Cashflow Documentation

Release 0.0.3

Author

May 16, 2018

Contents

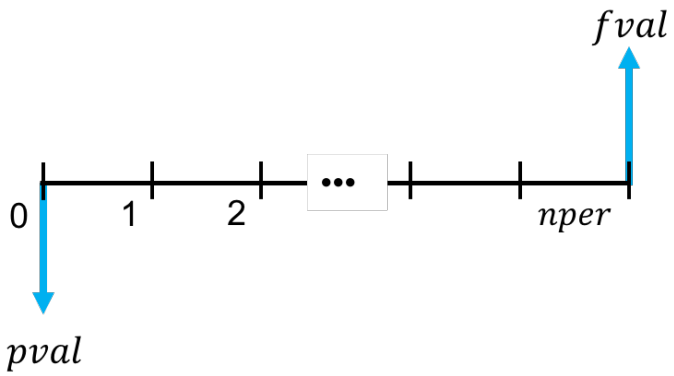
Contents:

CHAPTER 1

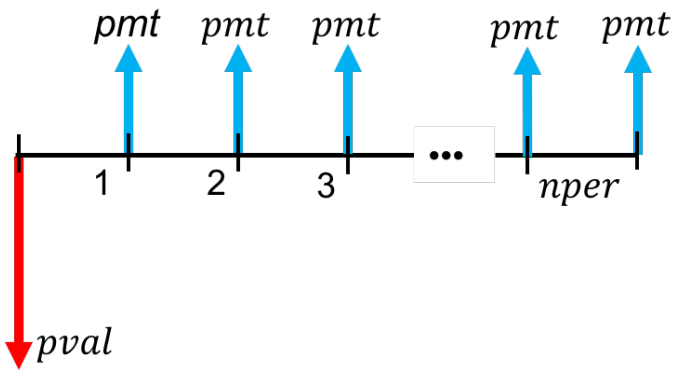
Time value of money models

This module contains functions for computing the time value of the money.

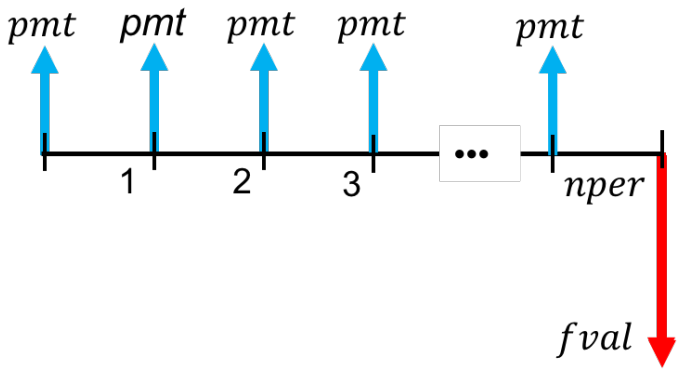
- `pvfv`: computes the missing value in the equation `fval = pval * (1 + rate) ** nper`, that represents the following cashflow.



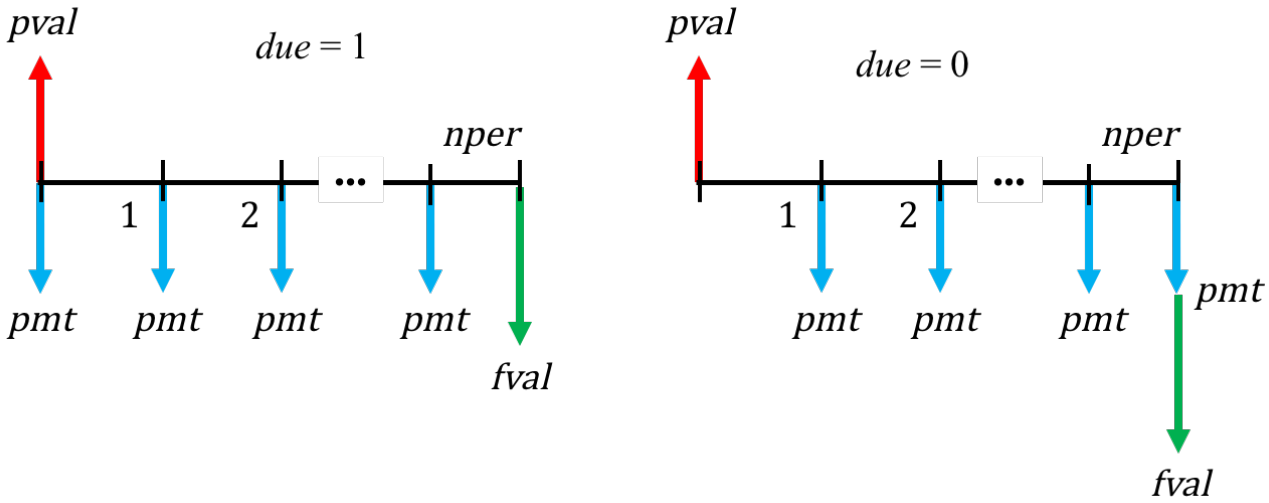
- `pvpmt`: computes the missing value (*pmt*, *pval*, *nper*, *nrate*) in a model relating a present value and a finite sequence of payments made at the end of the period (payments in arrears, or ordinary annuities), as indicated in the following cashflow diagram:



- `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:



`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

Examples.

```
>>> 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
```

```
>>> 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
```

```
>>> 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...]
```

```
>>> 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*)

CComputes the missing argument (set to None) in the function call.

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.

Effective interest rate per period is calculated as *nrate* / *pyr*.

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

Computes the missing argument (set to None) in the function call.

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.

Effective interest rate per period is calculated as *nrate* / *pyr*.

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

Computes the missing argument (set to None) in the function call.

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.

Effective interest rate per period is calculated as *nrate* / *pyr*.

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

Computes present and future values, periodic payments, nominal interest rate or number of periods.

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.

Effective interest rate per period is calculated as *nrate* / *pyr*.

Examples.

In 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 11.32% nominal)

- Periodic payment:

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

- Future value:

```
>>> tvmm(pval=5000, nrate=11.32/12, nper=48, pmt=pmt)
-0.0...
```

- Present value:

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

- Rate:

```
>>> tvmm(pval=5000, nper=48, pmt=pmt, fval = 0.0)
0.94...
```

- Number of periods:

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

- Periodic paymnts:

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

- Nominal rate:

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

```
>>> 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
```

```
>>> 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
```

CHAPTER 2

Interest rate transformations

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.
- **nrate** – 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.

```
>>> equivalent_rate(prate=interest_rate([10]*5))
10.0...
```

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

The function *iconv* computes the conversion among periodic, nominal and effective interest rates. Only an interest rate (periodic, nominal or effective) must be specified and the other two are computed. The periodic rate is the rate used in each compounding period. The effective rate is the equivalent rate that produces the same interest earnings that a periodic rate when there is P compounding periods in a year. The nominal rate is defined as the annual rate computed as P times the periodic rate.

Parameters

- **nrate** (*float, list,* TimeSeries) – nominal interest rate per year.
- **erate** (*float, list,* TimeSeries) – effective interest rate per year.
- **prate** (*float, list,* TimeSeries) – periodic rate
- **pyr** (*int, list*) – number of compounding periods per year

Returns

- (**nrate**, **prate**): when **erate** is specified.
- (**erate**, **prate**): when **nrate** is specified.

- (**nrate**, **erate**): when **prate** is specified.

Return type A tuple

Effective rate to nominal rate compounded monthly and monthly peridic rate.

```
>>> iconv(erate=10, pyr=12)
(9.56..., 0.79...)
```

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

```
>>> iconv(nrate=10, pyr=12)
(10.47..., 0.83...)
```

iconv accepts Python vectors.

```
>>> iconv(erate=10, pyr=[3, 6, 12])
([9.68..., 9.60..., 9.56...], [3.22..., 1.60..., 0.79...])
```

```
>>> iconv(prate=1, pyr=[3, 6, 12])
([3, 6, 12], [3.03..., 6.15..., 12.68...])
```

```
>>> iconv(nrate=10, pyr=[3, 6, 12])
([10.33..., 10.42..., 10.47...], [3.33..., 1.66..., 0.83...])
```

```
>>> iconv(erate=[10, 12, 14], pyr=12)
([9.56..., 11.38..., 13.17...], [0.79..., 0.94..., 1.09...])
```

```
>>> iconv(prate=[1, 2, 3], pyr=12)
([12, 24, 36], [12.68..., 26.82..., 42.57...])
```

```
>>> iconv(nrate=[10, 12, 14], pyr=12)
([10.47..., 12.68..., 14.93...], [0.83..., 1.0, 1.16...])
```

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.

```
>>> iconv(erate=[10, 12, 14], pyr=[3, 6, 12])
([9.68..., 11.44..., 13.17...], [3.22..., 1.90..., 1.09...])
```

```
>>> iconv(nrate=[10, 12, 14], pyr=[3, 6, 12])
([10.33..., 12.61..., 14.93...], [3.33..., 2.0, 1.16...])
```

```
>>> iconv(prate=[1, 2, 3], pyr=[3, 6, 12])
([3, 12, 36], [3.03..., 12.61..., 42.57...])
```

iconv accepts TimeSeries objects

```
>>> erate, prate = iconv(nrate = interest_rate(const_value=12, nper=12, pyr=2))
>>> prate
Time Series:
Start = (0, 0)
End = (5, 1)
pyr = 2
Data = (0, 0)-(5, 1) [12] 6.00
```

```
>>> erate
Time Series:
Start = (0, 0)
End = (5, 1)
pyr = 2
Data = (0, 0)-(5, 1) [12] 12.36
```

```
>>> erate, prate = iconv(nrate = interest_rate(const_value=12, nper=12, pyr=4))
>>> prate
  Qtr0 Qtr1 Qtr2 Qtr3
0 3.00 3.00 3.00 3.00
1 3.00 3.00 3.00 3.00
2 3.00 3.00 3.00 3.00
```

```
>>> erate
   Qtr0  Qtr1  Qtr2  Qtr3
0 12.55 12.55 12.55 12.55
1 12.55 12.55 12.55 12.55
2 12.55 12.55 12.55 12.55
```

```
>>> nrate, prate = iconv(erate = erate)
>>> nrate
   Qtr0  Qtr1  Qtr2  Qtr3
0 12.00 12.00 12.00 12.00
1 12.00 12.00 12.00 12.00
2 12.00 12.00 12.00 12.00
```

```
>>> prate
  Qtr0 Qtr1 Qtr2 Qtr3
0 3.00 3.00 3.00 3.00
1 3.00 3.00 3.00 3.00
2 3.00 3.00 3.00 3.00
```

```
>>> nrate, erate = iconv(prate = prate)
>>> nrate
   Qtr0  Qtr1  Qtr2  Qtr3
0 12.00 12.00 12.00 12.00
1 12.00 12.00 12.00 12.00
2 12.00 12.00 12.00 12.00
```

```
>>> erate
   Qtr0  Qtr1  Qtr2  Qtr3
0 12.55 12.55 12.55 12.55
1 12.55 12.55 12.55 12.55
2 12.55 12.55 12.55 12.55
```

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)

Examples

```
>>> nrate = interest_rate(const_value=4,nper=10, pyr=4)
>>> erate, prate = iconv(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=0*)

Returns a list of discount factors calculated as 1 / (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 List of float values

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

```
>>> nrate = interest_rate(const_value=4,nper=10, pyr=4)
>>> erate, prate = iconv(nrate=nrate)
>>> to_discount_factor(nrate=nrate, base_date=2)
[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=2)
[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=2)
[1.0201, 1.01, 1.0, 0.990..., 0.980..., 0.970..., 0.960..., 0.951..., 0.942..., 0.
↪932...]
```

CHAPTER 3

Time series objects

This module implements a *TimeSeries* object that is used to represent generic cashflows and interest rate.

class cashflows.gtimeseries.**TimeSeries**(*start=None*, *end=None*, *nper=None*, *pyr=1*)

Bases: object

Time series object for representing generic cashflows and interest rates.

Examples.

```
>>> TimeSeries(start=(2000, 0), end=(2002, 3), nper=12, pyr=4)
     Qtr0 Qtr1 Qtr2 Qtr3
2000 0.00 0.00 0.00 0.00
2001 0.00 0.00 0.00 0.00
2002 0.00 0.00 0.00 0.00
```

```
>>> TimeSeries(start=(2000, 0), end=(2002, 3), pyr=4)
     Qtr0 Qtr1 Qtr2 Qtr3
2000 0.00 0.00 0.00 0.00
2001 0.00 0.00 0.00 0.00
2002 0.00 0.00 0.00 0.00
```

copy()

returns a copy of the time series

cumsum()

returns the cumulative sum of the time series

tolist()

Returns the values as a list

cashflows.gtimeseries.**cashflow**(*const_value=0*, *start=None*, *end=None*, *nper=None*, *pyr=1*, *spec=None*)

Returns a time series as a generic cashflow.

```
>>> spec = ((2000, 3), 10)
>>> cashflow(const_value=1, start=(2000, 0), nper=8, pyr=4, spec=spec)
```

(continues on next page)

(continued from previous page)

```
      Qtr0  Qtr1  Qtr2  Qtr3
2000  1.00  1.00  1.00 10.00
2001  1.00  1.00  1.00  1.00
```

```
>>> spec = [((2000, 3), 10), ((2001, 3), 10)]
>>> cashflow(const_value=1, start=(2000, 0), nper=8, pyr=4, spec=spec)
      Qtr0  Qtr1  Qtr2  Qtr3
2000  1.00  1.00  1.00 10.00
2001  1.00  1.00  1.00 10.00
```

```
>>> spec = (3, 10)
>>> cashflow(const_value=1, start=(2000, 0), nper=8, pyr=4, spec=spec)
      Qtr0  Qtr1  Qtr2  Qtr3
2000  1.00  1.00  1.00 10.00
2001  1.00  1.00  1.00  1.00
```

```
>>> spec = [(3, 10), (7, 10)]
>>> cashflow(const_value=1, start=(2000, 0), nper=8, pyr=4, spec=spec)
      Qtr0  Qtr1  Qtr2  Qtr3
2000  1.00  1.00  1.00 10.00
2001  1.00  1.00  1.00 10.00
```

```
>>> cashflow(const_value=[10]*10, pyr=4)
   Qtr0  Qtr1  Qtr2  Qtr3
0 10.00 10.00 10.00 10.00
1 10.00 10.00 10.00 10.00
2 10.00 10.00
```

```
>>> cashflow(const_value=[-10]*4)
Time Series:
Start = (0,)
End = (3,)
pyr = 1
Data = (0,)-(3,) [4] -10.00
```

```
>>> x = cashflow(const_value=[0, 1, 2, 3], pyr=4)
>>> x[3] = 10
>>> x
   Qtr0  Qtr1  Qtr2  Qtr3
0  0.00  1.00  2.00 10.00
```

```
>>> x[3]
10
```

```
>>> x[(0, 3)] = 0
>>> x
   Qtr0  Qtr1  Qtr2  Qtr3
0  0.00  1.00  2.00  0.00
```

```
>>> x[(0,2)]
2
```

```
>>> cashflow(const_value=[0, 1, 2, 2, 4, 5, 6, 7, 8])
Time Series:
Start = (0,)
End = (8,)
pyr = 1
Data = (0,)          0.00
       (1,)          1.00
       (2,)-(3,) [2] 2.00
       (4,)          4.00
       (5,)          5.00
       (6,)          6.00
       (7,)          7.00
       (8,)          8.00
```

```
>>> cashflow(const_value=0, nper=15, pyr=1, spec=[(t,100) for t in range(5,10)])
Time Series:
Start = (0,)
End = (14,)
pyr = 1
Data = (0,)-(4,)   [5]   0.00
       (5,)-(9,)   [5] 100.00
       (10,)-(14,) [5]   0.00
```

```
>>> cashflow(const_value=[0, 1, 2, 3, 4, 5]).cumsum()
Time Series:
Start = (0,)
End = (5,)
pyr = 1
Data = (0,)          0.00
       (1,)          1.00
       (2,)          3.00
       (3,)          6.00
       (4,)         10.00
       (5,)         15.00
```

cashflows.gtimeseries.**cfloplot**(*cflo*)

Text plot of a cashflow.

```
>>> cflo = cashflow(const_value=[-10, 5, 0, 20] * 3, pyr=4)
>>> cfloplot(cflo)
time    value +------------------+------------------+
(0, 0) -10.00           **********
(0, 1)   5.00                    *****
(0, 2)   0.00                    *
(0, 3)  20.00                    ********************
(1, 0) -10.00           **********
(1, 1)   5.00                    *****
(1, 2)   0.00                    *
(1, 3)  20.00                    ********************
(2, 0) -10.00           **********
(2, 1)   5.00                    *****
(2, 2)   0.00                    *
(2, 3)  20.00                    ********************
```

cashflows.gtimeseries.**interest_rate**(*const_value=0*, *start=None*, *end=None*, *nper=None*, *pyr=1*, *spec=None*)

Creates a time series object specified as a interest rate.

```
>>> spec = ((2000, 3), 10)
>>> interest_rate(const_value=1, start=(2000, 0), nper=8, pyr=4, spec=spec)
      Qtr0  Qtr1  Qtr2  Qtr3
2000  1.00  1.00  1.00 10.00
2001 10.00 10.00 10.00 10.00
```

```
>>> spec = [((2000, 3), 10), ((2001, 1), 20)]
>>> interest_rate(const_value=1, start=(2000, 0), nper=8, pyr=4, spec=spec)
      Qtr0  Qtr1  Qtr2  Qtr3
2000  1.00  1.00  1.00 10.00
2001 10.00 20.00 20.00 20.00
```

```
>>> spec = (3, 10)
>>> interest_rate(const_value=1, start=(2000, 0), nper=8, pyr=4, spec=spec)
      Qtr0  Qtr1  Qtr2  Qtr3
2000  1.00  1.00  1.00 10.00
2001 10.00 10.00 10.00 10.00
```

```
>>> spec = [(3, 10), (6, 20)]
>>> interest_rate(const_value=1, start=(2000, 0), nper=8, pyr=4, spec=spec)
      Qtr0  Qtr1  Qtr2  Qtr3
2000  1.00  1.00  1.00 10.00
2001 10.00 10.00 20.00 20.00
```

```
>>> interest_rate(const_value=[10]*10, pyr=4)
   Qtr0  Qtr1  Qtr2  Qtr3
0 10.00 10.00 10.00 10.00
1 10.00 10.00 10.00 10.00
2 10.00 10.00
```

cashflows.gtimeseries.**repr_table**(*cols*, *header=None*)

cashflows.gtimeseries.**verify_eq_time_range**(*series1*, *series2*)

CHAPTER 4

After tax cashflow calculation.

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

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. *cflo* and *tax_rate* must have the same length and time specification.

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

Parameters

- **cflo** (TimeSeries) – generic cashflow.
- **tax_rate** (TimeSeries) – periodic income tax rate.

Returns TimeSeries objects with taxed values

Example*

```
>>> cflo = cashflow(const_value=[100] * 5, spec=(0, -100))
>>> tax_rate = interest_rate(const_value=[10] * 5)
>>> after_tax_cashflow(cflo=cflo, tax_rate=tax_rate)
Time Series:
Start = (0,)
End = (4,)
pyr = 1
Data = (0,)           0.00
       (1,)-(4,) [4] 10.00
```

CHAPTER 5

Currency conversion

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

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

Converts a cashflow of dollars to another currency.

Parameters

- **cflo** (TimeSeries) – A cashflow.
- **exchange_rate** (*float*) – Exchange rate at time *base_date*.
- **devaluation** (TimeSeries) – 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)
>>> currency_conversion(cflo=cflo, exchange_rate=2)
Time Series:
Start = (0,)
End = (4,)
pyr = 1
Data = (0,)-(4,) [5] 200.00
```

```
>>> currency_conversion(cflo=cflo, exchange_rate=2,
... devaluation=interest_rate(const_value=[5]*5), base_date=(2,))
Time Series:
Start = (0,)
End = (4,)
pyr = 1
Data = (0,)   181.41
```

(continues on next page)

(continued from previous page)

```
       (1,)   190.48
       (2,)   200.00
       (3,)   210.00
       (4,)   220.50
```

```
>>> cflo = cashflow(const_value=[100] * 8, pyr=4)
>>> currency_conversion(cflo=cflo,
...                     exchange_rate=2,
...                     devaluation=interest_rate([1]*8, pyr=4))
    Qtr0   Qtr1   Qtr2   Qtr3
0 200.00 202.00 204.02 206.06
1 208.12 210.20 212.30 214.43
```

```
>>> cflo = cashflow(const_value=[100] * 5)
>>> currency_conversion(cflo=[cflo, cflo], exchange_rate=2)
[Time Series:
Start = (0,)
End = (4,)
pyr = 1
Data = (0,)-(4,) [5] 200.00
,  Time Series:
Start = (0,)
End = (4,)
pyr = 1
Data = (0,)-(4,) [5] 200.00
]
```

CHAPTER 6

Constant dollar transformations

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.

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

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

Parameters

- **cflo** (TimeSeries) – A cashflow.
- **inflation** (TimeSeries) – Inflation rate per compounding period.
- **base_date** (*int, tuple*) – base date.

Returns A cashflow in current money (TimeSeries)

Examples.

```
>>> const2curr(cflo=cashflow(const_value=[100] * 5),
... inflation=interest_rate(const_value=[10, 10, 20, 20, 20]))
Time Series:
Start = (0,)
End = (4,)
pyr = 1
Data = (0,)   100.00
       (1,)   110.00
       (2,)   132.00
       (3,)   158.40
       (4,)   190.08
```

```
>>> const2curr(cflo=cashflow(const_value=[100] * 5),
... inflation=interest_rate(const_value=[10, 10, 20, 20, 20]), base_date=4)
Time Series:
Start = (0,)
End = (4,)
```

(continues on next page)

(continued from previous page)

```
pyr = 1
Data = (0,)    52.61
       (1,)    57.87
       (2,)    69.44
       (3,)    83.33
       (4,)   100.00
```

```
>>> const2curr(cflo=cashflow(const_value=[100] * 8, pyr=4),
... inflation=interest_rate(const_value=1, nper=8, pyr=4))
    Qtr0   Qtr1   Qtr2   Qtr3
0 100.00 101.00 102.01 103.03
1 104.06 105.10 106.15 107.21
```

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

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

Parameters

- **cflo** (*list, Cashflow*) – A cashflow.
- **inflation_rate** (*float, Rate*) – Inflation rate per compounding period.
- **base_date** (*int*) – base time..

Returns A cashflow in constant dollars

```
>>> curr2const(cflo=cashflow(const_value=[100] * 5),
... inflation=interest_rate(const_value=[10, 10, 20, 20, 20]))
Time Series:
Start = (0,)
End = (4,)
pyr = 1
Data = (0,)   100.00
       (1,)    90.91
       (2,)    75.76
       (3,)    63.13
       (4,)    52.61
```

```
>>> curr2const(cflo=cashflow(const_value=[100] * 5),
... inflation=interest_rate(const_value=[10, 10, 20, 20, 20]), base_date=4)
Time Series:
Start = (0,)
End = (4,)
pyr = 1
Data = (0,)   190.08
       (1,)   172.80
       (2,)   144.00
       (3,)   120.00
       (4,)   100.00
```

```
>>> curr2const(cflo=cashflow(const_value=[100] * 8, pyr=4),
... inflation=interest_rate(const_value=1, nper=8, pyr=4))
    Qtr0   Qtr1   Qtr2   Qtr3
0 100.00  99.01  98.03  97.06
1  96.10  95.15  94.20  93.27
```

CHAPTER 7

Analysis of cashflows

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.

`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`** (*`int float, Rate`*) – Periodic interest rate.
- **`cflo`** (*`cashflow, list`*) – Generic cashflow.
- **`base_date`** (*`int, list`*) – Time.

Returns Float or list of floats.

Examples.

```
>>> prate = interest_rate([2]*9, pyr=4)
>>> cflo = cashflow([-717.01] + [100]*8, pyr=4)
>>> benefit_cost_ratio(cflo, prate)
1.02...
```

```
>>> prate = interest_rate([12]*5)
>>> cflo = cashflow([-200] + [100]*4)
>>> benefit_cost_ratio(cflo, prate)
1.518...
```

```
>>> benefit_cost_ratio([cflo, cflo], prate)
[1.518..., 1.518...]
```

```
>>> benefit_cost_ratio(cflo, [prate, prate])
[1.518..., 1.518...]
```

```
>>> benefit_cost_ratio([cflo, cflo], [prate, prate])
[1.518..., 1.518...]
```

```
>>> benefit_cost_ratio([cflo, cflo], [prate, prate], [0, 0])
[1.518..., 1.518...]
```

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

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

Parameters **cflo** (TimeSeries) – Generic cashflow.

Returns Float or list of floats.

Examples.

```
>>> cflo = cashflow([-717.01] + [100]*8, pyr=4)
>>> irr(cflo)
2.50...
```

```
>>> cflo = cashflow([-200] + [100]*4)
>>> irr(cflo)
34.90...
```

```
>>> irr([cflo, cflo])
[34.90..., 34.90...]
```

cashflows.analysis.**list_as_table**(*data*)

Prints the list *data* as a table. This function is used to produce a human-readable format of a table for comparing financial indicators.

Parameters **data** (*list*) – List of numeric values.

Returns None

Example.

```
>>> list_as_table(data=[1, 2, 3, 4])
 #               Value
-----------------------
 0              1.0000
 1              2.0000
 2              3.0000
 3              4.0000
```

```
>>> prate = interest_rate([12]*5)
>>> cflo = cashflow([-200] + [100]*4)
>>> list_as_table(timevalue(cflo=[cflo, cflo, cflo], prate=prate))
 #               Value
------------------------
 0            103.7349
 1            103.7349
 2            103.7349
```

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** (*list, cashflow*) – 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)
>>> mirr(cflo)
18.92...
```

```
>>> mirr([cflo, cflo])
[18.92..., 18.92...]
```

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

Returns Float or list of floats.

Examples.

```
>>> prate = interest_rate([2]*9, pyr=4)
>>> cflo = cashflow([-732.54] + [100]*8, pyr=4)
>>> net_uniform_series(cflo, prate)
0.00...
```

```
>>> prate = interest_rate([12]*5)
>>> cflo = cashflow([-200] + [100]*4)
>>> net_uniform_series(cflo, prate)
116.18...
```

```
>>> net_uniform_series([cflo, cflo], prate)
[116.18..., 116.18...]
```

```
>>> net_uniform_series(cflo, [prate, prate])
[116.18..., 116.18...]
```

```
>>> net_uniform_series([cflo, cflo], [prate, prate])
[116.18..., 116.18...]
```

```
>>> net_uniform_series([cflo, cflo], [prate, prate], nper=5)
[28.77..., 28.77...]
```

```
>>> net_uniform_series([cflo, cflo], [prate, prate], nper=[5, 5])
[28.77..., 28.77...]
```

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** (*TimeSeries, list of TimeSeries*) – Generic cashflow.
- **prate** (*TimeSeries*) – 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, pyr=4)
>>> prate = interest_rate([2]*9, pyr=4)
>>> timevalue(cflo, prate)
0.00...
```

```
>>> prate = interest_rate([12]*5)
>>> cflo = cashflow([100]*5, spec = (0, -200))
>>> timevalue(cflo, prate)
103.73...
```

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

```
>>> timevalue(cflo, prate, base_date=0, utility=exp_utility_fun(200))
-84.15...
```

```
>>> timevalue(cflo, prate, base_date=0, utility=log_utility_fun(210))
369092793...
```

```
>>> timevalue(cflo, prate, base_date=0, utility=sqrt_utility_fun(210))
2998.12...
```

```
>>> prate = interest_rate([12]*5)
>>> cflo = cashflow([-200] + [100]*4)
>>> timevalue(cflo=cflo, prate=prate)
103.73...
```

```
>>> timevalue(cflo=[cflo, cflo], prate=prate)
[103.73..., 103.73...]
```

```
>>> timevalue(cflo=cflo, prate=[prate, prate])
[103.73..., 103.73...]
```

```
>>> timevalue(cflo=[cflo, cflo], prate=[prate, prate])
[103.73..., 103.73...]
```

```
>>> timevalue(cflo=[cflo, cflo], prate=[prate, prate], base_date=[4, 4])
[163.22..., 163.22...]
```

CHAPTER 8

Bond Valuation

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.

cashflows.bond.**bond**(*face_value=None*, *coupon_rate=None*, *coupon_value=None*, *num_coupons=None*, *value=None*, *ytm=None*)

Evaluation of bond investments.

Parameters

- **face_value** (*float*) – the bond's value-at-maturity.
- **coupon_rate** (*float*) – rate for calculate the coupon payment.
- **coupon_value** (*float*) – periodic payment.
- **num_coupons** (*int*) – number of couont payments before maturity.
- **value** (*float, list*) – present value of the bond
- **ytm** (*float, list*) – yield-to-maturity.

Returns

- *value*: when *ytm* is specified.
- *ytm*: when *value* is specified.
- *None*: when *ytm* and *value* are specified. Prints a sensibility table.

Return type None, a float value, or a list of float values

When *coupon_rate* is defined, *coupon_value* is calculated automaticly.

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...
```

Also, it is possible to make sensibility analysis for bond's data. In the following case, the present value of the bond is calculated for various values of the yield-to-maturity.

```
>>> bond(face_value=1000, coupon_rate=5.6, num_coupons=10,
... ytm=[4.0, 5.0, 5.6, 6.0, 7.0])
[1129.77..., 1046.33..., 1000.0..., 970.55..., 901.66...]
```

And for different values:

```
>>> bond(face_value=1000, coupon_rate=5.6, num_coupons=10,
... value=[900, 1000, 1100])
[7.0..., 5.6..., 4.3...]
```

When values for the yield-to-maturity and one value for present value of the bond are supplied, the function prints a report.

```
>>> bond(face_value=1000, coupon_rate=5.6, num_coupons=10,
... ytm=[4.0, 5.0, 5.6, 6.0, 7.0], value=1000)
Bond valuation analysis
Reference price: 1000
Analysis:
          Yield     Value Change
            (%)       ($)    (%)
         ------------------------
           4.00   1129.77  12.98
           5.00   1046.33   4.63
           5.60   1000.00   0.00
           6.00    970.56  -2.94
           7.00    901.67  -9.83
```

CHAPTER 9

Asset depreciation

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

- **cost** (TimeSeries) – the cost per period of the assets.
- **life** (TimeSeries) – number of depreciation periods for the asset.
- **salvalue** (TimeSeries) – 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 depreciation per period and accumulated depreciation per period

Return type A tuple (dep, accum) of lists (tuple)

Examples.

```
>>> costs1 = cashflow(const_value=0, nper=16, spec=(0, 1000), pyr=4)
>>> costs2 = cashflow(const_value=0, nper=16, spec=[(0, 1000), (8, 1000)], pyr=4)
>>> life1 = cashflow(const_value=0, nper=16, spec=(0, 4), pyr=4)
>>> life2 = cashflow(const_value=0, nper=16, spec=[(0, 4), (8, 4)], pyr=4)
>>> delay12 = cashflow(const_value=0, nper=16, spec=(0, 2), pyr=4)
>>> delay22 = cashflow(const_value=0, nper=16, spec=[(0, 2), (8, 2)], pyr=4)
>>> depreciation_db(costs=costs1, life=life1, factor=1.5, convert_to_sl=False)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00 375.00 234.38 146.48
1  91.55   0.00   0.00   0.00
2   0.00   0.00   0.00   0.00
3   0.00   0.00   0.00   0.00
```

```
>>> depreciation_db(costs=costs1, life=life1, factor=1.5, convert_to_sl=False,␣
↪noprint=False)
t          Beg.    Cost  Depre.  Accum.    End.
           Book                  Depre.    Book
          Value                           Value
-----------------------------------------------
(0, 0)     0.00 1000.00    0.00    0.00 1000.00
(0, 1) 1000.00    0.00  375.00  375.00  625.00
(0, 2)  625.00    0.00  234.38  609.38  390.62
(0, 3)  390.62    0.00  146.48  755.86  244.14
(1, 0)  244.14    0.00   91.55  847.41  152.59
(1, 1)  152.59    0.00    0.00  847.41  152.59
(1, 2)  152.59    0.00    0.00  847.41  152.59
(1, 3)  152.59    0.00    0.00  847.41  152.59
(2, 0)  152.59    0.00    0.00  847.41  152.59
(2, 1)  152.59    0.00    0.00  847.41  152.59
(2, 2)  152.59    0.00    0.00  847.41  152.59
(2, 3)  152.59    0.00    0.00  847.41  152.59
(3, 0)  152.59    0.00    0.00  847.41  152.59
(3, 1)  152.59    0.00    0.00  847.41  152.59
(3, 2)  152.59    0.00    0.00  847.41  152.59
(3, 3)  152.59    0.00    0.00  847.41  152.59
```

```
>>> depreciation_db(costs=costs1, life=life1, delay=delay12, factor=1.5, convert_
↪to_sl=False)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00   0.00   0.00 375.00
1 234.38 146.48  91.55   0.00
2   0.00   0.00   0.00   0.00
3   0.00   0.00   0.00   0.00
```

```
>>> depreciation_db(costs=costs2, life=life2, factor=1.5, convert_to_sl=False)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00 375.00 234.38 146.48
1  91.55   0.00   0.00   0.00
2   0.00 375.00 234.38 146.48
3  91.55   0.00   0.00   0.00
```

```
>>> depreciation_db(costs=costs2, life=life2, delay=delay22, factor=1.5, convert_
↪to_sl=False)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00   0.00   0.00 375.00
1 234.38 146.48  91.55   0.00
2   0.00   0.00   0.00 375.00
3 234.38 146.48  91.55   0.00
```

```
>>> depreciation_db(costs=costs2, life=life2, delay=delay22, factor=1.5, convert_
↪to_sl=False, noprint=False)
t          Beg.    Cost  Depre.  Accum.    End.
           Book                  Depre.    Book
          Value                           Value
-----------------------------------------------
(0, 0)     0.00 1000.00    0.00    0.00 1000.00
(0, 1) 1000.00    0.00    0.00    0.00 1000.00
(0, 2) 1000.00    0.00    0.00    0.00 1000.00
(0, 3) 1000.00    0.00  375.00  375.00  625.00
```

(continues on next page)

(continued from previous page)

```
(1, 0)  625.00    0.00  234.38  609.38  390.62
(1, 1)  390.62    0.00  146.48  755.86  244.14
(1, 2)  244.14    0.00   91.55  847.41  152.59
(1, 3)  152.59    0.00    0.00  847.41  152.59
(2, 0)  152.59 1000.00    0.00  847.41 1152.59
(2, 1) 1152.59    0.00    0.00  847.41 1152.59
(2, 2) 1152.59    0.00    0.00  847.41 1152.59
(2, 3) 1152.59    0.00  375.00 1222.41  777.59
(3, 0)  777.59    0.00  234.38 1456.79  543.21
(3, 1)  543.21    0.00  146.48 1603.27  396.73
(3, 2)  396.73    0.00   91.55 1694.82  305.18
(3, 3)  305.18    0.00    0.00 1694.82  305.18
```

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

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

Parameters

- **cost** (TimeSeries) – the cost per period of the assets.
- **life** (TimeSeries) – number of depreciation periods for the asset.
- **salvalue** (TimeSeries) – salvage value as a percentage of cost.
- **noprint** (*bool*) – when True, the procedure prints a depreciation table.

Returns depreciation, accum_depreciation (TimeSeries, TimeSeries).

Examples.

```
>>> costs1 = cashflow(const_value=0, nper=16, spec=(0, 1000), pyr=4)
>>> costs2 = cashflow(const_value=0, nper=16, spec=[(0, 1000), (8, 1000)], pyr=4)
>>> life1 = cashflow(const_value=0, nper=16, spec=(0, 4), pyr=4)
>>> life2 = cashflow(const_value=0, nper=16, spec=[(0, 4), (8, 4)], pyr=4)
>>> delay12 = cashflow(const_value=0, nper=16, spec=(0, 2), pyr=4)
>>> delay22 = cashflow(const_value=0, nper=16, spec=[(0, 2), (8, 2)], pyr=4)
>>> depreciation_sl(costs=costs1, life=life1)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00 250.00 250.00 250.00
1 250.00   0.00   0.00   0.00
2   0.00   0.00   0.00   0.00
3   0.00   0.00   0.00   0.00
```

```
>>> depreciation_sl(costs=costs1, life=life1, delay=delay12)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00   0.00   0.00 250.00
1 250.00 250.00 250.00   0.00
2   0.00   0.00   0.00   0.00
3   0.00   0.00   0.00   0.00
```

```
>>> depreciation_sl(costs=costs2, life=life2)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00 250.00 250.00 250.00
1 250.00   0.00   0.00   0.00
2   0.00 250.00 250.00 250.00
3 250.00   0.00   0.00   0.00
```

```
>>> depreciation_sl(costs=costs2, life=life2, delay=delay22)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00   0.00   0.00 250.00
1 250.00 250.00 250.00   0.00
2   0.00   0.00   0.00 250.00
3 250.00 250.00 250.00   0.00
```

```
>>> depreciation_sl(costs=costs2, life=life2, delay=delay22, noprint=False)
t          Beg.    Cost  Depre.  Accum.    End.
           Book                  Depre.    Book
         Value                          Value
-----------------------------------------------
(0, 0)     0.00 1000.00    0.00    0.00 1000.00
(0, 1) 1000.00    0.00    0.00    0.00 1000.00
(0, 2) 1000.00    0.00    0.00    0.00 1000.00
(0, 3) 1000.00    0.00  250.00  250.00  750.00
(1, 0)  750.00    0.00  250.00  500.00  500.00
(1, 1)  500.00    0.00  250.00  750.00  250.00
(1, 2)  250.00    0.00  250.00 1000.00    0.00
(1, 3)    0.00    0.00    0.00 1000.00    0.00
(2, 0)    0.00 1000.00    0.00 1000.00 1000.00
(2, 1) 1000.00    0.00    0.00 1000.00 1000.00
(2, 2) 1000.00    0.00    0.00 1000.00 1000.00
(2, 3) 1000.00    0.00  250.00 1250.00  750.00
(3, 0)  750.00    0.00  250.00 1500.00  500.00
(3, 1)  500.00    0.00  250.00 1750.00  250.00
(3, 2)  250.00    0.00  250.00 2000.00    0.00
(3, 3)    0.00    0.00    0.00 2000.00    0.00
```

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

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

Parameters

- **cost** (TimeSeries) – the cost per period of the assets.
- **life** (TimeSeries) – number of depreciation periods for the asset.
- **salvalue** (TimeSeries) – salvage value as a percentage of cost.
- **noprint** (*bool*) – when True, the procedure prints a depreciation table.

Returns depreciation per period and accumulated depreciation per period

Return type A tuple (dep, accum) of lists (tuple)

Examples.

```
>>> costs1 = cashflow(const_value=0, nper=16, spec=(0, 1000), pyr=4)
>>> costs2 = cashflow(const_value=0, nper=16, spec=[(0, 1000), (8, 1000)], pyr=4)
>>> life1 = cashflow(const_value=0, nper=16, spec=(0, 4), pyr=4)
>>> life2 = cashflow(const_value=0, nper=16, spec=[(0, 4), (8, 4)], pyr=4)
>>> delay12 = cashflow(const_value=0, nper=16, spec=(0, 2), pyr=4)
>>> delay22 = cashflow(const_value=0, nper=16, spec=[(0, 2), (8, 2)], pyr=4)
>>> depreciation_soyd(costs=costs1, life=life1)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00 400.00 300.00 200.00
1 100.00   0.00   0.00   0.00
```

(continues on next page)

(continued from previous page)

```
2   0.00   0.00   0.00   0.00
3   0.00   0.00   0.00   0.00
```

```
>>> depreciation_soyd(costs=costs1, life=life1, delay=delay12)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00   0.00   0.00 400.00
1 300.00 200.00 100.00   0.00
2   0.00   0.00   0.00   0.00
3   0.00   0.00   0.00   0.00
```

```
>>> depreciation_soyd(costs=costs2, life=life2)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00 400.00 300.00 200.00
1 100.00   0.00   0.00   0.00
2   0.00 400.00 300.00 200.00
3 100.00   0.00   0.00   0.00
```

```
>>> depreciation_soyd(costs=costs2, life=life2, delay=delay22)
    Qtr0   Qtr1   Qtr2   Qtr3
0   0.00   0.00   0.00 400.00
1 300.00 200.00 100.00   0.00
2   0.00   0.00   0.00 400.00
3 300.00 200.00 100.00   0.00
```

```
>>> depreciation_soyd(costs=costs2, life=life2, delay=delay22, noprint=False)
t          Beg.    Cost  Depre.  Accum.    End.
           Book                  Depre.    Book
          Value                           Value
-----------------------------------------------
(0, 0)     0.00 1000.00    0.00    0.00 1000.00
(0, 1) 1000.00    0.00    0.00    0.00 1000.00
(0, 2) 1000.00    0.00    0.00    0.00 1000.00
(0, 3) 1000.00    0.00  400.00  400.00  600.00
(1, 0)  600.00    0.00  300.00  700.00  300.00
(1, 1)  300.00    0.00  200.00  900.00  100.00
(1, 2)  100.00    0.00  100.00 1000.00    0.00
(1, 3)    0.00    0.00    0.00 1000.00    0.00
(2, 0)    0.00 1000.00    0.00 1000.00 1000.00
(2, 1) 1000.00    0.00    0.00 1000.00 1000.00
(2, 2) 1000.00    0.00    0.00 1000.00 1000.00
(2, 3) 1000.00    0.00  400.00 1400.00  600.00
(3, 0)  600.00    0.00  300.00 1700.00  300.00
(3, 1)  300.00    0.00  200.00 1900.00  100.00
(3, 2)  100.00    0.00  100.00 2000.00    0.00
(3, 3)    0.00    0.00    0.00 2000.00    0.00
```

cashflows.depreciation.**print_depr**(*depr*, *adepr*, *costs*, *begbook*, *endbook*)

Prints a depreciation table

Parameters

- **cost** (*int, float*) – Initial cost of the asset
- **depr** (*list*) – Depreciation per period
- **adepr** (*list*) – Accumulated depreciation per period
- **begbook** (*list*) – Beginning book value

- **endbook** (*list*) – Ending book value

Returns None

CHAPTER 10

Loan analysis

Computes the amorization schedule for the followint 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.

class cashflows.loan.**Loan**

Bases: `object`

Class for representing loans

begbal()

Returns the balance at the begining of each period as a Cashflow object.

endbal()

Returns the balance at the ending of each period as a Cashflow object.

interest()

Returns the interest paid as a Cashflow object.

ppalpmt()

Returns the principal payment for each period as a Cashflow object.

to_cashflow(*tax_rate=0*)

Converts the loan to the equivalent cashflow.

For the conversion, origination points are considered as exogenous costs and they are not taking in to account in the computation. In oposition, discount points are considered as prepaid interest and included in the cashflow.

When tax_rate is different from zero, tax benefits are considered.

true_rate (*tax_rate=0*)
Computes the true interest rate for the loan.

For the computation, the loan is converted to the equivalent cashflow, taking in to account the following aspects:

- Origination points are considered as non deducible costs and they are ignored in the computation.
- Discount points are prepaid interest and they are considered as deducibles in the computation.
- When *tax_rate* is different from zero, the After-Tax true interest rate is calculated. This is, only the (1 - *tax_rate*) of paid interests (including discount points) are used in the computation.

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

Returns A object of the class Loan.

```
>>> nrate = interest_rate(const_value=10, nper=11, pyr=4)
>>> bullet_loan(amount=1000, nrate=nrate, dispoints=0, orgpoints=0, prepmt=None)
t          Beg.    Per.   Total    Int.    Ppal  Ending
           Ppal    Rate     Pmt     Pmt     Pmt    Ppal
-----------------------------------------------------------
(0, 0)     0.00   10.00    0.00    0.00    0.00 1000.00
(0, 1) 1000.00   10.00   25.00   25.00    0.00 1000.00
(0, 2) 1000.00   10.00   25.00   25.00    0.00 1000.00
(0, 3) 1000.00   10.00   25.00   25.00    0.00 1000.00
(1, 0) 1000.00   10.00   25.00   25.00    0.00 1000.00
(1, 1) 1000.00   10.00   25.00   25.00    0.00 1000.00
(1, 2) 1000.00   10.00   25.00   25.00    0.00 1000.00
(1, 3) 1000.00   10.00   25.00   25.00    0.00 1000.00
(2, 0) 1000.00   10.00   25.00   25.00    0.00 1000.00
(2, 1) 1000.00   10.00   25.00   25.00    0.00 1000.00
(2, 2) 1000.00   10.00 1025.00   25.00 1000.00    0.00
```

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, TimeSeries*) – 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** (TimeSeries) – generic cashflow representing prepayments.

Returns A object of the class `Loan`.

```
>>> nrate = interest_rate(const_value=10, nper=11, pyr=4, spec=(5, 20))
>>> buydown_loan(amount=1000, nrate=nrate, dispoints=0, orgpoints=0, prepmt=None)
t          Beg.    Per.   Total    Int.    Ppal  Ending
           Ppal    Rate     Pmt     Pmt     Pmt    Ppal
-----------------------------------------------------
(0, 0) 1000.00   10.00    0.00    0.00    0.00 1000.00
(0, 1) 1000.00   10.00  114.26   25.00   89.26  910.74
(0, 2)  910.74   10.00  114.26   22.77   91.49  819.25
(0, 3)  819.25   10.00  114.26   20.48   93.78  725.47
(1, 0)  725.47   10.00  114.26   18.14   96.12  629.35
(1, 1)  629.35   20.00  123.99   31.47   92.53  536.83
(1, 2)  536.83   20.00  123.99   26.84   97.15  439.67
(1, 3)  439.67   20.00  123.99   21.98  102.01  337.66
(2, 0)  337.66   20.00  123.99   16.88  107.11  230.55
(2, 1)  230.55   20.00  123.99   11.53  112.47  118.09
(2, 2)  118.09   20.00  123.99    5.90  118.09    0.00
```

```
>>> pmt = cashflow(const_value=0, nper = 11, pyr=4, spec=((1, 3), 200))
>>> buydown_loan(amount=1000, nrate=nrate, dispoints=0, orgpoints=0, prepmt=pmt)
t          Beg.    Per.   Total    Int.    Ppal  Ending
           Ppal    Rate     Pmt     Pmt     Pmt    Ppal
-----------------------------------------------------
(0, 0) 1000.00   10.00    0.00    0.00    0.00 1000.00
(0, 1) 1000.00   10.00  114.26   25.00   89.26  910.74
(0, 2)  910.74   10.00  114.26   22.77   91.49  819.25
(0, 3)  819.25   10.00  114.26   20.48   93.78  725.47
(1, 0)  725.47   10.00  114.26   18.14   96.12  629.35
(1, 1)  629.35   20.00  123.99   31.47   92.53  536.83
(1, 2)  536.83   20.00  123.99   26.84   97.15  439.67
(1, 3)  439.67   20.00  323.99   21.98  302.01  137.66
(2, 0)  137.66   20.00   50.55    6.88   43.67   94.00
(2, 1)   94.00   20.00   50.55    4.70   45.85   48.14
(2, 2)   48.14   20.00   50.55    2.41   48.14    0.00
```

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*, TimeSeries) – 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** (TimeSeries) – generic cashflow representing prepayments.
- **balloonpmt** (TimeSeries) – generic cashflow representing balloon payments.

Returns A object of the class `Loan`.

```
>>> nrate = interest_rate(const_value=10, nper=11, pyr=4)
>>> fixed_ppal_loan(amount=1000, nrate=nrate, grace=0, dispoints=0, orgpoints=0,
...                 prepmt=None, balloonpmt=None)
```

(continues on next page)

(continued from previous page)

```
t          Beg.    Per.   Total    Int.    Ppal  Ending
           Ppal    Rate     Pmt     Pmt     Pmt    Ppal
-----------------------------------------------------
(0, 0)     0.00   10.00    0.00    0.00    0.00 1000.00
(0, 1)  1000.00   10.00  125.00   25.00  100.00  900.00
(0, 2)   900.00   10.00  122.50   22.50  100.00  800.00
(0, 3)   800.00   10.00  120.00   20.00  100.00  700.00
(1, 0)   700.00   10.00  117.50   17.50  100.00  600.00
(1, 1)   600.00   10.00  115.00   15.00  100.00  500.00
(1, 2)   500.00   10.00  112.50   12.50  100.00  400.00
(1, 3)   400.00   10.00  110.00   10.00  100.00  300.00
(2, 0)   300.00   10.00  107.50    7.50  100.00  200.00
(2, 1)   200.00   10.00  105.00    5.00  100.00  100.00
(2, 2)   100.00   10.00  102.50    2.50  100.00    0.00
```

```
>>> fixed_ppal_loan(amount=1000, nrate=nrate, grace=2, dispoints=0, orgpoints=0,
...                 prepmt=None, balloonpmt=None)
t          Beg.    Per.   Total    Int.    Ppal  Ending
           Ppal    Rate     Pmt     Pmt     Pmt    Ppal
-----------------------------------------------------
(0, 0)     0.00   10.00    0.00    0.00    0.00 1000.00
(0, 1)  1000.00   10.00   25.00   25.00    0.00 1000.00
(0, 2)  1000.00   10.00   25.00   25.00    0.00 1000.00
(0, 3)  1000.00   10.00  150.00   25.00  125.00  875.00
(1, 0)   875.00   10.00  146.88   21.88  125.00  750.00
(1, 1)   750.00   10.00  143.75   18.75  125.00  625.00
(1, 2)   625.00   10.00  140.62   15.62  125.00  500.00
(1, 3)   500.00   10.00  137.50   12.50  125.00  375.00
(2, 0)   375.00   10.00  134.38    9.38  125.00  250.00
(2, 1)   250.00   10.00  131.25    6.25  125.00  125.00
(2, 2)   125.00   10.00  128.12    3.12  125.00    0.00
```

```
>>> pmt = cashflow(const_value=0, nper = 11, pyr=4, spec=((1, 3), 200))
>>> fixed_ppal_loan(amount=1000, nrate=nrate, grace=2, dispoints=0, orgpoints=0,
...                 prepmt=pmt, balloonpmt=None)
t          Beg.    Per.   Total    Int.    Ppal  Ending
           Ppal    Rate     Pmt     Pmt     Pmt    Ppal
-----------------------------------------------------
(0, 0)     0.00   10.00    0.00    0.00    0.00 1000.00
(0, 1)  1000.00   10.00   25.00   25.00    0.00 1000.00
(0, 2)  1000.00   10.00   25.00   25.00    0.00 1000.00
(0, 3)  1000.00   10.00  150.00   25.00  125.00  875.00
(1, 0)   875.00   10.00  146.88   21.88  125.00  750.00
(1, 1)   750.00   10.00  143.75   18.75  125.00  625.00
(1, 2)   625.00   10.00  140.62   15.62  125.00  500.00
(1, 3)   500.00   10.00  337.50   12.50  325.00  175.00
(2, 0)   175.00   10.00  129.38    4.38  125.00   50.00
(2, 1)    50.00   10.00   51.25    1.25   50.00    0.00
(2, 2)     0.00   10.00    0.00    0.00    0.00    0.00
```

```
>>> pmt = cashflow(const_value=0, nper = 11, pyr=4, spec=((1, 3), 200))
>>> fixed_ppal_loan(amount=1000, nrate=nrate, grace=2, dispoints=0, orgpoints=0,
...                 prepmt=None, balloonpmt=pmt)
t          Beg.    Per.   Total    Int.    Ppal  Ending
           Ppal    Rate     Pmt     Pmt     Pmt    Ppal
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------
(0, 0)    0.00   10.00    0.00    0.00    0.00 1000.00
(0, 1) 1000.00   10.00   25.00   25.00    0.00 1000.00
(0, 2) 1000.00   10.00   25.00   25.00    0.00 1000.00
(0, 3) 1000.00   10.00  125.00   25.00  100.00  900.00
(1, 0)  900.00   10.00  122.50   22.50  100.00  800.00
(1, 1)  800.00   10.00  120.00   20.00  100.00  700.00
(1, 2)  700.00   10.00  117.50   17.50  100.00  600.00
(1, 3)  600.00   10.00  315.00   15.00  300.00  300.00
(2, 0)  300.00   10.00  107.50    7.50  100.00  200.00
(2, 1)  200.00   10.00  105.00    5.00  100.00  100.00
(2, 2)  100.00   10.00  102.50    2.50  100.00    0.00
```

cashflows.loan.**fixed_rate_loan**(*amount*, *nrate*, *life*, *start*, *pyr=1*, *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** (TimeSeries) – generic cashflow representing prepayments.
- **balloonpmt** (TimeSeries) – generic cashflow representing balloon payments.

Returns A object of the class Loan.

```
>>> pmt = cashflow(const_value=0, nper = 11, pyr=4, spec=((1, 3), 200))
>>> fixed_rate_loan(amount=1000, nrate=10, life=10, start=None, pyr=4, grace=0,␣
↪dispoints=0,
...                 orgpoints=0, prepmt=pmt, balloonpmt=None)
t          Beg.    Per.   Total     Int.    Ppal  Ending
           Ppal    Rate     Pmt      Pmt     Pmt    Ppal
------------------------------------------------------
(0, 0) 1000.00   10.00    0.00    0.00    0.00 1000.00
(0, 1) 1000.00   10.00  114.26   25.00   89.26  910.74
(0, 2)  910.74   10.00  114.26   22.77   91.49  819.25
(0, 3)  819.25   10.00  114.26   20.48   93.78  725.47
(1, 0)  725.47   10.00  114.26   18.14   96.12  629.35
(1, 1)  629.35   10.00  114.26   15.73   98.52  530.83
(1, 2)  530.83   10.00  114.26   13.27  100.99  429.84
(1, 3)  429.84   10.00  314.26   10.75  303.51  126.33
(2, 0)  126.33   10.00  114.26    3.16  111.10   15.23
(2, 1)   15.23   10.00   15.61    0.38   15.23    0.00
(2, 2)    0.00   10.00    0.00    0.00    0.00    0.00
```

CHAPTER 11

Savings

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

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

Parameters

- **deposits** (TimeSeries) – deposits to the account.
- **nrate** (TimeSeries) – nominal interest rate paid by the account.
- **initbal** (*float*) – initial balance of the account.
- **noprint** (*bool*) – prints summary report?

Returns interest, end_balance (TimeSeries, TimeSeries)

Examples

```
>>> cflo = cashflow(const_value=[100] * 12, pyr=4)
>>> nrate = interest_rate([10] * 12, pyr=4)
>>> savings(deposits=cflo, nrate=nrate, initbal=0, noprint=False)
t       Beginning    Deposit     Earned     Ending
          Balance             Interest    Balance
-----------------------------------------------
(0, 0)       0.00     100.00       0.00     100.00
(0, 1)     100.00     100.00       2.50     202.50
(0, 2)     202.50     100.00       5.06     307.56
(0, 3)     307.56     100.00       7.69     415.25
(1, 0)     415.25     100.00      10.38     525.63
(1, 1)     525.63     100.00      13.14     638.77
(1, 2)     638.77     100.00      15.97     754.74
(1, 3)     754.74     100.00      18.87     873.61
(2, 0)     873.61     100.00      21.84     995.45
(2, 1)     995.45     100.00      24.89    1120.34
(2, 2)    1120.34     100.00      28.01    1248.35
(2, 3)    1248.35     100.00      31.21    1379.56
```

```
>>> cflo = cashflow(const_value=[100] * 5, spec=[(0, 0), (2, 0)])
>>> nrate = interest_rate([0, 1, 2, 3, 4])
>>> savings(deposits=cflo, nrate=nrate, initbal=1000, noprint=False)
t    Beginning   Deposit    Earned    Ending
       Balance            Interest   Balance
-----------------------------------------------
(0,)   1000.00      0.00      0.00   1000.00
(1,)   1000.00    100.00     10.00   1110.00
(2,)   1110.00      0.00     22.20   1132.20
(3,)   1132.20    100.00     33.97   1266.17
(4,)   1266.17    100.00     50.65   1416.81
```

CHAPTER 12

Economic utility functions

cashflows.utilityfun.**exp_utility_fun**(*r_param*)
Exponential utility function.

U(x) = 1 - exp(-x / r_param)

cashflows.utilityfun.**log_utility_fun**(*r_param*)
Exponential utility function.

U(x) = log(x + r_param)

cashflows.utilityfun.**sqrt_utility_fun**(*r_param*)
Exponential utility function.

U(x) = + sqrt(x + r_param)

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

T

V