Package 'tidyactuarial'

May 29, 2026

Type Package

Title Tidy Tools for Actuarial Mathematics and Life Contingencies

Version 0.1.3

Description Provides tidyverse-aligned tools for actuarial mathematics and life contingencies, including life tables, survival probabilities, actuarial present values of cash flows, life annuities, life insurance, premiums, reserves, multiple-life calculations, Monte Carlo simulation, and deterministic cash-flow diagrams. The package emphasizes clear actuarial notation, reproducible workflows, and pipe-friendly tools for actuarial education and applied actuarial analysis.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports dplyr, ggplot2, rlang, scales, stats, tibble, utils

Suggests purrr, testthat (>= 3.0.0)

LazyData true

LazyDataCompression xz

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

Author Julian Fajardo [aut, cre]

Maintainer Julian Fajardo <julian.fajardo1908@gmail.com>

Repository CRAN

Date/Publication 2026-05-28 22:50:02 UTC

Contents

`amort_schedule` *Amortization schedule with optional prepayment adjustment*

Description

Builds an amortization schedule under a fixed annual interest-rate specification, allowing extra principal payments and optional adjustment of either the remaining term or the remaining payment amount.

Usage

```
amort_schedule(
  principal,
  n,
  rate,
  rate_type = "effective",
  m = 1L,
  periods_per_year = 1L,
  timing = c("immediate", "due"),
```

```
  payment = NULL,
  extra_principal = NULL,
  adjust = c("none", "term", "payment"),
  tol = 1e-08
)
```

Arguments

`principal`	Numeric scalar. Initial outstanding balance.
`n`	Positive integer. Number of contractual periods.
`rate`	Numeric scalar. Annual rate value.
`rate_type`	Character string indicating the annual rate type: `"effective"`, `"nominal_interest"`, `"nominal_discount"`, or `"force"`.
`m`	Positive integer. Compounding frequency for nominal annual rates.
`periods_per_year`	Positive integer. Number of schedule periods per year.
`timing`	Character string. One of `"immediate"` or `"due"`.
`payment`	Optional numeric scalar. Initial regular payment per period. If `NULL`, it is computed from the loan data.
`extra_principal`	Optional extra principal payments. Can be: • `NULL`, • a scalar, • an unnamed numeric vector of length `n`, • a named numeric vector with names interpreted as period numbers.
`adjust`	Character string. One of `"none"`, `"term"`, or `"payment"`.
`tol`	Numeric tolerance for zero-balance detection.

Details

The annual rate is converted internally to an effective rate per schedule period using `periods_per_year`.

Adjustment policies after extra principal payments:

- `"none"`: keep the original payment and contractual term, unless the loan is fully repaid early.
- `"term"`: keep the regular payment and shorten the term. No special logic is needed: the loop exits naturally when the outstanding balance reaches zero.
- `"payment"`: keep the remaining contractual term and recalculate the regular payment after each period.

For `timing = "immediate"` (annuity-immediate), interest accrues on the outstanding balance during the period, and the payment is made at the end. For `timing = "due"` (annuity-due), the payment is made at the start of the period and interest accrues on the balance after the payment.

If the user supplies a custom `payment` that is smaller than the periodic interest, principal repayment will be negative (negative amortization). This is permitted but the user should be aware.

Value

A tibble with one row per realized period and columns:

period Period index.

ob_start Outstanding balance at the start of the period.

interest Interest charged during the period.

payment Regular payment in the period.

extra_principal Extra principal paid in the period.

principal Principal repaid through the regular payment.

total_principal Total principal repaid in the period.

cashflow Total payment made in the period.

ob_end Outstanding balance at the end of the period.

i_effective_annual Equivalent annual effective rate.

i_effective_period Equivalent effective rate per schedule period.

periods_per_year Schedule frequency.

timing Payment timing convention.

adjust Adjustment rule used.

See Also

a_angle, present_value, pv_flow, standardize_interest

Other amortization: sinking_fund_schedule()

Examples

```
amort_schedule(
  principal = 100000,
  n = 12,
  rate = 0.12,
  rate_type = "nominal_interest",
  m = 12,
  periods_per_year = 12
)

amort_schedule(
  principal = 100000,
  n = 24,
  rate = 0.12,
  rate_type = "nominal_interest",
  m = 12,
  periods_per_year = 12,
  extra_principal = c("6" = 5000, "12" = 3000),
  adjust = "term"
)

amort_schedule(
```

```
  principal = 100000,
  n = 24,
  rate = 0.12,
  rate_type = "nominal_interest",
  m = 12,
  periods_per_year = 12,
  extra_principal = c("6" = 5000, "12" = 3000),
  adjust = "payment"
)
```

`annuity_arith` *Arithmetic annuity factor*

Description

Computes the actuarial present value or accumulated value factor for an arithmetic annuity.

Usage

```
annuity_arith(
  n_years,
  payments_per_year = 1L,
  rate,
  rate_type = "effective",
  m = 1L,
  deferral_years = 0,
  timing = "immediate",
  pattern = "increasing",
  first_payment = 1,
  increment = 1,
  valuation = c("present", "accumulated"),
  output = c("value", "table")
)
```

Arguments

`n_years`	Numeric vector of payment durations in years. Each value must be positive and finite.
`payments_per_year`	Positive integer vector giving the number of payments per year.
`rate`	Numeric vector of rate values.
`rate_type`	Character vector indicating the rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer vector giving the compounding frequency for nominal rates.
`deferral_years`	Numeric vector of deferral times in years. Must be greater than or equal to 0.

timing	Character vector. One of "immediate" or "due".
pattern	Character vector. One of "increasing", "decreasing", or "custom".
first_payment	Numeric vector. First payment for pattern = "custom". Ignored for "increasing" and "decreasing".
increment	Numeric vector. Arithmetic increment for pattern = "custom". Ignored for "increasing" and "decreasing".
valuation	Character string. Use "present" for present value or "accumulated" for accumulated value at the end of the term.
output	Character string. Use "value" to return a numeric factor, or "table" to return a tibble with intermediate quantities.

Details

This function covers increasing, decreasing, and custom arithmetic payment patterns. It replaces the more specific increasing and decreasing annuity functions.

Supported payment patterns:

- "increasing": payments $1, 2, \ldots, n$.
- "decreasing": payments $n, n-1, \ldots, 1$.
- "custom": payments following $P_r = P_1 + (r-1)g$.

Supported timing conventions:

- "immediate": payments at the end of each period.
- "due": payments at the beginning of each period.

The function first converts the supplied rate to the equivalent annual effective interest rate using standardize_interest.

For each scenario, the total number of payments is

$$N = n \times k$$

where n is n_years and k is payments_per_year.

The present value is computed by summing the discounted payment stream. The accumulated value is computed at the end of the annuity term. Under this convention, deferral_years affects the present value factor but not the accumulated value factor.

Value

If output = "value", a numeric vector of arithmetic annuity factors.

If output = "table", a tibble with input values, equivalent rates, period quantities, and both present and accumulated value factors.

See Also

a_angle, s_angle, standardize_interest

Other annuities: a_angle(), annuity_geom(), s_angle()

Examples

```
# Increasing arithmetic annuity
annuity_arith(n_years = 10, rate = 0.05, pattern = "increasing")

# Decreasing arithmetic annuity
annuity_arith(n_years = 10, rate = 0.05, pattern = "decreasing")

# Custom arithmetic annuity
annuity_arith(
  n_years = 10,
  rate = 0.05,
  pattern = "custom",
  first_payment = 100,
  increment = 25
)

# Accumulated value factor
annuity_arith(
  n_years = 10,
  rate = 0.05,
  pattern = "increasing",
  valuation = "accumulated"
)

# Tibble output
annuity_arith(
  n_years = 10,
  rate = 0.05,
  pattern = "decreasing",
  output = "table"
)
```

---

annuity_geom *Geometric annuity factor*

Description

Computes the present value or accumulated value factor for a geometric annuity.

Usage

```
annuity_geom(
  n_years = NULL,
  payments_per_year = 1L,
  rate,
  rate_type = "effective",
  m = 1L,
  growth_rate = 0,
```

```
  growth_rate_type = "effective",
  growth_m = 1L,
  deferral_years = 0,
  timing = "immediate",
  first_payment = 1,
  perpetuity = FALSE,
  valuation = c("present", "accumulated"),
  output = c("value", "table")
)
```

Arguments

n_years	Numeric vector of payment durations in years. Ignored only when perpetuity = TRUE and valuation = "present". Otherwise, each value must be positive and finite.
payments_per_year	Positive integer vector giving the number of payments per year.
rate	Numeric vector of annual interest-rate values.
rate_type	Character vector indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer vector giving the compounding frequency for nominal interest-rate inputs.
growth_rate	Numeric vector of annual growth-rate values for the payments.
growth_rate_type	Character vector indicating the growth-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
growth_m	Positive integer vector giving the compounding frequency for nominal growth-rate inputs.
deferral_years	Numeric vector of deferral times in years. Must be greater than or equal to 0. For present values, the deferral discounts the payment block. For accumulated values, it is recorded but does not change the factor under the adopted terminal-horizon convention.
timing	Character vector. One of "immediate" or "due".
first_payment	Numeric vector giving the first payment of the geometric sequence.
perpetuity	Logical vector. If TRUE, computes the geometric perpetuity present value. Perpetuities are not supported for valuation = "accumulated".
valuation	Character string. Use "present" for present value or "accumulated" for accumulated value at the end of the term.
output	Character string. Use "value" to return a numeric value, or "table" to return a tibble with intermediate quantities.

Details

This function replaces the more specific geometric present value and accumulated value functions.

Supported timing conventions:

- `"immediate"`: payments at the end of each period.
- `"due"`: payments at the beginning of each period.

Let i_p be the effective interest rate per payment period, g_p the effective growth rate per payment period, and $v_p = (1 + i_p)^{-1}$.

For a finite geometric annuity-immediate with N payment periods:

$$ga_N = \sum_{r=1}^{N} \frac{(1+g_p)^{r-1}}{(1+i_p)^r}$$

If $i_p \neq g_p$, then

$$ga_N = \frac{1 - \left(\frac{1+g_p}{1+i_p}\right)^N}{i_p - g_p}$$

The accumulated value factor is computed at the standard terminal horizon:

$$gs_N = \sum_{r=1}^{N} (1+g_p)^{r-1}(1+i_p)^{N-r}.$$

For annuities-due, the corresponding immediate factor is multiplied by $1 + i_p$.

Value

If `output = "value"`, a numeric vector.

If `output = "table"`, a tibble with input values, equivalent rates, period quantities, and both present and accumulated value factors.

See Also

`a_angle`, `s_angle`, `annuity_arith`, `standardize_interest`

Other annuities: `a_angle()`, `annuity_arith()`, `s_angle()`

Examples

```
# Present value of a geometric annuity
annuity_geom(
  n_years = 10,
  rate = 0.05,
  growth_rate = 0.02,
  valuation = "present"
)

# Accumulated value of a geometric annuity
annuity_geom(
  n_years = 10,
  rate = 0.05,
  growth_rate = 0.02,
  valuation = "accumulated"
```

```
)

# Tibble output
annuity_geom(
  n_years = 10,
  rate = 0.05,
  growth_rate = 0.02,
  output = "table"
)
```

`annuity_multi` *Actuarial present value of a multi-life annuity (up to 3 independent lives)*

Description

Computes the APV of a discrete annuity contingent on multiple independent lives. This implementation supports up to three lives (the most common practical case, e.g., parent–parent–child arrangements).

Usage

```
annuity_multi(
  lt,
  ages,
  annuity = c("cohort", "reversionary"),
  cohort = c("first", "last"),
  alpha = NULL,
  n = NULL,
  m = 0L,
  k = 1L,
  timing = c("immediate", "due"),
  woolhouse = c("none", "first", "second"),
  i
)
```

Arguments

`lt`	A lifetable object (data.frame/tibble) with column `x` and at least one of `lx`, `px`, `qx`. For different mortality assumptions by life, you may pass a list of lifetables of the same length as `ages` (e.g., `list(lt_male, lt_female, lt_child)`).
`ages`	Integer vector of actuarial ages for the lives at issue. Must have length 1–3.
`annuity`	Type of annuity logic: `"cohort"` uses the status defined in `cohort`; `"reversionary"` uses the α fractional reduction.
`cohort`	Survival status: `"first"` (joint-life, pays while all are alive) or `"last"` (last-survivor, pays while at least one remains alive). Used only when `annuity = "cohort"`.

alpha	Reversionary fraction (typically $0 \leq \alpha \leq 1$). Used only when annuity = "reversionary". Note: alpha = 0 matches joint-life; alpha = 1 matches last-survivor.
n	Integer term in years after deferment. If NULL, runs to the end of the available table horizon (conservatively based on the smallest omega across lifetables).
m	Integer deferment in years.
k	Integer payments per year. If k > 1, Woolhouse approximations may be applied.
timing	"immediate" or "due".
woolhouse	"none", "first", or "second".
i	Annual effective interest rate.

Details

Supports status-based annuities (joint-life / last-survivor) and a joint-and-survivor style annuity ("reversionary") that pays 1 while all lives are alive and then pays a fraction α while at least one life remains alive.

Under the assumption of independent future lifetimes (Finan, Section 51), the survival probability for the status is calculated as:

- **Joint-life (first-death):** ${}_tp_{x_1x_2\ldots x_n} = \prod_{j=1}^{n} {}_tp_{x_j}$
- **Last-survivor:** ${}_tp_{\overline{x_1x_2\ldots x_n}} = 1 - \prod_{j=1}^{n}(1 - {}_tp_{x_j})$

For annuity = "reversionary", the APV is a weighted combination of the two statuses (Finan, Section 53.3):

$$APV = APV(\text{joint-life}) + \alpha[APV(\text{last-survivor}) - APV(\text{joint-life})]$$

Value

A single numeric value representing the APV.

See Also

annuity_x for single-life annuities, t_px for survival probabilities.

---

annuity_x *Actuarial present value of a life annuity*

Description

Computes the actuarial present value of a discrete life annuity using a life table.

Usage

```
annuity_x(
  mortality_table,
  age,
  rate,
  rate_type = "effective",
  m = 1L,
  term_years = Inf,
  deferral_years = 0L,
  payments_per_year = 1L,
  timing = c("immediate", "due"),
  woolhouse = c("none", "first", "second"),
  output = c("value", "table")
)
```

Arguments

`mortality_table`	A life table as produced by `lifetable`. It must contain columns `x` and `lx`.
`age`	Integer actuarial age.
`rate`	Numeric scalar. Annual interest-rate input.
`rate_type`	Character string indicating the rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Compounding frequency for nominal rates. Ignored for `rate_type = "effective"` and `rate_type = "force"`.
`term_years`	Integer term in years. Use `Inf` for a whole-life annuity.
`deferral_years`	Integer deferral period in years.
`payments_per_year`	Positive integer. Number of payments per year. For example, use `12` for monthly payments.
`timing`	Character string. Either `"immediate"` for payments at the end of each payment period or `"due"` for payments at the beginning of each payment period.
`woolhouse`	Character string. For `payments_per_year > 1`, use `"none"` for exact UDD, `"first"` for the first-order Woolhouse approximation, or `"second"` for the second-order Woolhouse approximation.
`output`	Character string. Use `"value"` to return a numeric APV or `"table"` to return a one-row tibble with intermediate quantities.

Details

The function supports:

- whole-life annuities,
- temporary annuities,
- integer deferral,

- annual or k-thly payments,
- exact fractional survival under UDD,
- first- and second-order Woolhouse approximations.

For annual annuities-due,

$$\ddot{a}_{x:\overline{n}|} = \sum_{j=0}^{n-1} v^j \, {}_jp_x.$$

For annual annuities-immediate,

$$a_{x:\overline{n}|} = \sum_{j=1}^{n} v^j \, {}_jp_x.$$

Deferral is handled through

$$v^h \, {}_hp_x$$

where h is `deferral_years`.

For k-thly payments with `woolhouse = "none"`, fractional survival is computed under UDD.

Value

If `output = "value"`, a numeric scalar containing the actuarial present value.

If `output = "table"`, a one-row tibble with the main input values, equivalent interest rate, deferral factor, pure endowment factor, and APV.

See Also

`insurance_x`, `premium_x`, `reserve_x`, `t_px`, `t_Ex`

Other life-contingencies: `annuity_xy()`, `insurance_x()`, `insurance_xy()`, `life_contract()`, `premium_gross()`, `premium_x()`, `premium_xy()`, `reserve_x()`, `reserve_xy()`, `simulate_annuity_x()`, `simulate_insurance_x()`

Examples

```
lt <- data.frame(
  x  = 60:65,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000)
)

# Annual annuity-immediate
annuity_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
  timing = "immediate"
)

# Annual annuity-due
annuity_x(
  mortality_table = lt,
```

```
  age = 60,
  rate = 0.06,
  timing = "due"
)

# Temporary annuity
annuity_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
  term_years = 3,
  timing = "due"
)

# Deferred annuity
annuity_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
  deferral_years = 2,
  timing = "due"
)

# Table output
annuity_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
  term_years = 3,
  timing = "due",
  output = "table"
)
```

`annuity_xy` *Actuarial present value of a two-life annuity*

Description

Computes the actuarial present value of a discrete annuity contingent on two independent lives. Supports joint-life, last-survivor, and state-based reversionary-style payments.

Usage

```
annuity_xy(
  mortality_table,
  age_x = NULL,
  age_y = NULL,
  rate = NULL,
  rate_type = NULL,
```

```
  m = NULL,
  cohort = c("first", "last"),
  benefit = NULL,
  term_years = Inf,
  deferment_years = 0L,
  payments_per_year = 1L,
  timing = c("immediate", "due"),
  woolhouse = c("none", "first", "second"),
  frac,
  output = c("value", "table")
)
```

Arguments

`mortality_table`	Either a single life table used for both lives, a list of two life tables `list(table_x, table_y)`, or a `tidyact_life_contract` object created by `life_contract()`. Each life table must contain columns `x` and `lx`.
`age_x`	Integer actuarial age for the first life.
`age_y`	Integer actuarial age for the second life.
`rate`	Numeric scalar. Annual interest-rate input.
`rate_type`	Character string indicating the rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Compounding frequency for nominal rates.
`cohort`	Character string. Use `"first"` for joint-life payments while both lives are alive, or `"last"` for last-survivor payments while at least one life is alive. Ignored when `benefit` is supplied.
`benefit`	Optional list with weights `both`, `x_only`, and `y_only`.
`term_years`	Term in years. Use `Inf` for the maximum horizon allowed by the life tables.
`deferment_years`	Integer deferment period in years.
`payments_per_year`	Positive integer. Number of payments per year.
`timing`	Payment timing. Use `"immediate"` or `"due"`.
`woolhouse`	Woolhouse approximation for `payments_per_year` > 1: `"none"`, `"first"`, or `"second"`.
`frac`	Fractional-age assumption for exact k-thly computation: `"UDD"`, `"CF"`, `"CML"`, or `"Balducci"`.
`output`	Character string. Use `"value"` for a numeric APV or `"table"` for a one-row tibble.

Details

This function assumes independent future lifetimes.

Value

If `output = "value"`, a numeric scalar. If `output = "table"`, a one-row tibble.

See Also

`annuity_x()`, `insurance_xy()`, `premium_xy()`, `t_pxy()`

Other life-contingencies: `annuity_x()`, `insurance_x()`, `insurance_xy()`, `life_contract()`, `premium_gross()`, `premium_x()`, `premium_xy()`, `reserve_x()`, `reserve_xy()`, `simulate_annuity_x()`, `simulate_insurance_x()`

Examples

```
lt <- data.frame(
  x  = 60:66,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000, 86000)
)

annuity_xy(
  mortality_table = lt,
  age_x = 60,
  age_y = 62,
  rate = 0.05,
  cohort = "first",
  timing = "due"
)

lt |>
  life_contract(lives = "joint", age_x = 60, age_y = 62, rate = 0.05) |>
  annuity_xy(cohort = "last", timing = "due")
```

---

apv_life_flow *Actuarial present value of a payment stream under mortality*

Description

Computes the actuarial present value (APV) of a cash-flow stream contingent on survival. The life table is supplied as the first argument (pipe-friendly). Payments may be specified by numeric times (years from 0) or by calendar dates.

Usage

```
apv_life_flow(
  lt,
  ages,
  time = NULL,
  date = NULL,
  start_date = NULL,
```

```
  cf,
  i,
  status = c("single", "first", "last", "reversionary"),
  alpha = NULL,
  plot = FALSE
)
```

Arguments

lt	A life table data frame with column x and at least one of lx, px, or qx.
ages	Integer vector of actuarial ages (length 1 for single life, length 2+ for multiple lives).
time	Numeric vector of payment times in years (>= 0). Provide either time or date.
date	Optional vector of Date payment dates. Provide either time or date.
start_date	Optional Date used as time 0 when date is provided. If missing, the minimum of date is used.
cf	Numeric vector of cash flows (same length as time or date).
i	Annual effective interest rate (single numeric value).
status	Survival status: "single", "first", "last", or "reversionary".
alpha	Reversionary fraction for status = "reversionary" (single numeric value). While all lives are alive, full benefit is paid; while at least one but not all are alive, alpha times the benefit is paid.
plot	Logical; if TRUE, attaches a ggplot object in attr(result, "plot") showing cumulative APV over time.

Details

Multiple lives are supported under an independence assumption, through common statuses: single-life, first-death (all alive), last-survivor (any alive), and reversionary (joint-and-survivor) with fraction alpha.

For each payment at time t, the contribution to the APV is (Finan, Sections 33 and 37):

$$PV(t) = C(t) \times v^t \times P(StatusAliveMatT)$$

The survival probability depends on the status:

- "single": ${}_tp_x$ (single life).
- "first": ${}_tp_{x_1} \cdot {}_tp_{x_2} \ldots$ (all must be alive - joint life).
- "last": $1 - \prod(1 - {}_tp_{x_j})$ (at least one alive - last survivor).
- "reversionary": full benefit while all alive, fraction α while partially alive.

Fractional-year survival is computed under UDD within each year (Finan, Section 24.1).

Value

A tibble with one row per payment and columns: time, cf, surv_prob, discount, expected_cf, pv, pv_cum. If date was provided, a date column is included. The total APV is stored as attr(result, "apv").

a_angle *Level annuity factor a-angle-n*

Description

Computes the actuarial present value factor for a level annuity.

Usage

```
a_angle(
  n_years = NULL,
  payments_per_year = 1L,
  rate,
  rate_type = "effective",
  m = 1L,
  deferral_years = 0,
  timing = "immediate",
  perpetuity = FALSE,
  payment = 1,
  output = c("value", "table")
)
```

Arguments

n_years	Numeric vector of payment durations in years. Ignored when perpetuity = TRUE. If perpetuity = FALSE, each value must be positive and finite.
payments_per_year	Positive integer vector giving the number of discrete payments per year. Ignored for continuous annuities.
rate	Numeric vector of rate values.
rate_type	Character vector indicating the rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer vector giving the compounding frequency for nominal rates. Ignored for "effective" and "force".
deferral_years	Numeric vector of deferral times in years. Must be greater than or equal to 0.
timing	Character vector. One of "immediate", "due", or "continuous".
perpetuity	Logical vector. If TRUE, computes the perpetuity factor.
payment	Numeric vector of level payment amounts. Used only when output = "table" to report the corresponding present value. The annuity factor itself is always computed for unit payments.
output	Character string. Use "value" to return a numeric annuity factor, or "table" to return a tibble with intermediate calculations.

Details

Supported timing conventions:

- `"immediate"`: annuity-immediate with discrete payments.
- `"due"`: annuity-due with discrete payments.
- `"continuous"`: continuous annuity.

For discrete annuities, `payments_per_year = k` means payments are made every $1/k$ year. The function returns the annuity factor, assuming a unit payment at each payment time.

Deferral is supported through `deferral_years = h`. For discrete annuities, the deferral must align with the payment grid, that is, hk must be an integer.

If `perpetuity = TRUE`, the infinite-term annuity factor is returned.

The function first converts the supplied rate to the equivalent annual effective interest rate using `standardize_interest`.

For finite discrete annuities:

$$a_{\overline{n}|} = \frac{1 - v^n}{i}$$

For due annuities:

$$\ddot{a}_{\overline{n}|} = (1 + i) a_{\overline{n}|}$$

For continuous annuities:

$$\bar{a}_{\overline{n}|} = \frac{1 - e^{-\delta n}}{\delta}$$

Input vectors must have length 1 or a common length. Missing values are propagated.

Value

If `output = "value"`, a numeric vector of annuity factors.

If `output = "table"`, a tibble with input values, equivalent rates, annuity factors, payment amounts, and present values.

See Also

`s_angle`, `standardize_interest`, `present_value`

Other annuities: `annuity_arith()`, `annuity_geom()`, `s_angle()`

Examples

```
# Numeric annuity factor
a_angle(n_years = 10, rate = 0.05)

# Nominal interest converted monthly, with monthly payments
a_angle(
  n_years = 10,
  rate = 0.06,
  rate_type = "nominal_interest",
  m = 12,
```

```
  payments_per_year = 12
)

# Continuous annuity
a_angle(
  n_years = 15,
  rate = 0.04,
  rate_type = "force",
  timing = "continuous"
)

# Tibble output for teaching or auditing
a_angle(
  n_years = 10,
  rate = 0.05,
  payment = 1000,
  output = "table"
)

# Vectorized example
a_angle(
  n_years = c(5, 10, 20),
  payments_per_year = c(1, 12, 1),
  rate = c(0.05, 0.06, 0.04),
  rate_type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  deferral_years = c(0, 0, 2),
  timing = c("immediate", "immediate", "continuous"),
  perpetuity = c(FALSE, FALSE, FALSE)
)
```

`bonds_sample` *Sample bond contracts for fixed-income examples*

Description

A small pedagogical dataset containing bond contracts for pricing, yield-to-maturity, duration, and convexity examples.

A small pedagogical dataset containing bond contracts for pricing, yield-to-maturity, duration, and convexity examples.

Usage

```
bonds_sample
```

```
bonds_sample
```

Format

A tibble with 5 rows and 8 variables:

bond_id Bond identifier.

face_value Face value of the bond.

coupon_rate Annual coupon rate.

coupon_frequency Number of coupon payments per year.

maturity_years Maturity in years.

yield_rate Annual nominal yield rate consistent with coupon frequency.

bond_type Short bond type label.

price Theoretical bond price computed from the listed yield rate.

A tibble with 5 rows and 8 variables:

bond_id Bond identifier.

face_value Face value of the bond.

coupon_rate Annual coupon rate.

coupon_frequency Number of coupon payments per year.

maturity_years Maturity in years.

yield_rate Annual nominal yield rate consistent with coupon frequency.

bond_type Short bond type label.

price Theoretical bond price computed from the listed yield rate.

Source

Synthetic pedagogical data created for tidyactuarial examples.

Synthetic pedagogical data created for tidyactuarial examples.

Examples

```
data(bonds_sample)

bonds_sample |>
  dplyr::select(bond_id, face_value, coupon_rate, maturity_years, price)

data(bonds_sample)

bonds_sample |>
  dplyr::select(bond_id, face_value, coupon_rate, maturity_years, price)
```

bond_book_value *Book value of a level coupon bond at a coupon date*

Description

Computes the book value of a level coupon bond at one or more coupon dates, under a specified yield basis.

Usage

```
bond_book_value(
  face,
  coupon_rate,
  years_to_maturity,
  valuation_time,
  coupons_per_year = 1L,
  yield_effective_per_period = NULL,
  yield_rate = NULL,
  yield_rate_type = "effective",
  yield_m = 1L,
  redemption = NULL,
  tol = 1e-10,
  check = TRUE,
  output = c("value", "table")
)
```

Arguments

face	Numeric scalar. Face or par value of the bond.
coupon_rate	Numeric scalar. Annual coupon rate as a proportion.
years_to_maturity	Numeric scalar. Final maturity in years.
valuation_time	Numeric vector. Valuation time(s) in years, measured from issue. Each value must lie between 0 and years_to_maturity and must align with coupon dates.
coupons_per_year	Positive integer. Number of coupon payments per year.
yield_effective_per_period	Optional numeric scalar. Effective yield per coupon period. If supplied, it is used directly.
yield_rate	Optional numeric scalar. Annual yield rate value.
yield_rate_type	Character string indicating the annual yield type: "effective", "nominal_interest", "nominal_discount", or "force".
yield_m	Positive integer. Compounding frequency for nominal annual yields.
redemption	Numeric scalar. Redemption value at final maturity. If NULL, defaults to face.

tol	Numeric scalar. Tolerance used in alignment checks.
check	Logical scalar. If TRUE, performs input validation.
output	Character string. Use "value" to return a numeric vector of book values, or "table" to return a tibble with intermediate quantities.

Details

The book value is interpreted prospectively: at a valuation time that lies on the coupon grid, it equals the present value at that time of all remaining future coupons and the final redemption amount, discounted at the bond's yield basis.

This function interprets valuation_time as a time immediately after any coupon due at that date has been paid. Therefore:

- at valuation_time = 0, the book value equals the bond price,
- at valuation_time = years_to_maturity, the book value is 0.

Assumptions:

- Coupons are paid in arrears at regular intervals.
- years_to_maturity * coupons_per_year must be an integer.
- Each valuation_time * coupons_per_year must be an integer.
- Stub periods are not supported.
- Valuation is performed at coupon dates; no accrued interest is included.

Yield input conventions:

- If yield_effective_per_period is supplied, it takes precedence over yield_rate, yield_rate_type, and yield_m, and is interpreted as the effective yield per coupon period.
- Otherwise, yield_rate, yield_rate_type, and yield_m define an annual yield specification, which is converted first to annual effective yield and then to effective yield per coupon period.

Let the valuation time correspond to coupon period k, with total maturity period count N. If the remaining future cash flows are $C_{k+1}, \ldots, C_N$, and i_p is the effective yield per coupon period, then the book value at time k is

$$BV_k = \sum_{j=k+1}^{N} C_j (1 + i_p)^{-(j-k)}$$

This is the prospective book value on the bond's yield basis.

Value

If output = "value", a numeric vector of book values, one for each valuation_time.

If output = "table", a tibble with valuation times, valuation periods, book values, yield information, and bond inputs.

See Also

bond_price, bond_cash_flows, bond_duration, bond_convexity

Other bonds: bond_callable_price(), bond_convexity(), bond_duration(), bond_price(), bond_ytm(), portfolio_convexity(), portfolio_duration()

Examples

```
# Book value at time 0 equals price
bond_book_value(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 5,
  valuation_time = 0,
  coupons_per_year = 2,
  yield_rate = 0.06,
  yield_rate_type = "effective"
)

# Book value at several coupon dates
bond_book_value(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 5,
  valuation_time = c(0, 1, 2, 3, 4, 5),
  coupons_per_year = 1,
  yield_rate = 0.06,
  yield_rate_type = "effective"
)

# Table output
bond_book_value(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 5,
  valuation_time = c(0, 1, 2, 3, 4, 5),
  coupons_per_year = 1,
  yield_rate = 0.06,
  yield_rate_type = "effective",
  output = "table"
)

# Yield given directly per coupon period
bond_book_value(
  face = 1000,
  coupon_rate = 0.05,
  years_to_maturity = 10,
  valuation_time = c(0, 2, 4, 6),
  coupons_per_year = 2,
  yield_effective_per_period = 0.03
)
```

bond_callable_price *Price of a callable bond at a target minimum yield*

Description

Computes the maximum price an investor should pay for a callable bond in order to guarantee a specified minimum yield.

Usage

```
bond_callable_price(
  face,
  coupon_rate,
  years_to_maturity,
  coupons_per_year = 1L,
  call_times,
  call_prices,
  yield_effective_per_period = NULL,
  yield_rate = NULL,
  yield_rate_type = "effective",
  yield_m = 1L,
  redemption = NULL,
  tol = 1e-10,
  check = TRUE,
  output = c("value", "table")
)
```

Arguments

face	Numeric scalar. Face or par value of the bond.
coupon_rate	Numeric scalar. Annual coupon rate as a proportion.
years_to_maturity	Numeric scalar. Final maturity in years. Must be strictly positive.
coupons_per_year	Positive integer. Number of coupon payments per year.
call_times	Numeric vector of callable times in years. Each value must be strictly between 0 and years_to_maturity, and must align with coupon dates.
call_prices	Numeric vector of call prices corresponding to call_times.
yield_effective_per_period	Optional numeric scalar. Effective yield per coupon period. If supplied, it is used directly.
yield_rate	Optional numeric scalar. Annual yield rate value.
yield_rate_type	Character string indicating the annual yield type: "effective", "nominal_interest", "nominal_discount", or "force".

yield_m	Positive integer. Compounding frequency for nominal annual yields.
redemption	Numeric scalar. Redemption value at final maturity. If NULL, defaults to face.
tol	Numeric scalar. Tolerance used in alignment checks.
check	Logical scalar. If TRUE, performs input validation.
output	Character string. Use "value" to return the worst-case callable-bond price, or "table" to return a tibble with all redemption scenarios.

Details

The bond is evaluated under each possible redemption scenario:

- each callable date with its associated call price, and
- final maturity with its final redemption value.

For each scenario, the bond price is computed using the target yield. The callable-bond price returned by this function is the smallest of those scenario prices, that is, the maximum price consistent with the target yield under the least favorable redemption scenario for the investor.

This follows the standard actuarial/financial interpretation used in introductory fixed-income mathematics: when a bond is callable at the issuer's option, the investor must protect against the redemption scenario that is least favorable to the investor at the required yield.

Assumptions:

- Coupons are paid in arrears at regular intervals.
- years_to_maturity * coupons_per_year must be an integer.
- Each call_times * coupons_per_year must be an integer.
- Stub periods are not supported.
- Pricing is performed at a coupon date; no accrued interest is included.

Yield input conventions:

- If yield_effective_per_period is supplied, it takes precedence over yield_rate, yield_rate_type, and yield_m, and is interpreted as the effective yield per coupon period.
- Otherwise, yield_rate, yield_rate_type, and yield_m define an annual yield specification, which is converted first to annual effective yield and then to effective yield per coupon period.

Let the callable bond have possible redemption scenarios indexed by $j = 1, \ldots, J$, where each scenario corresponds either to a call date or to final maturity. For scenario j, let $P_j(y)$ denote the bond price computed at the target yield y assuming redemption occurs at that scenario time and value.

Then this function returns

$$\min_j P_j(y)$$

when output = "value".

This is the maximum price an investor can pay while still guaranteeing at least the target yield under the least favorable redemption scenario.

Value

If `output = "value"`, a numeric scalar: the worst-case callable-bond price consistent with the target yield.

If `output = "table"`, a tibble with one row per redemption scenario, including scenario prices and the worst-case indicator.

See Also

`bond_price`, `bond_cash_flows`, `bond_book_value`, `bond_ytm`

Other bonds: `bond_book_value()`, `bond_convexity()`, `bond_duration()`, `bond_price()`, `bond_ytm()`, `portfolio_convexity()`, `portfolio_duration()`

Examples

```
# Callable bond with two possible call dates
bond_callable_price(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 10,
  coupons_per_year = 2,
  call_times = c(5, 7),
  call_prices = c(105, 102),
  yield_rate = 0.06,
  yield_rate_type = "effective"
)

# Table output with all redemption scenarios
bond_callable_price(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 10,
  coupons_per_year = 2,
  call_times = c(5, 7),
  call_prices = c(105, 102),
  yield_rate = 0.06,
  yield_rate_type = "effective",
  output = "table"
)

# Target yield given directly per coupon period
bond_callable_price(
  face = 1000,
  coupon_rate = 0.05,
  years_to_maturity = 12,
  coupons_per_year = 2,
  call_times = c(4, 8),
  call_prices = c(1030, 1015),
  yield_effective_per_period = 0.028
)
```

bond_cash_flows *Cash flow structure of a level coupon bond*

Description

Builds the cash flow schedule of a level coupon bond with constant coupon rate and a single redemption payment at maturity.

Usage

```
bond_cash_flows(
  face,
  coupon_rate,
  years_to_maturity,
  coupons_per_year = 1L,
  redemption = NULL,
  tol = 1e-10,
  check = TRUE
)
```

Arguments

face	Numeric scalar. Face value.
coupon_rate	Numeric scalar. Annual coupon rate.
years_to_maturity	Numeric scalar. Years to maturity.
coupons_per_year	Positive integer. Payments per year.
redemption	Numeric scalar. Redemption value.
tol	Numeric scalar. Tolerance.
check	Logical scalar. Input validation.

Value

A tibble with the bond schedule.

bond_convexity *Discrete convexity of a level coupon bond under a flat yield*

Description

Computes discrete convexity measures for a level coupon bond valued under a flat yield-to-maturity assumption.

Usage

```
bond_convexity(
  face,
  coupon_rate,
  years_to_maturity,
  coupons_per_year = 1L,
  y_effective_per_period = NULL,
  y_rate = NULL,
  y_type = "effective",
  y_m = 1L,
  redemption = NULL,
  tol = 1e-10,
  check = TRUE
)
```

Arguments

face	Numeric scalar. Face (par) value of the bond.
coupon_rate	Numeric scalar. Annual coupon rate as a proportion.
years_to_maturity	Numeric scalar. Time to maturity in years.
coupons_per_year	Positive integer. Number of coupon payments per year.
y_effective_per_period	Optional numeric scalar. Effective yield per coupon period.
y_rate	Optional numeric scalar. Annual yield rate value.
y_type	Character string indicating the annual yield type: "effective", "nominal_interest", "nominal_discount", or "force".
y_m	Positive integer. Compounding frequency for nominal annual yields.
redemption	Numeric scalar. Redemption value at maturity. If NULL, defaults to face.
tol	Numeric scalar. Tolerance used to check maturity alignment.
check	Logical scalar. If TRUE, performs input validation.

Details

Assumptions:

- Coupons are paid in arrears at regular intervals.
- `years_to_maturity * coupons_per_year` must be an integer.
- Stub periods are not supported.
- Valuation is at a coupon date (no accrued interest).
- A single flat yield is used to discount all cash flows.

Yield input conventions:

- If `y_effective_per_period` is supplied, it is interpreted as the effective yield per coupon period.
- Otherwise, `y_rate`, `y_type`, and `y_m` define an annual yield specification, which is converted first to annual effective yield and then to effective yield per coupon period.

Let j be the effective yield per coupon period, m the number of coupon payments per year, and let cash flows C_k occur at coupon periods $k = 1, \ldots, N$. With $v = 1/(1+j)$ and $P = \sum_k C_k v^k$, the discrete convexity in coupon periods is

$$C_p = \frac{1}{P} \cdot \frac{\sum_{k=1}^{N} C_k k(k+1) v^k}{(1+j)^2}$$

Discrete convexity in years is C_p/m^2.

This is the second-order sensitivity of the bond price to changes in the yield per period. Together with `bond_duration`, it is used in the second-order Taylor approximation of price changes.

Value

A one-row tibble with:

price Dirty price at the given yield.

discrete_convexity_periods Discrete convexity in coupon periods.

discrete_convexity_years Discrete convexity in years.

yield_per_period Effective yield per coupon period.

yield_effective_annual Annual effective yield.

coupons_per_year Coupon frequency.

n_periods Total number of coupon periods.

See Also

`bond_duration`, `bond_price`, `bond_cash_flows`, `bond_book_value`, `bond_ytm`

Other bonds: `bond_book_value()`, `bond_callable_price()`, `bond_duration()`, `bond_price()`, `bond_ytm()`, `portfolio_convexity()`, `portfolio_duration()`

Examples

```
bond_convexity(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 5,
  coupons_per_year = 2,
  y_rate = 0.06,
  y_type = "effective"
)

bond_convexity(
  face = 1000,
  coupon_rate = 0.05,
  years_to_maturity = 10,
  coupons_per_year = 2,
  y_effective_per_period = 0.03
)
```

---

bond_duration *Macaulay and modified duration of a level coupon bond under a flat yield*

Description

Computes Macaulay duration and modified-duration measures for a level coupon bond valued under a flat yield-to-maturity assumption.

Usage

```
bond_duration(
  face,
  coupon_rate,
  years_to_maturity,
  coupons_per_year = 1L,
  y_effective_per_period = NULL,
  y_rate = NULL,
  y_type = "effective",
  y_m = 1L,
  redemption = NULL,
  tol = 1e-10,
  check = TRUE
)
```

Arguments

face	Numeric scalar. Face (par) value of the bond.
coupon_rate	Numeric scalar. Annual coupon rate as a proportion.

`years_to_maturity`
: Numeric scalar. Time to maturity in years.

`coupons_per_year`
: Positive integer. Number of coupon payments per year.

`y_effective_per_period`
: Optional numeric scalar. Effective yield per coupon period.

`y_rate`
: Optional numeric scalar. Annual yield rate value.

`y_type`
: Character string indicating the annual yield type: `"effective"`, `"nominal_interest"`, `"nominal_discount"`, or `"force"`.

`y_m`
: Positive integer. Compounding frequency for nominal annual yields.

`redemption`
: Numeric scalar. Redemption value at maturity. If `NULL`, defaults to `face`.

`tol`
: Numeric scalar. Tolerance used to check maturity alignment.

`check`
: Logical scalar. If `TRUE`, performs input validation.

Details

Assumptions:

- Coupons are paid in arrears at regular intervals.
- `years_to_maturity * coupons_per_year` must be an integer.
- Stub periods are not supported.
- Valuation is at a coupon date (no accrued interest).
- A single flat yield is used to discount all cash flows.

Yield input conventions:

- If `y_effective_per_period` is supplied, it is interpreted as the effective yield per coupon period.
- Otherwise, `y_rate`, `y_type`, and `y_m` define an annual yield specification, which is converted first to annual effective yield and then to effective yield per coupon period.

Let j be the effective yield per coupon period, m the number of coupon payments per year, and let cash flows C_k occur at coupon periods $k = 1, \ldots, N$. With $v = 1/(1+j)$ and $P = \sum_k C_k v^k$:

Macaulay duration in coupon periods:

$$D_p = \frac{\sum_{k=1}^{N} k\, C_k\, v^k}{P}.$$

Macaulay duration in years: $D = D_p/m$.

Modified duration with respect to j (in coupon periods): $D_j^* = D_p/(1+j)$.

Modified duration with respect to the annual effective rate i (in years): $D_i^* = D/(1+i)$, where $i = (1+j)^m - 1$.

Modified duration measures the first-order sensitivity of the bond price to yield changes. Together with `bond_convexity`, it forms the second-order Taylor approximation of price changes.

Value

A one-row tibble with:

price Dirty price at the given yield.

macaulay_duration_periods Macaulay duration in coupon periods.

macaulay_duration_years Macaulay duration in years.

modified_duration_periods_j Modified duration with respect to the effective yield per coupon period, expressed in coupon periods.

modified_duration_years_i Modified duration with respect to the annual effective yield, expressed in years.

yield_per_period Effective yield per coupon period.

yield_effective_annual Annual effective yield.

coupons_per_year Coupon frequency.

n_periods Total number of coupon periods.

See Also

bond_convexity, bond_price, bond_cash_flows, bond_book_value, bond_ytm

Other bonds: bond_book_value(), bond_callable_price(), bond_convexity(), bond_price(), bond_ytm(), portfolio_convexity(), portfolio_duration()

Examples

```
bond_duration(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 5,
  coupons_per_year = 2,
  y_rate = 0.06,
  y_type = "effective"
)

bond_duration(
  face = 1000,
  coupon_rate = 0.05,
  years_to_maturity = 10,
  coupons_per_year = 2,
  y_effective_per_period = 0.03
)
```

bond_price *Price of a level coupon bond from its yield*

Description

Computes the dirty price of a level coupon bond at time 0 from its yield.

Usage

```
bond_price(
  face,
  coupon_rate,
  years_to_maturity,
  coupons_per_year = 1L,
  y_effective_per_period = NULL,
  y_rate = NULL,
  y_type = "effective",
  y_m = 1L,
  redemption = NULL,
  tol = 1e-10,
  check = TRUE
)
```

Arguments

face	Numeric scalar. Face (par) value of the bond.
coupon_rate	Numeric scalar. Annual coupon rate as a proportion.
years_to_maturity	Numeric scalar. Time to maturity in years.
coupons_per_year	Positive integer. Number of coupon payments per year.
y_effective_per_period	Optional numeric scalar. Effective yield per coupon period. If supplied, it is used directly.
y_rate	Optional numeric scalar. Annual yield rate value.
y_type	Character string indicating the annual yield type: "effective", "nominal_interest", "nominal_discount", or "force".
y_m	Positive integer. Compounding frequency for nominal annual yields.
redemption	Numeric scalar. Redemption value at maturity. If NULL, defaults to face.
tol	Numeric scalar. Tolerance used when checking alignment of maturity with coupon periods.
check	Logical scalar. If TRUE, performs basic input validation.

Details

Assumptions:

- Coupons are paid in arrears at regular intervals.
- `years_to_maturity * coupons_per_year` must be an integer.
- Stub periods are not supported.
- No accrued interest is considered; the price is evaluated at a coupon date.

The yield may be supplied in either of two ways:

- directly as an effective yield per coupon period through `y_effective_per_period`, or
- as an annual rate specification through `y_rate`, `y_type`, and `y_m`.

If an annual rate specification is supplied, it is first converted to the equivalent annual effective yield and then to the effective yield per coupon period.

Let j be the effective yield per coupon period, m the number of coupon payments per year, and $N = T \times m$ the total number of periods. With coupon per period $C = Fr/m$ and discount factor $v = 1/(1+j)$, the price is:

$$P = C \cdot a_{\overline{N}||j} + R \cdot v^N = \sum_{k=1}^{N} C_k \, v^k$$

where the sum runs over all cash flows (coupons and redemption) indexed by coupon period k.

Value

Numeric scalar: dirty price of the bond at time 0.

See Also

`bond_ytm`, `bond_cash_flows`, `bond_duration`, `bond_convexity`, `bond_book_value`, `bond_callable_price`

Other bonds: `bond_book_value()`, `bond_callable_price()`, `bond_convexity()`, `bond_duration()`, `bond_ytm()`, `portfolio_convexity()`, `portfolio_duration()`

Examples

```
# 5-year annual coupon bond, yield given as annual effective
bond_price(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 5,
  coupons_per_year = 1,
  y_rate = 0.06,
  y_type = "effective"
)

# 10-year semiannual bond, yield given directly per coupon period
bond_price(
  face = 1000,
```

```
  coupon_rate = 0.05,
  years_to_maturity = 10,
  coupons_per_year = 2,
  y_effective_per_period = 0.03
)

# Semiannual coupons, nominal annual yield convertible quarterly
bond_price(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 5,
  coupons_per_year = 2,
  y_rate = 0.06,
  y_type = "nominal_interest",
  y_m = 4
)
```

---

bond_ytm *Yield to maturity of a level coupon bond*

Description

Computes the yield to maturity (YTM) of a level coupon bond given its observed dirty price at time 0.

Usage

```
bond_ytm(
  price,
  face,
  coupon_rate,
  years_to_maturity,
  coupons_per_year = 1L,
  redemption = NULL,
  interval = NULL,
  tol = 1e-12,
  maxiter = 1000,
  check = TRUE
)
```

Arguments

price	Numeric scalar. Observed dirty price of the bond at time 0.
face	Numeric scalar. Face (par) value of the bond.
coupon_rate	Numeric scalar. Annual coupon rate as a proportion.
years_to_maturity	Numeric scalar. Time to maturity in years. Must be strictly positive.

`coupons_per_year`
: Positive integer. Number of coupon payments per year.

`redemption`
: Numeric scalar. Redemption value at maturity. If `NULL`, defaults to `face`.

`interval`
: Optional numeric vector of length 2 giving a bracket for the effective yield per coupon period.

`tol`
: Numeric scalar. Tolerance passed to `uniroot`.

`maxiter`
: Positive integer. Maximum number of iterations passed to `uniroot`.

`check`
: Logical scalar. If `TRUE`, performs basic input checks.

Details

The YTM is solved first as the effective yield per coupon period and then reported together with common annual equivalents.

Assumptions:

- Coupons are paid in arrears at regular intervals.
- Price is observed at a coupon date (no accrued interest).
- `years_to_maturity * coupons_per_year` must be an integer.
- Stub periods are not supported.

The effective yield per coupon period j is the solution to

$$P = \sum_{k=1}^{N} C_k \, (1 + j)^{-k}$$

where P is the observed price and C_k are the bond's cash flows (coupons and redemption) at coupon periods $k = 1, \ldots, N$.

The root is found numerically using `uniroot`. If no `interval` is supplied, the function automatically brackets the root starting from $(-0.999999,\ 0.10)$ and progressively widens the upper bound until a sign change is detected.

From the per-period yield, the annual equivalents are:

$$j^{(m)} = m \cdot j, \qquad i = (1 + j)^m - 1.$$

Value

A one-row tibble with columns:

price Input dirty price.

i_period Effective yield per coupon period.

j_nominal Nominal annual yield convertible `coupons_per_year` times per year (= `coupons_per_year * i_period`). When `coupons_per_year = 2`, this is the bond-equivalent yield.

i_effective_annual Annual effective yield.

See Also

bond_price, bond_cash_flows, bond_duration, bond_convexity, bond_callable_price

Other bonds: bond_book_value(), bond_callable_price(), bond_convexity(), bond_duration(), bond_price(), portfolio_convexity(), portfolio_duration()

Examples

```
bond_ytm(
  price = 100,
  face = 100,
  coupon_rate = 0.06,
  years_to_maturity = 5,
  coupons_per_year = 1
)

bond_ytm(
  price = 950,
  face = 1000,
  coupon_rate = 0.05,
  years_to_maturity = 10,
  coupons_per_year = 2
)
```

---

cash_flows_sample *Sample cash flows for interest theory examples*

Description

A small pedagogical dataset containing cash-flow scenarios for present value, future value, net present value, internal rate of return, equations of value, and cash-flow diagrams.

A small pedagogical dataset containing cash-flow scenarios for present value, future value, net present value, internal rate of return, equations of value, and cash-flow diagrams.

Usage

```
cash_flows_sample

cash_flows_sample
```

Format

A tibble with 19 rows and 5 variables:

scenario_id Scenario identifier.

time Payment time.

amount Cash-flow amount. Negative values represent outflows and positive values represent inflows.

cashflow_type Type of cash flow.

description Short description of the cash flow.

A tibble with 19 rows and 5 variables:

scenario_id Scenario identifier.

time Payment time.

amount Cash-flow amount. Negative values represent outflows and positive values represent inflows.

cashflow_type Type of cash flow.

description Short description of the cash flow.

Source

Synthetic pedagogical data created for tidyactuarial examples.

Synthetic pedagogical data created for tidyactuarial examples.

Examples

```
data(cash_flows_sample)

cash_flows_sample |>
  dplyr::filter(scenario_id == "investment_project")

data(cash_flows_sample)

cash_flows_sample |>
  dplyr::filter(scenario_id == "investment_project")
```

---

commutation_table *Build an annual commutation table (discrete ages)*

Description

Constructs classical annual commutation functions Dx, Nx, Sx, Cx, Mx, Rx from a lifetable defined at integer ages and an annual effective interest rate i.

Usage

```
commutation_table(lt, i, check = TRUE)
```

Arguments

lt	A lifetable object as produced by lifetable. Must contain columns x and lx. If available, qx or px may also be used, but dx is computed robustly from successive lx.
i	Numeric. Annual effective interest rate (must satisfy i > -1).
check	Logical. If TRUE, performs basic input checks.

Value

A tibble with columns `x`, `lx`, `dx`, `v`, `Dx`, `Nx`, `Sx`, `Cx`, `Mx`, `Rx`.

`discount_factor_spot` *Spot discount factor*

Description

Computes the discount factor implied by a spot rate for a given time.

Usage

```
discount_factor_spot(
  time,
  rate,
  rate_type = "effective",
  m = 1L,
  output = c("value", "table")
)
```

Arguments

`time`	Numeric vector of times in years. Each value must be greater than or equal to 0.
`rate`	Numeric vector of spot-rate values.
`rate_type`	Character vector indicating the spot-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer vector giving the compounding frequency for nominal spot-rate inputs.
`output`	Character string. Use `"value"` to return a numeric discount factor, or `"table"` to return a tibble with intermediate calculations.

Details

The spot rate may be supplied in FM-style notation:

- annual effective rate,
- nominal annual interest rate,
- nominal annual discount rate,
- force of interest.

Internally, the supplied spot rate is first converted to the equivalent annual effective rate using `standardize_interest`. The discount factor is then computed as

$$v(t) = (1 + i)^{-t}$$

If $t = 0$, the discount factor is 1.

Input vectors must have length 1 or a common length. Missing values are propagated.

Value

If `output = "value"`, a numeric vector of discount factors.

If `output = "table"`, a tibble with input values, standardized rates, and discount factors.

See Also

`standardize_interest`, `present_value`

Other interest: `forward_rate()`, `interest_equivalents()`, `standardize_interest()`, `yield_curve()`

Examples

```
# Numeric discount factor
discount_factor_spot(
  time = 3,
  rate = 0.05
)

# Vectorized example
discount_factor_spot(
  time = c(1, 2, 3),
  rate = c(0.05, 0.055, 0.06)
)

# FM-style input with nominal annual interest
discount_factor_spot(
  time = 2,
  rate = 0.08,
  rate_type = "nominal_interest",
  m = 2
)

# Tibble output for teaching or auditing
discount_factor_spot(
  time = c(1, 2, 3),
  rate = c(0.05, 0.055, 0.06),
  output = "table"
)
```

---

e_x *Expected future lifetime from an annual life table*

Description

Computes the curtate or complete expected future lifetime at integer age x, optionally restricted to a temporary horizon of t years.

Usage

```
e_x(
  lt,
  x,
  t = NULL,
  type = c("curtate", "complete"),
  frac,
  tidy = FALSE,
  check = TRUE,
  tol = 1e-10
)
```

Arguments

`lt`	A life table object as produced by `lifetable` (must contain columns `x` and `lx`).
`x`	Integer age(s).
`t`	Optional nonnegative numeric duration(s). If `NULL` (default), the whole-life expectancy is computed (i.e., horizon extends to $\omega - x$). If a numeric value is provided, the t-year temporary life expectancy is returned.
`type`	Character: `"curtate"` (default) or `"complete"`.
`frac`	Fractional-age assumption for `type = "complete"`: `"UDD"`, `"CF"`, `"CML"` (alias of CF), or `"Balducci"`. If not specified and `lt` carries a `frac` attribute (set by `lifetable`), that value is used.
`tidy`	Logical. If `TRUE`, returns a tibble.
`check`	Logical. If `TRUE`, performs basic input checks.
`tol`	Numeric tolerance for integer checks.

Details

Curtate life expectancy (Finan, Section 23.7):

$$e_x = \sum_{k=1}^{\omega-x} {}_kp_x = \frac{1}{\ell_x} \sum_{k=1}^{\omega-x} \ell_{x+k}.$$

The t-year temporary curtate expectancy is (Finan, Sec. 23.7):

$$e_{x:\overline{t}|} = \sum_{k=1}^{t} {}_kp_x.$$

Complete life expectancy (Finan, Section 23.3):

$$\breve{e}_x = \int_0^{\omega-x} {}_tp_x \, dt = \frac{T_x}{\ell_x}.$$

The integral is decomposed year-by-year. Within each year, the within-year survival integral $\int_0^s {}_up_y \, du$ is evaluated in closed form under the selected fractional-age assumption (Finan, Section 24):

- UDD (Sec. 24.1): $\int_0^s {}_up_y\,du = s - \frac{1}{2}s^2 q_y$
- CF (Sec. 24.2): $\int_0^s {}_up_y\,du = (1 - p_y^s)/(-\ln p_y)$
- Balducci (Sec. 24.3): $\int_0^s {}_up_y\,du = \frac{p_y}{q_y}\ln\left(\frac{p_y + q_y s}{p_y}\right)$

Under UDD, the complete expectancy satisfies the well-known approximation (Finan, Example 20.24):

$$\breve{e}_x \approx e_x + \frac{1}{2}.$$

Value

A numeric vector of expected future lifetimes, or a tibble if `tidy = TRUE` with columns `x`, `t`, `type`, `frac`, `ex`.

`e_xy` *Expected future lifetime for two independent lives*

Description

Computes the expected future lifetime for two independent lives aged `x` and `y`, for either joint-life (first death) or last-survivor (second death).

Usage

```
e_xy(
  lt,
  x,
  y,
  t = NULL,
  type = c("curtate", "complete"),
  frac,
  cohort = c("first", "last"),
  tidy = FALSE,
  check = TRUE,
  tol = 1e-10
)
```

Arguments

`lt`	A life table data frame with columns `x` and `lx`.
`x`	Integer actuarial age for life 1.
`y`	Integer actuarial age for life 2.
`t`	Optional nonnegative numeric duration(s). If `NULL`, uses the maximum horizon allowed by the table.
`type`	Character: `"curtate"` or `"complete"`.

`frac`	Fractional-age assumption for `type = "complete"`, passed to `t_pxy`: `"UDD"`, `"CF"`, `"CML"`, or `"Balducci"`. If not specified and `lt` carries a `frac` attribute, that value is used.
`cohort`	Two-life cohort: `"first"` (joint-life) or `"last"` (last survivor).
`tidy`	Logical. If `TRUE`, returns a tibble.
`check`	Logical. If `TRUE`, performs basic input checks.
`tol`	Numeric tolerance for integer checks.

Details

Curtate expectation (Finan, Section 56.4 / Section 57):

$$e_{xy} = \sum_{k=1}^{\infty} {}_kp_{xy}, \quad e_{\overline{xy}} = \sum_{k=1}^{\infty} {}_kp_{\overline{xy}}.$$

Complete expectation (Finan, Section 56.4):

$$\mathring{e}_{xy} = \int_0^{\infty} {}_tp_{xy}\, dt.$$

The integral is decomposed year-by-year. Within each year, the survival integral for the two-life status is computed numerically via composite trapezoid (80-point grid), since closed-form expressions for joint/last survivor under fractional-age assumptions are complex.

Key identity (Finan, Example 57.4):

$$\mathring{e}_{\overline{xy}} = \mathring{e}_x + \mathring{e}_y - \mathring{e}_{xy}.$$

This can be used to cross-validate results.

Value

Numeric vector, or tibble if `tidy = TRUE`.

See Also

`e_x` for single-life expectancy, `t_pxy` for two-life survival probabilities, `annuity_xy` for two-life annuity APVs.

Examples

```
lt <- data.frame(
  x  = 60:66,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000, 86000)
)

# Curtate joint-life expectancy (Finan, Sec. 56.4)
e_xy(lt, x = 60, y = 62, type = "curtate", cohort = "first")

# Curtate last-survivor expectancy (Finan, Sec. 57)
```

```
e_xy(lt, x = 60, y = 62, type = "curtate", cohort = "last")

# Verify identity (Finan, Example 57.4):
# e_{xy-bar} = e_x + e_y - e_xy
e_joint <- e_xy(lt, x = 60, y = 62, type = "curtate", cohort = "first")
e_last  <- e_xy(lt, x = 60, y = 62, type = "curtate", cohort = "last")
e_x_val <- e_x(lt, x = 60, type = "curtate")
e_y_val <- e_x(lt, x = 62, type = "curtate")
c(last_surv = e_last, sum_minus_joint = e_x_val + e_y_val - e_joint)

# Complete joint-life expectancy under UDD
e_xy(lt, x = 60, y = 62, type = "complete", frac = "UDD", cohort = "first")

# Temporary: 3-year curtate joint-life
e_xy(lt, x = 60, y = 62, t = 3, type = "curtate", cohort = "first")

# Tidy output
e_xy(lt, x = 60, y = 62, type = "curtate", cohort = "first", tidy = TRUE)
```

`forward_rate` *Compute an implied forward rate from a discrete spot curve*

Description

Returns the annual effective forward rate implied between two maturities from a discrete yield curve stored in tibble-first format.

Usage

```
forward_rate(
  .data = NULL,
  term = NULL,
  spot = NULL,
  t_start = NULL,
  t_end = NULL,
  col_term = "term",
  col_spot = "spot",
  col_t_start = "t_start",
  col_t_end = "t_end",
  method = c("exact", "linear"),
  plot = FALSE,
  .out = "forward_rate",
  .out_plot = "forward_rate_plot",
  .keep = c("all", "used", "none"),
  .na = c("propagate", "error", "drop")
)
```

Arguments

.data	A data.frame or tibble. If NULL, term, spot, t_start, and t_end must be supplied.
term	Numeric vector of maturities when .data = NULL.
spot	Numeric vector of annual effective spot rates when .data = NULL.
t_start	Numeric scalar giving the start maturity when .data = NULL.
t_end	Numeric scalar giving the end maturity when .data = NULL.
col_term	Name of the list-column containing maturities.
col_spot	Name of the list-column containing spot rates.
col_t_start	Name of the numeric column containing the start maturity.
col_t_end	Name of the numeric column containing the end maturity.
method	Spot extraction method: "exact" or "linear".
plot	Logical; if TRUE, adds a list-column of ggplot2 objects.
.out	Name of the output column containing the forward rate.
.out_plot	Name of the output list-column containing ggplot2 objects. Used only if plot = TRUE.
.keep	One of "all", "used", or "none".
.na	NA handling policy: "propagate", "error", or "drop".

Details

Each row is treated as one curve (one case). For tibble input, col_term and col_spot must be list-columns of equal-length numeric vectors, and col_t_start and col_t_end must be numeric columns giving the forward interval for each row.

The implied forward rate is computed from the spot curve through:

$$(1+i_1)^{t_1}(1+f)^{t_2-t_1} = (1+i_2)^{t_2}$$

so that

$$f_{t_1,t_2} = \left(\frac{(1+i_2)^{t_2}}{(1+i_1)^{t_1}}\right)^{1/(t_2-t_1)} - 1$$

Two extraction methods are supported for the spot rates:

- "exact": requires that t_start and t_end match curve nodes.
- "linear": uses linear interpolation between adjacent nodes.

No extrapolation is performed outside the observed maturity range.

Value

A tibble. By default it returns the original columns plus a new numeric column named by .out. If plot = TRUE, it also adds a list-column named by .out_plot containing ggplot2 objects.

References

Marcel B. Finan, *A Basic Course in the Theory of Interest and Derivatives Markets: A Preparation for the Actuarial Exam FM/2*, Section 53: The Term Structure of Interest Rates and Yield Curves.

Kellison, S. G. *The Theory of Interest*, Chapter 10: The Term Structure of Interest Rates.

See Also

`yield_curve`, `discount_factor_spot`, `standardize_interest`

Other interest: `discount_factor_spot()`, `interest_equivalents()`, `standardize_interest()`, `yield_curve()`

Examples

```
# Simple example: exact forward rate
forward_rate(
  term = c(1, 2, 3, 4, 5),
  spot = c(0.040, 0.045, 0.048, 0.050, 0.051),
  t_start = 2,
  t_end = 5
)

# Medium example: interpolated forward rates for multiple curves
curves <- tibble::tibble(
  curve_id = c("A", "B"),
  term = list(c(1, 2, 3, 5), c(1, 3, 5, 7)),
  spot = list(c(0.04, 0.05, 0.055, 0.06),
              c(0.03, 0.035, 0.04, 0.045)),
  t_start = c(2, 2),
  t_end = c(4, 6)
)

forward_rate(
  curves,
  method = "linear",
  plot = TRUE
)
```

---

future_value *Future value of a single payment*

Description

Computes the future value of a payment invested at time 0 and accumulated to a given time, using the annual effective interest rate implied by the supplied rate specification.

Usage

```
future_value(
  amount,
  rate,
  rate_type = "effective",
  m = 1,
  time,
  output = c("value", "table")
)
```

Arguments

`amount`	Numeric vector of initial payment amounts.
`rate`	Numeric vector of rate values.
`rate_type`	Character vector indicating the rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer vector giving the compounding frequency for nominal rates. Ignored for `"effective"` and `"force"`.
`time`	Numeric vector of times in years from valuation to accumulation.
`output`	Character string. Use `"value"` to return a numeric future value, or `"table"` to return a tibble with intermediate calculations.

Details

The future value is computed as

$$FV = C(1+i)^t$$

where i is the annual effective interest rate.

The input interest rate may be supplied as:

- annual effective interest rate,
- nominal annual interest rate,
- nominal annual discount rate,
- force of interest.

Internally, all rate specifications are first converted to the equivalent annual effective interest rate using `standardize_interest`.

Input vectors must have length 1 or a common length. Missing values are propagated. This function does not accept dates.

Value

If `output = "value"`, a numeric vector of future values.

If `output = "table"`, a tibble with input values, equivalent rates, accumulation factors, and future values.

See Also

standardize_interest, present_value

Other time-value: fv_flow(), irr_flow(), irr_flow_multi(), plot_cash_flow(), present_value(), pv_flow()

Examples

```
# Numeric future value
future_value(amount = 1000, rate = 0.08, time = 3)

# Nominal interest converted monthly
future_value(
  amount = 1000,
  rate = 0.12,
  rate_type = "nominal_interest",
  m = 12,
  time = 5
)

# Tibble output for teaching or auditing
future_value(
  amount = 1000,
  rate = 0.08,
  time = 3,
  output = "table"
)

# Vectorized example
future_value(
  amount = c(1000, 2500, 4000),
  rate = c(0.08, 0.10, 0.12),
  rate_type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  time = c(3, 5, 2)
)
```

---

fv_flow *Future value of a general cash flow*

Description

Computes the future value of a cash-flow vector under either:

- a constant interest-rate specification, or
- a term structure of spot rates, one rate per cash flow.

Usage

```
fv_flow(
  payment,
  rate,
  type = "effective",
  m = 1L,
  time = NULL,
  date = NULL,
  day_count = c("act/365", "act/360"),
  tol = 1e-10
)
```

Arguments

payment	Numeric vector of cash flows.
rate	Numeric scalar or numeric vector of rate values.
type	Character vector indicating the rate type: "effective", "nominal_interest", "nominal_discount", or "force". May have length 1 or the same length as payment.
m	Positive integer vector giving the compounding frequency for nominal rates. May have length 1 or the same length as payment.
time	Optional numeric vector of payment times in years.
date	Optional vector of payment dates. If supplied, the earliest date is treated as time 0.
day_count	Day-count convention used to convert dates to year fractions. One of "act/365" or "act/360".
tol	Numeric tolerance used in internal checks.

Details

The cash flow is supplied explicitly through payment. Its timing is supplied either through time (in years) or date (calendar dates). If date is supplied, the earliest date is taken as time 0.

The future value is accumulated to the latest payment time (or latest date).

Interest-rate input:

- If rate has length 1, the same rate is used for all payments.
- If rate has the same length as payment, each rate is interpreted as the annual effective spot rate associated with the corresponding payment time.

Rate types may be supplied in FM-style notation:

- annual effective rate i,
- nominal annual interest rate $j^{(m)}$,
- nominal annual discount rate $d^{(m)}$,
- force of interest δ.

Internally, all supplied rates are converted to annual effective rates using standardize_interest. When rate is a vector, the accumulation uses the implied discount-factor ratio under the spot-rate interpretation.

Let T be the latest payment time (the accumulation horizon). For each payment C_k at time t_k with annual effective spot rate i_k, the future value contribution is

$$FV_k = C_k \cdot \frac{(1+i_T)^T}{(1+i_k)^{t_k}}$$

and the total future value is $FV = \sum_k FV_k$.

When a single constant rate is supplied, $i_k = i_T = i$ for all k and the formula simplifies to $FV = \sum_k C_k(1+i)^{T-t_k}$.

Value

Numeric scalar: the future value of the cash flow accumulated to the latest payment time.

See Also

pv_flow, future_value, standardize_interest

Other time-value: future_value(), irr_flow(), irr_flow_multi(), plot_cash_flow(), present_value(), pv_flow()

Examples

```
# Constant annual effective rate
fv_flow(
  payment = c(100, 150, 200),
  rate = 0.08,
  type = "effective",
  time = c(0, 1, 2)
)

# Spot rates, one per payment
fv_flow(
  payment = c(100, 150, 200),
  rate = c(0.05, 0.055, 0.06),
  type = "effective",
  time = c(1, 2, 3)
)

# Using dates; earliest date is taken as t = 0
fv_flow(
  payment = c(100, 150, 200),
  rate = c(0.05, 0.055, 0.06),
  type = "effective",
  date = as.Date(c("2026-01-10", "2027-01-10", "2028-01-10"))
)

# Nominal rates by payment
fv_flow(
```

```
  payment = c(100, 100, 100),
  rate = c(0.12, 0.12, 0.12),
  type = "nominal_interest",
  m = c(12, 12, 12),
  time = c(1, 2, 3)
)
```

`immunize_duration` *Duration-based immunization with multiple assets*

Description

Computes asset weights that duration-immunize a stream of liabilities using two or more assets. The method enforces:

1. Present value of assets = PV of liabilities;
2. Macaulay duration of assets = Macaulay duration of liabilities.

Usage

```
immunize_duration(liabilities, t_liabilities, pv_assets, duration_assets, i)
```

Arguments

`liabilities`	Numeric vector with liability payments.
`t_liabilities`	Numeric vector of the same length as `liabilities` with the times (in years or periods) at which each liability payment occurs.
`pv_assets`	Numeric vector with present values (prices) of each asset evaluated at the same yield rate `i`.
`duration_assets`	Numeric vector with the Macaulay duration of each asset, expressed in the same time units as `t_liabilities`.
`i`	Yield rate used to discount the liabilities (effective per period).

Details

For exactly two assets, a closed-form solution is used. For three or more assets, a minimum-norm solution is computed via linear algebra.

Let PV_L and D_L be the present value and Macaulay duration of the liability stream at yield i. Let PV_j and D_j be the present value and duration of asset j. The weights w_j are chosen so that:

$$\sum_j w_j PV_j = PV_L$$

and

$$\frac{\sum_j w_j PV_j D_j}{\sum_j w_j PV_j} = D_L$$

For two assets, the closed-form solution is:

$$w_1 = \frac{PV_L(D_L - D_2)}{PV_1(D_1 - D_2)}, \quad w_2 = \frac{PV_L - w_1 PV_1}{PV_2}$$

For $k \geq 3$ assets, the minimum-norm solution of the linear system $Aw = b$ is computed, where A is a $2 \times k$ matrix with rows $(PV_1, \ldots, PV_k)$ and $(PV_1 D_1, \ldots, PV_k D_k)$, and $b = (PV_L,\ PV_L D_L)^T$.

Value

A tibble with:

weight_asset Numeric vector of asset weights (amounts of each asset).

PV_liabilities Present value of the liabilities.

Duration_liabilities Macaulay duration of the liabilities.

PV_assets Present value of the immunized asset portfolio.

Duration_assets Macaulay duration of the asset portfolio.

n_assets Number of assets used.

See Also

immunize_duration_convexity, bond_duration, bond_convexity

Other immunization: immunize_duration_convexity(), plot_immunization_gap()

Examples

```
# Two-asset immunization
immunize_duration(
  liabilities = c(5000, 8000),
  t_liabilities = c(3, 7),
  pv_assets = c(100, 200),
  duration_assets = c(3, 7),
  i = 0.05
)

# Three-asset immunization (minimum-norm)
immunize_duration(
  liabilities = c(5000, 8000),
  t_liabilities = c(3, 7),
  pv_assets = c(100, 150, 200),
  duration_assets = c(2, 5, 8),
  i = 0.05
)
```

`immunize_duration_convexity` *Duration and convexity immunization with multiple assets*

Description

Computes asset weights that immunize a stream of liabilities using three or more assets, enforcing:

1. Present value of assets = PV of liabilities;
2. Macaulay duration of assets = Macaulay duration of liabilities;
3. Convexity of assets = convexity of liabilities.

Usage

```
immunize_duration_convexity(
  liabilities,
  t_liabilities,
  pv_assets,
  duration_assets,
  convexity_assets,
  i
)
```

Arguments

`liabilities`	Numeric vector with liability payments.
`t_liabilities`	Numeric vector with the times (in periods) at which each liability payment occurs. Must have the same length as `liabilities`.
`pv_assets`	Numeric vector with present values (prices) of each asset evaluated at the same yield rate `i`.
`duration_assets`	Numeric vector with the Macaulay duration of each asset, in the same time units as `t_liabilities`.
`convexity_assets`	Numeric vector with the discrete convexity of each asset, in the same time units (periods) as `t_liabilities`.
`i`	Yield rate used to discount the liabilities (effective per period).

Details

For exactly three assets, the system is solved directly. For four or more assets, a minimum-norm solution is computed via linear algebra.

Let PV_L, D_L, and C_L be the present value, Macaulay duration, and discrete convexity of the liability stream at yield i. The discrete convexity of the liabilities is computed as:

$$C_L = \frac{\sum_t L_t \, t(t+1) \, v^{t+2}}{PV_L}$$

where $v = 1/(1+i)$.

The weights w_j satisfy the $3 \times k$ system $Aw = b$, where the rows of A are (PV_j), $(PV_j D_j)$, $(PV_j C_j)$, and $b = (PV_L,\ PV_L D_L,\ PV_L C_L)^T$.

For $k = 3$ assets, the system is square and solved directly. For $k \geq 4$ assets, the minimum-norm solution $w = A^T(AA^T)^{-1}b$ is computed.

Value

A tibble with:

weight_asset Numeric vector of asset weights (amounts of each asset).

PV_liabilities Present value of the liabilities.

Duration_liabilities Macaulay duration of the liabilities.

Convexity_liabilities Discrete convexity of the liabilities.

PV_assets Present value of the asset portfolio.

Duration_assets Macaulay duration of the asset portfolio.

Convexity_assets Discrete convexity of the asset portfolio.

n_assets Number of assets used.

See Also

immunize_duration, bond_duration, bond_convexity

Other immunization: immunize_duration(), plot_immunization_gap()

Examples

```
# Three-asset immunization (exact solution)
immunize_duration_convexity(
  liabilities = c(5000, 8000, 10000),
  t_liabilities = c(3, 5, 7),
  pv_assets = c(100, 150, 200),
  duration_assets = c(2, 5, 8),
  convexity_assets = c(6, 30, 72),
  i = 0.05
)

# Four-asset immunization (minimum-norm)
immunize_duration_convexity(
  liabilities = c(5000, 8000, 10000),
  t_liabilities = c(3, 5, 7),
  pv_assets = c(100, 120, 150, 200),
  duration_assets = c(2, 4, 6, 8),
  convexity_assets = c(6, 20, 42, 72),
  i = 0.05
)
```

`insurance_variable_k` *Actuarial present value of a life insurance with variable k-thly benefits*

Description

Computes the actuarial present value of a life insurance where the death benefit may vary by subperiod (k payments per year) and is payable at the end of the subperiod of death.

Usage

```
insurance_variable_k(
  lt,
  x,
  i,
  benefit,
  n = NULL,
  m = 0,
  k = 12,
  frac,
  tidy = FALSE,
  check = TRUE
)
```

Arguments

`lt`	A life table object or data frame containing at least `x` and `lx`.
`x`	Integer actuarial age at issue.
`i`	Effective annual interest rate (must be `> -1`).
`benefit`	Numeric vector of benefits by subperiod, or a function of time returning the benefit at time t.
`n`	Optional term in years. If `NULL`, the term is inferred from the length of `benefit` (when numeric).
`m`	Nonnegative integer deferral period in years (default `0`).
`k`	Number of subperiods per year (default `12`).
`frac`	Fractional-age assumption used in survival probabilities: `"UDD"`, `"CF"`, `"CML"`, or `"Balducci"`. If not specified and `lt` carries a `frac` attribute, that value is used.
`tidy`	Logical. If `TRUE`, returns a one-row tibble.
`check`	Logical. If `TRUE`, performs basic input checks.

Details

Let k be the number of subperiods per year and $N = nk$ the total number of subperiods. With deferral m, the actuarial present value at age x is (generalizing Finan, Section 29):

$$\text{APV} = \sum_{j=1}^{N} v^{m+t_j} \cdot b_j \cdot \left({}_{m+t_{j-1}}p_x - {}_{m+t_j}p_x\right)$$

where $t_j = j/k$ and b_j is the benefit payable if death occurs in the interval $(m + t_{j-1},\, m + t_j]$.

This is the general form that encompasses:

- **Level insurance**: constant $b_j = 1$ reduces to $A^{(k)1}_{x:\overline{n}|}$ (Finan, Section 31).
- **Increasing insurance**: $b_j = \lceil j/k \rceil$ gives $(IA)^{(k)}_{x:\overline{n}|}$ (Finan, Section 29.3).
- **Decreasing insurance**: $b_j = n - \lfloor (j-1)/k \rfloor$ gives $(DA)^{(k)}_{x:\overline{n}|}$ (Finan, Section 29.3).
- **Credit insurance**: $b_j = B(t_j)$ where $B(t)$ is the outstanding loan balance.

The benefit may be supplied either as a numeric vector indexed by subperiod or as a function of time.

Fractional survival probabilities are computed via `t_px` under the selected assumption (Finan, Section 24).

Value

A numeric actuarial present value, or a one-row tibble if `tidy = TRUE`.

See Also

`insurance_x` for level-benefit life insurance, `annuity_x` for life annuity APVs, `t_px` for survival probabilities, Monte Carlo simulation tools may be used for empirical variance estimation.

Examples

```
lt <- data.frame(
  x  = 60:66,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000, 86000)
)

# Monthly insurance with increasing benefits (Finan, Sec. 29.3 style)
insurance_variable_k(
  lt, x = 60, i = 0.05,
  benefit = seq(100, 1200, length.out = 12),
  n = 1, k = 12
)

# Credit-style insurance with declining outstanding balance
balance <- function(t) 2000 * exp(-0.3 * t)
insurance_variable_k(
  lt, x = 60, i = 0.05,
  benefit = balance,
```

```
  n = 1, k = 12
)

# Level benefit = 1 should match a term insurance
# (approximately, since insurance_x uses annual and this uses k-thly)
insurance_variable_k(lt, x = 60, i = 0.05, benefit = 1, n = 5, k = 1)
insurance_x(mortality_table = lt, age = 60, rate = 0.05, term_years = 5, insurance_type = "term")

# 2-year deferred, 3-year term with monthly varying benefits
insurance_variable_k(
  lt, x = 60, i = 0.05,
  benefit = rep(1000, 36),
  n = 3, m = 2, k = 12
)

# Tidy output
insurance_variable_k(
  lt, x = 60, i = 0.05,
  benefit = rep(1000, 12), n = 1, k = 12, tidy = TRUE
)
```

`insurance_x` *Actuarial present value of a life insurance*

Description

Computes the actuarial present value of a discrete life insurance using a life table.

Usage

```
insurance_x(
  mortality_table,
  age,
  rate,
  rate_type = "effective",
  m = 1L,
  term_years = Inf,
  deferral_years = 0L,
  insurance_type = c("whole", "term", "endowment"),
  benefit = 1,
  output = c("value", "table")
)
```

Arguments

`mortality_table`	A life table as produced by `lifetable`. It must contain columns `x` and `lx`.
`age`	Integer actuarial age.

rate	Numeric scalar. Annual interest-rate input.
rate_type	Character string indicating the rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer. Compounding frequency for nominal rates. Ignored for rate_type = "effective" and rate_type = "force".
term_years	Integer term in years. Required for insurance_type = "term" and insurance_type = "endowment". Use Inf only for whole-life insurance.
deferral_years	Integer deferral period in years.
insurance_type	Character string. One of "whole", "term", or "endowment".
benefit	Numeric scalar. Benefit amount.
output	Character string. Use "value" to return a numeric APV or "table" to return a one-row tibble with intermediate quantities.

Details

The benefit is paid at the end of the year of death for whole-life and term insurance. For endowment insurance, the same benefit is paid either at death within the term or at the end of the term if the life survives.

Supported contracts:

- "whole": whole-life insurance.
- "term": n-year term insurance.
- "endowment": n-year endowment insurance.

This function uses standard annual discrete identities.

For whole-life insurance,

$$A_x = 1 - d\,\ddot{a}_x,$$

where $d = i/(1+i)$.

For n-year term insurance,

$$A^1_{x:\overline{n}|} = 1 - d\,\ddot{a}_{x:\overline{n}|} - v^n\,{}_np_x.$$

For n-year endowment insurance,

$$A_{x:\overline{n}|} = 1 - d\,\ddot{a}_{x:\overline{n}|}.$$

Deferral is handled by multiplying the value at age age + deferral_years by ${}_v{}^h{}_hp_x$, where $h =$ deferral_years.

Value

If output = "value", a numeric scalar containing the actuarial present value.

If output = "table", a one-row tibble with the main input values, equivalent interest rate, deferral factor, pure endowment factor, annuity-due value used in the identity, and APV.

See Also

annuity_x, premium_x, reserve_x, t_Ex, insurance_xy

Other life-contingencies: annuity_x(), annuity_xy(), insurance_xy(), life_contract(), premium_gross(), premium_x(), premium_xy(), reserve_x(), reserve_xy(), simulate_annuity_x(), simulate_insurance_x()

Examples

```
lt <- data.frame(
  x  = 60:65,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000)
)

# Whole-life insurance
insurance_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
  insurance_type = "whole"
)

# 5-year term insurance
insurance_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
  term_years = 5,
  insurance_type = "term"
)

# 5-year endowment insurance
insurance_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
  term_years = 5,
  insurance_type = "endowment"
)

# Deferred whole-life insurance
insurance_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
  deferral_years = 2,
  insurance_type = "whole"
)

# Table output
insurance_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
```

```
  term_years = 5,
  insurance_type = "term",
  output = "table"
)
```

`insurance_xj` *Cause-specific term/whole-life insurance APV under multiple decrements: insurance_xj*

Description

Computes the actuarial present value (APV) of an annual (discrete) insurance that pays `benefit` at the end of the year of decrement by a specified cause `j`, using a multiple decrement table produced by `md_table`.

Usage

```
insurance_xj(
  md,
  x,
  i,
  cause,
  product = c("whole", "term"),
  benefit = 1,
  n = NULL,
  m = 0L,
  tidy = FALSE,
  check = TRUE,
  tol = 1e-10
)

A_xj(
  md,
  x,
  i,
  cause,
  product = c("whole", "term"),
  benefit = 1,
  n = NULL,
  m = 0L,
  tidy = FALSE,
  check = TRUE,
  tol = 1e-10
)
```

Arguments

`md`	A multiple decrement table produced by `md_table`. Must contain columns `x`, `p_total`, and the requested `cause`.
`x`	Integer age(s) at issue.
`i`	Annual effective interest rate(s). Must satisfy `i > -1`.
`cause`	Character. Name of the cause column in `md` (e.g., `"q_death"`).
`product`	Character. Insurance type: `"whole"` or `"term"`.
`benefit`	Numeric benefit amount payable at the end of the year of decrement by the specified cause (default 1).
`n`	Integer term length in years (required when `product = "term"`).
`m`	Integer deferment in years (default `0`).
`tidy`	Logical. If `TRUE`, returns a tibble with inputs and `insurance_xj`.
`check`	Logical. If `TRUE`, performs input validation.
`tol`	Numeric tolerance for integer checks.

Details

Let $q_{x+k}^{(j)}$ be the one-year decrement probability for cause j at age $x+k$, and $p_{x+r}^{(\tau)}$ be the one-year total survival probability (any cause) at age $x+r$. For a product with deferment m and term n, with benefit payable at the end of the year of decrement by cause j, the APV is:

$$\sum_{k=m}^{m+n-1} v^{k+1} \left(\prod_{r=0}^{k-1} p_{x+r}^{(\tau)} \right) q_{x+k}^{(j)},$$

where $v = (1+i)^{-1}$ and i is the annual effective interest rate.

If `product = "whole"`, the function sets `n` to the remaining length of the table after deferment (i.e., whole life over the available ages).

Value

Numeric vector of APVs (or a tibble if `tidy = TRUE`).

See Also

`t_qxj` for cause-specific decrement probabilities, `lt_tau` to build a single-decrement lifetable for the total decrement.

Examples

```
qx_df <- tibble::tibble(
  x = 30:35,
  q_death = c(0.001, 0.0012, 0.0014, 0.0017, 0.0020, 1.0000),
  q_disability = c(0.002, 0.0021, 0.0022, 0.0023, 0.0024, 0.0000)
)
md <- md_table(qx_df, radix = 1e5, close = TRUE)
```

```
# 5-year term cause-specific insurance for death, i = 5%
insurance_xj(md, x = 30, i = 0.05, cause = "q_death", product = "term", n = 5)

# Whole-life (over available ages), 2-year deferred
insurance_xj(md, x = 30, i = 0.05, cause = "q_death", product = "whole", m = 2, tidy = TRUE)
```

`insurance_xy` *Actuarial present value of a two-life insurance*

Description

Computes the actuarial present value of a discrete two-life insurance with benefit payable at the end of the year of the triggering death, assuming independent future lifetimes.

Usage

```
insurance_xy(
  mortality_table,
  age_x = NULL,
  age_y = NULL,
  rate = NULL,
  rate_type = NULL,
  m = NULL,
  insurance_type = c("whole", "term", "endowment"),
  cohort = c("first", "last"),
  term_years = Inf,
  deferment_years = 0L,
  benefit = 1,
  output = c("value", "table")
)
```

Arguments

`mortality_table`	Either a single life table used for both lives, or a list of two life tables `list(table_x, table_y)`. Each table must contain columns `x` and `lx`. A `tidyact_life_contract` object created by `life_contract()` is also accepted.
`age_x`	Integer actuarial age for the first life.
`age_y`	Integer actuarial age for the second life.
`rate`	Numeric scalar. Annual interest-rate input.
`rate_type`	Character string indicating the rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Compounding frequency for nominal rates.
`insurance_type`	Character string. One of `"whole"`, `"term"`, or `"endowment"`.

`cohort`	Character string. Use `"first"` for joint-life insurance or `"last"` for last-survivor insurance.
`term_years`	Term in years. Required for term and endowment insurance. Use `Inf` or omit it for whole life insurance.
`deferment_years`	Integer deferment period in years.
`benefit`	Numeric scalar. Insurance benefit.
`output`	Character string. Use `"value"` for a numeric APV or `"table"` for a one-row tibble.

Details

Supported contracts:

- whole life insurance,
- term insurance,
- endowment insurance.

The cohort determines the two-life status:

- `cohort = "first"`: joint-life status; insurance is triggered by the first death.
- `cohort = "last"`: last-survivor status; insurance is triggered by the second death.

This function uses the standard identities that express insurance values through two-life annuity-due values.

For a whole life contract:

$$A = 1 - d\ddot{a}$$

For an n-year term insurance:

$$A^{1}_{:\overline{n}|} = 1 - d\ddot{a}_{:\overline{n}|} - v^{n}\ {}_{n}p$$

For an endowment insurance:

$$A_{:\overline{n}|} = 1 - d\ddot{a}_{:\overline{n}|}$$

The function assumes independent future lifetimes.

Value

If `output = "value"`, a numeric scalar. If `output = "table"`, a one-row tibble.

See Also

`annuity_xy()`, `premium_xy()`, `insurance_x()`, `t_pxy()`

Other life-contingencies: `annuity_x()`, `annuity_xy()`, `insurance_x()`, `life_contract()`, `premium_gross()`, `premium_x()`, `premium_xy()`, `reserve_x()`, `reserve_xy()`, `simulate_annuity_x()`, `simulate_insurance_x()`

Examples

```
lt <- data.frame(
  x = 60:110,
  lx = seq(100000, 0, length.out = 51)
)

insurance_xy(
  mortality_table = lt,
  age_x = 60,
  age_y = 62,
  rate = 0.06,
  insurance_type = "whole",
  cohort = "first"
)

lt |>
  life_contract(lives = "joint", age_x = 60, age_y = 62, rate = 0.06) |>
  insurance_xy(
    insurance_type = "term",
    term_years = 4,
    cohort = "first"
  )
```

---

interest_equivalents *Equivalent interest rates in FM actuarial notation*

Description

Converts a single interest-rate specification into equivalent actuarial rates for the same compounding frequency `m`.

Usage

```
interest_equivalents(
  type = c("effective", "nominal_interest", "nominal_discount", "force"),
  rate,
  m = 1L
)
```

Arguments

`type`	Character string indicating the input rate type. Must be one of `"effective"`, `"nominal_interest"`, `"nominal_discount"`, or `"force"`.
`rate`	Numeric scalar giving the rate value.
`m`	Positive integer scalar giving the compounding frequency for nominal rates.

Details

Internally, the supplied rate is first converted to the annual effective interest rate i using `standardize_interest`.

Given the annual effective interest rate i, the equivalents are:

Effective discount rate: $d = i/(1+i)$

Discount factor: $v = 1/(1+i)$

Force of interest: $\delta = \ln(1+i)$

Nominal interest: $j^{(m)} = m[(1+i)^{1/m} - 1]$

Nominal discount: $d^{(m)} = m[1-(1+i)^{-1/m}]$

Value

A tibble with columns:

family Rate family: `"effective"`, `"discount"`, `"force"`, `"nominal_interest"`, or `"nominal_discount"`.

notation Actuarial notation for the equivalent rate.

m Compounding frequency used for nominal rates.

description Human-readable description.

value Equivalent rate value.

See Also

`standardize_interest`, `discount_factor_spot`

Other interest: `discount_factor_spot()`, `forward_rate()`, `standardize_interest()`, `yield_curve()`

Examples

```
interest_equivalents("nominal_interest", rate = 0.18, m = 4)
interest_equivalents("nominal_discount", rate = 0.10, m = 12)
interest_equivalents("force", rate = 0.12)
interest_equivalents("effective", rate = 0.08)

# Batch use with purrr
if (requireNamespace("purrr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  cases <- tibble::tibble(
    type = c("effective", "force"),
    rate = c(0.08, 0.12),
    m = c(1, 1)
  )

  purrr::pmap(cases, interest_equivalents)
}
```

`irr_flow` *Internal rate of return for a cash flow*

Description

Computes the internal rate of return (IRR) of a cash flow by finding the annual effective rate that makes its present value equal to zero.

Usage

```
irr_flow(
  payment,
  time = NULL,
  date = NULL,
  nominal_m = 1L,
  interval = c(-0.99, 10),
  tol = 1e-10,
  maxiter = 1000,
  day_count = c("act/365", "act/360")
)
```

Arguments

`payment`	Numeric vector of cash flows.
`time`	Optional numeric vector of cash-flow times in years.
`date`	Optional vector of cash-flow dates. If supplied, the earliest date is treated as time 0.
`nominal_m`	Positive integer used only to report an equivalent nominal annual interest rate convertible `nominal_m` times per year.
`interval`	Numeric vector of length 2 giving the search interval for the annual effective IRR. Default is `c(-0.99, 10)`.
`tol`	Numeric tolerance passed to `uniroot`.
`maxiter`	Maximum number of iterations passed to `uniroot`.
`day_count`	Day-count convention used when `date` is supplied. One of `"act/365"` or `"act/360"`.

Details

The cash flow is supplied explicitly through `payment`. Its timing is given either through `time` (in years) or `date` (calendar dates). If `date` is supplied, the earliest date is treated as time 0.

The IRR returned is therefore interpreted as an annual effective rate.

The IRR is defined as the rate r satisfying

$$\sum_k C_k (1+r)^{-t_k} = 0$$

where C_k are the cash flows and t_k the corresponding times in years.

The root is found using `uniroot` over the specified `interval`. If the NPV does not change sign over the interval, no root can be bracketed and the function returns gracefully with `converged = FALSE`.

The number of sign changes in the nonzero cash-flow sequence is reported as a diagnostic. By Descartes' rule of signs, if there is exactly one sign change, the IRR is unique.

Value

A one-row tibble with:

irr Estimated IRR as an annual effective rate.

i_effective_annual Same as `irr`, reported explicitly.

j_nominal_interest Equivalent nominal annual interest rate convertible `nominal_m` times.

delta Equivalent force of interest.

npv Present value at the estimated IRR, close to zero.

interval_left Left endpoint of the search interval.

interval_right Right endpoint of the search interval.

converged Logical flag indicating whether a root was found.

n_iter Number of iterations used by `uniroot`.

n_cashflows Length of `payment`.

has_both_signs Whether the cash flow has at least one positive and one negative value.

n_sign_changes Number of sign changes in the nonzero cash-flow sequence.

If no sign change is present in the cash flow, or if the NPV does not change sign over `interval`, the function returns `converged = FALSE` and `irr = NA_real_`.

See Also

`pv_flow`, `irr_flow_multi`, `bond_ytm`

Other time-value: `future_value()`, `fv_flow()`, `irr_flow_multi()`, `plot_cash_flow()`, `present_value()`, `pv_flow()`

Examples

```
irr_flow(
  payment = c(-1000, 300, 400, 500),
  time = c(0, 1, 2, 3)
)

irr_flow(
  payment = c(-1000, 300, 400, 500),
  date = as.Date(c("2026-01-01", "2027-01-01", "2028-01-01", "2029-01-01"))
)
```

`irr_flow_multi` *Multiple internal rates of return for a cash flow*

Description

Searches for multiple internal rates of return (IRRs) of a cash flow by scanning a search interval and solving for all detectable roots of the net present value (NPV) function.

Usage

```
irr_flow_multi(
  payment,
  time = NULL,
  date = NULL,
  search_interval = c(-0.99, 10),
  grid_points = 2000L,
  nominal_m = 1L,
  tol = 1e-10,
  maxiter = 1000L,
  day_count = c("act/365", "act/360")
)
```

Arguments

`payment`	Numeric vector of cash flows.
`time`	Optional numeric vector of cash-flow times in years.
`date`	Optional vector of cash-flow dates. If supplied, the earliest date is treated as time 0.
`search_interval`	Numeric vector of length 2 giving the search interval for annual effective IRRs. Default is `c(-0.99, 10)`.
`grid_points`	Positive integer giving the number of grid points used to scan the interval. Larger values improve detection at the cost of speed.
`nominal_m`	Positive integer used only to report an equivalent nominal annual interest rate convertible `nominal_m` times per year.
`tol`	Numeric tolerance passed to `uniroot`.
`maxiter`	Positive integer passed to `uniroot`.
`day_count`	Day-count convention used when `date` is supplied. One of `"act/365"` or `"act/360"`.

Details

This function is intended for cash flows with multiple sign changes, where more than one IRR may exist. It evaluates the NPV on a fine grid over the search interval, identifies subintervals with sign changes (and grid points where the NPV is approximately zero), and applies `uniroot` to each candidate interval.

The IRRs returned are interpreted as annual effective rates.

Timing can be supplied either through `time` (in years) or `date` (calendar dates). If `date` is supplied, the earliest date is treated as time 0.

This function detects roots numerically over a finite search interval. It may miss roots if:

- the grid is too coarse,
- two roots are extremely close,
- the NPV touches zero without changing sign,
- or the root lies outside the search interval.

For a single-IRR workflow, use `irr_flow`.

Value

A tibble with one row per detected IRR and columns:

root_id Root index.

irr Detected IRR as an annual effective rate.

i_effective_annual Same as `irr`, reported explicitly.

j_nominal_interest Equivalent nominal annual interest rate convertible `nominal_m` times.

delta Equivalent force of interest.

npv NPV evaluated at the detected root (approximately zero).

interval_left Left endpoint of the local search bracket.

interval_right Right endpoint of the local search bracket.

n_cashflows Length of `payment`.

has_both_signs Whether the cash flow has at least one positive and one negative value.

n_sign_changes_cashflow Number of sign changes in the nonzero cash-flow sequence.

If no roots are detected, the function returns a tibble with zero rows.

See Also

`irr_flow`, `pv_flow`

Other time-value: `future_value()`, `fv_flow()`, `irr_flow()`, `plot_cash_flow()`, `present_value()`, `pv_flow()`

Examples

```
# A standard single-IRR cash flow
irr_flow_multi(
  payment = c(-1000, 300, 400, 500),
  time = c(0, 1, 2, 3)
)

# A cash flow with multiple sign changes
irr_flow_multi(
```

```
  payment = c(-1000, 5000, -4500, 200),
  time = c(0, 1, 2, 3),
  search_interval = c(-0.99, 5),
  grid_points = 5000
)

# Date-based version
irr_flow_multi(
  payment = c(-1000, 300, 400, 500),
  date = as.Date(c("2026-01-01", "2027-01-01", "2028-01-01", "2029-01-01"))
)
```

`km_lifetable` *Kaplan–Meier survival curve and a lifetable-style life table*

Description

Fits the nonparametric Kaplan–Meier estimator $\hat{S}(t)$ for right-censored time-to-event data, computes Greenwood's variance estimator for $\hat{S}(t)$, and constructs a discrete life table by evaluating $\hat{S}(t)$ at user-provided cut points (`breaks`).

Usage

```
km_lifetable(
  time,
  status,
  entry = NULL,
  breaks = NULL,
  radix = 1e+05,
  conf_level = 0.95,
  assumption = c("UDD", "CF", "Balducci")
)
```

Arguments

`time`	Numeric vector. Observed times (event or censoring times).
`status`	Integer/numeric vector of the same length as `time`. Use `1` for event (death), `0` for right-censoring.
`entry`	Optional numeric vector of entry times (left truncation / delayed entry). If provided, must have the same length as `time` and satisfy `entry <= time`. If `NULL`, all individuals are assumed to enter at time 0.
`breaks`	Optional numeric vector of increasing cut points used to build the discrete life table (e.g., `0:omega`). If `NULL`, defaults to integer ages from `0` to `ceiling(max(time))`.
`radix`	Numeric. Life table radix used to scale ℓ_x (default `1e5`).
`conf_level`	Numeric in `(0,1)`. Confidence level for pointwise intervals for $\hat{S}(t)$ computed via the log(-log) transformation (default `0.95`).

`assumption` Character. Fractional-age assumption used to compute L_x within each interval: `"UDD"`, `"CF"` (constant force), or `"Balducci"`.

Details

The resulting life table is intended for *experience-based* (empirical) life tables in actuarial/demographic contexts (e.g., cohort studies, population indicators). It is not a replacement for graduated/regulatory tables when smoothing, extrapolation, or product-specific selection effects are required.

Kaplan–Meier estimator. At each observed event time t_j:

$$\hat{S}(t) = \prod_{t_j \le t} \left(1 - \frac{d_j}{n_j}\right)$$

where n_j is the risk set size and d_j is the number of events. Greenwood's variance:

$$\widehat{\mathrm{Var}}(\hat{S}(t)) = \hat{S}(t)^2 \sum_{t_j \le t} \frac{d_j}{n_j(n_j - d_j)}.$$

Pointwise confidence intervals use the log(-log) transformation.

Life table mapping. For each interval $[x, x + \Delta)$:

$$\ell_x = \text{radix} \cdot \hat{S}(x), \quad d_x = \ell_x - \ell_{x+\Delta}, \quad q_x = d_x/\ell_x.$$

Exposure $L_x = \int_x^{x+\Delta} \ell(t)\, dt$ is computed using the selected fractional-age assumption (Finan, Section 24):

- UDD (Finan, Sec. 24.1): $L_x \approx \frac{\ell_x + \ell_{x+\Delta}}{2}\Delta$
- CF (constant force, Finan, Sec. 24.2): $L_x = \Delta \cdot (\ell_x - \ell_{x+\Delta}) / \ln(\ell_x/\ell_{x+\Delta})$
- Balducci (Finan, Sec. 24.3): $L_x = \Delta \cdot \ell_x \ell_{x+\Delta} / (\ell_x - \ell_{x+\Delta}) \cdot \ln(\ell_x/\ell_{x+\Delta})$

Additional columns follow Finan, Sections 23.3, 23.8–23.9:

- $T_x = \sum_{k \ge x} L_k$: total expected years lived after age x (Finan, Sec. 23.3).
- $\mathring{e}_x = T_x/\ell_x$: complete life expectancy (Finan, Sec. 23.3).
- $m_x = d_x/L_x$: central death rate (Finan, Sec. 23.9).
- $a_x = (\ell_x\Delta - L_x)/d_x$: average fraction of the interval lived by those who die.

Value

A list with two tibbles:

- `km`: tibble with columns `time`, `n_risk`, `d`, `censored`, `S`, `varS`, `seS`, `ci_low`, `ci_high`.
- `lifetable`: tibble with columns `x`, `x_next`, `width`, `lx`, `dx`, `qx`, `px`, `mx`, `ax`, `Lx`, `Tx`, `ex`. Carries class `"lifetable"` and standard attributes for compatibility with downstream functions.

See Also

`lifetable` for building tables from known mortality inputs, `plot_km` for plotting the KM curve.

Examples

```
set.seed(1)
n <- 200
trueT <- rexp(n, rate = 0.08)
censT <- rexp(n, rate = 0.04)
time <- pmin(trueT, censT)
status <- as.integer(trueT <= censT)

out <- km_lifetable(time, status, breaks = 0:25, radix = 100000)
head(out$km)
head(out$lifetable)

# Tidy pipeline: filter high-mortality intervals
out$lifetable |> dplyr::filter(qx > 0.05)

# Compare UDD vs CF assumptions
udd <- km_lifetable(time, status, breaks = 0:20, assumption = "UDD")
cfm <- km_lifetable(time, status, breaks = 0:20, assumption = "CF")
c(ex_udd = udd$lifetable$ex[1], ex_cf = cfm$lifetable$ex[1])

# Plot the KM curve with plot_km
plot_km(out$km, time_col = "time", surv_col = "S",
        lower_col = "ci_low", upper_col = "ci_high")
```

---

lifetable *Build an annual life table (tidy tibble) from lx, qx, px, or mx*

Description

Creates an annual life table with **integer, consecutive ages** and returns a **tibble** (tidyverse-friendly) with class "lifetable".

Usage

```
lifetable(
  x,
  lx = NULL,
  qx = NULL,
  px = NULL,
  mx = NULL,
  radix = NULL,
  omega = NULL,
  close = TRUE,
  ax = 0.5,
  type = c("ultimate", "select"),
  frac = c("UDD", "CF", "Balducci"),
  check = TRUE,
  tol = 1e-10
)
```

Arguments

x	Numeric vector of ages. Must be **integer** and **consecutive** (annual table), e.g. 0:110.
lx	Optional numeric vector of survivors ℓ_x. Must be nonnegative and nonincreasing.
qx	Optional numeric vector of one-year death probabilities q_x. NA values are allowed (useful at the last age when close=TRUE), but Inf/NaN are not allowed.
px	Optional numeric vector of one-year survival probabilities p_x. NA values are allowed, but Inf/NaN are not allowed. If provided, qx = 1 - px.
mx	Optional numeric vector of central death rates m_x. NA values are allowed, but Inf/NaN are not allowed. If provided, converted to qx using ax.
radix	Optional positive scalar. Required if building lx from (qx/px/mx) and lx is not provided.
omega	Optional integer limiting age. If omega < max(x), the table is truncated to omega. If omega > max(x), an error is raised (the function will not invent missing ages).
close	Logical. If TRUE (default), closes the table at omega (forces terminal conditions).
ax	Scalar in [0,1]. Average fraction of the year lived by those who die in the interval $[x, x+1)$. Under UDD (Finan, Sec. 24.1), ax = 0.5. Under constant force, $a_x = 1/\mu - 1/(\exp(\mu) - 1)$. At the terminal age with close = TRUE, mx equals $1/(1 - a_x)$, which is 2 for ax = 0.5. Default is 0.5.
type	Character. "ultimate" or "select" (metadata). Stored as an attribute and used by downstream functions.
frac	Character. "UDD", "CF", or "Balducci" (metadata). Stored as an attribute and used by fractional-age functions such as t_px.
check	Logical. If TRUE (default), performs strict validity and consistency checks.
tol	Numeric tolerance for integer checks and consistency checks.

Details

The table can be built from exactly one of:

- lx (survivors), or
- qx (one-year death probabilities), or
- px (one-year survival probabilities), or
- mx (central death rates),

and the function will compute the remaining columns consistently: dx, qx, px, and mx.

When multiple inputs are provided, priority is: lx > qx > px > mx. If lx is provided together with qx, cross-consistency is validated (both must agree via $q_x = (\ell_x - \ell_{x+1})/\ell_x$).

By default, the table is actuarially closed at omega:

$$\ell_{\omega+1} = 0 \Rightarrow d_\omega = \ell_\omega \Rightarrow q_\omega = 1 \Rightarrow p_\omega = 0.$$

The life table follows the standard actuarial construction described in Finan, Sections 22–24 (Exam MLC preparation).

The basic identities are (Finan, Section 22):

$$\ell_x = \ell_0 \cdot s(x), \quad d_x = \ell_x - \ell_{x+1}, \quad q_x = d_x/\ell_x, \quad p_x = \ell_{x+1}/\ell_x.$$

The central death rate `mx` is computed via the discrete approximation (Finan, Section 23.9):

$$m_x = \frac{q_x}{1 - a_x \cdot q_x}$$

which under UDD (`ax = 0.5`) reduces to the classical formula $m_x = q_x/(1-0.5\,q_x)$ (Finan, Section 24.1). This arises because under UDD, $L_x = \ell_x - \frac{1}{2}d_x$, and therefore $m_x = d_x/L_x$.

At the terminal age ω with `close = TRUE`, closure forces $q_\omega = 1$, $p_\omega = 0$, and $d_\omega = \ell_\omega$. The corresponding m_ω equals $1/(1 - a_x)$, which is 2 under UDD (`ax = 0.5`). If `ax = 1`, $m_\omega = \infty$.

Value

A `tibble` with class `c("lifetable","tbl_df","tbl","data.frame")` and columns:

- `x`: integer ages
- `lx`: survivors at exact age x
- `dx`: deaths in $[x, x+1)$
- `qx`: probability of death in $[x, x+1)$
- `px`: probability of survival to $x+1$
- `mx`: central death rate (derived using `ax`). At the terminal age with `close = TRUE`, `mx` equals $1/(1 - a_x)$ and may be `Inf` if `ax = 1`.

Attributes include: `radix`, `omega`, `type`, `frac`, `closed`, `ax`.

See Also

`km_lifetable` for Kaplan–Meier construction, `t_px` and `t_qx` for survival and death probabilities (including fractional ages), `e_x` for curtate and complete life expectancy, `annuity_x` and `insurance_x` for life contingency valuations that consume a life table.

Examples

```
# Example 1: build from lx (Finan, Section 22 style)
x  <- 0:5
lx <- c(100000, 99500, 99000, 98200, 97000, 95000)
lt1 <- lifetable(x = x, lx = lx, omega = 5, close = TRUE)
lt1

# Example 2: build from qx (radix required)
qx <- c(0.005, 0.005, 0.008, 0.012, 0.020, 1)
lt2 <- lifetable(x = x, qx = qx, radix = 100000, omega = 5, close = TRUE)
lt2

# Example 3: build from px
px <- 1 - c(0.005, 0.005, 0.008, 0.012, 0.020, 1)
lt3 <- lifetable(x = x, px = px, radix = 100000, omega = 5, close = TRUE)
```

```
lt3

# Example 4: build from mx
mx <- c(0.005, 0.006, 0.008, 0.012, 0.020, 0.030)
lt4 <- lifetable(x = x, mx = mx, radix = 100000, omega = 5, close = TRUE, ax = 0.5)
lt4

# Example 5: truncate to a smaller omega
lt5 <- lifetable(x = 0:10, lx = 100000 * exp(-0.01 * (0:10)), omega = 7, close = TRUE)
lt5

# Example 6: Finan Example 22.1 - exponential survival s(x) = exp(-0.005x)
lt_exp <- lifetable(
  x  = 0:7,
  lx = 1000 * exp(-0.005 * (0:7)),
  close = TRUE
)
lt_exp

# Example 7: verify survival identity (Finan, Section 22)
# 2_p_2 = l_4 / l_2 = 97000 / 99000
lt1$lx[lt1$x == 4] / lt1$lx[lt1$x == 2]

# Example 8: without closure - qx at omega is not forced to 1
lt_open <- lifetable(x = 0:3, lx = c(1000, 900, 750, 500), close = FALSE)
lt_open$qx  # last element is NA

# Example 9: access table metadata
attr(lt1, "omega")   # 5
attr(lt1, "closed")  # TRUE
attr(lt1, "frac")    # "UDD"
attr(lt1, "ax")      # 0.5
```

`life_contract` *Create a life-contingency contract specification*

Description

Creates a lightweight contract object that stores common life-contingency inputs for use in pipe workflows.

Usage

```
life_contract(
  mortality_table,
  lives = c("single", "joint", "last_survivor"),
  age = NULL,
  age_x = NULL,
  age_y = NULL,
```

```
  rate,
  rate_type = "effective",
  m = 1L,
  ...
)
```

Arguments

`mortality_table`	A life table or a list of two life tables. For single-life contracts, provide one data.frame or tibble. For two-life contracts, provide either one data.frame used for both lives, or `list(table_x, table_y)` with one table for each life. Each life table must contain columns `x` and `lx`.
`lives`	Character string. Use `"single"` for a single-life contract, `"joint"` for a joint-life two-life contract, or `"last_survivor"` for a last-survivor two-life contract.
`age`	Numeric scalar. Age for a single-life contract.
`age_x`	Numeric scalar. First age for two-life contracts.
`age_y`	Numeric scalar. Second age for two-life contracts.
`rate`	Numeric scalar. Annual interest-rate input.
`rate_type`	Character string indicating the rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Compounding frequency for nominal rates. Ignored for `rate_type = "effective"` and `rate_type = "force"`.
`...`	Reserved for future extensions.

Details

This function does not compute actuarial values. It validates and stores common inputs such as the mortality table, life status, ages, and interest rate specification. Calculation functions such as `annuity_x()`, `insurance_x()`, `premium_x()`, `reserve_x()`, `annuity_xy()`, `insurance_xy()`, `premium_xy()`, and simulation functions can then consume this object.

Value

An object of class `"tidyact_life_contract"`.

See Also

Other life-contingencies: `annuity_x()`, `annuity_xy()`, `insurance_x()`, `insurance_xy()`, `premium_gross()`, `premium_x()`, `premium_xy()`, `reserve_x()`, `reserve_xy()`, `simulate_annuity_x()`, `simulate_insurance_x()`

`loans_sample` *Sample loan contracts for amortization examples*

Description

A small pedagogical dataset containing level-payment loan contracts for amortization schedule and outstanding balance examples.

A small pedagogical dataset containing level-payment loan contracts for amortization schedule and outstanding balance examples.

Usage

```
loans_sample
```

```
loans_sample
```

Format

A tibble with 4 rows and 7 variables:

loan_id Loan identifier.

principal Initial loan principal.

annual_effective_rate Annual effective interest rate.

term_months Loan term in months.

payments_per_year Number of payments per year.

loan_type Short loan type label.

payment Level payment amount.

A tibble with 4 rows and 7 variables:

loan_id Loan identifier.

principal Initial loan principal.

annual_effective_rate Annual effective interest rate.

term_months Loan term in months.

payments_per_year Number of payments per year.

loan_type Short loan type label.

payment Level payment amount.

Source

Synthetic pedagogical data created for tidyactuarial examples.

Synthetic pedagogical data created for tidyactuarial examples.

Examples

```
data(loans_sample)

loans_sample |>
  dplyr::select(loan_id, principal, annual_effective_rate, term_months, payment)

data(loans_sample)

loans_sample |>
  dplyr::select(loan_id, principal, annual_effective_rate, term_months, payment)
```

`lt_tau` *Total-decrement lifetable from a multiple decrement table: lt_tau*

Description

Builds a single-decrement lifetable for the *total* decrement (any cause), using $q_x^{(\tau)}$ from a multiple decrement table produced by `md_table`. This enables direct re-use of single-life functions (e.g., `t_px`, `t_qx`, `t_Ex`, annuities, insurances) under the total decrement model.

Usage

```
lt_tau(md, ...)
```

Arguments

`md`	A multiple decrement table (typically the output of `md_table`), containing columns `x` and `q_total`.
`...`	Additional arguments passed to `lifetable` (e.g., `radix`, `omega`, `close`, `ax`, `type`, `frac`, `check`, `tol`).

Details

Given cause-specific decrement probabilities $q_x^{(j)}$, the total decrement is $q_x^{(\tau)} = \sum_j q_x^{(j)}$. This function simply passes `x = md$x` and `qx = md$q_total` to `lifetable`.

Value

A lifetable object as produced by `lifetable`.

Examples

```
qx_df <- tibble::tibble(
  x = 30:35,
  q_death = c(0.001, 0.0012, 0.0014, 0.0017, 0.0020, 1.0000),
  q_disability = c(0.002, 0.0021, 0.0022, 0.0023, 0.0024, 0.0000)
)
```

```
md <- md_table(qx_df, radix = 1e5, close = TRUE)
lt <- lt_tau(md, radix = 1e5, close = TRUE, frac = "UDD")
t_px(lt, x = 30, t = 5)
```

`mc_annuity` *Compute simulated present values for life annuities*

Description

Computes Monte Carlo simulated present values of life annuity payments from simulated future lifetimes.

Usage

```
mc_annuity(
  data,
  rate,
  payment = 1,
  payments_per_year = 1,
 annuity = c("whole_life", "temporary", "deferred", "deferred_temporary", "certain",
    "guaranteed"),
  term = NULL,
  deferral_years = 0,
  guarantee_years = NULL,
  timing = c("immediate", "due"),
  interest_type = c("effective", "nominal", "force"),
  m = 1,
  k_col = "Kx",
  tx_col = "Tx",
  annuity_col = "pv_annuity"
)
```

Arguments

`data`	A data frame or tibble containing simulated future lifetimes, typically returned by `simulate_lifetime()` or by `mc_multilife_status()` when working with multiple-life statuses.
`rate`	Numeric scalar. Interest rate used for discounting.
`payment`	Numeric scalar. Amount of each annuity payment. Default is `1`.
`payments_per_year`	Positive integer-like scalar. Number of annuity payments per year. Default is `1`, corresponding to annual payments. For example, use `payments_per_year = 12` for monthly payments, `4` for quarterly payments, and `2` for semiannual payments.
`annuity`	Character string specifying the annuity type. Available options are `"whole_life"`, `"temporary"`, `"deferred"`, `"deferred_temporary"`, `"certain"`, and `"guaranteed"`.

term	Numeric scalar. Term of the annuity in years. Required for "temporary", "deferred_temporary", and "certain" annuities.
deferral_years	Numeric scalar. Deferral period in years. Default is 0. Required to be positive for "deferred" and "deferred_temporary" annuities.
guarantee_years	Numeric scalar. Guaranteed payment period in years. Required for "guaranteed" annuities.
timing	Character string specifying the annuity payment timing. Available options are "immediate" and "due". Default is "immediate".
interest_type	Character string specifying the interest rate convention. Available options are "effective", "nominal", and "force". Default is "effective".
m	Numeric scalar. Number of interest conversion periods per year when interest_type = "nominal". Default is 1. This argument controls the interest-rate conversion frequency only. It does not represent annuity payment frequency.
k_col	Character string. Name of the column containing simulated curtate future lifetimes. Default is "Kx".
tx_col	Character string. Name of the column containing simulated complete future lifetimes. Default is "Tx". This column is required when payments_per_year > 1, except for annuities certain.
annuity_col	Character string. Name of the output column containing simulated present values of annuity payments. Default is "pv_annuity".

Details

This function is designed to be used after simulate_lifetime(). It takes simulated values of the curtate future lifetime K_x, and when needed the complete future lifetime T_x, and evaluates the present value random variable associated with several classical annuity benefits.

Let K_x denote the curtate future lifetime of a life aged x, let T_x denote the complete future lifetime, and let v denote the annual discount factor.

The arguments m and payments_per_year have different meanings:

- m is used only when interest_type = "nominal" and controls the frequency of interest conversion.
- payments_per_year controls how frequently annuity payments are made.

The argument payment represents the amount of each annuity payment. Thus, for a monthly annuity with total annual payment equal to 1, use payment = 1 / 12 and payments_per_year = 12.

For annual payments, payments_per_year = 1, the function works directly with K_x. For fractional payments, such as monthly, quarterly, or semiannual payments, the function uses T_x to determine whether the life is alive at each fractional payment time.

For annual whole life annuity-immediate, the simulated present value is

$$Y = \sum_{j=1}^{K_x} cv^j,$$

where c is the amount of each payment.

For annual whole life annuity-due, the simulated present value is

$$\ddot{Y} = \sum_{j=0}^{K_x} c v^j.$$

For fractional payments with payment frequency m_p, annuity-immediate payments are made at times $1/m_p, 2/m_p, \ldots$ while the life is alive. Annuity-due payments are made at times $0, 1/m_p, 2/m_p, \ldots$ while the life is alive.

The following annuity types are supported:

- `"whole_life"`: payments continue while the life is alive.
- `"temporary"`: payments continue while the life is alive, but for at most `term` years.
- `"deferred"`: payments begin after `deferral_years` years and continue while the life is alive.
- `"deferred_temporary"`: payments begin after `deferral_years` years and continue while the life is alive, but for at most `term` years after the deferral period.
- `"certain"`: payments are made for `term` years regardless of survival.
- `"guaranteed"`: payments continue while the life is alive, with at least `guarantee_years` years of payments guaranteed.

The function returns simulated present values, not only their expected value. Therefore the resulting column can be summarized with `summary_mc()`, plotted with `ggplot2`, or used to construct premiums, losses, and reserves.

Value

A tibble with the original simulation columns and additional columns:

- `rate`: original rate supplied.
- `interest_type`: interest rate convention.
- `m`: interest conversion frequency.
- `effective_rate`: equivalent annual effective interest rate.
- `discount_factor`: annual discount factor.
- `annuity`: annuity type.
- `payment`: amount of each annuity payment.
- `payments_per_year`: annuity payment frequency.
- `term`: annuity term, if applicable.
- `deferral_years`: deferral period.
- `guarantee_years`: guaranteed period, if applicable.
- `timing`: annuity payment timing.
- `n_payments`: number of payments made in the simulated scenario.
- `first_payment_time`: first payment time in the simulated scenario.
- `last_payment_time`: last payment time in the simulated scenario.
- `pv_annuity`: simulated present value of annuity payments, or another name supplied through `annuity_col`.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*. Second Edition. Society of Actuaries.

See Also

simulate_lifetime(), simulate_lifetimes(), mc_multilife_status(), mc_insurance(), mc_premium(), mc_loss(), mc_reserve(), summary_mc()

Examples

```
life_table <- tibble::tibble(
  age = 40:100,
  qx = seq(0.002, 1, length.out = 61)
)

# Annual whole life annuity-due
life_table |>
  simulate_lifetime(age = 40, n_sim = 1000, seed = 123) |>
  mc_annuity(
    rate = 0.05,
    annuity = "whole_life",
    payment = 1,
    payments_per_year = 1,
    timing = "due"
  )

# Monthly whole life annuity-due with total annual payment equal to 1
life_table |>
  simulate_lifetime(
    age = 40,
    n_sim = 1000,
    fractional = "udd",
    seed = 123
  ) |>
  mc_annuity(
    rate = 0.05,
    annuity = "whole_life",
    payment = 1 / 12,
    payments_per_year = 12,
    timing = "due"
  )

# Quarterly temporary life annuity-immediate
life_table |>
  simulate_lifetime(
    age = 40,
    n_sim = 1000,
    fractional = "udd",
    seed = 123
  ) |>
  mc_annuity(
```

```
    rate = 0.05,
    annuity = "temporary",
    term = 20,
    payment = 1 / 4,
    payments_per_year = 4,
    timing = "immediate"
  )

# Monthly joint-life annuity using a multiple-life status
life_table |>
  simulate_lifetimes(
    ages = c(60, 58),
    n_sim = 1000,
    fractional = "udd",
    seed = 123
  ) |>
  mc_multilife_status(status = "joint_life") |>
  mc_annuity(
    rate = 0.04,
    annuity = "whole_life",
    payment = 1 / 12,
    payments_per_year = 12,
    timing = "due",
    k_col = "K_status",
    tx_col = "T_status"
  )

# Nominal rate convertible monthly, with quarterly payments
life_table |>
  simulate_lifetime(
    age = 40,
    n_sim = 1000,
    fractional = "udd",
    seed = 123
  ) |>
  mc_annuity(
    rate = 0.06,
    interest_type = "nominal",
    m = 12,
    annuity = "whole_life",
    payment = 1 / 4,
    payments_per_year = 4,
    timing = "due"
  )
```

---

mc_insurance *Compute simulated present values for life insurance benefits*

Description

Computes Monte Carlo simulated present values of life insurance benefits from simulated future lifetimes.

Usage

```
mc_insurance(
  data,
  rate,
  benefit = 1,
  insurance = c("whole_life", "term", "deferred", "deferred_term", "pure_endowment",
    "endowment"),
  term = NULL,
  deferral_years = 0,
  payment_timing = c("end_of_year", "moment_of_death"),
  interest_type = c("effective", "nominal", "force"),
  m = 1,
  k_col = "Kx",
  tx_col = "Tx",
  benefit_col = "pv_benefit"
)
```

Arguments

data	A data frame or tibble containing simulated future lifetimes, typically returned by simulate_lifetime() or by mc_multilife_status() when working with multiple-life statuses.
rate	Numeric scalar. Interest rate used for discounting.
benefit	Numeric scalar. Benefit amount payable under the insurance. Default is 1.
insurance	Character string specifying the type of insurance. Available options are "whole_life", "term", "deferred", "deferred_term", "pure_endowment", and "endowment".
term	Numeric scalar. Term of the insurance in years. Required for "term", "deferred_term", "pure_endowment", and "endowment".
deferral_years	Numeric scalar. Deferral period in years. Default is 0. Required to be positive for "deferred" and "deferred_term" insurance.
payment_timing	Character string specifying when death benefits are paid. Available options are "end_of_year" and "moment_of_death". Default is "end_of_year".
interest_type	Character string specifying the interest rate convention. Available options are "effective", "nominal", and "force". Default is "effective".
m	Numeric scalar. Number of interest conversion periods per year when interest_type = "nominal". Default is 1. This argument controls the interest-rate conversion frequency only. It does not represent benefit frequency or premium payment frequency.
k_col	Character string. Name of the column containing simulated curtate future lifetimes. Default is "Kx".

`tx_col` Character string. Name of the column containing simulated complete future lifetimes. Required when `payment_timing = "moment_of_death"`. Default is `"Tx"`.

`benefit_col` Character string. Name of the output column containing simulated present values of benefits. Default is `"pv_benefit"`.

Details

This function is designed to be used after `simulate_lifetime()`. It takes simulated values of the curtate future lifetime K_x, and optionally the complete future lifetime T_x, and evaluates the present value random variable associated with classical life insurance benefits.

Let K_x denote the curtate future lifetime of a life aged x, let T_x denote the complete future lifetime, and let v denote the annual discount factor.

If `payment_timing = "end_of_year"`, death benefits are discounted using K_x+1. If `payment_timing = "moment_of_death"`, death benefits are discounted using T_x.

The arguments `interest_type` and `m` determine how the supplied rate is converted into an annual effective rate before discounting. When `interest_type = "effective"`, `rate` is interpreted as an annual effective interest rate. When `interest_type = "nominal"`, `rate` is interpreted as a nominal annual interest rate convertible `m` times per year. When `interest_type = "force"`, `rate` is interpreted as a constant force of interest.

The following insurance types are supported:

- `"whole_life"`: benefit is paid whenever death occurs.
- `"term"`: benefit is paid if death occurs within `term` years.
- `"deferred"`: benefit is paid if death occurs after the deferral period.
- `"deferred_term"`: benefit is paid if death occurs after the deferral period and within the following `term` years.
- `"pure_endowment"`: benefit is paid at time `term` if the life survives to that time.
- `"endowment"`: death benefit is paid if death occurs within `term` years; otherwise, a survival benefit is paid at time `term`.

The function returns simulated present values, not only their expected values. Therefore the resulting column can be summarized with `summary_mc()`, plotted with `ggplot2`, or used to construct premiums, losses, and reserves.

Value

A tibble with the original simulation columns and additional columns:

- `rate`: original rate supplied.
- `interest_type`: interest rate convention.
- `m`: interest conversion frequency.
- `effective_rate`: equivalent annual effective interest rate.
- `discount_factor`: annual discount factor.
- `insurance`: insurance type.

- benefit: benefit amount.
- term: insurance term, if applicable.
- deferral_years: deferral period.
- payment_timing: timing used for death benefits.
- benefit_time: simulated payment time of the benefit.
- benefit_indicator: indicator that the benefit is paid.
- pv_benefit: simulated present value of the benefit, or another name supplied through benefit_col.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*. Second Edition. Society of Actuaries.

See Also

simulate_lifetime(), simulate_lifetimes(), mc_multilife_status(), mc_annuity(), mc_premium(), mc_loss(), mc_reserve(), summary_mc()

Examples

```
life_table <- tibble::tibble(
  age = 40:100,
  qx = seq(0.002, 1, length.out = 61)
)

# Whole life insurance payable at the end of the year of death
life_table |>
  simulate_lifetime(age = 40, n_sim = 1000, seed = 123) |>
  mc_insurance(
    rate = 0.05,
    insurance = "whole_life",
    benefit = 1
  )

# 20-year term insurance
life_table |>
  simulate_lifetime(age = 40, n_sim = 1000, seed = 123) |>
  mc_insurance(
    rate = 0.05,
    insurance = "term",
    term = 20,
    benefit = 100000
  )

# 10-year deferred whole life insurance
life_table |>
  simulate_lifetime(age = 40, n_sim = 1000, seed = 123) |>
  mc_insurance(
    rate = 0.05,
    insurance = "deferred",
```

```
    deferral_years = 10,
    benefit = 1
  )

# 10-year deferred, 20-year term insurance
life_table |>
  simulate_lifetime(age = 40, n_sim = 1000, seed = 123) |>
  mc_insurance(
    rate = 0.05,
    insurance = "deferred_term",
    deferral_years = 10,
    term = 20,
    benefit = 1
  )

# Pure endowment
life_table |>
  simulate_lifetime(age = 40, n_sim = 1000, seed = 123) |>
  mc_insurance(
    rate = 0.05,
    insurance = "pure_endowment",
    term = 20,
    benefit = 1
  )

# Endowment insurance payable at the moment of death if death occurs
life_table |>
  simulate_lifetime(age = 40, n_sim = 1000, fractional = "udd", seed = 123) |>
  mc_insurance(
    rate = 0.05,
    insurance = "endowment",
    term = 20,
    payment_timing = "moment_of_death",
    benefit = 1
  )

# Nominal rate convertible monthly
life_table |>
  simulate_lifetime(age = 40, n_sim = 1000, seed = 123) |>
  mc_insurance(
    rate = 0.06,
    interest_type = "nominal",
    m = 12,
    insurance = "whole_life",
    benefit = 1
  )

# First-death insurance using a multiple-life status
life_table |>
  simulate_lifetimes(
    ages = c(60, 58),
    n_sim = 1000,
    fractional = "udd",
```

```
  seed = 123
) |>
mc_multilife_status(status = "first_death") |>
mc_insurance(
  rate = 0.04,
  insurance = "whole_life",
  benefit = 100000,
  k_col = "K_status",
  tx_col = "T_status"
)
```

`mc_loss` *Compute Monte Carlo loss random variables for life contingencies*

Description

Computes simulated actuarial loss random variables from Monte Carlo present values of benefits, premium annuities, and premiums.

Usage

```
mc_loss(
  data,
  benefit_col = "pv_benefit",
  annuity_col = "pv_annuity",
  premium_col = "premium",
  loss_col = "loss",
  premium = NULL
)
```

Arguments

`data`	A data frame or tibble containing simulated present values of benefits and premium annuities.
`benefit_col`	Character string. Name of the column containing the simulated present value of benefits. Default is `"pv_benefit"`.
`annuity_col`	Character string. Name of the column containing the simulated present value of premium annuities. Default is `"pv_annuity"`.
`premium_col`	Character string. Name of the column containing the premium. Default is `"premium"`.
`loss_col`	Character string. Name of the output column containing the simulated loss random variable. Default is `"loss"`.
`premium`	Optional numeric scalar. If supplied, this value is used as the premium instead of reading the premium from `premium_col`.

Details

This function constructs the simulated loss random variable

$$L = Z - PY,$$

where Z is the present value random variable of the insurance benefit, Y is the present value random variable of the premium annuity, and P is the premium.

In actuarial notation, the loss at issue is commonly written as

$$L_0 = Z - PY.$$

The random variable Z represents the present value of future benefits, while Y represents the present value of future premium payments.

This function does not estimate the premium. It only constructs the simulated loss random variable. The premium may come from a column previously created by `mc_premium()`, or it may be supplied directly through the `premium` argument.

The interpretation of `premium` depends on how the premium annuity present value was constructed. If `pv_annuity` was generated with annual payments, then `premium` corresponds to that annual premium structure. If `pv_annuity` was generated using fractional payments in `mc_annuity()`, such as `payment = 1 / 12` and `payments_per_year = 12`, then the premium is applied to that same fractional payment pattern.

Thus, `mc_loss()` can be used without modification for annual premiums, monthly premiums, quarterly premiums, semiannual premiums, and multiple-life premium structures, provided that `annuity_col` contains the appropriate simulated premium annuity present value.

The resulting loss column can be used to estimate quantities such as:

- expected loss;
- variance and standard deviation of loss;
- probability of positive loss;
- loss quantiles;
- empirical value-at-risk type measures;
- sensitivity of loss distributions across ages, benefits, products, payment frequencies, interest rates, or multiple-life statuses.

Value

A tibble with the original columns and one additional column containing the simulated loss random variable. The name of this column is controlled by `loss_col`.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*. Second Edition. Society of Actuaries.

See Also

simulate_lifetime(), simulate_lifetimes(), mc_multilife_status(), mc_insurance(), mc_annuity(), mc_premium(), mc_reserve(), summary_mc()

Examples

```
# Example 1: direct use with simulated present values
sim_values <- tibble::tibble(
  sim_id = 1:5,
  pv_benefit = c(0.80, 0.75, 0.95, 0.60, 0.70),
  pv_annuity = c(8.0, 7.5, 9.0, 6.0, 7.0),
  premium = 0.10
)

sim_values |>
  mc_loss()

# Example 2: using a premium supplied directly
sim_values |>
  mc_loss(premium = 0.12)

# Example 3: annual whole life loss
life_table <- tibble::tibble(
  age = 40:100,
  qx = seq(0.002, 1, length.out = 61)
)

life_table |>
  simulate_lifetime(age = 40, n_sim = 500, seed = 123) |>
  mc_insurance(
    rate = 0.05,
    insurance = "whole_life",
    benefit = 1
  ) |>
  mc_annuity(
    rate = 0.05,
    annuity = "whole_life",
    payment = 1,
    payments_per_year = 1,
    timing = "due"
  ) |>
  mc_premium() |>
  mc_loss()

# Example 4: monthly whole life loss
life_table |>
  simulate_lifetime(
    age = 40,
    n_sim = 500,
    fractional = "udd",
    seed = 123
  ) |>
```

```
  mc_insurance(
    rate = 0.05,
    insurance = "whole_life",
    benefit = 1
  ) |>
  mc_annuity(
    rate = 0.05,
    annuity = "whole_life",
    payment = 1 / 12,
    payments_per_year = 12,
    timing = "due"
  ) |>
  mc_premium() |>
  mc_loss()

# Example 5: summarising simulated losses
life_table |>
  simulate_lifetime(
    age = 45,
    n_sim = 500,
    fractional = "udd",
    seed = 123
  ) |>
  mc_insurance(
    rate = 0.04,
    insurance = "term",
    term = 20,
    benefit = 100000
  ) |>
  mc_annuity(
    rate = 0.04,
    annuity = "temporary",
    term = 20,
    payment = 1 / 12,
    payments_per_year = 12,
    timing = "due"
  ) |>
  mc_premium() |>
  mc_loss() |>
  summary_mc(value_col = "loss")

# Example 6: joint-life loss with monthly premium annuity
life_table |>
  simulate_lifetimes(
    ages = c(60, 58),
    n_sim = 500,
    fractional = "udd",
    seed = 123
  ) |>
  mc_multilife_status(status = "joint_life") |>
  mc_insurance(
    rate = 0.04,
    insurance = "whole_life",
```

```
  benefit = 100000,
  k_col = "K_status",
  tx_col = "T_status"
) |>
mc_annuity(
  rate = 0.04,
  annuity = "whole_life",
  payment = 1 / 12,
  payments_per_year = 12,
  timing = "due",
  k_col = "K_status",
  tx_col = "T_status"
) |>
mc_premium() |>
mc_loss()
```

`mc_multilife_status` *Construct multiple-life status lifetimes from simulated lives*

Description

Constructs simulated multiple-life status lifetimes from simulated individual lifetimes in long format.

Usage

```
mc_multilife_status(
  data,
  status = c("joint_life", "last_survivor", "first_death", "last_death", "kth_death",
    "at_least_k_alive"),
  k = NULL,
  sim_col = "sim_id",
  life_col = "life_id",
  k_col = "Kx",
  tx_col = "Tx",
  k_status_col = "K_status",
  t_status_col = "T_status",
  keep_lifetimes = FALSE
)
```

Arguments

`data`	A data frame or tibble containing simulated multiple-life lifetimes, typically returned by `simulate_lifetimes()`.
`status`	Character string specifying the multiple-life status. Available options are `"joint_life"`, `"last_survivor"`, `"first_death"`, `"last_death"`, `"kth_death"`, and `"at_least_k_alive"`.

k	Optional positive integer. Required when status = "kth_death" or status = "at_least_k_alive".
sim_col	Character string. Name of the simulation identifier column. Default is "sim_id".
life_col	Character string. Name of the life identifier column. Default is "life_id".
k_col	Character string. Name of the column containing simulated curtate future lifetimes. Default is "Kx".
tx_col	Character string. Name of the column containing simulated complete future lifetimes. Default is "Tx".
k_status_col	Character string. Name of the output column containing the curtate lifetime of the multiple-life status. Default is "K_status".
t_status_col	Character string. Name of the output column containing the complete lifetime of the multiple-life status. Default is "T_status".
keep_lifetimes	Logical. If TRUE, the output includes a list-column named lifetimes containing the simulated individual lifetimes used in each simulation. Default is FALSE.

Details

This function is designed to be used after simulate_lifetimes(). It takes one row per simulation and per life, and returns one row per simulation with simulated curtate and complete lifetimes for the selected multiple-life status.

The key output columns are K_status and T_status. They are intentionally named so they can be used directly in downstream functions such as mc_insurance() and mc_annuity() through their k_col and tx_col arguments.

Suppose that, for a given simulation, the complete future lifetimes of r lives are

$$T_1, T_2, \ldots, T_r.$$

Let

$$T_{(1)} \le T_{(2)} \le \cdots \le T_{(r)}$$

denote the ordered lifetimes. The selected multiple-life status is converted into an order statistic:

- "joint_life": fails at the first death, $T_{(1)}$.
- "first_death": time until the first death, $T_{(1)}$.
- "last_survivor": fails at the last death, $T_{(r)}$.
- "last_death": time until the last death, $T_{(r)}$.
- "kth_death": time until the k-th death, $T_{(k)}$.
- "at_least_k_alive": status remains active while at least k lives are alive, and fails at the $(r - k + 1)$-th death.

The same order-statistic logic is applied to the curtate future lifetimes $K_1, K_2, \ldots, K_r$, producing K_status.

This construction allows multiple-life quantities to be evaluated using the single-life Monte Carlo functions. For example, a joint-life annuity can be computed by passing k_col = "K_status" to

mc_annuity(), and a first-death insurance can be computed by passing k_col = "K_status" and tx_col = "T_status" to mc_insurance().

The function assumes the individual lifetimes have already been simulated. If the input comes from simulate_lifetimes(), the default model is independence across lives.

Value

A tibble with one row per simulation and the following columns:

- sim_id: simulation identifier, or the column named by sim_col.
- status: multiple-life status.
- n_lives: number of lives in the simulation.
- k: value of k, when applicable.
- status_rank: order statistic rank used to define the status.
- K_status: simulated curtate lifetime of the status, or another name supplied through k_status_col.
- T_status: simulated complete lifetime of the status, or another name supplied through t_status_col. If complete lifetimes are unavailable, this column contains NA.
- event_life_id: identifier of the life or lives associated with the status event time.
- n_event_lives: number of lives tied at the status event time.

If keep_lifetimes = TRUE, a list-column named lifetimes is also returned.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*. Second Edition. Society of Actuaries.

See Also

simulate_lifetimes(), simulate_lifetime(), mc_insurance(), mc_annuity(), summary_mc()

Examples

```
life_table <- tibble::tibble(
  age = 60:90,
  qx = seq(0.01, 1, length.out = 31)
)

# Simulated lifetimes for two independent lives
sim_two <- life_table |>
  simulate_lifetimes(
    ages = c(60, 58),
    n_sim = 25,
    seed = 123
  )

# Joint-life status: time until the first death
mc_multilife_status(
  sim_two,
```

```
  status = "joint_life"
)

# Last-survivor status: time until the last death
mc_multilife_status(
  sim_two,
  status = "last_survivor"
)

# Simulated lifetimes for three independent lives
sim_three <- life_table |>
  simulate_lifetimes(
    ages = c(60, 58, 55),
    life_id = c("x", "y", "z"),
    n_sim = 25,
    seed = 123
  )

# Time until the second death among three lives
mc_multilife_status(
  sim_three,
  status = "kth_death",
  k = 2
)

# Status active while at least two lives are alive
mc_multilife_status(
  sim_three,
  status = "at_least_k_alive",
  k = 2
)
```

`mc_premium` *Compute Monte Carlo net premiums for life contingencies*

Description

Computes simulated net premiums from Monte Carlo present values of insurance benefits and premium annuities.

Usage

```
mc_premium(
  data,
  benefit_col = "pv_benefit",
  annuity_col = "pv_annuity",
  premium_col = "premium",
  by = NULL,
  na_rm = TRUE
)
```

Arguments

data	A data frame or tibble containing simulated present values. Usually this object is obtained after applying mc_insurance() and mc_annuity() to the same simulated lifetime sample.
benefit_col	Character string. Name of the column containing the simulated present value of the insurance benefit. Default is "pv_benefit".
annuity_col	Character string. Name of the column containing the simulated present value of the premium annuity. Default is "pv_annuity".
premium_col	Character string. Name of the output column containing the simulated net premium. Default is "premium".
by	Optional character vector with grouping columns. If supplied, the premium is computed separately within each group. If by = NULL and data is already grouped with dplyr::group_by(), the current grouping structure is used.
na_rm	Logical. Should missing values be removed when computing simulated means? Default is TRUE.

Details

This function applies the actuarial equivalence principle to simulated present value random variables. If Z denotes the present value random variable of the benefit and Y denotes the present value random variable of the premium annuity, the net premium is estimated as

$$\hat{P} = \frac{\bar{Z}}{\bar{Y}}.$$

Equivalently, this is the Monte Carlo estimator of

$$P = \frac{E[Z]}{E[Y]}.$$

The function does not simulate lifetimes and does not calculate present values directly. It only computes the Monte Carlo net premium from columns that already contain simulated present values.

In a typical workflow, simulate_lifetime() generates simulated values of K_x and possibly T_x; mc_insurance() creates the simulated benefit present value Z; mc_annuity() creates the simulated premium annuity present value Y; and mc_premium() estimates the net level premium.

The estimated premium is attached to every row of the input data. This is intentional: it makes it easy to construct the simulated loss random variable with mc_loss(), for example

$$L = Z - \hat{P}Y.$$

The function is also valid for premiums payable more than once per year. In that case, the payment frequency is not specified in mc_premium(); it is already embedded in the simulated premium annuity present value supplied through annuity_col.

For example, if mc_annuity() was called with payment = 1 / 12 and payments_per_year = 12, then pv_annuity represents the present value of a monthly premium stream whose total annual

amount is 1. The premium estimated by `mc_premium()` is then consistent with that monthly payment structure and corresponds to a Monte Carlo estimate of a premium such as $P_x^{(12)}$.

If `mc_annuity()` was called with `payment = 1` and `payments_per_year = 12`, then `pv_annuity` represents a stream of payments of 1 each month, and the resulting premium should be interpreted relative to that payment pattern.

This function computes net premiums only. It does not include expenses, safety loadings, profit margins, taxes, surrender charges, commissions, or other practical pricing adjustments.

Value

A tibble with the original columns and one additional column containing the simulated net premium. The name of this column is controlled by `premium_col`.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*. Second Edition. Society of Actuaries.

See Also

`simulate_lifetime()`, `simulate_lifetimes()`, `mc_insurance()`, `mc_annuity()`, `mc_loss()`, `mc_reserve()`, `summary_mc()`

Examples

```
# Example 1: direct use with simulated present values
sim_values <- tibble::tibble(
  sim_id = 1:6,
  pv_benefit = c(0.82, 0.74, 0.61, 0.95, 0.70, 0.88),
  pv_annuity = c(8.2, 7.5, 6.1, 9.0, 7.2, 8.8)
)

sim_values |>
  mc_premium()

# Example 2: grouped premiums by age
sim_by_age <- tibble::tibble(
  sim_id = rep(1:6, times = 2),
  age = rep(c(40, 50), each = 6),
  pv_benefit = c(
    0.82, 0.74, 0.61, 0.95, 0.70, 0.88,
    0.91, 0.86, 0.79, 0.98, 0.83, 0.94
  ),
  pv_annuity = c(
    8.2, 7.5, 6.1, 9.0, 7.2, 8.8,
    6.8, 6.4, 5.9, 7.1, 6.2, 6.7
  )
)

sim_by_age |>
  mc_premium(by = "age")
```

```
# Example 3: using dplyr grouping
sim_by_age |>
  dplyr::group_by(age) |>
  mc_premium()

# Example 4: annual whole life net premium
life_table <- tibble::tibble(
  age = 40:100,
  qx = seq(0.002, 1, length.out = 61)
)

life_table |>
  simulate_lifetime(age = 40, n_sim = 500, seed = 123) |>
  mc_insurance(
    rate = 0.05,
    insurance = "whole_life",
    benefit = 1
  ) |>
  mc_annuity(
    rate = 0.05,
    annuity = "whole_life",
    payment = 1,
    payments_per_year = 1,
    timing = "due"
  ) |>
  mc_premium()

# Example 5: monthly whole life net premium
life_table |>
  simulate_lifetime(
    age = 40,
    n_sim = 500,
    fractional = "udd",
    seed = 123
  ) |>
  mc_insurance(
    rate = 0.05,
    insurance = "whole_life",
    benefit = 1
  ) |>
  mc_annuity(
    rate = 0.05,
    annuity = "whole_life",
    payment = 1 / 12,
    payments_per_year = 12,
    timing = "due"
  ) |>
  mc_premium()

# Example 6: term insurance with monthly premium annuity
life_table |>
  simulate_lifetime(
```

```
    age = 45,
    n_sim = 500,
    fractional = "udd",
    seed = 123
  ) |>
  mc_insurance(
    rate = 0.04,
    insurance = "term",
    term = 20,
    benefit = 100000
  ) |>
  mc_annuity(
    rate = 0.04,
    annuity = "temporary",
    term = 20,
    payment = 1 / 12,
    payments_per_year = 12,
    timing = "due"
  ) |>
  mc_premium()

# Example 7: joint-life monthly net premium
life_table |>
  simulate_lifetimes(
    ages = c(60, 58),
    n_sim = 500,
    fractional = "udd",
    seed = 123
  ) |>
  mc_multilife_status(status = "joint_life") |>
  mc_insurance(
    rate = 0.04,
    insurance = "whole_life",
    benefit = 100000,
    k_col = "K_status",
    tx_col = "T_status"
  ) |>
  mc_annuity(
    rate = 0.04,
    annuity = "whole_life",
    payment = 1 / 12,
    payments_per_year = 12,
    timing = "due",
    k_col = "K_status",
    tx_col = "T_status"
  ) |>
  mc_premium()
```

`mc_reserve` *Compute Monte Carlo prospective reserves for life contingencies*

Description

Computes simulated prospective reserve losses at one or more policy durations from simulated future lifetimes.

Usage

```
mc_reserve(
  data,
  duration = 0,
  rate,
  premium = NULL,
  premium_col = "premium",
  benefit = 1,
  payment = 1,
  payments_per_year = 1,
 insurance = c("whole_life", "term", "deferred", "deferred_term", "pure_endowment",
    "endowment"),
 annuity = c("whole_life", "temporary", "deferred", "deferred_temporary", "certain",
    "guaranteed"),
  term = NULL,
  deferral_years = 0,
  guarantee_years = NULL,
  payment_timing = c("end_of_year", "moment_of_death"),
  premium_timing = c("due", "immediate"),
  reserve_timing = c("before_payment", "after_payment"),
  interest_type = c("effective", "nominal", "force"),
  m = 1,
  k_col = "Kx",
  tx_col = "Tx",
  in_force_basis = c("auto", "complete", "curtate"),
  not_in_force = c("na", "zero"),
  reserve_col = "reserve_loss"
)
```

Arguments

`data`	A data frame or tibble containing simulated future lifetimes, typically returned by `simulate_lifetime()` or `mc_multilife_status()`.
`duration`	Numeric vector. Policy duration or durations at which the reserve is evaluated. Default is `0`.
`rate`	Numeric scalar. Interest rate used for discounting.
`premium`	Optional numeric scalar. Premium used in the reserve loss. If `NULL`, the function first tries to use `premium_col` from `data`. If `premium_col` is not found, a Monte Carlo net premium at issue is estimated internally as $\bar{Z}_0/\bar{Y}_0$.
`premium_col`	Character string. Name of the premium column in `data`. Default is `"premium"`.
`benefit`	Numeric scalar. Benefit amount payable under the insurance. Default is `1`.
`payment`	Numeric scalar. Amount of each premium annuity payment. Default is `1`.

payments_per_year
: Positive integer-like scalar. Number of premium payments per year. Default is 1, corresponding to annual premiums. Use 12 for monthly premiums, 4 for quarterly premiums, and 2 for semiannual premiums.

insurance
: Character string specifying the insurance type. Available options are "whole_life", "term", "deferred", "deferred_term", "pure_endowment", and "endowment".

annuity
: Character string specifying the premium annuity type. Available options are "whole_life", "temporary", "deferred", "deferred_temporary", "certain", and "guaranteed".

term
: Numeric scalar. Contract term in years. Required for insurance types "term", "deferred_term", "pure_endowment", and "endowment", and for annuity types "temporary", "deferred_temporary", and "certain".

deferral_years
: Numeric scalar. Deferral period in years. Default is 0.

guarantee_years
: Numeric scalar. Guaranteed payment period in years. Required when annuity = "guaranteed".

payment_timing
: Character string specifying when death benefits are paid. Available options are "end_of_year" and "moment_of_death". Default is "end_of_year".

premium_timing
: Character string specifying the premium payment timing. Available options are "due" and "immediate". Default is "due".

reserve_timing
: Character string specifying whether payments due exactly at the valuation duration are included. Available options are "before_payment" and "after_payment". Default is "before_payment".

interest_type
: Character string specifying the interest rate convention. Available options are "effective", "nominal", and "force". Default is "effective".

m
: Numeric scalar. Number of interest conversion periods per year when interest_type = "nominal". Default is 1. This argument controls the interest-rate conversion frequency only. It does not represent premium payment frequency.

k_col
: Character string. Name of the column containing simulated curtate future lifetimes. Default is "Kx".

tx_col
: Character string. Name of the column containing simulated complete future lifetimes. Default is "Tx".

in_force_basis
: Character string specifying how the in-force indicator is evaluated. Available options are "auto", "complete", and "curtate". Default is "auto".

not_in_force
: Character string specifying what to return for scenarios that are not in force at the valuation duration. Available options are "na" and "zero". Default is "na".

reserve_col
: Character string. Name of the output reserve loss column. Default is "reserve_loss".

Details

This function recalculates future benefit and future premium present values from each valuation duration. It is not a wrapper around mc_loss(), because reserves require valuing only the cash flows that remain after the valuation time.

For a policy in force at duration t, the simulated prospective loss is

$$L_t = Z_t - PY_t,$$

where Z_t is the present value at duration t of future benefits, Y_t is the present value at duration t of future premium payments, and P is the premium.

The arguments `m` and `payments_per_year` have different meanings:

- `m` is used only when `interest_type = "nominal"` and controls the frequency of interest conversion.
- `payments_per_year` controls how frequently future premium payments are made.

The argument `payment` represents the amount of each premium payment. Thus, for monthly premiums with total annual premium equal to 1, use `payment = 1 / 12` and `payments_per_year = 12`.

If `payments_per_year = 1`, future premiums are annual. If `payments_per_year > 1`, future premiums are made at fractional times, and a valid complete future lifetime column supplied through `tx_col` is required.

Durations may be integer or fractional. Fractional reserve durations require complete future lifetimes. For example, monthly reserve calculations may use `duration = seq(0, 20, by = 1 / 12)`.

If `reserve_timing = "before_payment"`, cash flows occurring exactly at the valuation duration are included. If `reserve_timing = "after_payment"`, cash flows occurring exactly at the valuation duration are excluded.

If `not_in_force = "na"`, scenarios that are not in force at a given duration receive `NA` values for future present values and reserve losses. This is useful for estimating reserves conditional on the policy still being in force. If `not_in_force = "zero"`, those scenarios receive zero values, which may be useful for portfolio run-off summaries.

This function computes prospective reserves under the simulated model. It does not include expenses, surrender values, taxes, profit loadings, or statutory reserving adjustments.

Value

A tibble with one row per original simulation and per requested duration. It contains the original simulation columns and additional columns:

- `duration`: valuation duration.
- `in_force`: logical indicator for survival or in-force status.
- `reserve_timing`: timing convention used at valuation.
- `not_in_force`: convention used for scenarios not in force.
- `rate`: original rate supplied.
- `interest_type`: interest rate convention.
- `m`: interest conversion frequency.
- `effective_rate`: equivalent annual effective interest rate.
- `discount_factor`: annual discount factor.
- `insurance`: insurance type.
- `annuity`: premium annuity type.

- `benefit`: benefit amount.
- `payment`: amount of each premium payment.
- `payments_per_year`: premium payment frequency.
- `future_pv_benefit`: present value at duration of future benefits.
- `future_pv_premiums`: present value at duration of future premium payments.
- `premium`: premium used in the simulated reserve loss.
- `reserve_loss`: simulated reserve loss, or another name supplied through `reserve_col`.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*. Second Edition. Society of Actuaries.

See Also

`simulate_lifetime()`, `simulate_lifetimes()`, `mc_multilife_status()`, `mc_insurance()`, `mc_annuity()`, `mc_premium()`, `mc_loss()`, `summary_mc()`

Examples

```
life_table <- tibble::tibble(
  age = 40:100,
  qx = seq(0.002, 1, length.out = 61)
)

# Annual prospective reserves for whole life insurance
life_table |>
  simulate_lifetime(age = 40, n_sim = 1000, seed = 123) |>
  mc_reserve(
    duration = c(0, 5, 10, 20),
    rate = 0.05,
    insurance = "whole_life",
    annuity = "whole_life",
    benefit = 1,
    payment = 1,
    payments_per_year = 1,
    premium_timing = "due"
  )

# Monthly premium reserve curve
life_table |>
  simulate_lifetime(
    age = 40,
    n_sim = 1000,
    fractional = "udd",
    seed = 123
  ) |>
  mc_reserve(
    duration = seq(0, 10, by = 1),
    rate = 0.05,
```

```
    insurance = "whole_life",
    annuity = "whole_life",
    benefit = 1,
    payment = 1 / 12,
    payments_per_year = 12,
    premium_timing = "due"
  )

# Monthly reserves by fractional policy duration
life_table |>
  simulate_lifetime(
    age = 40,
    n_sim = 1000,
    fractional = "udd",
    seed = 123
  ) |>
  mc_reserve(
    duration = seq(0, 5, by = 1 / 12),
    rate = 0.05,
    insurance = "whole_life",
    annuity = "whole_life",
    benefit = 1,
    payment = 1 / 12,
    payments_per_year = 12,
    premium_timing = "due"
  ) |>
  summary_mc(value_col = "reserve_loss", by = "duration")

# Joint-life reserve using a multiple-life status
life_table |>
  simulate_lifetimes(
    ages = c(60, 58),
    n_sim = 1000,
    fractional = "udd",
    seed = 123
  ) |>
  mc_multilife_status(status = "joint_life") |>
  mc_reserve(
    duration = c(0, 5, 10),
    rate = 0.04,
    insurance = "whole_life",
    annuity = "whole_life",
    benefit = 1,
    payment = 1,
    payments_per_year = 1,
    k_col = "K_status",
    tx_col = "T_status"
  )
```

---

md_table *Multiple decrement table (annual, discrete ages)*

Description

Builds a multiple decrement table from cause-specific annual decrement probabilities $q_x^{(j)}$. This function is annual/discrete: ages must be integer-valued and the input probabilities are interpreted as one-year decrement probabilities for each cause.

Usage

```
md_table(
  qx_df,
  age_col = "x",
  cause_cols = NULL,
  radix = 1e+05,
  close = TRUE,
  check = TRUE,
  tol = 1e-10
)
```

Arguments

qx_df	A data.frame/tibble with an age column (default x) and one or more cause columns containing annual probabilities in $[0, 1]$. Recommended naming convention: cause columns start with "q_" (e.g., q_death, q_disability).
age_col	Character. Name of the age column (default "x").
cause_cols	Character vector. Names of the cause columns. If NULL (default), all columns other than age_col are treated as causes.
radix	Numeric. Starting cohort size at the first age (default 1e5).
close	Logical. If TRUE, requires $q_\omega^{(\tau)} = 1$ at the last age (default TRUE).
check	Logical. If TRUE, performs input validation (default TRUE).
tol	Numeric tolerance used in checks (default 1e-10).

Details

Let the cause columns be $q_x^{(1)}, \ldots, q_x^{(J)}$. The total decrement probability is $q_x^{(\tau)} = \sum_j q_x^{(j)}$ and the total survival probability is $p_x^{(\tau)} = 1 - q_x^{(\tau)}$. The cohort is generated recursively by $\ell_{x+1} = \ell_x \, p_x^{(\tau)}$ with starting radix $\ell_{x_0} =$ radix.

If close = TRUE, the last age (omega) must satisfy $q_\omega^{(\tau)} = 1$ (within tolerance), so that the table closes naturally.

Value

A tibble with columns:

- x: integer ages.
- lx: cohort ℓ_x.
- q_total: total decrement probability $q_x^{(\tau)}$.

- `p_total`: total survival probability $p_x^{(\tau)}$.
- `d_total`: total decrements $d_x^{(\tau)} = \ell_x q_x^{(\tau)}$.
- cause columns (as provided).
- cause-specific decrements `d_*` with $d_x^{(j)} = \ell_x q_x^{(j)}$.

Examples

```
qx_df <- tibble::tibble(
  x = 30:35,
  q_death = c(0.001, 0.0012, 0.0014, 0.0017, 0.0020, 1.0000),
  q_disability = c(0.002, 0.0021, 0.0022, 0.0023, 0.0024, 0.0000)
)
md <- md_table(qx_df, radix = 1e5, close = TRUE)
md
```

---

mortality_colombia_tables

Colombian mortality tables

Description

A tidy collection of Colombian mortality tables for life-contingency examples. The dataset includes regulatory and pedagogical mortality tables used in Colombian actuarial applications.

Usage

```
mortality_colombia_tables
```

Format

A tibble with 12 variables:

table Mortality table identifier.

sex Sex category, typically `"male"` or `"female"`.

x Integer age.

lx Number of survivors at exact age `x`.

dx Expected number of deaths between ages `x` and `x + 1`.

qx One-year death probability between ages `x` and `x + 1`.

px One-year survival probability between ages `x` and `x + 1`.

mu Force of mortality, if available.

ex Complete life expectancy at age `x`, if available.

source Source identifier or reference.

qx_calculated Death probability recalculated from `lx` and `dx`, when available.

qx_difference Difference between reported `qx` and recalculated `qx`, when available.

Details

The dataset is intended for actuarial examples involving Colombian mortality tables, survival probabilities, life annuities, life insurance present values, and validation of life-table calculations.

The variables `qx_calculated` and `qx_difference` are included as validation aids. They allow users to compare reported death probabilities against probabilities reconstructed from `lx` and `dx`.

Some tables may start at different initial ages, use different terminal ages, or use different radix values. Users should filter the desired table and sex before passing the data to life-contingency functions.

Source

Colombian mortality tables cleaned for tidyactuarial examples. Source identifiers are provided in the `source` column.

Examples

```
data(mortality_colombia_tables)

head(mortality_colombia_tables)

mortality_colombia_tables |>
  dplyr::count(table, sex)

rv08_male <- mortality_colombia_tables |>
  dplyr::filter(table == "RV08_Rentistas_2005_2008", sex == "male") |>
  dplyr::select(x, lx, dx, qx, px, ex)

head(rv08_male)
```

---

mortality_law_table *Generate a tidy life table from a theoretical mortality law*

Description

Creates a tidy life table with one row per integer age from a parametric mortality law. The output follows tidyactuarial conventions and includes columns such as `x`, `qx`, `px`, `lx`, `dx`, and `mx`.

Usage

```
mortality_law_table(
  law = c("Exponential", "Gompertz", "Makeham", "Weibull", "Logistic", "DeMoivre",
    "Beta", "HeligmanPollard"),
  x_min,
  x_max,
  ...,
  params = NULL,
```

```
  frac = c("CF", "UDD", "Balducci"),
  radix = 1e+05,
  close = TRUE,
  ax = 0.5,
  check = TRUE,
  tol = 1e-10
)
```

Arguments

law	Character. Mortality law. One of "Exponential", "Gompertz", "Makeham", "Weibull", "Logistic", "DeMoivre", "Beta", or "HeligmanPollard".
x_min	Integer. Minimum age, inclusive.
x_max	Integer. Maximum age, inclusive. Must satisfy x_min < x_max.
...	Named law parameters. These values override params. Placing ... before arguments such as close and check avoids partial-matching conflicts with the Gompertz/Makeham parameter c.
params	Named list of law parameters, or NULL. Direct parameters supplied through ... override values in params.
frac	Character. Within-year assumption used to convert μ_x to q_x. One of "CF", "UDD", or "Balducci".
radix	Numeric. Starting cohort size at age x_min.
close	Logical. If TRUE, forces the last age to close.
ax	Numeric. Average fraction of the year lived by those dying.
check	Logical. If TRUE, performs strict input validation.
tol	Numeric. Tolerance used in checks.

Value

A tibble with columns x, law, frac, mu_x, qx, px, lx, dx, Lx, Tx, ex, and mx.

Supported laws

- **Exponential**: $\mu_x = \lambda$
- **Gompertz**: $\mu_x = Bc^x$
- **Makeham**: $\mu_x = A + Bc^x$
- **Weibull**: $\mu_x = (k/\lambda)(x/\lambda)^{k-1}$
- **Logistic**: $\mu_x = (A + Bc^x)/(1 + Cc^x)$
- **DeMoivre**: finite lifetime law with $q_x = 1/(\omega - x)$ for $x < \omega$
- **Beta**: scaled lifetime model $X/\omega \sim Beta(\alpha, \beta)$
- **HeligmanPollard**: odds model returning q_x directly

Converting mu(x) to qx

For laws defined by a force of mortality μ_x, the one-year death probability q_x is obtained using `frac`:

- `"CF"`: $q_x = 1 - \exp(-\mu_x)$
- `"UDD"`: $q_x = \mu_x$
- `"Balducci"`: $q_x = \mu_x/(1 + \mu_x)$

Direct qx laws

`"DeMoivre"`, `"Beta"`, and `"HeligmanPollard"` define `qx` directly. For these laws, `frac` is not used to derive `qx`.

Closure

If `close = TRUE`, the last age is forced to close the table by setting `qx[x_max] = 1` and `px[x_max] = 0`.

Examples

```
mortality_law_table("Exponential", 0, 110, lambda = 0.01)

mortality_law_table("Gompertz", 0, 110, B = 1e-5, c = 1.08)

mortality_law_table("Makeham", 0, 110, A = 5e-4, B = 1e-6, c = 1.10)

mortality_law_table("Weibull", 1, 110, k = 2.5, lambda = 90)

mortality_law_table("Logistic", 0, 110, A = 1e-4, B = 1e-6, c = 1.10, C = 1e-3)

mortality_law_table("DeMoivre", 0, 100, omega = 100)

mortality_law_table("Beta", 0, 100, alpha = 2, beta = 5, omega = 101)

mortality_law_table(
  "HeligmanPollard", 1, 110,
  A = 0.0002, B = 0.1, C = 0.03, D = 10, E = 20, F = 0.00005, G = 1.08
)
```

---

mortality_world_sample_2015_2023

World mortality sample panel, 2015–2023

Description

A compact international panel of period life tables for selected countries from 2015 to 2023. The dataset is intended for comparative mortality examples, especially before, during, and after the COVID-19 pandemic period.

Usage

```
mortality_world_sample_2015_2023
```

Format

A tibble with 35,451 rows and 14 variables:

country Country name.

country_code Numeric ISO country code.

continent Continent.

region Geographic region.

year Calendar year, from 2015 to 2023.

pandemic_period Period label: `"pre_pandemic"`, `"pre_pandemic_reference"`, `"pandemic"`, `"transition"`, or `"post_pandemic"`.

sex Sex category: `"male"`, `"female"`, or `"both"`.

age Integer age, from 0 to 100.

mx Central death rate at age `x`.

qx One-year death probability between ages `x` and `x + 1`.

px One-year survival probability between ages `x` and `x + 1`.

lx Number of survivors at exact age `x`, based on radix 100,000.

dx Expected number of deaths between ages `x` and `x + 1`.

source Data source.

Details

The dataset is derived from central death rates `mx`. Death probabilities `qx` were computed using the annual approximation

$$q_x = \frac{m_x}{1 + (1 - a_x)m_x},$$

with $a_x = 0.5$. The last available age is closed by setting `qx = 1`. Survivors `lx` and expected deaths `dx` are then reconstructed recursively from a radix of 100,000.

This dataset is a selected-country panel, not a complete world mortality database.

Source

United Nations, World Population Prospects 2024.

Examples

```
data(mortality_world_sample_2015_2023)

mortality_world_sample_2015_2023 |>
  dplyr::filter(country == "Colombia", sex == "both", age == 70) |>
  dplyr::select(year, pandemic_period, qx, lx)
```

`mortality_world_sample_2023`
World mortality sample, 2023

Description

A compact international sample of period life tables for selected countries in 2023. The dataset is intended for recent and simple examples involving central death rates, one-year death probabilities, survival probabilities, and life-table calculations.

Usage

```
mortality_world_sample_2023
```

Format

A tibble with 3,939 rows and 13 variables:

country Country name.

country_code Numeric ISO country code.

continent Continent.

region Geographic region.

year Calendar year.

sex Sex category: `"male"`, `"female"`, or `"both"`.

age Integer age, from 0 to 100.

mx Central death rate at age `x`.

qx One-year death probability between ages `x` and `x + 1`.

px One-year survival probability between ages `x` and `x + 1`.

lx Number of survivors at exact age `x`, based on radix 100,000.

dx Expected number of deaths between ages `x` and `x + 1`.

source Data source.

Details

The dataset is derived from central death rates `mx`. Death probabilities `qx` were computed using the annual approximation

$$q_x = \frac{m_x}{1 + (1 - a_x)m_x},$$

with $a_x = 0.5$. The last available age is closed by setting `qx = 1`. Survivors `lx` and expected deaths `dx` are then reconstructed recursively from a radix of 100,000.

This dataset is a selected-country sample, not a complete world mortality database.

Source

United Nations, World Population Prospects 2024.

Examples

```
data(mortality_world_sample_2023)

mortality_world_sample_2023 |>
  dplyr::filter(country == "Colombia", sex == "both") |>
  dplyr::select(age, mx, qx, px, lx, dx)
```

---

multiple_decrement_sample

Sample multiple decrement probabilities

Description

A small pedagogical annual multiple decrement dataset with three causes: death, disability, and withdrawal. It is intended for examples involving multiple decrement tables, total-decrement life tables, cause-specific decrement probabilities, and cause-specific insurance benefits.

A small pedagogical annual multiple decrement dataset with three causes: death, disability, and withdrawal. It is intended for examples involving multiple decrement tables, total-decrement life tables, cause-specific decrement probabilities, and cause-specific insurance benefits.

Usage

```
multiple_decrement_sample

multiple_decrement_sample
```

Format

A tibble with 7 rows and 6 variables:

x Integer age.

q_death One-year death decrement probability.

q_disability One-year disability decrement probability.

q_withdrawal One-year withdrawal decrement probability.

q_total Total one-year decrement probability.

p_total Total one-year survival probability.

A tibble with 7 rows and 6 variables:

x Integer age.

q_death One-year death decrement probability.

q_disability One-year disability decrement probability.

q_withdrawal One-year withdrawal decrement probability.

q_total Total one-year decrement probability.

p_total Total one-year survival probability.

Source

Synthetic pedagogical data created for tidyactuarial examples.

Synthetic pedagogical data created for tidyactuarial examples.

Examples

```
data(multiple_decrement_sample)

md <- md_table(
  qx_df = multiple_decrement_sample |>
    dplyr::select(x, q_death, q_disability, q_withdrawal),
  radix = 100000,
  close = FALSE
)

md
```

```
data(multiple_decrement_sample)

md <- md_table(
  qx_df = multiple_decrement_sample |>
    dplyr::select(x, q_death, q_disability, q_withdrawal),
  radix = 100000,
  close = FALSE
)

md
```

---

plot_cash_flow *Plot a cash-flow diagram*

Description

Creates a professional cash-flow diagram with arrows representing inflows and outflows over time.

Usage

```
plot_cash_flow(
  .data = NULL,
  payment,
  time = NULL,
```

```
  date = NULL,
  rate = NULL,
  i = NULL,
  pv = NULL,
  title = NULL,
  subtitle = NULL,
  x_label = NULL,
  amount_label = "Cash flow",
  financial = TRUE,
  normalize = FALSE,
  aggregate = TRUE,
  show_labels = TRUE,
  label_size = 3.5,
  arrow_size = 0.8,
  timeline_size = 0.7,
  label_digits = 2L,
  currency = "",
  col_inflow = "#1B9E77",
  col_outflow = "#D95F02",
  date_labels = "%Y-%m-%d",
  day_count = c("act/365", "act/360")
)
```

Arguments

.data	Optional data.frame or tibble containing cash-flow columns. If supplied, payment, time, and date can be passed as unquoted column names or as strings.
payment	Numeric vector of cash flows, or a column name when .data is supplied.
time	Optional numeric vector of times, or a column name when .data is supplied.
date	Optional date vector, or a column name when .data is supplied. If supplied, the earliest date is treated as time 0 for present-value calculations.
rate	Optional annual effective interest rate used to compute present value if pv is NULL.
i	Deprecated alias for rate. Use rate in new code.
pv	Optional numeric present value to display.
title	Optional plot title.
subtitle	Optional plot subtitle. If NULL, a subtitle is built from rate and pv when available.
x_label	Optional x-axis label.
amount_label	Optional y-axis label when normalize = FALSE.
financial	Logical. If TRUE, positive payments point upward and negative payments point downward. If FALSE, all arrows point upward with height proportional to absolute value.
normalize	Logical. If TRUE, arrow heights are normalized by the largest absolute payment. If FALSE, the vertical axis uses real payment magnitudes.

aggregate	Logical. If TRUE, cash flows occurring at the same time or date are summed before plotting.
show_labels	Logical. If TRUE, labels are shown next to cash-flow arrows.
label_size	Numeric text size for labels.
arrow_size	Numeric line width for arrows.
timeline_size	Numeric line width for the time axis.
label_digits	Integer number of decimal digits for payment labels.
currency	Optional currency or unit prefix, such as "$".
col_inflow	Character color for inflow arrows.
col_outflow	Character color for outflow arrows.
date_labels	Character date-label format passed to ggplot2::scale_x_date().
day_count	Day-count convention used when date is supplied. One of "act/365" or "act/360".

Details

The function supports both classical vector input and tidyverse-style data input through the pipe. It also supports either numeric times or calendar dates.

By default, the plot uses real cash-flow magnitudes on the vertical axis. This is preferable when the diagram is intended to communicate financial scale. Use normalize = TRUE only when an intentionally schematic diagram is desired.

If an interest rate is supplied and pv is NULL, the present value is computed as

$$PV = \sum_k C_k (1+i)^{-t_k}$$

and displayed in the subtitle.

Value

A ggplot2 object.

See Also

pv_flow(), fv_flow(), irr_flow()

Other time-value: future_value(), fv_flow(), irr_flow(), irr_flow_multi(), present_value(), pv_flow()

Examples

```
plot_cash_flow(
  payment = c(-1000, 300, 400, 500),
  time = c(0, 1, 2, 3),
  rate = 0.08,
  currency = "$"
)

cashflows <- tibble::tibble(
```

```
  time = c(0, 1, 2, 3),
  payment = c(-1000, 300, 400, 500)
)

cashflows |>
  plot_cash_flow(payment = payment, time = time, rate = 0.08)

dated_flows <- tibble::tibble(
  date = as.Date(c("2026-01-01", "2026-07-01", "2027-01-01")),
  payment = c(-1000, 450, 700)
)

dated_flows |>
  plot_cash_flow(payment = payment, date = date, rate = 0.08)
```

`plot_immunization_gap` *Plot immunization performance under interest rate shifts*

Description

Computes and plots the difference between the present value of liabilities and the present value of an immunized asset portfolio under small interest rate changes. This allows visual evaluation of duration or duration-convexity immunization quality.

Usage

```
plot_immunization_gap(
  liabilities,
  t_liabilities,
  asset_cashflows,
  weights,
  i0,
  delta = 0.01,
  n_grid = 200L
)
```

Arguments

`liabilities`	Numeric vector of liability payments.
`t_liabilities`	Numeric vector of times (periods) of each liability payment.
`asset_cashflows`	A list where each element is a list with components `$payment` and `$time`, defining each asset's cash flow.
`weights`	Numeric vector of portfolio weights (amount invested in each asset). Must have same length as `asset_cashflows`.
`i0`	Base effective interest rate per period.
`delta`	A numeric value defining the range of rates: from `i0 - delta` to `i0 + delta`.
`n_grid`	Number of rate values to evaluate.

Details

Let $v(i) = 1/(1+i)$. For a liability stream L_k at time t_k:

$$PV_L(i) = \sum_k L_k \, v(i)^{t_k}$$

For a portfolio of assets with weights w_j:

$$PV_A(i) = \sum_j w_j \, PV_j(i)$$

The curve $\Delta(i) = PV_A(i) - PV_L(i)$ illustrates immunization robustness. Under perfect duration immunization, this curve is tangent to zero at $i = i_0$ and non-negative nearby if the convexity condition is also met.

Value

A ggplot2 object showing the PV difference curve $PV_A(i) - PV_L(i)$ and a zero reference line.

See Also

immunize_duration, immunize_duration_convexity, bond_duration, bond_convexity

Other immunization: immunize_duration(), immunize_duration_convexity()

Examples

```
# Two-asset duration immunization gap
plot_immunization_gap(
  liabilities = c(5000, 8000),
  t_liabilities = c(3, 7),
  asset_cashflows = list(
    list(payment = c(0, 0, 100), time = c(1, 2, 3)),
    list(payment = c(0, 0, 0, 0, 0, 0, 200), time = 1:7)
  ),
  weights = c(5, 2.5),
  i0 = 0.05,
  delta = 0.02
)
```

---

plot_km *Plot a Kaplan–Meier survival curve*

Description

Creates a step-function plot of the Kaplan–Meier survival estimate $\hat{S}(t)$ with optional pointwise confidence bands. Designed to work directly with the output of km_lifetable.

Usage

```
plot_km(
  km,
  time_col = "time",
  surv_col = "S",
  lower_col = "ci_low",
  upper_col = "ci_high",
  conf_int = TRUE,
  title = NULL
)
```

Arguments

km	A data frame or tibble with at least columns for time and survival. Can also be the full list returned by km_lifetable, in which case the $km component is extracted automatically.
time_col	Character. Name of the time column. Default "time".
surv_col	Character. Name of the survival column. Default "S" (matching km_lifetable output).
lower_col	Character. Name of the lower CI column. Default "ci_low" (matching km_lifetable output).
upper_col	Character. Name of the upper CI column. Default "ci_high" (matching km_lifetable output).
conf_int	Logical. If TRUE (default) and CI columns exist in km, plot a step-wise confidence ribbon.
title	Optional character string for the plot title.

Details

Both the survival curve and the confidence band are rendered as step functions (using geom_step), which is the correct representation for the KM estimator - a right-continuous step function that drops at each observed event time.

The confidence band uses geom_stepribbon logic: the data is internally expanded so that a ribbon-fill follows the step pattern rather than interpolating linearly between event times.

Value

A ggplot object that can be further customised with additional ggplot2 layers.

See Also

km_lifetable for fitting the KM estimator and building the empirical life table.

Examples

```
set.seed(42)
n <- 150
time   <- rexp(n, rate = 0.05)
status <- rbinom(n, 1, prob = 0.7)

# Fit KM and plot directly
out <- km_lifetable(time, status, breaks = 0:30)

# Pass the full list - $km is extracted automatically
plot_km(out)

# Or pass just the km tibble
plot_km(out$km)

# Without confidence band
plot_km(out, conf_int = FALSE, title = "KM Survival Curve")

# Customise with ggplot2 layers
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot_km(out) +
    ggplot2::geom_hline(yintercept = 0.5, linetype = "dashed") +
    ggplot2::labs(subtitle = "Dashed line = median survival")
}
```

---

portfolio_convexity *Compute portfolio convexity as a market-value-weighted average*

Description

Computes portfolio convexity from individual position convexities using market values as weights.

Usage

```
portfolio_convexity(
  .data = NULL,
  portfolio_id = NULL,
  market_value = NULL,
  convexity = NULL,
  col_portfolio = "portfolio_id",
  col_market_value = "market_value",
  col_convexity = "convexity",
  .out = "portfolio_convexity",
  .out_value = "portfolio_market_value",
  .out_n = "n_positions",
  .na = c("propagate", "error", "drop")
)
```

Arguments

`.data`	A data.frame or tibble. If `NULL`, `market_value` and `convexity` must be supplied as vectors.
`portfolio_id`	Optional vector of portfolio identifiers when `.data = NULL`. If omitted, all positions are treated as belonging to a single portfolio.
`market_value`	Numeric vector of market values when `.data = NULL`.
`convexity`	Numeric vector of individual convexities when `.data = NULL`.
`col_portfolio`	Name of the portfolio identifier column. If `NULL`, all rows are treated as one portfolio.
`col_market_value`	Name of the numeric column containing market values.
`col_convexity`	Name of the numeric column containing individual convexities.
`.out`	Name of the output column containing portfolio convexity.
`.out_value`	Name of the output column containing total portfolio market value.
`.out_n`	Name of the output column containing the number of positions used in the calculation.
`.na`	NA handling policy: `"propagate"`, `"error"`, or `"drop"`.

Details

This is a summarise-style tibble-first function. Each input row represents one position, and each output row represents one portfolio.

The function does not compute individual convexities from bond terms or yields. Instead, it assumes that the input convexity column already contains valid convexity measures on a common basis within each portfolio.

The portfolio convexity is computed as:

$$C_P = \frac{\sum_{k=1}^{n} P_k C_k}{\sum_{k=1}^{n} P_k}$$

where P_k is the market value of position k and C_k is its convexity.

Value

A tibble with one row per portfolio and columns for portfolio convexity, total market value, and number of positions used.

References

Marcel B. Finan, *A Basic Course in the Theory of Interest and Derivatives Markets: A Preparation for the Actuarial Exam FM/2*, Section 55: Redington Immunization and Convexity.

Kellison, S. G. *The Theory of Interest*, Chapter 11: Duration, Convexity and Immunization.

See Also

portfolio_duration, bond_convexity, bond_duration

Other bonds: bond_book_value(), bond_callable_price(), bond_convexity(), bond_duration(), bond_price(), bond_ytm(), portfolio_duration()

Examples

```
# Simple example: one portfolio
portfolio_convexity(
  market_value = c(1000, 2000, 500),
  convexity = c(20, 12, 35)
)

# Medium example: two portfolios
positions <- tibble::tibble(
  portfolio_id = c("A", "A", "B", "B"),
  market_value = c(1000, 2000, 1000 / 1.08^2, 1000 / 1.08^4),
  convexity = c(20, 12, 6, 18)
)

portfolio_convexity(
  positions,
  col_portfolio = "portfolio_id",
  col_market_value = "market_value",
  col_convexity = "convexity"
)
```

---

portfolio_duration *Compute portfolio duration as a market-value-weighted average*

Description

Computes portfolio duration from individual position durations using market values as weights.

Usage

```
portfolio_duration(
  .data = NULL,
  portfolio_id = NULL,
  market_value = NULL,
  duration = NULL,
  col_portfolio = "portfolio_id",
  col_market_value = "market_value",
  col_duration = "duration",
  .out = "portfolio_duration",
  .out_value = "portfolio_market_value",
  .out_n = "n_positions",
```

```
  .na = c("propagate", "error", "drop")
)
```

Arguments

`.data`	A data.frame or tibble. If `NULL`, `market_value` and `duration` must be supplied as vectors.
`portfolio_id`	Optional vector of portfolio identifiers when `.data = NULL`. If omitted, all positions are treated as belonging to a single portfolio.
`market_value`	Numeric vector of market values when `.data = NULL`.
`duration`	Numeric vector of individual durations when `.data = NULL`.
`col_portfolio`	Name of the portfolio identifier column. If `NULL`, all rows are treated as one portfolio.
`col_market_value`	Name of the numeric column containing market values.
`col_duration`	Name of the numeric column containing individual durations.
`.out`	Name of the output column containing portfolio duration.
`.out_value`	Name of the output column containing total portfolio market value.
`.out_n`	Name of the output column containing the number of positions used in the calculation.
`.na`	NA handling policy: `"propagate"`, `"error"`, or `"drop"`.

Details

This is a summarise-style tibble-first function. Each input row represents one position, and each output row represents one portfolio.

The function does not compute individual durations from bond terms or yields. Instead, it assumes that the input duration column already contains valid duration measures on a common basis within each portfolio.

The portfolio duration is computed as:

$$D_P = \frac{\sum_{k=1}^{n} P_k D_k}{\sum_{k=1}^{n} P_k}$$

where P_k is the market value of position k and D_k is its duration.

Value

A tibble with one row per portfolio and columns for portfolio duration, total market value, and number of positions used.

References

Marcel B. Finan, *A Basic Course in the Theory of Interest and Derivatives Markets: A Preparation for the Actuarial Exam FM/2*, Section 54: Macaulay and Modified Durations.

Kellison, S. G. *The Theory of Interest*, Chapter 11: Duration, Convexity and Immunization.

See Also

portfolio_convexity, bond_duration, bond_convexity

Other bonds: bond_book_value(), bond_callable_price(), bond_convexity(), bond_duration(), bond_price(), bond_ytm(), portfolio_convexity()

Examples

```
# Simple example: one portfolio
portfolio_duration(
  market_value = c(1000, 2000, 500),
  duration = c(7, 5, 10)
)

# Medium example: two portfolios
positions <- tibble::tibble(
  portfolio_id = c("A", "A", "B", "B"),
  market_value = c(1000, 2000, 1000 / 1.08^2, 1000 / 1.08^4),
  duration = c(7, 5, 2, 4)
)

portfolio_duration(
  positions,
  col_portfolio = "portfolio_id",
  col_market_value = "market_value",
  col_duration = "duration"
)
```

---

premium_gross *Gross (expense-loaded) premium from net premium*

Description

Adjusts a net premium using a simple expense structure (α, β, γ) to obtain the gross or commercial premium through the extended equivalence principle.

Usage

```
premium_gross(
  prem,
  alpha = 0,
  beta = 0,
  gamma = 0,
  output = c("value", "table")
)
```

Arguments

prem	A one-row data frame or tibble containing at least: • premium: net premium per payment. • apv_premiums: APV of the premium annuity.
alpha	Numeric scalar greater than or equal to 0. Initial acquisition expense as a multiple of one gross premium payment. The initial expense is αG, paid once at issue.
beta	Numeric scalar in $[0, 1)$. Proportional collection expense as a fraction of each gross premium payment.
gamma	Numeric scalar greater than or equal to 0. Fixed maintenance expense per premium payment period, in monetary units.
output	Character string. Use "value" to return a numeric gross premium, or "table" to return a one-row tibble with the expense breakdown.

Details

The function is designed to work with the detailed output of premium_x using output = "table". It can also be used with any one-row tibble containing the columns premium and apv_premiums.

The extended equivalence principle equates the APV of gross premiums with the APV of benefits plus expenses:

$$G\ddot{a} = P_{\text{net}}\ddot{a} + \alpha G + \beta G\ddot{a} + \gamma\ddot{a}.$$

Solving for the gross premium gives:

$$G = \frac{P_{\text{net}} + \gamma}{(1-\beta) - \alpha/\ddot{a}}.$$

In this function, $\ddot{a}$ is supplied through the apv_premiums column of prem.

Value

If output = "value", a numeric gross premium per payment.

If output = "table", a one-row tibble with columns gross_premium, net_premium, alpha, beta, gamma, loading_pct, and apv_premiums.

See Also

premium_x for single-life net premiums, premium_xy for two-life net premiums, annuity_x for building custom expense APVs.

Other life-contingencies: annuity_x(), annuity_xy(), insurance_x(), insurance_xy(), life_contract(), premium_x(), premium_xy(), reserve_x(), reserve_xy(), simulate_annuity_x(), simulate_insurance_x()

Examples

```
lt <- data.frame(
  x  = 60:66,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000, 86000)
)

# Full workflow: net premium -> gross premium
net <- premium_x(
  mortality_table = lt,
  age = 60,
  rate = 0.05,
  insurance_type = "whole",
  benefit = 100000,
  output = "table"
)

premium_gross(net, alpha = 0.5, beta = 0.05, gamma = 50)

# Finan-style expense structure:
# 10% of each premium plus fixed expenses of 275 per payment period
premium_gross(net, alpha = 0, beta = 0.10, gamma = 275)

# Detailed output with expense breakdown
premium_gross(
  net,
  alpha = 0.5,
  beta = 0.05,
  gamma = 50,
  output = "table"
)

# No expenses: gross = net
premium_gross(net)
```

---

premium_x *Net premium for life insurance by the equivalence principle*

Description

Computes the net benefit premium of a life insurance contract using the equivalence principle:

$$P = \frac{\text{APV of benefits}}{\text{APV of premium annuity}}.$$

Usage

```
premium_x(
  mortality_table,
  age,
```

```
  rate,
  rate_type = "effective",
  m = 1L,
  insurance_type = c("whole", "term", "endowment", "variable_k"),
  benefit = 1,
  term_years = Inf,
  deferral_years = 0L,
  payments_per_year = 1L,
  frac = c("UDD", "CF", "CML", "Balducci"),
  premium_timing = c("due", "immediate"),
  premium_start = c("issue", "deferred"),
  premium_term_years = NULL,
  woolhouse = c("none", "first", "second"),
  output = c("value", "table"),
  check = TRUE
)
```

Arguments

`mortality_table`	A life table data frame containing at least columns `x` and `lx`.
`age`	Integer actuarial age at issue.
`rate`	Numeric scalar. Annual interest-rate input.
`rate_type`	Character string indicating the rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Compounding frequency for nominal rates. Ignored for `rate_type = "effective"` and `rate_type = "force"`.
`insurance_type`	Type of insurance contract. One of `"whole"`, `"term"`, `"endowment"`, or `"variable_k"`.
`benefit`	Benefit amount. For standard products, a single nonnegative numeric value. For `insurance_type = "variable_k"`, a numeric vector or a function of time may be supplied and is passed to `insurance_variable_k`.
`term_years`	Term of the insurance contract in years. Use `Inf` for whole-life insurance. Required as a finite integer for term and endowment insurance.
`deferral_years`	Integer deferral period in years.
`payments_per_year`	Positive integer. Number of premium payments per year.
`frac`	Fractional-age assumption used only for `insurance_type = "variable_k"`. One of `"UDD"`, `"CF"`, `"CML"`, or `"Balducci"`.
`premium_timing`	Timing of premium payments. Use `"due"` for payments in advance or `"immediate"` for payments in arrears.
`premium_start`	Start of premium payments. Use `"issue"` for premiums starting at issue, or `"deferred"` for premiums starting after `deferral_years`.
`premium_term_years`	Optional premium-paying term in years, counted from `premium_start`. If `NULL`, it defaults to whole life for whole life insurance and to `term_years` for temporary products. For finite term contracts, premiums must not extend beyond the end of coverage.

woolhouse	Woolhouse order for the premium annuity when payments_per_year > 1. One of "none", "first", or "second".
output	Character string. Use "value" to return a numeric premium, or "table" to return a one-row tibble with details.
check	Logical. If TRUE, performs input validation.

Details

The premium returned corresponds to one premium payment. For example, when payments_per_year = 12, the returned value is the monthly premium.

The benefit premium is the level payment satisfying the equivalence principle: the APV of premiums equals the APV of benefits at issue.

For standard products, the APV of benefits is computed with insurance_x. The APV of the premium annuity is computed with annuity_x, supporting k-thly payments and Woolhouse approximations.

Value

If output = "value", a numeric scalar with the net premium per payment.

If output = "table", a one-row tibble with the main inputs, APV of benefits, APV of premiums, premium per payment, and annualized premium.

See Also

insurance_x, annuity_x, premium_xy, premium_gross

Other life-contingencies: annuity_x(), annuity_xy(), insurance_x(), insurance_xy(), life_contract(), premium_gross(), premium_xy(), reserve_x(), reserve_xy(), simulate_annuity_x(), simulate_insurance_x()

Examples

```
lt <- data.frame(
  x  = 60:66,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000, 86000)
)

# Whole life insurance, annual premium
premium_x(
  mortality_table = lt,
  age = 60,
  rate = 0.05,
  insurance_type = "whole",
  benefit = 100000
)

# Verify manually: P = A / a-double-dot
A <- insurance_x(
  mortality_table = lt,
  age = 60,
  rate = 0.05,
```

```
  insurance_type = "whole",
  benefit = 100000
)

ad <- annuity_x(
  mortality_table = lt,
  age = 60,
  rate = 0.05,
  timing = "due"
)

A / ad

# Five-year term insurance
premium_x(
  mortality_table = lt,
  age = 60,
  rate = 0.05,
  insurance_type = "term",
  term_years = 5,
  benefit = 100000
)

# Table output
premium_x(
  mortality_table = lt,
  age = 60,
  rate = 0.05,
  insurance_type = "term",
  term_years = 5,
  benefit = 100000,
  output = "table"
)

# Monthly premiums paid for a shorter period than the coverage term
premium_x(
  mortality_table = lt,
  age = 60,
  rate = 0.05,
  insurance_type = "term",
  term_years = 5,
  benefit = 100000,
  payments_per_year = 12,
  premium_term_years = 3
)
```

`premium_xy` *Net premium for two-life insurance by the equivalence principle*

Description

Computes the net benefit premium for a two-life insurance contract using the equivalence principle.

Usage

```
premium_xy(
  mortality_table,
  age_x = NULL,
  age_y = NULL,
  rate = NULL,
  rate_type = NULL,
  m = NULL,
  insurance_type = c("whole", "term", "endowment", "pure_endowment"),
  benefit = 1,
  term_years = Inf,
  deferment_years = 0L,
  payments_per_year = 1L,
  frac = c("UDD", "CF", "CML", "Balducci"),
  premium_timing = c("due", "immediate"),
  premium_start = c("issue", "deferred"),
  premium_term_years = NULL,
  cohort = c("first", "last"),
  output = c("value", "table"),
  check = TRUE,
  tol = 1e-10
)
```

Arguments

mortality_table	Either a single life table used for both lives, or a list of two life tables `list(table_x, table_y)`. Each table must contain column `x` and at least one of `lx`, `px`, or `qx`. A `tidyact_life_contract` object created by `life_contract()` is also accepted.
age_x	Integer actuarial age for the first life.
age_y	Integer actuarial age for the second life.
rate	Numeric scalar. Annual interest-rate input.
rate_type	Character string indicating the rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
m	Positive integer. Compounding frequency for nominal rates.
insurance_type	Type of insurance: `"whole"`, `"term"`, `"endowment"`, or `"pure_endowment"`.
benefit	Benefit amount. Must be a single nonnegative number.
term_years	Optional insurance term in years after deferment. Required for `"term"`, `"endowment"`, and `"pure_endowment"`.
deferment_years	Nonnegative integer deferral period in years.

payments_per_year
: Number of premium payments per year.

frac
: Fractional-age assumption used for fractional premium payment times: "UDD", "CF", "CML", or "Balducci".

premium_timing
: Timing of premium payments: "due" for payments in advance or "immediate" for payments in arrears.

premium_start
: Start of premium payments: "issue" for time 0 or "deferred" for time deferment_years.

premium_term_years
: Optional premium-paying term in years, counted from premium_start. If NULL, defaults to the available status horizon for whole-life products and to term_years for finite products.

cohort
: Status definition: "first" for joint-life status or "last" for last-survivor status.

output
: Character string. Use "value" for a numeric premium or "table" for a one-row tibble with details.

check
: Logical. If TRUE, performs input validation.

tol
: Numeric tolerance for integer checks.

Details

The premium returned corresponds to one premium payment. For example, if payments_per_year = 1, it is an annual premium; if payments_per_year = 12, it is a monthly premium.

This function separates:

- the insurance coverage period, controlled by insurance_type, term_years, and deferment_years;
- the premium-paying period, controlled by premium_term_years, premium_start, premium_timing, and payments_per_year.

The function assumes independent future lifetimes.

The supported two-life statuses are:

- cohort = "first": joint-life status. The status survives while both lives survive.
- cohort = "last": last-survivor status. The status survives while at least one life survives.

Let P(t) denote the probability that the selected two-life status survives t years from issue. For integer death benefits paid at the end of the year of status failure, the benefit APV for a term insurance deferred h years and lasting n years is

$$B \sum_{r=h}^{h+n-1} v^{r+1}\{P(r) - P(r+1)\}.$$

For an endowment insurance, the pure endowment benefit B v^(h+n) P(h+n) is added.

Premiums are contingent on the selected two-life status being in force at the premium payment time.

Value

If output = "value", a numeric net premium per payment. If output = "table", a one-row tibble with premium details.

See Also

premium_x(), insurance_xy(), annuity_xy(), reserve_xy()

Other life-contingencies: annuity_x(), annuity_xy(), insurance_x(), insurance_xy(), life_contract(), premium_gross(), premium_x(), reserve_x(), reserve_xy(), simulate_annuity_x(), simulate_insurance_x()

Examples

```
lt <- data.frame(
  x = 60:110,
  lx = seq(100000, 0, length.out = 51)
)

premium_xy(
  mortality_table = lt,
  age_x = 60,
  age_y = 62,
  rate = 0.05,
  insurance_type = "term",
  term_years = 5,
  premium_term_years = 3,
  payments_per_year = 12,
  cohort = "last",
  benefit = 100000,
  output = "table"
)

lt |>
  life_contract(lives = "joint", age_x = 60, age_y = 62, rate = 0.05) |>
  premium_xy(
    insurance_type = "term",
    term_years = 5,
    premium_term_years = 3,
    payments_per_year = 12,
    cohort = "first",
    benefit = 100000
  )
```

---

present_value *Present value of a single payment*

Description

Computes the present value of a future payment due at a given time, using the annual effective interest rate implied by the supplied rate specification.

Usage

```
present_value(
  amount,
  rate,
  rate_type = "effective",
  m = 1,
  time,
  output = c("value", "table")
)
```

Arguments

`amount`	Numeric vector of future payment amounts.
`rate`	Numeric vector of rate values.
`rate_type`	Character vector indicating the rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer vector giving the compounding frequency for nominal rates. Ignored for `"effective"` and `"force"`.
`time`	Numeric vector of times in years until payment.
`output`	Character string. Use `"value"` to return a numeric present value, or `"table"` to return a tibble with intermediate calculations.

Details

The present value is computed as

$$PV = Cv^t = \frac{C}{(1+i)^t}$$

where i is the annual effective interest rate and $v = (1+i)^{-1}$ is the annual discount factor.

The input interest rate may be supplied as:

- annual effective interest rate,
- nominal annual interest rate,
- nominal annual discount rate,
- force of interest.

Internally, all rate specifications are first converted to the equivalent annual effective interest rate using `standardize_interest`.

Input vectors must have length 1 or a common length. Missing values are propagated. This function does not accept dates.

Value

If `output = "value"`, a numeric vector of present values.

If `output = "table"`, a tibble with input values, equivalent rates, discount factors, and present values.

See Also

standardize_interest, future_value

Other time-value: future_value(), fv_flow(), irr_flow(), irr_flow_multi(), plot_cash_flow(), pv_flow()

Examples

```
# Numeric present value
present_value(amount = 1000, rate = 0.08, time = 3)

# Nominal interest converted monthly
present_value(
  amount = 1000,
  rate = 0.12,
  rate_type = "nominal_interest",
  m = 12,
  time = 5
)

# Tibble output for teaching or auditing
present_value(
  amount = 1000,
  rate = 0.08,
  time = 3,
  output = "table"
)

# Vectorized example
present_value(
  amount = c(1000, 2500, 4000),
  rate = c(0.08, 0.10, 0.12),
  rate_type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  time = c(3, 5, 2)
)
```

---

pv_flow *Present value of a general cash flow*

Description

Computes the present value of a cash-flow vector under either:

- a constant interest-rate specification, or
- a term structure of spot rates, one rate per cash flow.

Usage

```
pv_flow(
  payment,
  rate,
  type = "effective",
  m = 1L,
  time = NULL,
  date = NULL,
  day_count = c("act/365", "act/360")
)
```

Arguments

payment	Numeric vector of cash flows.
rate	Numeric scalar or numeric vector of rate values.
type	Character vector indicating the rate type: "effective", "nominal_interest", "nominal_discount", or "force". May have length 1 or the same length as payment.
m	Positive integer vector giving the compounding frequency for nominal rates. May have length 1 or the same length as payment.
time	Optional numeric vector of payment times in years.
date	Optional vector of payment dates. If supplied, the earliest date is treated as time 0.
day_count	Day-count convention used to convert dates to year fractions. One of "act/365" or "act/360".

Details

The cash flow is supplied explicitly through payment. Its timing is supplied either through time (in years) or date (calendar dates). If date is supplied, the earliest date is taken as time 0.

Interest-rate input:

- If rate has length 1, the same rate is used for all payments.
- If rate has the same length as payment, each rate is interpreted as the spot rate associated with the corresponding payment time.

Rate types may be supplied in FM-style notation:

- annual effective rate i,
- nominal annual interest rate $j^{(m)}$,
- nominal annual discount rate $d^{(m)}$,
- force of interest δ.

Internally, all supplied rates are converted to annual effective rates using standardize_interest.

When `rate` is a vector of spot rates, the discounting formula is

$$PV = \sum_{k=1}^{n} \frac{C_k}{(1 + i_k)^{t_k}}$$

where i_k is the annual effective spot rate corresponding to payment k. When a single constant rate is supplied, $i_k = i$ for all k.

Value

Numeric scalar: the present value of the cash flow.

See Also

`fv_flow`, `present_value`, `irr_flow`, `standardize_interest`

Other time-value: `future_value()`, `fv_flow()`, `irr_flow()`, `irr_flow_multi()`, `plot_cash_flow()`, `present_value()`

Examples

```
# Constant annual effective rate
pv_flow(
  payment = c(100, 150, 200),
  rate = 0.08,
  type = "effective",
  time = c(0, 1, 2)
)

# Spot rates, one per payment
pv_flow(
  payment = c(100, 150, 200),
  rate = c(0.05, 0.055, 0.06),
  type = "effective",
  time = c(1, 2, 3)
)

# Using dates; earliest date is taken as t = 0
pv_flow(
  payment = c(100, 150, 200),
  rate = c(0.05, 0.055, 0.06),
  type = "effective",
  date = as.Date(c("2026-01-10", "2027-01-10", "2028-01-10"))
)

# Nominal rates by payment
pv_flow(
  payment = c(100, 100, 100),
  rate = c(0.12, 0.12, 0.12),
  type = "nominal_interest",
  m = c(12, 12, 12),
  time = c(1, 2, 3)
)
```

reserve_x *Benefit reserve schedule for single-life insurance*

Description

Computes terminal benefit reserves at selected policy durations for a fully discrete single-life insurance contract.

Usage

```
reserve_x(
  mortality_table,
  age,
  rate,
  rate_type = "effective",
  m = 1L,
  insurance_type = c("whole", "term", "endowment"),
  term_years = Inf,
  benefit = 1,
  premium = NULL,
  premium_term_years = NULL,
  durations = NULL,
  method = c("prospective", "recursive"),
  output = c("table", "value")
)
```

Arguments

mortality_table	A life table data frame with columns x and lx.
age	Integer actuarial age at issue.
rate	Numeric scalar. Annual interest-rate input.
rate_type	Character string indicating the rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer. Compounding frequency for nominal rates. Ignored for rate_type = "effective" and rate_type = "force".
insurance_type	Character string. One of "whole", "term", or "endowment".
term_years	Insurance term in years. Use Inf for whole-life insurance. For term and endowment insurance, this must be finite.
benefit	Numeric scalar. Insurance benefit amount.
premium	Optional numeric scalar. Premium per annual payment. If NULL, the net premium is computed internally using premium_x.

premium_term_years
: Optional premium-paying term in years. If NULL, premiums are payable for the full contract duration.

durations
: Optional integer vector of policy durations at which to compute reserves. If NULL, reserves are computed for all integer durations from issue to the contract horizon.

method
: Character string. Either "prospective" or "recursive".

output
: Character string. Use "table" to return a reserve schedule, or "value" to return a named numeric vector.

Details

The function supports whole-life, term, and endowment insurance. Reserves may be computed prospectively or recursively. Premiums are assumed payable annually in advance. Limited-payment policies are supported through premium_term_years.

The prospective reserve is computed as

$$ {}_kV = APV_k(\text{future benefits}) - P\,APV_k(\text{future premiums}). $$

The recursive method uses the annual fully discrete recursion

$$ {}_{k+1}V = \frac{({}_kV + P_k)(1+i) - b_{k+1}q_{x+k}}{p_{x+k}}. $$

Value

If output = "table", a tibble with one row per selected duration. If output = "value", a named numeric vector of reserves.

See Also

premium_x, insurance_x, annuity_x, t_px

Other life-contingencies: annuity_x(), annuity_xy(), insurance_x(), insurance_xy(), life_contract(), premium_gross(), premium_x(), premium_xy(), reserve_xy(), simulate_annuity_x(), simulate_insurance_x()

Examples

```
lt <- data.frame(
  x  = 60:70,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000,
         86000, 81000, 75000, 68000, 60000)
)

reserve_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
  insurance_type = "whole"
)
```

```
reserve_x(
  mortality_table = lt,
  age = 60,
  rate = 0.06,
  insurance_type = "endowment",
  term_years = 5,
  benefit = 100000
)
```

`reserve_xy` *Benefit reserve schedule for two-life insurance*

Description

Computes terminal benefit reserves at selected policy durations for a fully discrete two-life insurance contract, assuming independent future lifetimes.

Usage

```
reserve_xy(
  mortality_table,
  age_x = NULL,
  age_y = NULL,
  rate = NULL,
  rate_type = NULL,
  m = NULL,
  insurance_type = c("whole", "term", "endowment"),
  cohort = c("first", "last"),
  term_years = Inf,
  benefit = 1,
  premium = NULL,
  premium_term_years = NULL,
  payments_per_year = 1L,
  premium_timing = c("due", "immediate"),
  at = NULL,
  method = c("prospective", "recursive"),
  output = c("table", "value")
)
```

Arguments

`mortality_table`
: Either a single life table used for both lives, or a list of two life tables `list(table_x, table_y)`. Each table must contain columns `x` and `lx`. A `tidyact_life_contract` object created by `life_contract()` is also accepted.

`age_x`
: Integer actuarial age for the first life at issue.

age_y	Integer actuarial age for the second life at issue.
rate	Numeric scalar. Annual interest-rate input.
rate_type	Character string indicating the rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer. Compounding frequency for nominal rates.
insurance_type	Insurance type. One of "whole", "term", or "endowment".
cohort	Status definition. Use "first" for joint-life status or "last" for last-survivor status.
term_years	Insurance term in years. Required for term and endowment insurance. For whole life insurance, use Inf or omit it.
benefit	Numeric benefit amount.
premium	Net premium per payment. If NULL, it is computed internally using premium_xy().
premium_term_years	Premium-paying term in years. If NULL, premiums are payable for the full contract term.
payments_per_year	Number of premium payments per year. Default is 1.
premium_timing	Timing of premium payments. Use "due" or "immediate". The recursive method currently requires annual due premiums.
at	Integer vector of policy durations at which to compute reserves. If NULL, reserves are computed for all integer durations.
method	Computation method. Use "prospective" or "recursive".
output	Character string. Use "table" for a tibble schedule or "value" for a named numeric vector.

Details

The prospective reserve at duration k is computed as:

$$ {}_kV = APV(\text{future benefits}) - P \cdot APV(\text{future premiums}). $$

Future benefit values are computed with insurance_xy() at shifted ages age_x + k and age_y + k. Future premium values are computed with annuity_xy() on the same two-life status.

The recursive method is implemented for annual due premiums and follows the usual one-year recursion for the selected two-life status.

This function assumes independent future lifetimes.

Value

If output = "table", a tibble with reserve schedule details. If output = "value", a named numeric vector.

See Also

reserve_x(), premium_xy(), insurance_xy(), annuity_xy(), t_pxy()

Other life-contingencies: annuity_x(), annuity_xy(), insurance_x(), insurance_xy(), life_contract(), premium_gross(), premium_x(), premium_xy(), reserve_x(), simulate_annuity_x(), simulate_insurance_x()

Examples

```
lt <- data.frame(
  x  = 60:70,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000,
         86000, 81000, 75000, 68000, 60000)
)

reserve_xy(
  mortality_table = lt,
  age_x = 60,
  age_y = 62,
  rate = 0.06,
  insurance_type = "term",
  cohort = "first",
  term_years = 4
)

reserve_xy(
  mortality_table = lt,
  age_x = 60,
  age_y = 62,
  rate = 0.06,
  insurance_type = "endowment",
  cohort = "last",
  term_years = 5,
  benefit = 100000
)
```

---

simulate_annuity_x *Monte Carlo simulation of a life annuity*

Description

Simulates the present value of a discrete life annuity using a mortality table and annual curtate future lifetimes.

Usage

```
simulate_annuity_x(
  mortality_table,
  age = NULL,
```

```
  rate = NULL,
  rate_type = NULL,
  m = NULL,
  term_years = Inf,
  payments_per_year = 1L,
  timing = c("due", "immediate"),
  payment = 1,
  method = c("inverse"),
  n_sim = 10000L,
  seed = NULL,
  output = c("simulations", "summary")
)
```

Arguments

mortality_table	A life table or a tidyact_life_contract object. A life table must contain columns x and lx.
age	Integer actuarial age. Optional when mortality_table is a tidyact_life_contract.
rate	Numeric scalar. Annual interest-rate input. Optional when mortality_table is a tidyact_life_contract.
rate_type	Character string indicating the rate type.
m	Positive integer. Compounding frequency for nominal rates.
term_years	Term in years. Use Inf for whole life.
payments_per_year	Positive integer. Number of annuity payments per year.
timing	"due" or "immediate".
payment	Numeric scalar. Level payment per payment period.
method	Simulation method. Currently "inverse" is supported.
n_sim	Positive integer. Number of simulations.
seed	Optional seed for reproducibility.
output	"simulations" for the simulated data or "summary" for summary statistics using summary_mc().

Details

The function supports direct use with mortality_table, age, and rate, and pipe workflows with life_contract().

Value

A tibble.

See Also

Other simulation: simulate_insurance_x(), summary_mc()

Other life-contingencies: annuity_x(), annuity_xy(), insurance_x(), insurance_xy(), life_contract(), premium_gross(), premium_x(), premium_xy(), reserve_x(), reserve_xy(), simulate_insurance_x()

---

simulate_insurance_x *Monte Carlo simulation of a life insurance*

Description

Simulates the present value of a discrete life insurance using a mortality table and annual curtate future lifetimes.

Usage

```
simulate_insurance_x(
  mortality_table,
  age = NULL,
  rate = NULL,
  rate_type = NULL,
  m = NULL,
  insurance_type = c("whole", "term", "endowment"),
  term_years = Inf,
  benefit = 1,
  method = c("inverse"),
  n_sim = 10000L,
  seed = NULL,
  output = c("simulations", "summary")
)
```

Arguments

mortality_table	A life table or a tidyact_life_contract object.
age	Integer actuarial age. Optional when mortality_table is a tidyact_life_contract.
rate	Numeric scalar. Annual interest-rate input. Optional when mortality_table is a tidyact_life_contract.
rate_type	Character string indicating the rate type.
m	Positive integer. Compounding frequency for nominal rates.
insurance_type	"whole", "term", or "endowment".
term_years	Term in years. Required for term and endowment insurance. Use Inf for whole life.
benefit	Numeric scalar. Insurance benefit.
method	Simulation method. Currently "inverse" is supported.

n_sim	Positive integer. Number of simulations.
seed	Optional seed for reproducibility.
output	"simulations" for the simulated data or "summary" for summary statistics using summary_mc().

Details

The function supports direct use with mortality_table, age, and rate, and pipe workflows with life_contract().

Value

A tibble.

See Also

Other simulation: simulate_annuity_x(), summary_mc()

Other life-contingencies: annuity_x(), annuity_xy(), insurance_x(), insurance_xy(), life_contract(), premium_gross(), premium_x(), premium_xy(), reserve_x(), reserve_xy(), simulate_annuity_x()

---

simulate_lifetime *Simulate future lifetimes from a life table*

Description

Simulates curtate and, optionally, complete future lifetimes from a life table containing one-year death probabilities.

Usage

```
simulate_lifetime(
  data,
  age,
  n_sim = 10000,
  age_col = "age",
  qx_col = "qx",
  method = c("inverse", "multinomial", "antithetic"),
  fractional = c("udd", "constant_force", "none"),
  seed = NULL,
  include_distribution = FALSE
)
```

Arguments

`data`	A data frame or tibble containing the life table.
`age`	Numeric scalar. Initial age x of the individual.
`n_sim`	Positive integer. Number of Monte Carlo simulations. Default is `10000`.
`age_col`	Character string. Name of the age column in `data`. Default is `"age"`.
`qx_col`	Character string. Name of the one-year death probability column in `data`. Default is `"qx"`.
`method`	Character string specifying the simulation method for K_x. Available options are `"inverse"`, `"multinomial"`, and `"antithetic"`. Default is `"inverse"`.
`fractional`	Character string specifying how the fractional part of the complete future lifetime is generated within the year of death. Available options are `"udd"`, `"constant_force"`, and `"none"`. Default is `"udd"`.
`seed`	Optional integer seed for reproducibility. Default is `NULL`.
`include_distribution`	Logical. If `TRUE`, the probability mass function used to simulate K_x is attached as a list-column named `distribution`. Default is `FALSE`.

Details

This function is designed as the simulation engine for Monte Carlo life contingency calculations in `tidyactuarial`. It generates simulated values of the curtate future lifetime K_x and a simulated complete future lifetime T_x. The output is a tidy tibble that can be used naturally with pipes and downstream functions such as `mc_insurance()`, `mc_annuity()`, `mc_premium()`, and `mc_loss()`.

For a life aged x, the curtate future lifetime K_x follows

$$P(K_x = k) = {}_kp_x q_{x+k},$$

where ${}_kp_x$ is the probability of surviving k complete years from age x, and q_{x+k} is the one-year probability of death at attained age $x + k$.

The function first constructs the conditional distribution of K_x from the selected age onward. Then it generates simulated values according to the selected method.

The available simulation methods are:

- `"inverse"`: inverse transform simulation using the cumulative distribution of K_x.
- `"multinomial"`: direct sampling from the probability mass function of K_x.
- `"antithetic"`: inverse transform simulation using antithetic uniforms U and $1 - U$, a basic variance reduction technique.

The `fractional` argument controls the simulated complete future lifetime T_x. If `fractional = "udd"`, a uniform fractional lifetime is added to K_x. If `fractional = "constant_force"`, the fractional lifetime within the year of death is generated under a constant force of mortality assumption conditional on death during that year. If `fractional = "none"`, the complete lifetime is returned as `NA`.

The probabilities are normalized internally to handle finite life tables. If the life table is truncated, the simulated distribution is conditional on death occurring within the available range of ages. In practical work, it is recommended that the last available death probability be equal to 1.

Value

A tibble with one row per simulation and the following columns:

- `sim_id`: simulation identifier.
- `age`: initial age.
- `method`: simulation method used.
- `fractional`: fractional age assumption used for T_x.
- `Kx`: simulated curtate future lifetime.
- `Tx`: simulated complete future lifetime. If `fractional = "none"`, this column contains `NA`.

If `include_distribution = TRUE`, the output also includes a list-column named `distribution` containing the probability mass function used for the simulation.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*. Second Edition. Society of Actuaries.

See Also

`mc_insurance()`, `mc_annuity()`, `mc_premium()`, `mc_loss()`

Examples

```
life_table <- tibble::tibble(
  age = 40:100,
  qx = seq(0.002, 1, length.out = 61)
)

# Basic simulation using inverse transform sampling
life_table |>
  simulate_lifetime(
    age = 40,
    n_sim = 1000,
    method = "inverse",
    seed = 123
  )

# Antithetic simulation
life_table |>
  simulate_lifetime(
    age = 40,
    n_sim = 1000,
    method = "antithetic",
    seed = 123
  )

# Returning the distribution used for simulation
life_table |>
  simulate_lifetime(
```

```
    age = 40,
    n_sim = 100,
    include_distribution = TRUE,
    seed = 123
  )

# Full pipeline with a life insurance present value
life_table |>
  simulate_lifetime(age = 40, n_sim = 1000, seed = 123) |>
  mc_insurance(
    rate = 0.05,
    insurance = "whole_life",
    benefit = 1
  )

# Complete future lifetime for benefits payable at the moment of death
life_table |>
  simulate_lifetime(
    age = 40,
    n_sim = 1000,
    fractional = "udd",
    seed = 123
  ) |>
  mc_insurance(
    rate = 0.05,
    insurance = "whole_life",
    payment_timing = "moment_of_death",
    tx_col = "Tx"
  )
```

`simulate_lifetimes` *Simulate multiple independent future lifetimes*

Description

Simulates curtate and complete future lifetimes for several lives under an independence assumption.

Usage

```
simulate_lifetimes(
  data,
  ages,
  n_sim = 10000,
  life_id = NULL,
  age_col = "age",
  qx_col = "qx",
  method = c("inverse", "multinomial", "antithetic"),
  fractional = c("udd", "constant_force", "none"),
  seed = NULL
)
```

Arguments

data	A life table data frame or a list of life table data frames. If a single data frame is supplied, the same life table is used for all lives. If a list is supplied, it must have length 1 or the same length as ages.
ages	Numeric vector. Initial ages of the lives to simulate.
n_sim	Positive integer. Number of Monte Carlo simulations per life. Default is 10000.
life_id	Optional character vector identifying each life. If NULL, IDs are created automatically as "life_1", "life_2", and so on. If data is a named list and life_id = NULL, the list names are used when possible.
age_col	Character string. Name of the age column in the life table or life tables. Default is "age".
qx_col	Character string. Name of the one-year death probability column in the life table or life tables. Default is "qx".
method	Character string specifying the simulation method for each curtate future lifetime. Available options are "inverse", "multinomial", and "antithetic". Default is "inverse".
fractional	Character string specifying how the fractional part of the complete future lifetime is generated within the year of death. Available options are "udd", "constant_force", and "none". Default is "udd".
seed	Optional integer seed for reproducibility. Default is NULL.

Details

This function extends simulate_lifetime() to multiple lives. It is designed as the simulation engine for Monte Carlo multiple-life actuarial calculations. Each life is simulated independently, and the output is returned in tidy long format with one row per simulation and per life.

For lives aged $x_1, x_2, \ldots, x_r$, the function simulates

$$K_{x_1}, K_{x_2}, \ldots, K_{x_r}$$

independently. The same applies to the complete future lifetimes $T_{x_1}, T_{x_2}, \ldots, T_{x_r}$ when fractional is not "none".

The independence assumption means that no common shock, copula, frailty, or other dependence structure is imposed. This is a natural first model for multiple-life Monte Carlo calculations and allows direct construction of joint-life, last-survivor, first-death, last-death, and k-th death quantities through downstream functions.

The output is intentionally long:

```
sim_id | life_id | life_index | age | Kx | Tx
```

This format works naturally with dplyr::group_by(), dplyr::summarise(), tidyr::pivot_wider(), and downstream functions such as mc_multilife_status().

If different mortality tables are needed for different lives, pass data as a list of life tables. For example, one table may be used for a male life and another for a female life.

Value

A tibble with one row per simulation and per life. It contains:

- `sim_id`: simulation identifier.
- `life_id`: life identifier.
- `life_index`: position of the life in `ages`.
- `age`: initial age of the life.
- `method`: simulation method used.
- `fractional`: fractional age assumption.
- `Kx`: simulated curtate future lifetime.
- `Tx`: simulated complete future lifetime. If `fractional = "none"`, this column contains `NA`.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*. Second Edition. Society of Actuaries.

See Also

`simulate_lifetime()`, `summary_mc()`

Examples

```
life_table <- tibble::tibble(
  age = 40:110,
  qx = seq(0.002, 1, length.out = 71)
)

# Two independent lives using the same life table
life_table |>
  simulate_lifetimes(
    ages = c(60, 58),
    n_sim = 1000,
    seed = 123
  )

# Three independent lives
life_table |>
  simulate_lifetimes(
    ages = c(60, 58, 55),
    life_id = c("x", "y", "z"),
    n_sim = 1000,
    seed = 123
  )

# Different life tables for different lives
male_table <- tibble::tibble(
  age = 40:110,
  qx = seq(0.003, 1, length.out = 71)
```

```
)

female_table <- tibble::tibble(
  age = 40:110,
  qx = seq(0.002, 1, length.out = 71)
)

list(male = male_table, female = female_table) |>
  simulate_lifetimes(
    ages = c(60, 58),
    n_sim = 1000,
    seed = 123
  )
```

`sinking_fund_schedule` *Sinking fund amortization schedule for a loan*

Description

Builds a sinking fund schedule under a fixed loan rate and a fixed accumulation rate for the sinking fund.

Usage

```
sinking_fund_schedule(
  principal,
  n,
  i_loan,
  i_fund,
  deposit = NULL,
  tol = 1e-08
)
```

Arguments

`principal`	Numeric scalar. Initial loan amount.
`n`	Positive integer. Number of periods.
`i_loan`	Numeric scalar. Effective interest rate per period on the loan.
`i_fund`	Numeric scalar. Effective interest rate per period on the sinking fund.
`deposit`	Optional numeric scalar. Level sinking-fund deposit per period. If `NULL`, it is computed so that the fund accumulates to `principal` at time `n`.
`tol`	Numeric tolerance used for zero checks and final-balance checks.

Details

The borrower pays:

- interest on the loan each period, and
- a level deposit into the sinking fund.

At maturity, the sinking fund is used to redeem the principal.

If `deposit` is `NULL` and `i_fund` is approximately zero, the deposit is computed as `principal / n`.

Otherwise, the standard sinking-fund formula is used:

$$\text{deposit} = \frac{\text{principal}}{s_n}$$

where

$$s_n = \frac{(1 + i_{\text{fund}})^n - 1}{i_{\text{fund}}}$$

Value

A tibble with one row per period and columns:

period Period index.

loan_balance_start Outstanding loan balance at the start of the period.

interest_loan Interest paid on the loan during the period.

sinking_deposit Deposit made into the sinking fund.

fund_balance_start Fund balance at the start of the period.

interest_fund Interest earned by the fund during the period.

fund_balance_end_before_redemption Fund balance before final redemption.

redemption_from_fund Amount withdrawn from the fund to redeem the loan at maturity.

fund_balance_end Fund balance after redemption.

loan_balance_end Outstanding loan balance after redemption.

total_cashflow_borrower Borrower's external cash outflow during the period.

See Also

`amort_schedule`, `s_angle`

Other amortization: `amort_schedule()`

Examples

```
sinking_fund_schedule(
  principal = 100000,
  n = 12,
  i_loan = 0.01,
  i_fund = 0.008
)
```

```
sinking_fund_schedule(
  principal = 50000,
  n = 10,
  i_loan = 0.02,
  i_fund = 0,
  deposit = NULL
)
```

`soa08lt` *SOA Illustrative Life Table*

Description

This dataset is intended for reproducible examples, internal validation, and benchmark tests for life-contingency functions such as `annuity_x()`, `insurance_x()`, `premium_x()`, `reserve_x()`, `annuity_xy()`, `insurance_xy()`, `premium_xy()`, and `reserve_xy()`.

Usage

```
soa08lt
```

Format

A data frame with one row per integer age and the following columns:

x Integer age.

lx Number of lives surviving to exact age `x`.

dx Number of deaths between exact ages `x` and `x + 1`.

qx One-year probability of death between exact ages `x` and `x + 1`.

px One-year probability of survival from exact age `x` to `x + 1`.

Details

A tidy version of the Society of Actuaries illustrative life table commonly used in life-contingencies examples and benchmark calculations.

The table is included as a convenient benchmark table for actuarial calculations. The build script in `data-raw/soa08lt.R` constructs this tidy dataset from the SOA illustrative actuarial table object distributed in the `lifecontingencies` package.

Source

Society of Actuaries illustrative life table, commonly referenced in Bowers et al. (1997), *Actuarial Mathematics*, Appendix 2A.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*. Second edition. Society of Actuaries.

Spedicato, G. A. (2013). The `lifecontingencies` package: performing financial and actuarial mathematics calculations in R.

Examples

```
data(soa08lt)
head(soa08lt)

annuity_x(
  mortality_table = soa08lt,
  age = 65,
  rate = 0.06,
  timing = "due"
)
```

---

standardize_interest *Standardize an interest rate to the annual effective rate i*

Description

Converts common interest-rate specifications to the equivalent annual effective interest rate `i`.

Usage

```
standardize_interest(type = "effective", rate, m = 1)
```

Arguments

`type`	Character vector indicating the rate type. Must be one of `"effective"`, `"nominal_interest"`, `"nominal_discount"`, or `"force"`.
`rate`	Numeric vector of rate values.
`m`	Positive integer vector giving the compounding frequency for nominal rates. Ignored for `"effective"` and `"force"`.

Details

The conversion formulas are:

effective: $i =$ rate (identity)

nominal_interest: $i = (1 + j^{(m)}/m)^m - 1$

nominal_discount: $i = (1 - d^{(m)}/m)^{-m} - 1$

force: $i = e^{\delta} - 1$

Input vectors must have length 1 or a common length.

Value

Numeric vector of annual effective rates. Missing values are propagated.

See Also

interest_equivalents, discount_factor_spot

Other interest: discount_factor_spot(), forward_rate(), interest_equivalents(), yield_curve()

Examples

```
# Scalar cases
standardize_interest("nominal_interest", rate = 0.18, m = 4)
standardize_interest("nominal_discount", rate = 0.10, m = 12)
standardize_interest("force", rate = 0.12)

# Vectorized case
standardize_interest(
  type = c("nominal_interest", "force", "effective"),
  rate = c(0.06, 0.05, 0.04),
  m = c(12, 1, 1)
)

# Use inside a data pipeline
if (requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  portfolio <- tibble::tibble(
    policy_id  = 1:3,
    gross_rate = c(0.05, 0.08, 0.10),
    rate_type  = c("force", "nominal_interest", "nominal_discount"),
    frequency  = c(NA, 4, 12)
  )

  dplyr::mutate(
    portfolio,
    effective_rate = standardize_interest(
      type = rate_type,
      rate = gross_rate,
      m = frequency
    )
  )
}
```

---

summary_mc *Summarise Monte Carlo simulation output*

Description

Computes tidy summary statistics from simulated actuarial values.

Usage

```
summary_mc(
  .data,
  value_col = "present_value",
  group_cols = NULL,
  by = NULL,
  probs = c(0.025, 0.5, 0.975),
  var_probs = c(0.95, 0.99),
  na_rm = TRUE
)
```

Arguments

.data	A data.frame or tibble containing simulation output.
value_col	Character string. Name of the numeric column to summarise. Defaults to "present_value".
group_cols	Optional character vector of grouping columns.
by	Optional character vector of grouping columns. This is a convenient alias for group_cols, useful in examples and pipelines.
probs	Numeric vector of probabilities for quantiles.
var_probs	Numeric vector of probabilities for VaR and TVaR.
na_rm	Logical. If TRUE, missing values are removed.

Details

This function is intentionally generic. It can summarise simulated present values from annuities, insurances, premiums, reserves, losses, or any other numeric actuarial indicator stored in a tidy simulation table.

The function computes the number of simulations, mean, variance, standard deviation, standard error, minimum, maximum, selected quantiles, VaR, and TVaR.

TVaR is computed empirically as the mean of simulated values greater than or equal to the corresponding empirical VaR.

Value

A tibble with summary statistics.

See Also

Other simulation: simulate_annuity_x(), simulate_insurance_x()

Examples

```
sim <- tibble::tibble(
  simulation_id = 1:5,
  duration = c(0, 0, 1, 1, 1),
  present_value = c(10, 12, 8, 15, 11)
```

```
)

summary_mc(sim)
summary_mc(sim, by = "duration")
```

`s_angle` *Level annuity accumulation factor s-angle-n*

Description

Computes the actuarial accumulation factor for a level annuity.

Usage

```
s_angle(
  n_years,
  payments_per_year = 1L,
  rate,
  rate_type = "effective",
  m = 1L,
  deferral_years = 0,
  timing = "immediate",
  payment = 1,
  output = c("value", "table")
)
```

Arguments

`n_years`	Numeric vector of payment durations in years. Each value must be positive and finite.
`payments_per_year`	Positive integer vector giving the number of discrete payments per year. Ignored for continuous annuities.
`rate`	Numeric vector of rate values.
`rate_type`	Character vector indicating the rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer vector giving the compounding frequency for nominal rates. Ignored for `"effective"` and `"force"`.
`deferral_years`	Numeric vector of deferral times in years. Must be greater than or equal to 0. Under the adopted horizon convention, this is metadata only for accumulation factors.
`timing`	Character vector. One of `"immediate"`, `"due"`, or `"continuous"`.
`payment`	Numeric vector of level payment amounts. Used only when `output = "table"` to report the corresponding future value. The accumulation factor itself is always computed for unit payments.
`output`	Character string. Use `"value"` to return a numeric accumulation factor, or `"table"` to return a tibble with intermediate calculations.

Details

Supported timing conventions:

- `"immediate"`: annuity-immediate with discrete payments.
- `"due"`: annuity-due with discrete payments.
- `"continuous"`: continuous annuity.

For discrete annuities, `payments_per_year = k` means payments are made every $1/k$ year. The function returns the accumulation factor, assuming a unit payment at each payment time.

Horizon convention: the future value is measured at the time of the last payment. Under this convention, a pure deferral that shifts the entire payment block forward in time does not change the accumulation factor when the payment pattern is otherwise unchanged. Therefore, `deferral_years` is recorded and validated, but it does not modify the factor.

The future value of a perpetuity diverges, so perpetuities are not supported in `s_angle()`.

The function first converts the supplied rate to the equivalent annual effective interest rate using `standardize_interest`.

For finite discrete annuities:

$$s_{\overline{n|}} = \frac{(1+i)^n - 1}{i}$$

For due annuities:

$$\ddot{s}_{\overline{n|}} = (1+i)s_{\overline{n|}}$$

For continuous annuities:

$$\bar{s}_{\overline{n|}} = \frac{e^{\delta n} - 1}{\delta}$$

Input vectors must have length 1 or a common length. Missing values are propagated.

Value

If `output = "value"`, a numeric vector of accumulation factors.

If `output = "table"`, a tibble with input values, equivalent rates, accumulation factors, payment amounts, and future values.

See Also

`a_angle`, `standardize_interest`, `future_value`

Other annuities: `a_angle()`, `annuity_arith()`, `annuity_geom()`

Examples

```
# Numeric accumulation factor
s_angle(n_years = 10, rate = 0.05)

# Nominal interest converted monthly, with monthly payments
s_angle(
  n_years = 10,
  rate = 0.06,
```

```
  rate_type = "nominal_interest",
  m = 12,
  payments_per_year = 12
)

# Continuous annuity
s_angle(
  n_years = 15,
  rate = 0.04,
  rate_type = "force",
  timing = "continuous"
)

# Tibble output for teaching or auditing
s_angle(
  n_years = 10,
  rate = 0.05,
  payment = 1000,
  output = "table"
)

# Vectorized example
s_angle(
  n_years = c(5, 10, 20),
  payments_per_year = c(1, 12, 1),
  rate = c(0.05, 0.06, 0.04),
  rate_type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  deferral_years = c(0, 2, 3),
  timing = c("immediate", "due", "continuous")
)
```

---

t_Ex *Pure endowment (discounted survival): _tE_x*

Description

Computes the actuarial present value of a pure endowment, i.e., the expected present value of a payment of 1 made at time t if and only if a life aged x survives to age $x + t$:

$$ {}_tE_x = v^t \cdot {}_tp_x = (1+i)^{-t} \cdot \frac{\ell_{x+t}}{\ell_x}. $$

Usage

```
t_Ex(lt, x, t, i, frac, tidy = FALSE, check = TRUE, tol = 1e-10)
```

Arguments

`lt`	A lifetable object as produced by `lifetable`. Must contain columns `x` and `lx`.
`x`	Integer age(s) at which the endowment starts.
`t`	Nonnegative numeric duration(s) in years (can be fractional).
`i`	Annual effective interest rate(s). Must satisfy `i > -1`.
`frac`	Fractional-age assumption passed to `t_px`: `"UDD"`, `"CF"`, `"CML"` (alias of CF), or `"Balducci"`. If not specified and `lt` carries a `frac` attribute, that value is used.
`tidy`	Logical. If `TRUE`, returns a tibble with columns `x`, `t`, `i`, `frac`, `nEx`.
`check`	Logical. If `TRUE`, performs validity checks.
`tol`	Numeric tolerance for integer checks on `x`.

Details

The pure endowment is a fundamental building block in life contingency mathematics (Finan, Section 26.3.1). It serves as the actuarial discount factor, combining financial discounting with mortality:

$$ {}_nE_x = v^n \cdot {}_np_x. $$

Key identities involving ${}_nE_x$:

- Actuarial accumulated value: $\ddot{s}_{x:\overline{n}|} = \ddot{a}_{x:\overline{n}|}/{}_nE_x$ (Finan, Sec. 34).
- Endowment insurance decomposition: $A_{x:\overline{n}|} = A^1_{x:\overline{n}|} + {}_nE_x$ (Finan, Sec. 26.3.2).
- Deferred annuity: ${}_{n|}\ddot{a}_x = {}_nE_x \cdot \ddot{a}_{x+n}$ (Finan, Sec. 35).

For a constant force of mortality μ and force of interest δ:

$$ {}_nE_x = e^{-n(\mu+\delta)} $$

(Finan, Example 26.14).

The variance of the pure endowment random variable is (Finan, Sec. 26.3.1):

$$ \mathrm{Var}(\bar{Z}_{\ x:\overline{n}|}) = v^{2n} \cdot {}_np_x \cdot {}_nq_x. $$

Value

Numeric vector of ${}_tE_x$, or a tibble if `tidy = TRUE`.

See Also

`t_px` for survival probabilities (without discounting), `insurance_x` for term, whole life, and endowment insurance APVs, `annuity_x` for life annuity APVs that use ${}_nE_x$ internally.

Examples

```
x  <- 0:5
lx <- c(100000, 99500, 99000, 98200, 97000, 95000)
lt <- lifetable(x = x, lx = lx, omega = 5, close = TRUE)

# Basic pure endowment: 3_E_0 = v^3 * 3_p_0
t_Ex(lt, x = 0, t = 3, i = 0.06)
# Verify manually:
(1.06)^(-3) * t_px(lt, x = 0, t = 3)

# Finan Example 26.14 style: constant force mu, delta
# For exponential survival with mu = 0.05, delta = 0.10:
# 10_E_30 = exp(-10*(0.05 + 0.10)) = exp(-1.5)
lt_exp <- lifetable(x = 0:50, lx = 100000 * exp(-0.05 * (0:50)))
t_Ex(lt_exp, x = 30, t = 10, i = exp(0.10) - 1)
exp(-1.5)  # theoretical value

# Finan Problem 26.16: 5-year pure endowment for (30), i = 6%
lt_ilt <- lifetable(
  x  = 30:35,
  lx = c(9501381, 9486854, 9471591, 9455522, 9438571, 9420657)
)
t_Ex(lt_ilt, x = 30, t = 5, i = 0.06)
# = v^5 * l_35 / l_30 = (1.06)^(-5) * 9420657/9501381

# Vectorized: multiple ages at once
t_Ex(lt, x = c(0, 1, 2), t = 3, i = 0.05, tidy = TRUE)

# Use in a tidy pipeline
if (requireNamespace("dplyr", quietly = TRUE)) {
  tibble::tibble(age = 0:4, term = c(5, 4, 3, 2, 1)) |>
    dplyr::mutate(pure_endow = t_Ex(lt, x = age, t = term, i = 0.06))
}
```

---

t_px *t-year survival probability from a life table*

Description

Computes the t-year survival probability

$$ {}_tp_x = P[T(x) > t] $$

using an annual life table, allowing for fractional ages under standard actuarial assumptions.

Usage

```
t_px(lt, x, t, frac, tidy = FALSE, check = TRUE, tol = 1e-10)
```

Arguments

`lt`	A lifetable object as produced by `lifetable`. Must contain columns `x` and `lx`. Columns `qx` or `px` are used if present.
`x`	Integer age(s) at which survival starts.
`t`	Nonnegative numeric duration(s) in years (can be fractional).
`frac`	Fractional-age assumption: `"UDD"`, `"CF"`, `"CML"` (alias of CF), or `"Balducci"`. If not specified and `lt` carries a `frac` attribute (set by `lifetable`), that value is used.
`tidy`	Logical. If `TRUE`, returns a tibble with columns `x`, `t`, `frac`, `tpx`.
`check`	Logical. If `TRUE`, performs validity checks.
`tol`	Numeric tolerance for integer checks on `x`.

Details

The integer-year survival is obtained directly from the life table (Finan, Section 22):

$$ {}_np_x = \frac{\ell_{x+n}}{\ell_x} $$

For non-integer durations, let $t = n + s$ with $n = \lfloor t \rfloor$ and $s \in [0, 1)$. Then (Finan, Section 24):

$$ {}_tp_x = {}_np_x \times {}_sp_{x+n} $$

The fractional-year factor ${}_sp_y$ depends on the assumption:

- UDD (Finan, Sec. 24.1): ${}_sp_y = 1 - s \times q_y$
- CF (Finan, Sec. 24.2): ${}_sp_y = (p_y)^s$
- Balducci (Finan, Sec. 24.3): ${}_sp_y = \frac{p_y}{1-(1-s)\times q_y}$

If $x + t > \omega$ (the terminal age), the function returns 0 since no survival is possible beyond the table's limiting age.

Value

Numeric vector of ${}_tp_x$, or a tibble if `tidy = TRUE`.

`t_pxy` *Two-life survival probability for independent lives*

Description

Computes the survival probability for two independent lives with actuarial ages `x` and `y` at time 0 under a joint-life or last-survivor status.

Usage

```
t_pxy(lt, x, y, t, frac, status = c("joint", "last"))
```

Arguments

`lt`	Either: • a single life table data frame, used for both lives; or • a list of two life tables `list(lt_x, lt_y)`, one for each life. Each life table must contain columns `x` and `lx`.
`x`	Integer actuarial age of the first life at time 0.
`y`	Integer actuarial age of the second life at time 0.
`t`	Nonnegative time (may be fractional).
`frac`	Fractional-age assumption: `"UDD"`, `"CF"`, `"CML"` (alias of CF), or `"Balducci"`. If not specified and the supplied life table(s) carry a `frac` attribute, that value is used. If two tables are supplied and their `frac` attributes differ, `frac` must be supplied explicitly. Passed to `t_px`.
`status`	Two-life status: `"joint"` or `"last"`.

Details

Independence is assumed throughout (Finan, Sections 56–57).

Joint-life status (Finan, Section 56): the status survives as long as *both* lives are alive.

$$ {}_tp_{xy} = {}_tp_x \cdot {}_tp_y. $$

Last-survivor status (Finan, Section 57): the status survives as long as *at least one* life is alive.

$$ {}_tp_{\overline{xy}} = {}_tp_x + {}_tp_y - {}_tp_x \cdot {}_tp_y. $$

Individual survival probabilities ${}_tp_x$ and ${}_tp_y$ are computed via `t_px`, which supports fractional ages under UDD, constant force, and Balducci assumptions (Finan, Section 24).

Value

A single numeric value.

See Also

t_px for single-life survival, e_xy for joint-life expectancy, annuity_xy for two-life annuity APVs, insurance_xy for two-life insurance APVs.

Examples

```
lt <- data.frame(
  x  = 60:66,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000, 86000)
)

# Joint life, 2.5 years, UDD (Finan, Sec. 56)
t_pxy(lt, x = 60, y = 62, t = 2.5, frac = "UDD", status = "joint")

# Verify: joint = product of marginals
t_px(lt, x = 60, t = 2.5, frac = "UDD") *
  t_px(lt, x = 62, t = 2.5, frac = "UDD")

# Last survivor, 2.5 years, constant force (Finan, Sec. 57)
t_pxy(lt, x = 60, y = 62, t = 2.5, frac = "CF", status = "last")

# Same result if the same table is supplied explicitly for both lives
t_pxy(list(lt, lt), x = 60, y = 62, t = 2.5, frac = "UDD", status = "joint")

# Different life tables for the two lives
lt_m <- data.frame(
  x  = 60:66,
  lx = c(100000, 98500, 96800, 94800, 92400, 89500, 86000)
)
lt_f <- data.frame(
  x  = 60:66,
  lx = c(100000, 99000, 97800, 96400, 94700, 92700, 90300)
)
t_pxy(list(lt_m, lt_f), x = 60, y = 62, t = 2.5, frac = "UDD", status = "joint")

# Finan Example 56.2 style: integer survival
# 10_p_{50:60} = 10_p_50 * 10_p_60
lt_ilt <- data.frame(
  x  = 50:70,
  lx = c(8950901, 8879913, 8804189, 8723382, 8637048,
         8544731, 8445974, 8340310, 8227261, 8106334,
         7977338, 7839775, 7693040, 7536522, 7369603,
         7191658, 7002051, 6800139, 6585264, 6356752,
         6114913)
)
t_pxy(lt_ilt, x = 50, y = 60, t = 10, status = "joint")

# Finan Problem 56.1: t_q_xy = t_q_x + t_q_y - t_q_x * t_q_y
p_joint <- t_pxy(lt, x = 60, y = 62, t = 3, status = "joint")
q_joint <- 1 - p_joint
qx <- 1 - t_px(lt, x = 60, t = 3)
qy <- 1 - t_px(lt, x = 62, t = 3)
```

```
c(q_joint = q_joint, q_sum = qx + qy - qx * qy)  # should match
```

---

t_qx *t-year death probability from a life table*

Description

Computes the t-year death probability

$$ {}_tq_x = \Pr[T(x) \le t] = 1 - {}_tp_x $$

using an annual life table, allowing for fractional ages under standard actuarial assumptions.

Usage

```
t_qx(lt, x, t, frac, tidy = FALSE, check = TRUE, tol = 1e-10)
```

Arguments

lt	A lifetable object as produced by lifetable. Must contain columns x and lx. Columns qx or px are used if present.
x	Integer age(s) at which the interval starts.
t	Nonnegative numeric duration(s) in years (can be fractional).
frac	Fractional-age assumption: "UDD", "CF", "CML" (alias of CF), or "Balducci". If not specified and lt carries a frac attribute (set by lifetable), that value is used. Passed directly to t_px.
tidy	Logical. If TRUE, returns a tibble with columns x, t, frac, tqx.
check	Logical. If TRUE, performs validity checks.
tol	Numeric tolerance for integer checks on x.

Details

This is a thin wrapper around t_px:

$$ {}_tq_x = 1 - {}_tp_x. $$

The identity ${}_tq_x = {}_td_x/\ell_x$ (Finan, Section 22) holds for integer t, where ${}_td_x = \ell_x - \ell_{x+t}$ is the expected number of deaths between ages x and $x + t$.

For fractional durations, the result depends on the chosen assumption (UDD, CF, or Balducci); see t_px for details and formulas (Finan, Section 24).

The deferred death probability ${}_{n|}q_x$ can be obtained as (Finan, Section 23.4):

$$ {}_{n|}q_x = {}_np_x \cdot q_{x+n} = {}_{n+1}q_x - {}_nq_x. $$

Value

Numeric vector of ${}_tq_x$, or a tibble if tidy = TRUE.

See Also

t_px for the complementary survival probability, t_Ex for the pure endowment (discounted survival), e_x for life expectancy, lifetable for building the life table input.

Examples

```
x  <- 0:5
lx <- c(100000, 99500, 99000, 98200, 97000, 95000)
lt <- lifetable(x = x, lx = lx, omega = 5, close = TRUE)

# Integer death probability (Finan, Section 22)
t_qx(lt, x = 0, t = 3)  # (l0 - l3) / l0

# t = 0 always returns 0
t_qx(lt, x = 0, t = 0)

# Fractional age under UDD (Finan, Section 24.1)
t_qx(lt, x = 0, t = 2.5, frac = "UDD")

# Finan Example 22.2a: number of deaths between ages 2 and 5
# 3_d_2 = l_2 - l_5 = 98995 - 97468 = 1527
lt_22 <- lifetable(
  x = 0:5,
  lx = c(100000, 99499, 98995, 98489, 97980, 97468)
)
t_qx(lt_22, x = 2, t = 3) * lt_22$lx[lt_22$x == 2]  # 1527

# Deferred death probability (Finan, Section 23.4):
# 2|1_q_0 = 2_p_0 * q_2 = 3_q_0 - 2_q_0
t_qx(lt, x = 0, t = 3) - t_qx(lt, x = 0, t = 2)

# Vectorized with tidy output
t_qx(lt, x = c(0, 1), t = c(1.5, 2.2), frac = "Balducci", tidy = TRUE)

# Use in a tidy pipeline
if (requireNamespace("dplyr", quietly = TRUE)) {
  tibble::tibble(age = c(0, 1, 2), duration = c(3, 2.5, 1.7)) |>
    dplyr::mutate(
      surv  = t_px(lt, x = age, t = duration),
      death = t_qx(lt, x = age, t = duration)
    )
}
```

---

t_qxj *t-year probability of decrement by cause j: t_qxj*

Description

Computes ${}_tq_x^{(j)}$, the probability that a life aged x decrements by a specific cause j within t years, using a multiple decrement table produced by md_table.

Usage

```
t_qxj(md, x, t, cause, frac = NULL, tidy = FALSE, check = TRUE, tol = 1e-10)
```

Arguments

md	A multiple decrement table produced by md_table. Must contain columns x, p_total, and the requested cause.
x	Numeric vector. Starting age(s) (integer-valued).
t	Numeric vector. Time horizon(s) in years (t >= 0). Can be non-integer if frac is supplied.
cause	Character. Name of the cause column in md (e.g., "q_death").
frac	Character. Fractional-age assumption for non-integer t: one of "UDD", "CF", "Balducci", or NULL. If NULL (default), t must be integer-valued.
tidy	Logical. If TRUE, returns a tibble with columns x, t, cause, frac, and tqxj.
check	Logical. If TRUE, performs input validation (default TRUE).
tol	Numeric tolerance for integer checks (default 1e-10).

Details

Let $q_x^{(j)}$ be the annual decrement probability for cause j and $q_x^{(\tau)}$ be the total decrement probability. For integer t,

$$ {}_tq_x^{(j)} = \sum_{k=0}^{t-1} \left(\prod_{r=0}^{k-1} p_{x+r}^{(\tau)} \right) q_{x+k}^{(j)}. $$

For non-integer $t = n+s$ with $n = \lfloor t \rfloor$ and $s \in [0, 1)$, this function supports fractional-age assumptions specified by frac and uses the additional convention that the within-year cause proportions remain constant: ${}_sq_x^{(j)} = w_j \, {}_sq_x^{(\tau)}$ where $w_j = q_x^{(j)}/q_x^{(\tau)}$ (and 0 when $q_x^{(\tau)} = 0$).

Supported fractional-age assumptions for the total decrement:

- "UDD": ${}_sq_x^{(\tau)} = sq_x^{(\tau)}$.
- "CF": ${}_sp_x^{(\tau)} = (p_x^{(\tau)})^s$.
- "Balducci": ${}_sq_x^{(\tau)} = \frac{sq_x^{(\tau)}}{1-(1-s)q_x^{(\tau)}}$.

Value

Numeric vector of ${}_tq_x^{(j)}$ (or tibble if tidy=TRUE).

Examples

```
qx_df <- tibble::tibble(
  x = 30:35,
  q_death = c(0.001, 0.0012, 0.0014, 0.0017, 0.0020, 1.0000),
  q_disability = c(0.002, 0.0021, 0.0022, 0.0023, 0.0024, 0.0000)
)
md <- md_table(qx_df, radix = 1e5, close = TRUE)
```

```
t_qxj(md, x = 30, t = 5, cause = "q_death")
t_qxj(md, x = 30, t = 2.5, cause = "q_death", frac = "CF", tidy = TRUE)
```

`yield_curve` *Validate a yield curve, compute discount factors, and optionally create a plot*

Description

Builds a tibble-first representation of a discrete yield curve using annual effective spot rates and computes the corresponding discount factors.

Usage

```
yield_curve(
  .data = NULL,
  term = NULL,
  spot = NULL,
  col_term = "term",
  col_spot = "spot",
  plot = FALSE,
  .out = "discount",
  .out_plot = "yield_curve_plot",
  .keep = c("all", "used", "none"),
  .na = c("propagate", "error", "drop")
)
```

Arguments

`.data`	A data.frame or tibble. If `NULL`, `term` and `spot` must be supplied as numeric vectors.
`term`	Numeric vector of maturities when `.data = NULL`.
`spot`	Numeric vector of annual effective spot rates when `.data = NULL`.
`col_term`	Name of the list-column containing maturities.
`col_spot`	Name of the list-column containing spot rates.
`plot`	Logical; if `TRUE`, adds a list-column of `ggplot2` objects.
`.out`	Name of the output list-column containing discount factors.
`.out_plot`	Name of the output list-column containing `ggplot2` objects. Used only if `plot = TRUE`.
`.keep`	One of `"all"`, `"used"`, or `"none"`.
`.na`	NA handling policy: `"propagate"`, `"error"`, or `"drop"`.

Details

Each row is treated as one curve (one case). For tibble input, `col_term` and `col_spot` must be list-columns of equal-length numeric vectors. When `.data = NULL`, `term` and `spot` must be numeric vectors and a one-row tibble is returned.

The discount factors are computed as:

$$v_t = (1 + i_t)^{-t}$$

where i_t is the annual effective spot rate for maturity t.

If `plot = TRUE`, the function also returns a list-column of `ggplot2` objects showing the spot yield curve for each row.

Value

A tibble. By default it returns the original columns plus a new list-column named by `.out` containing discount-factor vectors. If `plot = TRUE`, it also adds a list-column named by `.out_plot` containing `ggplot2` objects.

References

Marcel B. Finan, *A Basic Course in the Theory of Interest and Derivatives Markets: A Preparation for the Actuarial Exam FM/2*, Section 53: The Term Structure of Interest Rates and Yield Curves.

Kellison, S. G. *The Theory of Interest*.

See Also

`forward_rate`, `discount_factor_spot`, `standardize_interest`

Other interest: `discount_factor_spot()`, `forward_rate()`, `interest_equivalents()`, `standardize_interest()`

Examples

```
# Simple example
res <- yield_curve(
  term = c(1, 2, 3, 4, 5),
  spot = c(0.040, 0.045, 0.048, 0.050, 0.051),
  plot = TRUE
)

res$yield_curve_plot[[1]]

# Medium example
curves <- tibble::tibble(
  curve_id = c("A", "B"),
  term = list(c(1, 2, 3), c(1, 3, 5)),
  spot = list(c(0.04, 0.05, 0.06), c(0.03, 0.035, 0.04))
)

res2 <- yield_curve(
  curves,
  col_term = "term",
```

```
  col_spot = "spot",
  plot = TRUE,
  .out = "v",
  .out_plot = "curve_plot"
)

res2$curve_plot[[2]]
```

Index