value=[800, 900, 1000])
    Coupon_Rate  Coupon_Value  Face_Value  Num_Coupons  Value       YTM
0      5.600000          56.0        1000           10    800  8.671484
1      5.600000          56.0        1000           10    900  7.025450
2      5.600000          56.0        1000           10   1000  5.600000
3      5.700000          57.0        1000           10    800  8.787284
4      5.700000          57.0        1000           10    900  7.132508
```

(continues on next page)

(continued from previous page)

```
5      5.700000          57.0         1000           10   1000   5.700000
6      5.800000          58.0         1000           10    800   8.903126
7      5.800000          58.0         1000           10    900   7.239584
8      5.800000          58.0         1000           10   1000   5.800000
9      5.090909          56.0         1100           10    800   9.419301
10     5.090909          56.0         1100           10    900   7.772838
11     5.090909          56.0         1100           10   1000   6.346424
12     5.181818          57.0         1100           10    800   9.531367
13     5.181818          57.0         1100           10    900   7.876353
14     5.181818          57.0         1100           10   1000   6.443037
15     5.272727          58.0         1100           10    800   9.643488
16     5.272727          58.0         1100           10    900   7.979898
17     5.272727          58.0         1100           10   1000   6.539664
18     4.666667          56.0         1200           10    800  10.119360
19     4.666667          56.0         1200           10    900   8.472129
20     4.666667          56.0         1200           10   1000   7.044510
21     4.750000          57.0         1200           10    800  10.228122
22     4.750000          57.0         1200           10    900   8.572512
23     4.750000          57.0         1200           10   1000   7.138133
24     4.833333          58.0         1200           10    800  10.336951
25     4.833333          58.0         1200           10    900   8.672936
26     4.833333          58.0         1200           10   1000   7.231779
```

```
>>> bond(face_value=1000, coupon_rate=5.6, num_coupons=10, value=1000, ytm=[5.1,␣
↪5.6, 6.1])
   Basis_Value        Change  Coupon_Rate  Coupon_Value  Face_Value  \
0         1000  3.842187e+00          5.6          56.0        1000
1         1000  1.136868e-14          5.6          56.0        1000
2         1000 -3.662671e+00          5.6          56.0        1000

   Num_Coupons        Value  YTM
0           10  1038.421866  5.1
1           10  1000.000000  5.6
2           10   963.373290  6.1
```

```
>>> bond(face_value=1000, coupon_rate=5.6, num_coupons=10, value=[1000, 1100],␣
↪ytm=[5.1, 5.6, 6.1])
   Basis_Value        Change  Coupon_Rate  Coupon_Value  Face_Value  \
0         1000  3.842187e+00          5.6          56.0        1000
1         1000  1.136868e-14          5.6          56.0        1000
2         1000 -3.662671e+00          5.6          56.0        1000
3         1100 -5.598012e+00          5.6          56.0        1000
4         1100 -9.090909e+00          5.6          56.0        1000
5         1100 -1.242061e+01          5.6          56.0        1000

   Num_Coupons        Value  YTM
0           10  1038.421866  5.1
1           10  1000.000000  5.6
2           10   963.373290  6.1
3           10  1038.421866  5.1
4           10  1000.000000  5.6
5           10   963.373290  6.1
```

>>> bond(face_value=1000, coupon_rate=5.6, num_coupons=[20], value=[1000, 1100], ytm=[5.6, 6.1]) # doctest: +ELLIPSIS +NORMALIZE_WHITESPACE # Basis_Value Change Coupon_Rate Coupon_Value Face_Value Num_Coupons # 0 1000 0.000000 5.6 56.0 1000 20 # 1 1000 -5.688693 5.6 56.0 1000 20 # 2

1100 -9.090909 5.6 56.0 1000 20 # 3 1100 -14.262448 5.6 56.0 1000 20 # <BLANKLINE> # Value YTM # 0 1000.000000 5.6 # 1 943.113073 6.1 # 2 1000.000000 5.6 # 3 943.113073 6.1

1.9 Asset depreciation

1.9.1 Overview

This module implements the following functions to compute the depreciation of an asset:

- `depreciation_sl`: Computes the depreciation of an asset using straight line depreciation method.
- `depreciation_soyd`: Computes the depreciation of an asset using the sum-of-year's-digits method.
- `depreciation_db`: Computes the depreciation of an asset using the declining balance method.

1.9.2 Functions in this module

`cashflows.depreciation.`**`depreciation_db`**(*costs*, *life*, *salvalue=None*, *factor=1*, *convert_to_sl=True*, *delay=None*, *noprint=True*)

Computes the depreciation of an asset using the declining balance method.

Parameters

- **`costs`** (*`pandas.Series`*) – the cost per period of the assets.
- **`life`** (*`pandas.Series`*) – number of depreciation periods for the asset.
- **`salvalue`** (*`pandas.Series`*) – salvage value as a percentage of cost.
- **`factor`** (*`float`*) – acelerating factor for depreciation.
- **`convert_to_sl`** (*`bool`*) – converts to straight line method?
- **`noprint`** (*`bool`*) – when True, the procedure prints a depreciation table.

Returns Returns a pandas DataFrame with the computations.

Examples.

```
>>> costs = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> costs[0] = 1000
>>> life = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> life[0] = 4
>>> depreciation_db(costs=costs, life=life, factor=1.5, convert_to_sl=False)
        Beg_Book    Depr  Accum_Depr  End_Book
2000Q1   1000.00  375.00      375.00    625.00
2000Q2    625.00  234.38      609.38    390.62
2000Q3    390.62  146.48      755.86    244.14
2000Q4    244.14   91.55      847.41    152.59
2001Q1    152.59    0.00      847.41    152.59
2001Q2    152.59    0.00      847.41    152.59
2001Q3    152.59    0.00      847.41    152.59
2001Q4    152.59    0.00      847.41    152.59
2002Q1    152.59    0.00      847.41    152.59
2002Q2    152.59    0.00      847.41    152.59
2002Q3    152.59    0.00      847.41    152.59
2002Q4    152.59    0.00      847.41    152.59
2003Q1    152.59    0.00      847.41    152.59
```

(continues on next page)

(continued from previous page)

```
2003Q2    152.59    0.00      847.41    152.59
2003Q3    152.59    0.00      847.41    152.59
2003Q4    152.59    0.00      847.41    152.59
```

```
>>> costs = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> costs[0] = 1000
>>> costs[8] = 1000
>>> life = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> life[0] = 4
>>> life[8] = 4
>>> depreciation_db(costs=costs, life=life, factor=1.5, convert_to_sl=False)
        Beg_Book    Depr  Accum_Depr  End_Book
2000Q1   1000.00  375.00      375.00    625.00
2000Q2    625.00  234.38      609.38    390.62
2000Q3    390.62  146.48      755.86    244.14
2000Q4    244.14   91.55      847.41    152.59
2001Q1    152.59    0.00      847.41    152.59
2001Q2    152.59    0.00      847.41    152.59
2001Q3    152.59    0.00      847.41    152.59
2001Q4    152.59    0.00      847.41    152.59
2002Q1   1152.59  375.00     1222.41    777.59
2002Q2    777.59  234.38     1456.79    543.21
2002Q3    543.21  146.48     1603.27    396.73
2002Q4    396.73   91.55     1694.82    305.18
2003Q1    305.18    0.00     1694.82    305.18
2003Q2    305.18    0.00     1694.82    305.18
2003Q3    305.18    0.00     1694.82    305.18
2003Q4    305.18    0.00     1694.82    305.18
```

cashflows.depreciation.**depreciation_sl**(*costs*, *life*, *salvalue=None*)

Computes the depreciation of an asset using straight line depreciation method.

Parameters

- **costs** (*pandas.Series*) – the cost per period of the assets.
- **life** (*pandas.Series*) – number of depreciation periods for the asset.
- **salvalue** (*pandas.Series*) – salvage value as a percentage of cost.

Returns Returns a pandas DataFrame with the computations.

Examples.

```
>>> costs = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> costs[0] = 1000
>>> life = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> life[0] = 4
>>> depreciation_sl(costs=costs, life=life)
           Beg_Book   Depr  Accum_Depr  End_Book
   2000Q1    1000.0  250.0       250.0     750.0
   2000Q2     750.0  250.0       500.0     500.0
   2000Q3     500.0  250.0       750.0     250.0
   2000Q4     250.0  250.0      1000.0       0.0
   2001Q1       0.0    0.0      1000.0       0.0
   2001Q2       0.0    0.0      1000.0       0.0
   2001Q3       0.0    0.0      1000.0       0.0
   2001Q4       0.0    0.0      1000.0       0.0
```

(continues on next page)

(continued from previous page)

```
   2002Q1       0.0    0.0      1000.0       0.0
   2002Q2       0.0    0.0      1000.0       0.0
   2002Q3       0.0    0.0      1000.0       0.0
   2002Q4       0.0    0.0      1000.0       0.0
   2003Q1       0.0    0.0      1000.0       0.0
   2003Q2       0.0    0.0      1000.0       0.0
   2003Q3       0.0    0.0      1000.0       0.0
   2003Q4       0.0    0.0      1000.0       0.0
```

```
>>> costs = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> costs[0] = 1000
>>> costs[8] = 1000
>>> life = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> life[0] = 4
>>> life[8] = 4
>>> depreciation_sl(costs=costs, life=life)
          Beg_Book   Depr  Accum_Depr  End_Book
   2000Q1    1000.0  250.0       250.0     750.0
   2000Q2     750.0  250.0       500.0     500.0
   2000Q3     500.0  250.0       750.0     250.0
   2000Q4     250.0  250.0      1000.0       0.0
   2001Q1       0.0    0.0      1000.0       0.0
   2001Q2       0.0    0.0      1000.0       0.0
   2001Q3       0.0    0.0      1000.0       0.0
   2001Q4       0.0    0.0      1000.0       0.0
   2002Q1    1000.0  250.0      1250.0     750.0
   2002Q2     750.0  250.0      1500.0     500.0
   2002Q3     500.0  250.0      1750.0     250.0
   2002Q4     250.0  250.0      2000.0       0.0
   2003Q1       0.0    0.0      2000.0       0.0
   2003Q2       0.0    0.0      2000.0       0.0
   2003Q3       0.0    0.0      2000.0       0.0
   2003Q4       0.0    0.0      2000.0       0.0
```

cashflows.depreciation.**depreciation_soyd**(*costs*, *life*, *salvalue=None*)

Computes the depreciation of an asset using the sum-of-year's-digits method.

Parameters

- **costs** (*pandas.Series*) – the cost per period of the assets.
- **life** (*pandas.Series*) – number of depreciation periods for the asset.
- **salvalue** (*pandas.Series*) – salvage value as a percentage of cost.

Returns Returns a pandas DataFrame with the computations.

Examples.

```
>>> costs = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> costs[0] = 1000
>>> life = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> life[0] = 4
>>> depreciation_soyd(costs=costs, life=life)
          Beg_Book   Depr  Accum_Depr  End_Book
   2000Q1    1000.0  400.0       400.0     600.0
   2000Q2     600.0  300.0       700.0     300.0
   2000Q3     300.0  200.0       900.0     100.0
```

(continues on next page)

(continued from previous page)

```
   2000Q4     100.0  100.0      1000.0       0.0
   2001Q1       0.0    0.0      1000.0       0.0
   2001Q2       0.0    0.0      1000.0       0.0
   2001Q3       0.0    0.0      1000.0       0.0
   2001Q4       0.0    0.0      1000.0       0.0
   2002Q1       0.0    0.0      1000.0       0.0
   2002Q2       0.0    0.0      1000.0       0.0
   2002Q3       0.0    0.0      1000.0       0.0
   2002Q4       0.0    0.0      1000.0       0.0
   2003Q1       0.0    0.0      1000.0       0.0
   2003Q2       0.0    0.0      1000.0       0.0
   2003Q3       0.0    0.0      1000.0       0.0
   2003Q4       0.0    0.0      1000.0       0.0
```

```
>>> costs = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> costs[0] = 1000
>>> costs[8] = 1000
>>> life = cashflow(const_value=0, periods=16, start='2000Q1', freq='Q')
>>> life[0] = 4
>>> life[8] = 4
>>> depreciation_soyd(costs=costs, life=life)
          Beg_Book   Depr  Accum_Depr  End_Book
   2000Q1    1000.0  400.0       400.0     600.0
   2000Q2     600.0  300.0       700.0     300.0
   2000Q3     300.0  200.0       900.0     100.0
   2000Q4     100.0  100.0      1000.0       0.0
   2001Q1       0.0    0.0      1000.0       0.0
   2001Q2       0.0    0.0      1000.0       0.0
   2001Q3       0.0    0.0      1000.0       0.0
   2001Q4       0.0    0.0      1000.0       0.0
   2002Q1    1000.0  400.0      1400.0     600.0
   2002Q2     600.0  300.0      1700.0     300.0
   2002Q3     300.0  200.0      1900.0     100.0
   2002Q4     100.0  100.0      2000.0       0.0
   2003Q1       0.0    0.0      2000.0       0.0
   2003Q2       0.0    0.0      2000.0       0.0
   2003Q3       0.0    0.0      2000.0       0.0
   2003Q4       0.0    0.0      2000.0       0.0
```

1.10 Loan analysis

1.10.1 Overview

Computes the amorization schedule for the following types of loans:

- `fixed_rate_loan`: In this loan, the interest rate is fixed and the total payments are equal during the life of the loan.
- `buydown_loan`: the interest rate changes during the life of the loan; the value of the payments are calculated using the current value of the interest rate. When the interest rate is constant during the life of the loan, the results are equals to the function `fixed_rate_loan`.
- `fixed_ppal_loan`: the payments to the principal are constant during the life of loan.
- `bullet_loan`: the principal is payed at the end of the life of the loan.

1.10.2 Functions in this module

class cashflows.loan.**Loan**(*life*, *amount*, *grace*, *nrate*, *dispoints=0*, *orgpoints=0*, *data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=False*)

Bases: pandas.core.frame.DataFrame

tocashflow(*tax_rate=None*)

true_rate(*tax_rate=None*)

cashflows.loan.**bullet_loan**(*amount*, *nrate*, *dispoints=0*, *orgpoints=0*, *prepmt=None*)

In this type of loan, the principal is payed at the end for the life of the loan. Periodic payments correspond only to interests.

Parameters

- **amount** (*float*) – Loan amount.
- **nrate** (*float, pandas.Series*) – nominal interest rate per year.
- **dispoints** (*float*) – Discount points of the loan.
- **orgpoints** (*float*) – Origination points of the loan.
- **prepmt** (*pandas.Series*) – generic cashflow representing prepayments.

Returns A object of the class Loan.

```
>>> nrate = interest_rate(const_value=[10]*11, start='2018Q1', freq='Q')
>>> bullet_loan(amount=1000, nrate=nrate, dispoints=0, orgpoints=0, prepmt=None)
Amount:             1000.00
Total interest:     250.00
Total payment:      1250.00
Discount points:    0.00
Origination points: 0.00

        Beg_Ppal_Amount  Nom_Rate  Tot_Payment  Int_Payment  Ppal_Payment  \
2018Q1              0.0      10.0          0.0          0.0           0.0
2018Q2           1000.0      10.0         25.0         25.0           0.0
2018Q3           1000.0      10.0         25.0         25.0           0.0
2018Q4           1000.0      10.0         25.0         25.0           0.0
2019Q1           1000.0      10.0         25.0         25.0           0.0
2019Q2           1000.0      10.0         25.0         25.0           0.0
2019Q3           1000.0      10.0         25.0         25.0           0.0
2019Q4           1000.0      10.0         25.0         25.0           0.0
2020Q1           1000.0      10.0         25.0         25.0           0.0
2020Q2           1000.0      10.0         25.0         25.0           0.0
2020Q3           1000.0      10.0       1025.0         25.0        1000.0

        End_Ppal_Amount
2018Q1           1000.0
2018Q2           1000.0
2018Q3           1000.0
2018Q4           1000.0
2019Q1           1000.0
2019Q2           1000.0
2019Q3           1000.0
2019Q4           1000.0
2020Q1           1000.0
2020Q2           1000.0
2020Q3              0.0
```

cashflows.loan.**buydown_loan**(*amount*, *nrate*, *grace=0*, *dispoints=0*, *orgpoints=0*, *prepmt=None*)

In this loan, the periodic payments are recalculated when there are changes in the value of the interest rate.

Parameters

- **amount** (*float*) – Loan amount.
- **nrate** (*float, pandas.Series*) – nominal interest rate per year.
- **grace** (*int*) – numner of grace periods without paying the principal.
- **dispoints** (*float*) – Discount points of the loan.
- **orgpoints** (*float*) – Origination points of the loan.
- **prepmt** (*pandas.Series*) – generic cashflow representing prepayments.

Returns A object of the class Loan.

```
>>> nrate = interest_rate(const_value=10, start='2016Q1', periods=11, freq='Q',␣
↪chgpts={'2017Q2':20})
>>> buydown_loan(amount=1000, nrate=nrate, dispoints=0, orgpoints=0, prepmt=None)
Amount:             1000.00
Total interest:     200.99
Total payment:      1200.99
Discount points:    0.00
Origination points: 0.00

        Beg_Ppal_Amount  Nom_Rate  Tot_Payment  Int_Payment  Ppal_Payment  \
2016Q1      1000.000000      10.0     0.000000     0.000000      0.000000
2016Q2      1000.000000      10.0   114.258763    25.000000     89.258763
2016Q3       910.741237      10.0   114.258763    22.768531     91.490232
2016Q4       819.251005      10.0   114.258763    20.481275     93.777488
2017Q1       725.473517      10.0   114.258763    18.136838     96.121925
2017Q2       629.351591      20.0   123.993257    31.467580     92.525677
2017Q3       536.825914      20.0   123.993257    26.841296     97.151961
2017Q4       439.673952      20.0   123.993257    21.983698    102.009559
2018Q1       337.664393      20.0   123.993257    16.883220    107.110037
2018Q2       230.554356      20.0   123.993257    11.527718    112.465539
2018Q3       118.088816      20.0   123.993257     5.904441    118.088816

        End_Ppal_Amount
2016Q1     1.000000e+03
2016Q2     9.107412e+02
2016Q3     8.192510e+02
2016Q4     7.254735e+02
2017Q1     6.293516e+02
2017Q2     5.368259e+02
2017Q3     4.396740e+02
2017Q4     3.376644e+02
2018Q1     2.305544e+02
2018Q2     1.180888e+02
2018Q3     1.136868e-13
```

```
>>> pmt = cashflow(const_value=0, start='2016Q1', periods=11, freq='Q')
>>> pmt['2017Q4'] = 200
>>> buydown_loan(amount=1000, nrate=nrate, dispoints=0, orgpoints=0, prepmt=pmt)
Amount:             1000.00
Total interest:     180.67
Total payment:      1180.67
```

(continues on next page)

(continued from previous page)

```
Discount points:    0.00
Origination points: 0.00

        Beg_Ppal_Amount  Nom_Rate  Tot_Payment  Int_Payment  Ppal_Payment  \
2016Q1      1000.000000      10.0     0.000000     0.000000      0.000000
2016Q2      1000.000000      10.0   114.258763    25.000000     89.258763
2016Q3       910.741237      10.0   114.258763    22.768531     91.490232
2016Q4       819.251005      10.0   114.258763    20.481275     93.777488
2017Q1       725.473517      10.0   114.258763    18.136838     96.121925
2017Q2       629.351591      20.0   123.993257    31.467580     92.525677
2017Q3       536.825914      20.0   123.993257    26.841296     97.151961
2017Q4       439.673952      20.0   323.993257    21.983698    302.009559
2018Q1       137.664393      20.0    50.551544     6.883220     43.668324
2018Q2        93.996068      20.0    50.551544     4.699803     45.851741
2018Q3        48.144328      20.0    50.551544     2.407216     48.144328

        End_Ppal_Amount
2016Q1     1.000000e+03
2016Q2     9.107412e+02
2016Q3     8.192510e+02
2016Q4     7.254735e+02
2017Q1     6.293516e+02
2017Q2     5.368259e+02
2017Q3     4.396740e+02
2017Q4     1.376644e+02
2018Q1     9.399607e+01
2018Q2     4.814433e+01
2018Q3     4.263256e-14
```

cashflows.loan.**fixed_ppal_loan**(*amount*, *nrate*, *grace=0*, *dispoints=0*, *orgpoints=0*, *prepmt=None*, *balloonpmt=None*)

Loan with fixed principal payment.

Parameters

- **amount** (*float*) – Loan amount.
- **nrate** (*float, pandas.Series*) – nominal interest rate per year.
- **grace** (*int*) – number of grace periiods without paying principal.
- **dispoints** (*float*) – Discount points of the loan.
- **orgpoints** (*float*) – Origination points of the loan.
- **prepmt** (*pandas.Series*) – generic cashflow representing prepayments.
- **balloonpmt** (*pandas.Series*) – generic cashflow representing balloon payments.

Returns A object of the class Loan.

Examples

```
>>> nrate = interest_rate(const_value=[10]*11, start='2018Q1', freq='Q')
>>> tax_rate = interest_rate(const_value=[35]*11, start='2018Q1', freq='Q')
>>> fixed_ppal_loan(amount=1000, nrate=nrate, grace=0, dispoints=0, orgpoints=0,
...                 prepmt=None, balloonpmt=None)
Amount:             1000.00
Total interest:      137.50
Total payment:      1137.50
```

(continues on next page)

(continued from previous page)

```
Discount points:    0.00
Origination points: 0.00

        Beg_Ppal_Amount  Nom_Rate  Tot_Payment  Int_Payment  Ppal_Payment  \
2018Q1              0.0      10.0          0.0          0.0           0.0
2018Q2           1000.0      10.0        125.0         25.0         100.0
2018Q3            900.0      10.0        122.5         22.5         100.0
2018Q4            800.0      10.0        120.0         20.0         100.0
2019Q1            700.0      10.0        117.5         17.5         100.0
2019Q2            600.0      10.0        115.0         15.0         100.0
2019Q3            500.0      10.0        112.5         12.5         100.0
2019Q4            400.0      10.0        110.0         10.0         100.0
2020Q1            300.0      10.0        107.5          7.5         100.0
2020Q2            200.0      10.0        105.0          5.0         100.0
2020Q3            100.0      10.0        102.5          2.5         100.0

        End_Ppal_Amount
2018Q1           1000.0
2018Q2            900.0
2018Q3            800.0
2018Q4            700.0
2019Q1            600.0
2019Q2            500.0
2019Q3            400.0
2019Q4            300.0
2020Q1            200.0
2020Q2            100.0
2020Q3              0.0
```

```
>>> fixed_ppal_loan(amount=1000, nrate=nrate, grace=2, dispoints=0, orgpoints=0,
...                 prepmt=None, balloonpmt=None)
Amount:             1000.00
Total interest:     162.50
Total payment:      1162.50
Discount points:    0.00
Origination points: 0.00

        Beg_Ppal_Amount  Nom_Rate  Tot_Payment  Int_Payment  Ppal_Payment  \
2018Q1              0.0      10.0        0.000        0.000           0.0
2018Q2           1000.0      10.0       25.000       25.000           0.0
2018Q3           1000.0      10.0       25.000       25.000           0.0
2018Q4           1000.0      10.0      150.000       25.000         125.0
2019Q1            875.0      10.0      146.875       21.875         125.0
2019Q2            750.0      10.0      143.750       18.750         125.0
2019Q3            625.0      10.0      140.625       15.625         125.0
2019Q4            500.0      10.0      137.500       12.500         125.0
2020Q1            375.0      10.0      134.375        9.375         125.0
2020Q2            250.0      10.0      131.250        6.250         125.0
2020Q3            125.0      10.0      128.125        3.125         125.0

        End_Ppal_Amount
2018Q1           1000.0
2018Q2           1000.0
2018Q3           1000.0
2018Q4            875.0
2019Q1            750.0
```

(continues on next page)

(continued from previous page)

```
2019Q2            625.0
2019Q3            500.0
2019Q4            375.0
2020Q1            250.0
2020Q2            125.0
2020Q3              0.0
```

```
>>> pmt = cashflow(const_value=[0]*11, start='2018Q1', freq='Q')
>>> pmt['2019Q4'] = 200
>>> fixed_ppal_loan(amount=1000, nrate=nrate, grace=2, dispoints=0, orgpoints=0,
...                 prepmt=pmt, balloonpmt=None)
Amount:             1000.00
Total interest:      149.38
Total payment:      1149.38
Discount points:    0.00
Origination points: 0.00

        Beg_Ppal_Amount  Nom_Rate  Tot_Payment  Int_Payment  Ppal_Payment  \
2018Q1              0.0      10.0        0.000        0.000           0.0
2018Q2           1000.0      10.0       25.000       25.000           0.0
2018Q3           1000.0      10.0       25.000       25.000           0.0
2018Q4           1000.0      10.0      150.000       25.000         125.0
2019Q1            875.0      10.0      146.875       21.875         125.0
2019Q2            750.0      10.0      143.750       18.750         125.0
2019Q3            625.0      10.0      140.625       15.625         125.0
2019Q4            500.0      10.0      337.500       12.500         325.0
2020Q1            175.0      10.0      129.375        4.375         125.0
2020Q2             50.0      10.0       51.250        1.250          50.0
2020Q3              0.0      10.0        0.000        0.000           0.0

        End_Ppal_Amount
2018Q1           1000.0
2018Q2           1000.0
2018Q3           1000.0
2018Q4            875.0
2019Q1            750.0
2019Q2            625.0
2019Q3            500.0
2019Q4            175.0
2020Q1             50.0
2020Q2              0.0
2020Q3              0.0
```

```
>>> fixed_ppal_loan(amount=1000, nrate=nrate, grace=2, dispoints=0, orgpoints=0,
...                 prepmt=None, balloonpmt=pmt)
Amount:             1000.00
Total interest:      165.00
Total payment:      1165.00
Discount points:    0.00
Origination points: 0.00

        Beg_Ppal_Amount  Nom_Rate  Tot_Payment  Int_Payment  Ppal_Payment  \
2018Q1              0.0      10.0          0.0          0.0           0.0
2018Q2           1000.0      10.0         25.0         25.0           0.0
2018Q3           1000.0      10.0         25.0         25.0           0.0
2018Q4           1000.0      10.0        125.0         25.0         100.0
```

(continues on next page)

(continued from previous page)

```
2019Q1            900.0      10.0        122.5         22.5          100.0
2019Q2            800.0      10.0        120.0         20.0          100.0
2019Q3            700.0      10.0        117.5         17.5          100.0
2019Q4            600.0      10.0        315.0         15.0          300.0
2020Q1            300.0      10.0        107.5          7.5          100.0
2020Q2            200.0      10.0        105.0          5.0          100.0
2020Q3            100.0      10.0        102.5          2.5          100.0

        End_Ppal_Amount
2018Q1           1000.0
2018Q2           1000.0
2018Q3           1000.0
2018Q4            900.0
2019Q1            800.0
2019Q2            700.0
2019Q3            600.0
2019Q4            300.0
2020Q1            200.0
2020Q2            100.0
2020Q3              0.0
```

```
>>> x = fixed_ppal_loan(amount=1000, nrate=nrate, grace=2, dispoints=0,␣
↪orgpoints=0,
...                      prepmt=None, balloonpmt=pmt)
>>> x.true_rate()
10.00...
```

```
>>> x.true_rate(tax_rate)
6.50...
```

```
>>> x.tocashflow()
2018Q1    1000.0
2018Q2     -25.0
2018Q3     -25.0
2018Q4    -125.0
2019Q1    -122.5
2019Q2    -120.0
2019Q3    -117.5
2019Q4    -315.0
2020Q1    -107.5
2020Q2    -105.0
2020Q3    -102.5
Freq: Q-DEC, dtype: float64
```

```
>>> x.tocashflow(tax_rate)
2018Q1    1000.000
2018Q2     -16.250
2018Q3     -16.250
2018Q4    -116.250
2019Q1    -114.625
2019Q2    -113.000
2019Q3    -111.375
2019Q4    -309.750
2020Q1    -104.875
2020Q2    -103.250
```

(continues on next page)

(continued from previous page)

```
2020Q3    -101.625
Freq: Q-DEC, dtype: float64
```

```
>>> x = fixed_ppal_loan(amount=1000, nrate=nrate, grace=2, dispoints=0,␣
↪orgpoints=10,
...                      prepmt=None, balloonpmt=pmt)
>>> x
Amount:             1000.00
Total interest:     165.00
Total payment:      1265.00
Discount points:    0.00
Origination points: 10.00

        Beg_Ppal_Amount  Nom_Rate  Tot_Payment  Int_Payment  Ppal_Payment  \
2018Q1              0.0      10.0        100.0          0.0           0.0
2018Q2           1000.0      10.0         25.0         25.0           0.0
2018Q3           1000.0      10.0         25.0         25.0           0.0
2018Q4           1000.0      10.0        125.0         25.0         100.0
2019Q1            900.0      10.0        122.5         22.5         100.0
2019Q2            800.0      10.0        120.0         20.0         100.0
2019Q3            700.0      10.0        117.5         17.5         100.0
2019Q4            600.0      10.0        315.0         15.0         300.0
2020Q1            300.0      10.0        107.5          7.5         100.0
2020Q2            200.0      10.0        105.0          5.0         100.0
2020Q3            100.0      10.0        102.5          2.5         100.0

        End_Ppal_Amount
2018Q1           1000.0
2018Q2           1000.0
2018Q3           1000.0
2018Q4            900.0
2019Q1            800.0
2019Q2            700.0
2019Q3            600.0
2019Q4            300.0
2020Q1            200.0
2020Q2            100.0
2020Q3              0.0
```

```
>>> x.true_rate()
17.1725...
```

```
>>> x.tocashflow()
2018Q1    900.0
2018Q2    -25.0
2018Q3    -25.0
2018Q4   -125.0
2019Q1   -122.5
2019Q2   -120.0
2019Q3   -117.5
2019Q4   -315.0
2020Q1   -107.5
2020Q2   -105.0
2020Q3   -102.5
Freq: Q-DEC, dtype: float64
```

```
>>> x.true_rate(tax_rate)
13.4232...
```

```
>>> x.tocashflow(tax_rate)
2018Q1    900.000
2018Q2    -16.250
2018Q3    -16.250
2018Q4   -116.250
2019Q1   -114.625
2019Q2   -113.000
2019Q3   -111.375
2019Q4   -309.750
2020Q1   -104.875
2020Q2   -103.250
2020Q3   -101.625
Freq: Q-DEC, dtype: float64
```

cashflows.loan.**fixed_rate_loan**(*amount*, *nrate*, *life*, *start*, *freq='A'*, *grace=0*, *dispoints=0*, *orgpoints=0*, *prepmt=None*, *balloonpmt=None*)

Fixed rate loan.

Parameters

- **amount** (*float*) – Loan amount.
- **nrate** (*float*) – nominal interest rate per year.
- **life** (*float*) – life of the loan.
- **start** (*int, tuple*) – init period for the loan.
- **pyr** (*int*) – number of compounding periods per year.
- **grace** (*int*) – number of periods of grace (without payment of the principal)
- **dispoints** (*float*) – Discount points of the loan.
- **orgpoints** (*float*) – Origination points of the loan.
- **prepmt** (*pandas.Series*) – generic cashflow representing prepayments.
- **balloonpmt** (*pandas.Series*) – generic cashflow representing balloon payments.

Returns A object of the class Loan.

```
>>> pmt = cashflow(const_value=0, start='2016Q1', periods=11, freq='Q')
>>> pmt['2017Q4'] = 200
>>> fixed_rate_loan(amount=1000, nrate=10, life=10, start='2016Q1', freq='Q',
...                 grace=0, dispoints=0,
...                 orgpoints=0, prepmt=pmt, balloonpmt=None)
Amount:             1000.00
Total interest:     129.68
Total payment:      1129.68
Discount points:    0.00
Origination points: 0.00

        Beg_Ppal_Amount  Nom_Rate  Tot_Payment  Int_Payment  Ppal_Payment  \
2016Q1      1000.000000      10.0     0.000000     0.000000      0.000000
2016Q2      1000.000000      10.0   114.258763    25.000000     89.258763
2016Q3       910.741237      10.0   114.258763    22.768531     91.490232
2016Q4       819.251005      10.0   114.258763    20.481275     93.777488
```

(continues on next page)

(continued from previous page)

```
2017Q1       725.473517      10.0   114.258763    18.136838     96.121925
2017Q2       629.351591      10.0   114.258763    15.733790     98.524973
2017Q3       530.826618      10.0   114.258763    13.270665    100.988098
2017Q4       429.838520      10.0   314.258763    10.745963    303.512800
2018Q1       126.325720      10.0   114.258763     3.158143    111.100620
2018Q2        15.225100      10.0    15.605727     0.380627     15.225100
2018Q3         0.000000      10.0     0.000000     0.000000      0.000000

        End_Ppal_Amount
2016Q1      1000.000000
2016Q2       910.741237
2016Q3       819.251005
2016Q4       725.473517
2017Q1       629.351591
2017Q2       530.826618
2017Q3       429.838520
2017Q4       126.325720
2018Q1        15.225100
2018Q2         0.000000
2018Q3         0.000000
```

1.11 Savings

1.11.1 Overview

The function `savings` computes the final balance for a savings account with arbitrary deposits and withdrawls and variable interset rate.

1.11.2 Functions in this module

`cashflows.savings.`**`savings`**(*deposits*, *nrate*, *initbal=0*)

Computes the final balance for a savings account with arbitrary deposits and withdrawls and variable interset rate.

Parameters

- **`cflo`** (*`pandas.Series`*) – Generic cashflow.
- **`deposits`** (*`pandas.Series`*) – deposits to the account.
- **`nrate`** (*`pandas.Series`*) – nominal interest rate paid by the account.
- **`initbal`** (*`float`*) – initial balance of the account.

Returns A pandas.DataFrame.

Examples

```
>>> cflo = cashflow(const_value=[100]*12, start='2000Q1', freq='Q')
>>> nrate = interest_rate([10]*12, start='2000Q1', freq='Q')
>>> savings(deposits=cflo, nrate=nrate, initbal=0)
        Beginning_Balance  Deposits  Earned_Interest  Ending_Balance  \
2000Q1           0.000000     100.0         0.000000      100.000000
2000Q2         100.000000     100.0         2.500000      202.500000
```

(continues on next page)

(continued from previous page)

```
2000Q3         202.500000     100.0         5.062500      307.562500
2000Q4         307.562500     100.0         7.689063      415.251562
2001Q1         415.251562     100.0        10.381289      525.632852
2001Q2         525.632852     100.0        13.140821      638.773673
2001Q3         638.773673     100.0        15.969342      754.743015
2001Q4         754.743015     100.0        18.868575      873.611590
2002Q1         873.611590     100.0        21.840290      995.451880
2002Q2         995.451880     100.0        24.886297     1120.338177
2002Q3        1120.338177     100.0        28.008454     1248.346631
2002Q4        1248.346631     100.0        31.208666     1379.555297

        Nominal_Rate
2000Q1          10.0
2000Q2          10.0
2000Q3          10.0
2000Q4          10.0
2001Q1          10.0
2001Q2          10.0
2001Q3          10.0
2001Q4          10.0
2002Q1          10.0
2002Q2          10.0
2002Q3          10.0
2002Q4          10.0
```

```
>>> cflo = cashflow(const_value=[0, 100, 0, 100, 100], start='2000Q1', freq='A')
>>> nrate = interest_rate([0, 1, 2, 3, 4], start='2000Q1', freq='A')
>>> savings(deposits=cflo, nrate=nrate, initbal=1000)
      Beginning_Balance  Deposits  Earned_Interest  Ending_Balance  \
2000           1000.000       0.0          0.00000      1000.00000
2001           1000.000     100.0         10.00000      1110.00000
2002           1110.000       0.0         22.20000      1132.20000
2003           1132.200     100.0         33.96600      1266.16600
2004           1266.166     100.0         50.64664      1416.81264

      Nominal_Rate
2000           0.0
2001           1.0
2002           2.0
2003           3.0
2004           4.0
```

CHAPTER 2

Indices and Tables

- genindex
- modindex
- search

CHAPTER 3

MIT license

Copyright (c) 2018 Juan David Velásquez-Henao, Ibeth Karina Vergara-Baquero

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Python Module Index

C

Index