Package 'tidyactuarial'

April 10, 2026

Type Package

Title Tidy Tools for Actuarial Mathematics and Life Contingencies

Version 0.1.1

Description Provides tidyverse-aligned tools for actuarial mathematics and life contingencies, including life tables, survival probabilities, actuarial present values of cash flows, life annuities, multi-life benefits, and related quantities. The package emphasizes clear actuarial notation consistent with standard curricula (e.g. SOA exams) and supports reproducible workflows using modern R.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

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

RoxygenNote 7.3.3

Suggests purrr

NeedsCompilation no

Author Julian Fajardo [aut, cre]

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

Repository CRAN

Date/Publication 2026-04-10 10:10:08 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_multi *Actuarial present value of a multi-life annuity (n lives)*

Description

Computes the APV of a discrete annuity contingent on multiple independent lives. 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.

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	Life table with column x and at least one of lx, px, qx.
ages	Integer vector of actuarial ages for the lives at issue.
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 table.
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

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 APV of a discrete life annuity at actuarial age x using a life table. Supports term n (temporary), integer deferral m, k-thly payments (exact under UDD), and Woolhouse approximations up to second order.

Usage

```
annuity_x(
  lt,
  x,
  i,
  n = NULL,
  m = 0L,
  k = 1L,
  timing = c("immediate", "due"),
  woolhouse = c("none", "first", "second"),
  tidy = FALSE
)
```

Arguments

`lt`	A lifetable object as produced by `lifetable`. Must contain columns `x` and `lx`.
`x`	Integer actuarial age.
`i`	Effective annual interest rate (must satisfy `i > -1`).
`n`	Integer term in years. If `NULL` (default), whole life to end of table.
`m`	Integer deferral in years (default `0`).
`k`	Integer payments per year (default `1`). Example: `k = 12` for monthly.
`timing`	`"immediate"` (payments at end of period) or `"due"` (beginning). Default `"immediate"`.
`woolhouse`	For `k > 1`: `"none"` (exact UDD), `"first"` (2-term Woolhouse), or `"second"` (3-term Woolhouse).
`tidy`	Logical. If `TRUE`, returns a one-row tibble.

Details

Annual annuity-due (Finan, Section 37.2, Example 37.9):

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

Annual annuity-immediate (Finan, Section 37.5):

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

Deferral (Finan, Section 37.3):

$${}_{m|}\ddot{a}_x = v^m \times {}_mp_x \times \ddot{a}_{x+m}$$

k-thly exact under UDD (Finan, Section 38): each $1/k$-year fractional survival is computed under UDD, giving an exact result:

$$\ddot{a}^{(k)}_{x:\overline{n}|} = \frac{1}{k} \sum_{j=0}^{kn-1} v^{j/k} \times {}_{j/k}p_x$$

Woolhouse approximations (Finan, Problems 38.9-38.10):

- 2-term (first order): $\ddot{a}^{(k)}_{x:\overline{n}|} \approx \ddot{a}_{x:\overline{n}|} - \frac{k-1}{2k}(1 - {}_nE_x)$
- 3-term (second order): $\ddot{a}^{(k)}_{x:\overline{n}|} \approx \ddot{a}_{x:\overline{n}|} - \frac{k-1}{2k}(1 - {}_nE_x) - \frac{k^2-1}{12k^2}[\delta + \mu_x - {}_nE_x(\delta + \mu_{x+n})]$

Conversion from due to immediate for k-thly temporaries (Finan, Problem 38.6):

$$a^{(k)}_{x:\overline{n}|} = \ddot{a}^{(k)}_{x:\overline{n}|} - \frac{1}{k}(1 - {}_nE_x)$$

Value

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

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

Description

Computes the APV of a discrete annuity contingent on two independent lives aged x and y. Supports joint-life, last-survivor, and reversionary-style benefits via state-based weights.

Usage

```
annuity_xy(
  lt,
  x,
  y,
  i,
  cohort = c("first", "last"),
  benefit = NULL,
  n = NULL,
  m = 0L,
  k = 1L,
  timing = c("immediate", "due"),
  woolhouse = c("none", "first", "second"),
  frac,
  tidy = FALSE
)
```

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.
i	Annual effective interest rate (must be > -1).
cohort	Status cohort: "first" (pays while both alive) or "last" (pays while at least one alive). Ignored if benefit is explicitly supplied.
benefit	Optional list with weights both, x_only, y_only. If supplied, the payment at time t is weighted by the probability of each state at t.
n	Integer term (years). If NULL, whole life to end of table.
m	Integer deferment (years). Default 0.
k	Integer payments per year. Default 1.
timing	Payment timing: "immediate" or "due".
woolhouse	Woolhouse approximation for k > 1: "none" (exact UDD k-thly), "first", or "second".
frac	Fractional-age assumption for k > 1 exact computation: "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.

Details

This function assumes independence between lives (Finan, Sections 56–59).

Joint-life annuity-due (Finan, Section 58):

$$\ddot{a}_{xy} = \sum_{k=0}^{\infty} v^k \cdot {}_kp_{xy}.$$

Last-survivor annuity-due (Finan, Section 59):

$$\ddot{a}_{\overline{xy}} = \ddot{a}_x + \ddot{a}_y - \ddot{a}_{xy}.$$

State-based benefits: for reversionary annuities, use `benefit = list(both = 1, x_only = alpha, y_only = alpha)` where α is the fraction paid to the survivor.

For `k > 1` with `woolhouse = "none"`, exact k-thly computation is performed under the selected fractional-age assumption using `t_px`.

Value

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

See Also

`annuity_x` for single-life annuity APVs, `insurance_xy` for two-life insurance, `t_pxy` for two-life survival, `premium_xy` for two-life premiums.

Examples

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

# Joint-life annuity-due (Finan, Sec. 58)
annuity_xy(lt, x = 60, y = 62, i = 0.05, cohort = "first", timing = "due")

# Last-survivor annuity-due (Finan, Sec. 59)
annuity_xy(lt, x = 60, y = 62, i = 0.05, cohort = "last", timing = "due")

# Verify identity: a_{xy-bar} = a_x + a_y - a_{xy}
a_joint <- annuity_xy(lt, x = 60, y = 62, i = 0.05,
                      cohort = "first", timing = "due")
a_last  <- annuity_xy(lt, x = 60, y = 62, i = 0.05,
                      cohort = "last", timing = "due")
a_x <- annuity_x(lt, x = 60, i = 0.05, timing = "due")
a_y <- annuity_x(lt, x = 62, i = 0.05, timing = "due")
c(last = a_last, sum_minus_joint = a_x + a_y - a_joint)

# Reversionary: full while both alive, 60% to survivor
annuity_xy(lt, x = 60, y = 62, i = 0.05,
           benefit = list(both = 1, x_only = 0.6, y_only = 0.6),
```

```
            timing = "due")

# 2-year deferred joint-life annuity
annuity_xy(lt, x = 60, y = 62, i = 0.05, m = 2,
            cohort = "first", timing = "due")

# Tidy output
annuity_xy(lt, x = 60, y = 62, i = 0.05, cohort = "first",
            timing = "due", tidy = TRUE)
```

`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").

---

arithmetic_annuity_av_tbl

Accumulated value of an arithmetic progression annuity

Description

Computes the accumulated value of an annuity whose payments follow an arithmetic progression.

Usage

```
arithmetic_annuity_av_tbl(
  .data = NULL,
  amount = NULL,
  step = NULL,
  n = NULL,
  i = NULL,
  timing = c("immediate", "due", "vencida", "anticipada"),
  perpetuity = FALSE,
  col_amount = "amount",
  col_step = "step",
  col_n = "n",
  col_i = "i",
  .out = "av",
  .keep = c("all", "used", "none"),
  .na = c("propagate", "error", "drop")
)
```

Arguments

.data	A data.frame or tibble. If NULL, inputs must be supplied directly as scalars or equal-length vectors.
amount	Numeric first payment when .data = NULL.
step	Numeric arithmetic increment per payment period when .data = NULL.
n	Number of payments when .data = NULL.
i	Effective interest rate per payment period when .data = NULL.
timing	Payment timing. Accepted values are "immediate", "due", "vencida", and "anticipada".
perpetuity	Logical. Must be FALSE. Included only for interface consistency with the present-value companion function.
col_amount	Name of the first-payment column.
col_step	Name of the arithmetic-step column.
col_n	Name of the number-of-payments column.
col_i	Name of the interest-rate column.
.out	Name of the output column containing accumulated value.
.keep	One of "all", "used", or "none".
.na	NA handling policy: "propagate", "error", or "drop".

Details

The first payment is amount, and each subsequent payment changes by step. The annuity may be either:

- annuity-immediate (timing = "immediate" or "vencida"),
- annuity-due (timing = "due" or "anticipada").

This is a tibble-first mutate-style function: each input row is one case.

Assumptions:

- Time is discrete.
- `i` is the effective interest rate per payment period.
- `n` is the number of payments.
- All payments must remain nonnegative: the final payment `amount + (n - 1) * step` must be `>= 0`.
- `perpetuity = TRUE` is not supported because the accumulated value of a perpetuity is not finite.

For an annuity-immediate with `n` payments, first payment `P`, step `Q`, and effective rate `i` per period, the accumulated value at time `n` is:

$$AV = P\, s_{\overline{n}|} + Q\, \frac{s_{\overline{n}|} - n}{i}$$

where $s_{\overline{n}|} = \frac{(1+i)^n - 1}{i}$.

For an annuity-due the result is $(1 + i)$ times the immediate value.

When $i = 0$, the accumulated value simplifies to:

$$AV = nP + Q\frac{n(n-1)}{2}.$$

Value

A tibble with a new numeric column named by `.out`.

References

Marcel B. Finan, *A Basic Course in the Theory of Interest and Derivatives Markets: A Preparation for the Actuarial Exam FM/2*.

Kellison, S. G. *The Theory of Interest*.

See Also

`arithmetic_annuity_pv_tbl`, `a_angle`, `s_angle`

Other annuities: `Da_angle()`, `Da_angle_tbl()`, `Ia_angle()`, `Ia_angle_tbl()`, `a_angle()`, `a_angle_tbl()`, `arithmetic_annuity_pv_tbl()`, `ga_angle()`, `ga_angle_tbl()`, `gs_angle()`, `gs_angle_tbl()`, `s_angle()`, `s_angle_tbl()`

Examples

```
arithmetic_annuity_av_tbl(
  amount = 100,
  step = 5,
  n = 5,
  i = 0.05,
  timing = "immediate"
```

```
)

cases <- tibble::tibble(
  amount = c(100, 200, 50),
  step   = c(5, 10, 0),
  n      = c(5, 4, 10),
  i      = c(0.05, 0.04, 0.03)
)

arithmetic_annuity_av_tbl(
  cases,
  timing = "due",
  .out = "av_due"
)
```

`arithmetic_annuity_pv_tbl` *Present value of an arithmetic progression annuity*

Description

Computes the present value of an annuity whose payments follow an arithmetic progression.

Usage

```
arithmetic_annuity_pv_tbl(
  .data = NULL,
  amount = NULL,
  step = NULL,
  n = NULL,
  i = NULL,
  timing = c("immediate", "due", "vencida", "anticipada"),
  perpetuity = FALSE,
  col_amount = "amount",
  col_step = "step",
  col_n = "n",
  col_i = "i",
  .out = "pv",
  .keep = c("all", "used", "none"),
  .na = c("propagate", "error", "drop")
)
```

Arguments

`.data`	A data.frame or tibble. If `NULL`, inputs must be supplied directly as scalars or equal-length vectors.
`amount`	Numeric first payment when `.data = NULL`.

step	Numeric arithmetic increment per payment period when .data = NULL.
n	Number of payments when .data = NULL and perpetuity = FALSE.
i	Effective interest rate per payment period when .data = NULL.
timing	Payment timing. Accepted values are "immediate", "due", "vencida", and "anticipada".
perpetuity	Logical; if TRUE, computes a perpetuity instead of a finite annuity.
col_amount	Name of the first-payment column.
col_step	Name of the arithmetic-step column.
col_n	Name of the number-of-payments column.
col_i	Name of the interest-rate column.
.out	Name of the output column containing present value.
.keep	One of "all", "used", or "none".
.na	NA handling policy: "propagate", "error", or "drop".

Details

The first payment is amount, and each subsequent payment changes by step. The annuity may be either:

- annuity-immediate (timing = "immediate" or "vencida"),
- annuity-due (timing = "due" or "anticipada"),
- finite (perpetuity = FALSE),
- or perpetual (perpetuity = TRUE).

This is a tibble-first mutate-style function: each input row is one case.

Assumptions:

- Time is discrete.
- i is the effective interest rate per payment period.
- If perpetuity = FALSE, n is the number of payments.
- If perpetuity = TRUE, n is ignored.
- All payments must remain nonnegative: for finite annuities the final payment amount + (n - 1) * step must be >= 0; for perpetuities, step >= 0 is required.

For a finite annuity-immediate with n payments, first payment P, arithmetic step Q, and effective rate i per period:

$$PV = P\, a_{\overline{n|}} + Q\, \frac{a_{\overline{n|}} - n v^n}{i}$$

where $a_{\overline{n|}} = (1 - v^n)/i$ and $v = 1/(1 + i)$.

For an annuity-due, replace $a_{\overline{n|}}$ by $\ddot{a}_{\overline{n|}} = (1+i)a_{\overline{n|}}$ and divide the step component by $d = i/(1+i)$ instead of i.

For a perpetuity-immediate ($i > 0$):

$$PV = \frac{P}{i} + \frac{Q}{i^2}.$$

For a perpetuity-due:

$$PV = \frac{P}{d} + \frac{Q}{i\,d} = (1+i)\left(\frac{P}{i} + \frac{Q}{i^2}\right).$$

When $i = 0$ (finite case only), the present value simplifies to:

$$PV = nP + Q\frac{n(n-1)}{2}.$$

Value

A tibble with a new numeric column named by `.out`.

References

Marcel B. Finan, *A Basic Course in the Theory of Interest and Derivatives Markets: A Preparation for the Actuarial Exam FM/2*.

Kellison, S. G. *The Theory of Interest*.

See Also

`arithmetic_annuity_av_tbl`, `a_angle`, `s_angle`

Other annuities: `Da_angle()`, `Da_angle_tbl()`, `Ia_angle()`, `Ia_angle_tbl()`, `a_angle()`, `a_angle_tbl()`, `arithmetic_annuity_av_tbl()`, `ga_angle()`, `ga_angle_tbl()`, `gs_angle()`, `gs_angle_tbl()`, `s_angle()`, `s_angle_tbl()`

Examples

```
# Simple example: finite annuity-immediate
arithmetic_annuity_pv_tbl(
  amount = 100,
  step = 5,
  n = 5,
  i = 0.05,
  timing = "immediate"
)

# Medium example: finite annuity-due for multiple rows
cases <- tibble::tibble(
  amount = c(100, 200, 50),
  step   = c(5, 10, 0),
  n      = c(5, 4, 10),
  i      = c(0.05, 0.04, 0.03)
)

arithmetic_annuity_pv_tbl(
  cases,
  timing = "due",
  perpetuity = FALSE,
  .out = "pv_due"
)
```

```
# Perpetuity-due
arithmetic_annuity_pv_tbl(
  amount = 1000,
  step = 40,
  i = 0.10,
  timing = "due",
  perpetuity = TRUE
)
```

---

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,
  type = "effective",
  m = 1L,
  deferral_years = 0,
  timing = "immediate",
  perpetuity = FALSE
)
```

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.
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.

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 only, 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

Numeric vector of annuity factors.

See Also

`a_angle_tbl`, `s_angle`, `Da_angle`, `Ia_angle`, `standardize_interest`, `present_value`

Other annuities: `Da_angle()`, `Da_angle_tbl()`, `Ia_angle()`, `Ia_angle_tbl()`, `a_angle_tbl()`, `arithmetic_annuity_av_tbl()`, `arithmetic_annuity_pv_tbl()`, `ga_angle()`, `ga_angle_tbl()`, `gs_angle()`, `gs_angle_tbl()`, `s_angle()`, `s_angle_tbl()`

Examples

```
# Simple scalar examples
a_angle(n_years = 10, rate = 0.05, type = "effective")
a_angle(n_years = 10, rate = 0.06, type = "nominal_interest", m = 12,
        payments_per_year = 12)
a_angle(n_years = 15, rate = 0.04, type = "force", timing = "continuous")

# Medium 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),
  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)
)

# Use inside a data pipeline
if (requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  contracts <- tibble::tibble(
    n_years = c(10, 15, 20),
    rate = c(0.05, 0.06, 0.04),
    type = c("effective", "nominal_interest", "force"),
    m = c(1, 12, 1),
    payments_per_year = c(1, 12, NA),
    deferral_years = c(0, 0, 3),
    timing = c("immediate", "immediate", "continuous"),
    perpetuity = c(FALSE, FALSE, FALSE)
  )

  dplyr::mutate(
    contracts,
    factor = a_angle(
      n_years = n_years,
      payments_per_year = dplyr::coalesce(payments_per_year, 1L),
      rate = rate,
      type = type,
      m = m,
      deferral_years = deferral_years,
      timing = timing,
      perpetuity = perpetuity
    )
  )
}
```

`a_angle_tbl` *Level annuity details in tibble form*

Description

Computes the actuarial present value factor for a level annuity and returns a tibble with the main input values, implied rates, annuity factor, payment amount, and present value.

Usage

```
a_angle_tbl(
```

```
  n_years = NULL,
  payments_per_year = 1L,
  rate,
  type = "effective",
  m = 1L,
  deferral_years = 0,
  timing = "immediate",
  payment = 1,
  perpetuity = FALSE
)
```

Arguments

`n_years`	Numeric vector of payment durations in years.
`payments_per_year`	Positive integer vector giving the number of discrete payments per year. Ignored for continuous annuities.
`rate`	Numeric vector of rate values.
`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.
`timing`	Character vector. One of `"immediate"`, `"due"`, or `"continuous"`.
`payment`	Numeric vector of level payment amounts.
`perpetuity`	Logical vector. If `TRUE`, computes the perpetuity factor.

Details

This is a reporting wrapper around `a_angle`. The annuity factor assumes unit payments. The present value is then computed as

$$PV = R \times a$$

where R is the payment amount and a is the annuity factor.

Value

A tibble with columns:

n_years Payment duration in years.

payments_per_year Number of payments per year.

deferral_years Deferral period in years.

timing Payment timing convention.

perpetuity Whether the annuity is perpetual.

rate_input Original supplied rate.

rate_type Type of supplied rate.

m Compounding frequency for nominal rates.

i_effective Equivalent annual effective rate.

delta Equivalent force of interest.

i_period Equivalent per-payment effective rate for discrete annuities.

v_period Equivalent per-payment discount factor for discrete annuities.

n_periods Number of payment periods for finite discrete annuities.

deferral_periods Number of deferred periods for discrete annuities.

annuity_factor Computed annuity factor.

payment Level payment amount.

present_value Present value of the annuity.

See Also

a_angle, s_angle, standardize_interest

Other annuities: Da_angle(), Da_angle_tbl(), Ia_angle(), Ia_angle_tbl(), a_angle(), arithmetic_annuity_av_tbl(), arithmetic_annuity_pv_tbl(), ga_angle(), ga_angle_tbl(), gs_angle(), gs_angle_tbl(), s_angle(), s_angle_tbl()

Examples

```
# Simple scalar example
a_angle_tbl(n_years = 10, rate = 0.05, payment = 1000)

# Medium vectorized example
a_angle_tbl(
  n_years = c(10, 15, 20),
  payments_per_year = c(1, 12, 1),
  rate = c(0.05, 0.06, 0.04),
  type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  deferral_years = c(0, 0, 2),
  timing = c("immediate", "immediate", "continuous"),
  payment = c(1000, 200, 5000),
  perpetuity = c(FALSE, FALSE, FALSE)
)
```

---

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,
  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. 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.
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 final maturity. If NULL, defaults to face.
tol	Numeric scalar. Tolerance used in alignment checks.
check	Logical scalar. If TRUE, performs input validation.

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).

Yield input conventions:

- If `y_effective_per_period` is supplied, it takes precedence over `y_rate`, `y_type`, and `y_m`, and 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 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

Numeric vector of book values, one for each `valuation_time`.

See Also

`bond_book_value_tbl`, `bond_price`, `bond_cash_flows`, `bond_duration`, `bond_convexity`

Other bonds: `bond_book_value_tbl()`, `bond_callable_price()`, `bond_callable_price_tbl()`, `bond_convexity()`, `bond_duration()`, `bond_price()`, `bond_ytm()`, `portfolio_convexity_tbl()`, `portfolio_duration_tbl()`

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,
  y_rate = 0.06,
  y_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,
  y_rate = 0.06,
  y_type = "effective"
)

# 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,
  y_effective_per_period = 0.03
)
```

---

bond_book_value_tbl *Book value table of a level coupon bond at coupon dates*

Description

Computes the book value of a level coupon bond at one or more coupon dates and returns the results in tibble form.

Usage

```
bond_book_value_tbl(
  face,
  coupon_rate,
  years_to_maturity,
  valuation_time,
  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. Final maturity in years.
valuation_time	Numeric vector. Valuation time(s) 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 final maturity. If NULL, defaults to face.
tol	Numeric scalar. Tolerance used in alignment checks.
check	Logical scalar. If TRUE, performs input validation.

Details

This is a reporting wrapper around the book-value calculation. The book value is interpreted prospectively and is evaluated immediately after any coupon due at the valuation time has been paid.

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).

Yield input conventions:

- If y_effective_per_period is supplied, it takes precedence over y_rate, y_type, and y_m, and 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.

Value

A tibble with columns:

valuation_time Valuation time in years.

valuation_period Valuation period index on the coupon grid.

book_value Prospective book value at the valuation time.

yield_per_period Effective yield per coupon period used in valuation.

yield_effective_annual Equivalent annual effective yield.

coupons_per_year Coupon frequency.

face Face value of the bond.

coupon_rate Annual coupon rate.

years_to_maturity Final maturity in years.

redemption Final redemption value.

See Also

bond_book_value, bond_price, bond_cash_flows, bond_duration, bond_convexity

Other bonds: bond_book_value(), bond_callable_price(), bond_callable_price_tbl(), bond_convexity(), bond_duration(), bond_price(), bond_ytm(), portfolio_convexity_tbl(), portfolio_duration_tbl()

Examples

```
bond_book_value_tbl(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 5,
  valuation_time = c(0, 1, 2, 3, 4, 5),
  coupons_per_year = 1,
  y_rate = 0.06,
  y_type = "effective"
)
```

---

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,
  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. Final maturity in years. Must be strictly positive (a callable bond needs at least one call date before maturity).
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.
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 final maturity. If NULL, defaults to face.
tol	Numeric scalar. Tolerance used in alignment checks.
check	Logical scalar. If TRUE, performs input validation.

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 (dirty price, no accrued interest).

Yield input conventions:

- If `y_effective_per_period` is supplied, it takes precedence over `y_rate`, `y_type`, and `y_m`, and 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 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

$$\min_j P_j(y)$$

that is, the smallest price across all redemption scenarios.

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

Value

Numeric scalar: the worst-case callable-bond price consistent with the target yield.

See Also

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

Other bonds: `bond_book_value()`, `bond_book_value_tbl()`, `bond_callable_price_tbl()`, `bond_convexity()`, `bond_duration()`, `bond_price()`, `bond_ytm()`, `portfolio_convexity_tbl()`, `portfolio_duration_tbl()`

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),
  y_rate = 0.06,
  y_type = "effective"
)

# 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),
  y_effective_per_period = 0.028
)
```

`bond_callable_price_tbl` *Callable-bond pricing table at a target minimum yield*

Description

Computes the price of a callable bond under each possible redemption scenario implied by the call schedule and final maturity.

Usage

```
bond_callable_price_tbl(
  face,
  coupon_rate,
  years_to_maturity,
  coupons_per_year = 1L,
  call_times,
  call_prices,
  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. Final maturity in years.
`coupons_per_year`	Positive integer. Number of coupon payments per year.
`call_times`	Numeric vector of callable times in years.

call_prices	Numeric vector of call prices corresponding to call_times.
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 final maturity. If NULL, defaults to face.
tol	Numeric scalar. Tolerance used in alignment checks.
check	Logical scalar. If TRUE, performs input validation.

Details

This is a reporting wrapper around the callable-bond pricing logic. It returns one row per scenario and identifies the worst-case scenario for the investor, that is, the scenario producing the smallest price at the target 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 (dirty price, no accrued interest).

Yield input conventions:

- If y_effective_per_period is supplied, it takes precedence over y_rate, y_type, and y_m, and 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.

This function evaluates the callable bond under:

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

For each scenario, the price consistent with the target yield is computed. The column is_worst_case marks the scenario(s) producing the smallest price, which corresponds to the least favorable redemption scenario for the investor.

Value

A tibble with columns:

scenario_id Scenario index.

scenario_type `"call"` or `"maturity"`.

scenario_time Redemption time in years for the scenario.

redemption_value Call price or final redemption value.

price_at_target_yield Price consistent with the target yield under that scenario.

is_worst_case Logical flag indicating the worst-case scenario(s).

yield_per_period Effective yield per coupon period used in pricing.

yield_effective_annual Equivalent annual effective yield.

coupons_per_year Coupon frequency.

face Face value of the bond.

coupon_rate Annual coupon rate.

years_to_maturity Final maturity in years.

See Also

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

Other bonds: `bond_book_value()`, `bond_book_value_tbl()`, `bond_callable_price()`, `bond_convexity()`, `bond_duration()`, `bond_price()`, `bond_ytm()`, `portfolio_convexity_tbl()`, `portfolio_duration_tbl()`

Examples

```
bond_callable_price_tbl(
  face = 100,
  coupon_rate = 0.08,
  years_to_maturity = 10,
  coupons_per_year = 2,
  call_times = c(5, 7),
  call_prices = c(105, 102),
  y_rate = 0.06,
  y_type = "effective"
)

bond_callable_price_tbl(
  face = 1000,
  coupon_rate = 0.05,
  years_to_maturity = 12,
  coupons_per_year = 2,
  call_times = c(4, 8),
  call_prices = c(1030, 1015),
  y_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_book_value_tbl()`, `bond_callable_price()`, `bond_callable_price_tbl()`, `bond_duration()`, `bond_price()`, `bond_ytm()`, `portfolio_convexity_tbl()`, `portfolio_duration_tbl()`

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_book_value_tbl(), bond_callable_price(), bond_callable_price_tbl(), bond_convexity(), bond_price(), bond_ytm(), portfolio_convexity_tbl(), portfolio_duration_tbl()

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_book_value_tbl()`, `bond_callable_price()`, `bond_callable_price_tbl()`, `bond_convexity()`, `bond_duration()`, `bond_ytm()`, `portfolio_convexity_tbl()`, `portfolio_duration_tbl()`

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_book_value_tbl(), bond_callable_price(), bond_callable_price_tbl(), bond_convexity(), bond_duration(), bond_price(), portfolio_convexity_tbl(), portfolio_duration_tbl()

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
)
```

Da_angle *Decreasing annuity factor (Da-angle-n)*

Description

Computes the actuarial present value factor for a decreasing annuity.

Usage

```
Da_angle(
  n_years,
  payments_per_year = 1L,
  rate,
  type = "effective",
  m = 1L,
  deferral_years = 0,
  timing = "immediate"
)
```

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.
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".

Details

The payment pattern is arithmetic decreasing. For a unit pattern, the payments over n periods are:

$$n, n-1, \ldots, 2, 1.$$

Supported timing conventions:

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

For discrete payments with payments_per_year = k, the total number of payment periods is nk. Deferral is supported through deferral_years; for discrete annuities, the deferral must align with the payment grid.

The input interest rate may be supplied as:

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

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

Let i_p be the effective rate per payment period, $v_p = (1+i_p)^{-1}$, and let n be the total number of payment periods. Then the decreasing annuity-immediate factor is

$$(Da)_{\overline{n|}} = \sum_{k=1}^{n} (n+1-k) v_p^k.$$

For $i_p \neq 0$, a closed-form expression is

$$(Da)_{\overline{n|}} = \frac{n}{i_p} - \frac{1}{i_p^2} + \frac{v_p^n}{i_p^2}.$$

For $i_p = 0$, the limit is

$$(Da)_{\overline{n|}} = \frac{n(n+1)}{2}.$$

The annuity-due version is obtained by multiplying the immediate factor by $1 + i_p$. Deferred versions are obtained by multiplying by v_p^h, where h is the number of deferred periods.

A useful identity relates the decreasing and increasing annuity factors:

$$(Da)_{\overline{n|}} + (Ia)_{\overline{n|}} = (n+1)\, a_{\overline{n|}}.$$

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

Value

Numeric vector of decreasing annuity factors.

See Also

Da_angle_tbl, Ia_angle, a_angle, s_angle, standardize_interest

Other annuities: Da_angle_tbl(), Ia_angle(), Ia_angle_tbl(), a_angle(), a_angle_tbl(), arithmetic_annuity_av_tbl(), arithmetic_annuity_pv_tbl(), ga_angle(), ga_angle_tbl(), gs_angle(), gs_angle_tbl(), s_angle(), s_angle_tbl()

Examples

```
# Simple scalar example
Da_angle(
  n_years = 10,
  rate = 0.05,
  type = "effective",
  timing = "immediate"
)

# Medium vectorized example
Da_angle(
  n_years = c(10, 20),
  payments_per_year = c(1, 12),
  rate = c(0.05, 0.08),
  type = c("effective", "nominal_interest"),
  m = c(1, 12),
  deferral_years = c(0, 2),
  timing = c("immediate", "due")
)
```

Da_angle_tbl *Decreasing annuity details in tibble form*

Description

Computes the actuarial present value factor for a decreasing annuity and returns a tibble with the main input values, implied rates, decreasing annuity factor, payment scale, and present value.

Usage

```
Da_angle_tbl(
  n_years,
  payments_per_year = 1L,
  rate,
  type = "effective",
  m = 1L,
  deferral_years = 0,
  timing = "immediate",
  payment = 1
)
```

Arguments

n_years	Numeric vector of payment durations in years.
payments_per_year	Positive integer vector giving the number of payments per year.
rate	Numeric vector of rate values.
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.
timing	Character vector. One of "immediate" or "due".
payment	Numeric vector of payment scale factors.

Details

This is a reporting wrapper around Da_angle. The decreasing annuity factor assumes a unit pattern. The present value is then computed as

$$PV = R \times (Da)$$

where R is the payment scale.

Value

A tibble with columns:

n_years Payment duration in years.

payments_per_year Number of payments per year.

deferral_years Deferral period in years.

timing Payment timing convention.

rate_input Original supplied rate.

rate_type Type of supplied rate.

m Compounding frequency for nominal rates.

i_effective Equivalent annual effective rate.

delta Equivalent force of interest.

i_period Equivalent per-payment effective rate.

v_period Equivalent per-payment discount factor.

n_periods Number of payment periods.

deferral_periods Number of deferred periods.

Da_factor Computed decreasing annuity factor.

payment Payment scale factor.

present_value Present value of the decreasing annuity.

See Also

Da_angle, Ia_angle_tbl, a_angle_tbl, standardize_interest

Other annuities: Da_angle(), Ia_angle(), Ia_angle_tbl(), a_angle(), a_angle_tbl(), arithmetic_annuity_av_tbl(), arithmetic_annuity_pv_tbl(), ga_angle(), ga_angle_tbl(), gs_angle(), gs_angle_tbl(), s_angle(), s_angle_tbl()

Examples

```
# Simple scalar example
Da_angle_tbl(
  n_years = 10,
  rate = 0.05,
  type = "effective",
  payment = 100
)

# Medium vectorized example
Da_angle_tbl(
  n_years = c(10, 20),
  payments_per_year = c(1, 12),
  rate = c(0.05, 0.08),
  type = c("effective", "nominal_interest"),
  m = c(1, 12),
  deferral_years = c(0, 2),
  timing = c("immediate", "due"),
  payment = c(100, 50)
)
```

---

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, spot_rate, spot_type = "effective", spot_m = 1L)
```

Arguments

time	Numeric vector of times in years. Each value must be greater than or equal to 0.
spot_rate	Numeric vector of spot-rate values.
spot_type	Character vector indicating the spot-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
spot_m	Positive integer vector giving the compounding frequency for nominal spot-rate inputs.

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

Numeric vector of discount factors.

See Also

discount_factor_spot_tbl, standardize_interest, present_value

Other interest: discount_factor_spot_tbl(), forward_rate_tbl(), interest_equivalents(), standardize_interest(), yield_curve_tbl()

Examples

```
# Simple scalar example
discount_factor_spot(
  time = 3,
  spot_rate = 0.05,
  spot_type = "effective"
)
```

```
# Vectorized example
discount_factor_spot(
  time = c(1, 2, 3),
  spot_rate = c(0.05, 0.055, 0.06),
  spot_type = "effective"
)

# FM-style input with nominal annual interest
discount_factor_spot(
  time = 2,
  spot_rate = 0.08,
  spot_type = "nominal_interest",
  spot_m = 2
)
```

`discount_factor_spot_tbl` *Spot discount factor in tibble form*

Description

Computes the discount factor implied by a spot rate for a given time and returns a tibble with the main input values, the standardized annual effective rate, and the discount factor.

Usage

```
discount_factor_spot_tbl(time, spot_rate, spot_type = "effective", spot_m = 1L)
```

Arguments

`time`	Numeric vector of times in years. Each value must be greater than or equal to 0.
`spot_rate`	Numeric vector of spot-rate values.
`spot_type`	Character vector indicating the spot-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`spot_m`	Positive integer vector giving the compounding frequency for nominal spot-rate inputs.

Details

This is a reporting wrapper around `discount_factor_spot`.

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

A tibble with columns:

time Time in years.

spot_rate_input Original supplied spot rate.

spot_type Type of supplied spot rate.

spot_m Compounding frequency for nominal spot-rate inputs.

i_effective Equivalent annual effective spot rate.

discount_factor Computed discount factor.

See Also

discount_factor_spot, standardize_interest

Other interest: discount_factor_spot(), forward_rate_tbl(), interest_equivalents(), standardize_interest(), yield_curve_tbl()

Examples

```
# Simple scalar example
discount_factor_spot_tbl(
  time = 3,
  spot_rate = 0.05,
  spot_type = "effective"
)

# Vectorized example
discount_factor_spot_tbl(
  time = c(1, 2, 3),
  spot_rate = c(0.05, 0.055, 0.06),
  spot_type = "effective"
)

# FM-style input with nominal annual discount
discount_factor_spot_tbl(
  time = c(1, 2),
  spot_rate = c(0.04, 0.05),
  spot_type = "nominal_discount",
  spot_m = 2
)
```

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_tbl *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_tbl(
  .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()`, `discount_factor_spot_tbl()`, `interest_equivalents()`, `standardize_interest()`, `yield_curve_tbl()`

Examples

```
# Simple example: exact forward rate
forward_rate_tbl(
  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_tbl(
  curves,
  method = "linear",
  plot = TRUE
)
```

`future_value` *Future value of a single payment*

Description

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

Usage

```
future_value(C, rate, type = "effective", m = 1, t)
```

Arguments

`C`	Numeric vector of initial payment amounts.
`rate`	Numeric vector of rate values.
`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"`.
`t`	Numeric vector of times in years from valuation to accumulation.

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

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

This is the core numeric version of the calculation. It is designed to work naturally with vectors and with `dplyr::mutate()`.

Input vectors must have length 1 or a common length. Standard recycling is supported only under that rule.

Missing values are propagated. This function does not accept dates. If you need a tabular output with actuarial fields, use `future_value_tbl`.

Value

Numeric vector of future values.

See Also

standardize_interest, present_value, future_value_tbl

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

Examples

```
# Simple scalar example
future_value(C = 1000, rate = 0.08, type = "effective", t = 3)

# Medium vectorized example
future_value(
  C = c(1000, 2500, 4000),
  rate = c(0.08, 0.10, 0.12),
  type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  t = c(3, 5, 2)
)

# Use inside a data pipeline
if (requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  investments <- tibble::tibble(
    deposit = c(1000, 1500, 2000),
    rate    = c(0.08, 0.12, 0.09),
    type    = c("effective", "force", "nominal_interest"),
    m       = c(1, 1, 4),
    t       = c(2, 3, 5)
  )

  dplyr::mutate(
    investments,
    fv = future_value(C = deposit, rate = rate, type = type, m = m, t = t)
  )
}
```

---

future_value_tbl *Future value details in tibble form*

Description

Computes the future value of a payment and returns a tibble containing both the inputs and the actuarial quantities used in the calculation.

Usage

```
future_value_tbl(C, rate, type = "effective", m = 1, t)
```

Arguments

C	Numeric vector of initial payment amounts.
rate	Numeric vector of rate values.
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".
t	Numeric vector of times in years from valuation to accumulation.

Details

This is a reporting wrapper around future_value. It is useful when you want a tabular summary including the input rate specification, the annual effective rate, and the final future value.

This function follows the same recycling and validation rules as future_value. Input vectors must have length 1 or a common length.

Missing values are propagated.

Value

A tibble with columns:

C Initial payment amount.

t Time in years to accumulation.

rate_input Original supplied rate.

rate_type Type of supplied rate.

m Compounding frequency.

i_effective Equivalent annual effective interest rate.

future_value Computed future value.

See Also

future_value, standardize_interest, present_value

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

Examples

```
# Simple scalar example
future_value_tbl(C = 1000, rate = 0.08, type = "effective", t = 3)

# Medium vectorized example
future_value_tbl(
  C = c(1000, 2500, 4000),
  rate = c(0.08, 0.10, 0.12),
  type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
```

```
  t = c(3, 5, 2)
)

# Combine with dplyr
if (requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  investments <- tibble::tibble(
    deposit = c(1000, 1500, 2000),
    rate    = c(0.08, 0.12, 0.09),
    type    = c("effective", "force", "nominal_interest"),
    m       = c(1, 1, 4),
    t       = c(2, 3, 5)
  )

  future_value_tbl(
    C = investments$deposit,
    rate = investments$rate,
    type = investments$type,
    m = investments$m,
    t = investments$t
  )
}
```

`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(), future_value_tbl(), irr_flow(), irr_flow_multi(), plot_cash_flow(), present_value(), present_value_tbl(), 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)
)
```

---

ga_angle *Geometric annuity factor ga-angle-n*

Description

Computes the actuarial present value factor for a geometric annuity.

Usage

```
ga_angle(
  n_years = NULL,
  payments_per_year = 1L,
  rate,
  type = "effective",
  m = 1L,
  growth = 0,
  growth_type = "effective",
  growth_m = 1L,
  deferral_years = 0,
  timing = c("immediate", "due"),
  perpetuity = FALSE
)
```

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 payments per year.
rate	Numeric vector of annual rate values for discounting.
type	Character vector indicating the annual discount-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer vector giving the compounding frequency for nominal discount-rate inputs.
growth	Numeric vector of annual growth-rate values.
growth_type	Character vector indicating the annual 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.
timing	Character vector. One of "immediate" or "due".
perpetuity	Logical vector. If TRUE, computes the perpetuity factor.

Details

The payment pattern is geometric. For a unit first payment, the payments are:

- immediate: $1, (1+g_p), (1+g_p)^2, \ldots$
- due: same geometric pattern, but shifted one period earlier

where g_p is the effective growth rate per payment period.

Supported timing conventions:

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

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, where k is the number of payments per year.

The interest rate and the growth rate may each be supplied in FM-style notation:

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

Internally, both are first converted to annual effective rates and then to effective rates per payment 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:

$$ga_n = \sum_{t=1}^{n} \frac{(1+g_p)^{t-1}}{(1+i_p)^t}$$

If $i_p \neq g_p$, then

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

If $i_p = g_p$, the limiting formula is

$$ga_n = \frac{n}{1+i_p}$$

The annuity-due version is obtained by multiplying the immediate factor by $1 + i_p$. Deferred versions are obtained by multiplying by v_p^h, where h is the number of deferred periods.

For perpetuities, convergence requires $i_p > g_p$. In that case:

$$ga_\infty = \frac{1}{i_p - g_p}$$

and for due:

$$\ddot{ga}_\infty = \frac{1+i_p}{i_p - g_p}$$

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

Value

Numeric vector of geometric annuity factors.

See Also

ga_angle_tbl, a_angle, gs_angle, standardize_interest

Other annuities: Da_angle(), Da_angle_tbl(), Ia_angle(), Ia_angle_tbl(), a_angle(), a_angle_tbl(), arithmetic_annuity_av_tbl(), arithmetic_annuity_pv_tbl(), ga_angle_tbl(), gs_angle(), gs_angle_tbl(), s_angle(), s_angle_tbl()

Examples

```
# Simple scalar example
ga_angle(
  n_years = 10,
  rate = 0.05,
  type = "effective",
  growth = 0.02,
  growth_type = "effective",
  timing = "immediate"
)

# Growth = 0 collapses to a level annuity
ga_angle(
  n_years = 10,
  rate = 0.05,
  type = "effective",
  growth = 0,
  growth_type = "effective",
  timing = "immediate"
)

# Medium vectorized example
ga_angle(
  n_years = c(10, 20),
  payments_per_year = c(1, 12),
  rate = c(0.05, 0.08),
  type = c("effective", "nominal_interest"),
  m = c(1, 12),
  growth = c(0.02, 0.03),
  growth_type = c("effective", "effective"),
  growth_m = c(1, 1),
  deferral_years = c(0, 2),
  timing = c("immediate", "due")
)
```

---

ga_angle_tbl *Geometric annuity details in tibble form*

Description

Computes the actuarial present value factor for a geometric annuity and returns a tibble with the main input values, implied rates, factor, payment amount, and present value.

Usage

```
ga_angle_tbl(
  n_years = NULL,
  payments_per_year = 1L,
  rate,
  type = "effective",
  m = 1L,
  growth = 0,
  growth_type = "effective",
  growth_m = 1L,
  deferral_years = 0,
  timing = c("immediate", "due"),
  payment = 1,
  perpetuity = FALSE
)
```

Arguments

n_years	Numeric vector of payment durations in years.
payments_per_year	Positive integer vector giving the number of payments per year.
rate	Numeric vector of annual rate values for discounting.
type	Character vector indicating the annual discount-rate type.
m	Positive integer vector giving the compounding frequency for nominal discount-rate inputs.
growth	Numeric vector of annual growth-rate values.
growth_type	Character vector indicating the annual growth-rate type.
growth_m	Positive integer vector giving the compounding frequency for nominal growth-rate inputs.
deferral_years	Numeric vector of deferral times in years.
timing	Character vector. One of "immediate" or "due".
payment	Numeric vector giving the first payment of the geometric sequence.
perpetuity	Logical vector. If TRUE, computes the perpetuity factor.

Details

This is a reporting wrapper around ga_angle. The factor assumes that the first payment equals 1. The actual present value is then computed as

$$PV = P_1 \times ga$$

where P_1 is the first payment of the geometric sequence.

Value

A tibble with columns:

n_years Payment duration in years.

payments_per_year Number of payments per year.

deferral_years Deferral period in years.

timing Payment timing convention.

perpetuity Whether the annuity is perpetual.

rate_input Original supplied discount rate.

rate_type Type of supplied discount rate.

m Compounding frequency for nominal discount-rate inputs.

growth_input Original supplied growth rate.

growth_type Type of supplied growth rate.

growth_m Compounding frequency for nominal growth-rate inputs.

i_effective Equivalent annual effective discount rate.

g_effective Equivalent annual effective growth rate.

i_period Equivalent per-payment discount rate.

g_period Equivalent per-payment growth rate.

v_period Equivalent per-payment discount factor.

n_periods Number of payment periods for finite annuities.

deferral_periods Number of deferred periods.

ga_factor Computed geometric annuity factor.

payment First payment of the geometric sequence.

present_value Present value of the geometric annuity.

See Also

ga_angle, a_angle_tbl, standardize_interest

Other annuities: Da_angle(), Da_angle_tbl(), Ia_angle(), Ia_angle_tbl(), a_angle(), a_angle_tbl(), arithmetic_annuity_av_tbl(), arithmetic_annuity_pv_tbl(), ga_angle(), gs_angle(), gs_angle_tbl(), s_angle(), s_angle_tbl()

Examples

```
# Simple scalar example
ga_angle_tbl(
  n_years = 10,
  rate = 0.05,
  type = "effective",
  growth = 0.02,
  growth_type = "effective",
  payment = 100
)
```

```
# Medium vectorized example
ga_angle_tbl(
  n_years = c(10, 20),
  payments_per_year = c(1, 12),
  rate = c(0.05, 0.08),
  type = c("effective", "nominal_interest"),
  m = c(1, 12),
  growth = c(0.02, 0.03),
  growth_type = c("effective", "effective"),
  growth_m = c(1, 1),
  deferral_years = c(0, 2),
  timing = c("immediate", "due"),
  payment = c(100, 50)
)
```

---

gs_angle *Geometric annuity accumulation factor gs-angle-n*

Description

Computes the actuarial accumulation factor for a geometric annuity.

Usage

```
gs_angle(
  n_years,
  payments_per_year = 1L,
  rate,
  type = "effective",
  m = 1L,
  growth = 0,
  growth_type = "effective",
  growth_m = 1L,
  deferral_years = 0,
  timing = c("immediate", "due")
)
```

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 annual rate values for accumulation.
type	Character vector indicating the annual accumulation-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".

`m`	Positive integer vector giving the compounding frequency for nominal accumulation-rate inputs.
`growth`	Numeric vector of annual growth-rate values.
`growth_type`	Character vector indicating the annual 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. Under the adopted horizon convention, this is metadata only for accumulation factors.
`timing`	Character vector. One of `"immediate"` or `"due"`.

Details

The payment pattern is geometric. For a unit first payment, the payments are:

- immediate: $1, (1+g_p), (1+g_p)^2, \ldots$
- due: same geometric pattern, but shifted one period earlier

where g_p is the effective growth rate per payment period.

Supported timing conventions:

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

Horizon convention: the future value is measured at the standard terminal horizon for the annuity accumulation factor. 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 interest rate and the growth rate may each be supplied in FM-style notation:

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

Internally, both are first converted to annual effective rates and then to effective rates per payment period.

Let i_p be the effective accumulation rate per payment period, and g_p the effective growth rate per payment period.

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

$$gs_n = \sum_{t=1}^{n} (1+g_p)^{t-1}(1+i_p)^{n-t}$$

If $i_p \neq g_p$, then

$$gs_n = \frac{(1+i_p)^n - (1+g_p)^n}{i_p - g_p}$$

If $i_p = g_p$, the limiting formula is

$$gs_n = n(1+i_p)^{n-1}$$

The annuity-due version is obtained by multiplying the immediate factor by $1 + i_p$.

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

Geometric perpetuities are not supported by this accumulation-factor function.

Value

Numeric vector of geometric accumulation factors.

See Also

gs_angle_tbl, s_angle, ga_angle, standardize_interest

Other annuities: Da_angle(), Da_angle_tbl(), Ia_angle(), Ia_angle_tbl(), a_angle(), a_angle_tbl(), arithmetic_annuity_av_tbl(), arithmetic_annuity_pv_tbl(), ga_angle(), ga_angle_tbl(), gs_angle_tbl(), s_angle(), s_angle_tbl()

Examples

```
# Simple scalar example
gs_angle(
  n_years = 10,
  rate = 0.05,
  type = "effective",
  growth = 0.02,
  growth_type = "effective",
  timing = "immediate"
)

# Growth = 0 collapses to a level accumulation factor
gs_angle(
  n_years = 10,
  rate = 0.05,
  type = "effective",
  growth = 0,
  growth_type = "effective",
  timing = "immediate"
)

# Medium vectorized example
gs_angle(
  n_years = c(10, 20),
  payments_per_year = c(1, 12),
  rate = c(0.05, 0.08),
  type = c("effective", "nominal_interest"),
  m = c(1, 12),
```

```
  growth = c(0.02, 0.03),
  growth_type = c("effective", "effective"),
  growth_m = c(1, 1),
  deferral_years = c(0, 2),
  timing = c("immediate", "due")
)
```

---

gs_angle_tbl *Geometric annuity accumulation details in tibble form*

Description

Computes the actuarial accumulation factor for a geometric annuity and returns a tibble with the main input values, implied rates, factor, payment amount, and future value.

Usage

```
gs_angle_tbl(
  n_years,
  payments_per_year = 1L,
  rate,
  type = "effective",
  m = 1L,
  growth = 0,
  growth_type = "effective",
  growth_m = 1L,
  deferral_years = 0,
  timing = c("immediate", "due"),
  payment = 1
)
```

Arguments

n_years	Numeric vector of payment durations in years.
payments_per_year	Positive integer vector giving the number of payments per year.
rate	Numeric vector of annual rate values for accumulation.
type	Character vector indicating the annual accumulation-rate type.
m	Positive integer vector giving the compounding frequency for nominal accumulation-rate inputs.
growth	Numeric vector of annual growth-rate values.
growth_type	Character vector indicating the annual growth-rate type.
growth_m	Positive integer vector giving the compounding frequency for nominal growth-rate inputs.

`deferral_years`	Numeric vector of deferral times in years.
`timing`	Character vector. One of `"immediate"` or `"due"`.
`payment`	Numeric vector giving the first payment of the geometric sequence.

Details

This is a reporting wrapper around `gs_angle`. The factor assumes that the first payment equals 1. The actual future value is then computed as

$$FV = P_1 \times gs$$

where P_1 is the first payment of the geometric sequence.

Under the adopted horizon convention, `deferral_years` is metadata only and does not affect the accumulation factor.

Value

A tibble with columns:

n_years Payment duration in years.

payments_per_year Number of payments per year.

deferral_years Deferral period in years.

timing Payment timing convention.

rate_input Original supplied accumulation rate.

rate_type Type of supplied accumulation rate.

m Compounding frequency for nominal accumulation-rate inputs.

growth_input Original supplied growth rate.

growth_type Type of supplied growth rate.

growth_m Compounding frequency for nominal growth-rate inputs.

i_effective Equivalent annual effective accumulation rate.

g_effective Equivalent annual effective growth rate.

i_period Equivalent per-payment accumulation rate.

g_period Equivalent per-payment growth rate.

n_periods Number of payment periods.

deferral_periods Number of deferred periods.

gs_factor Computed geometric accumulation factor.

payment First payment of the geometric sequence.

future_value Future value of the geometric annuity.

See Also

`gs_angle`, `s_angle`, `ga_angle_tbl`, `standardize_interest`

Other annuities: `Da_angle()`, `Da_angle_tbl()`, `Ia_angle()`, `Ia_angle_tbl()`, `a_angle()`, `a_angle_tbl()`, `arithmetic_annuity_av_tbl()`, `arithmetic_annuity_pv_tbl()`, `ga_angle()`, `ga_angle_tbl()`, `gs_angle()`, `s_angle()`, `s_angle_tbl()`

Examples

```
# Simple scalar example
gs_angle_tbl(
  n_years = 10,
  rate = 0.05,
  type = "effective",
  growth = 0.02,
  growth_type = "effective",
  payment = 100
)

# Medium vectorized example
gs_angle_tbl(
  n_years = c(10, 20),
  payments_per_year = c(1, 12),
  rate = c(0.05, 0.08),
  type = c("effective", "nominal_interest"),
  m = c(1, 12),
  growth = c(0.02, 0.03),
  growth_type = c("effective", "effective"),
  growth_m = c(1, 1),
  deferral_years = c(0, 2),
  timing = c("immediate", "due"),
  payment = c(100, 50)
)
```

Ia_angle *Increasing annuity factor (Ia-angle-n)*

Description

Computes the actuarial present value factor for an increasing annuity.

Usage

```
Ia_angle(
  n_years,
  payments_per_year = 1L,
  rate,
  type = "effective",
  m = 1L,
  deferral_years = 0,
  timing = "immediate"
)
```

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.
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".

Details

The payment pattern is arithmetic increasing. For a unit gradient, the payments over n periods are:

$$1, 2, \ldots, n-1, n.$$

Supported timing conventions:

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

For discrete payments with payments_per_year = k, the total number of payment periods is nk. Deferral is supported through deferral_years; for discrete annuities, the deferral must align with the payment grid.

The input interest rate may be supplied as:

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

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

Let i_p be the effective rate per payment period, $v_p = (1 + i_p)^{-1}$, and let n be the total number of payment periods. Also let

$$a_n = \frac{1 - v_p^n}{i_p}$$

Then the increasing annuity-immediate factor is

$$(Ia)_n = \frac{a_n - n v_p^n}{i_p}$$

For $i_p = 0$, the limit is

$$(Ia)_n = \frac{n(n+1)}{2}$$

The annuity-due version is obtained by multiplying the immediate factor by $1 + i_p$. Deferred versions are obtained by multiplying by v_p^h, where h is the number of deferred periods.

A useful identity relates the increasing and decreasing annuity factors:

$$(Ia)_n + (Da)_n = (n+1)\, a_n$$

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

Value

Numeric vector of increasing annuity factors.

See Also

Ia_angle_tbl, Da_angle, a_angle, s_angle, standardize_interest

Other annuities: Da_angle(), Da_angle_tbl(), Ia_angle_tbl(), a_angle(), a_angle_tbl(), arithmetic_annuity_av_tbl(), arithmetic_annuity_pv_tbl(), ga_angle(), ga_angle_tbl(), gs_angle(), gs_angle_tbl(), s_angle(), s_angle_tbl()

Examples

```
# Simple scalar example
Ia_angle(
  n_years = 10,
  rate = 0.05,
  type = "effective",
  timing = "immediate"
)

# Medium vectorized example
Ia_angle(
  n_years = c(10, 20),
  payments_per_year = c(1, 12),
  rate = c(0.05, 0.08),
  type = c("effective", "nominal_interest"),
  m = c(1, 12),
  deferral_years = c(0, 2),
  timing = c("immediate", "due")
)
```

Ia_angle_tbl *Increasing annuity details in tibble form*

Description

Computes the actuarial present value factor for an increasing annuity and returns a tibble with the main input values, implied rates, increasing annuity factor, payment scale, and present value.

Usage

```
Ia_angle_tbl(
  n_years,
  payments_per_year = 1L,
  rate,
  type = "effective",
  m = 1L,
  deferral_years = 0,
  timing = "immediate",
  payment = 1
)
```

Arguments

`n_years`	Numeric vector of payment durations in years.
`payments_per_year`	Positive integer vector giving the number of payments per year.
`rate`	Numeric vector of rate values.
`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.
`timing`	Character vector. One of `"immediate"` or `"due"`.
`payment`	Numeric vector of payment scale factors.

Details

This is a reporting wrapper around `Ia_angle`. The increasing annuity factor assumes a unit gradient. The present value is then computed as

$$PV = R \times (Ia)$$

where R is the payment scale.

Value

A tibble with columns:

n_years Payment duration in years.

payments_per_year Number of payments per year.

deferral_years Deferral period in years.

timing Payment timing convention.

rate_input Original supplied rate.

rate_type Type of supplied rate.

m Compounding frequency for nominal rates.

i_effective Equivalent annual effective rate.

delta Equivalent force of interest.

i_period Equivalent per-payment effective rate.

v_period Equivalent per-payment discount factor.

n_periods Number of payment periods.

deferral_periods Number of deferred periods.

Ia_factor Computed increasing annuity factor.

payment Payment scale factor.

present_value Present value of the increasing annuity.

See Also

Ia_angle, Da_angle_tbl, a_angle_tbl, standardize_interest

Other annuities: Da_angle(), Da_angle_tbl(), Ia_angle(), a_angle(), a_angle_tbl(), arithmetic_annuity_av_tbl(), arithmetic_annuity_pv_tbl(), ga_angle(), ga_angle_tbl(), gs_angle(), gs_angle_tbl(), s_angle(), s_angle_tbl()

Examples

```
# Simple scalar example
Ia_angle_tbl(
  n_years = 10,
  rate = 0.05,
  type = "effective",
  payment = 100
)

# Medium vectorized example
Ia_angle_tbl(
  n_years = c(10, 20),
  payments_per_year = c(1, 12),
  rate = c(0.05, 0.08),
  type = c("effective", "nominal_interest"),
  m = c(1, 12),
  deferral_years = c(0, 2),
  timing = c("immediate", "due"),
  payment = c(100, 50)
)
```

---

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_multi *Actuarial present value of a multi-life insurance (N independent lives)*

Description

Computes the APV of a discrete life insurance contingent on a multi-life status built from multiple independent lives. The benefit is paid at the end of the year of death of the status.

Usage

```
insurance_multi(
  lt,
  ages,
  i,
  product = c("whole", "term", "endowment", "pure_endowment"),
  cohort = c("first", "last"),
  benefit = 1,
  n = NULL,
  m = 0L,
  tidy = FALSE,
  check = TRUE
)
```

Arguments

`lt`	Life table data frame with columns `x` and `lx`.
`ages`	Integer vector of issue ages for the lives.
`i`	Annual effective interest rate (must be `> -1`).
`product`	Insurance type: `"whole"`, `"term"`, `"endowment"`, or `"pure_endowment"`.
`cohort`	Status definition: `"first"` (joint-life, first death) or `"last"` (last-survivor, second death).
`benefit`	Numeric scalar benefit amount (default `1`).
`n`	Integer term in years after deferment. Required for `"term"`, `"endowment"`, and `"pure_endowment"`.
`m`	Integer deferment in years (default `0`).
`tidy`	Logical. If `TRUE`, returns a one-row tibble.
`check`	Logical. If `TRUE`, performs basic input checks.

Details

The implementation uses the standard equivalence between insurance and annuity-due on the same status (generalizing Finan, Sections 27 and 37 to N-life statuses from Sections 56–59):

$$A_{\text{status}} = 1 - d\,\ddot{a}_{\text{status}}, \quad d = \frac{i}{1+i}.$$

Supported products:

- **Whole life**: $A = 1 - d\,\ddot{a}$
- **n-year term**: $A^1 = A_{\text{endow}} - {}_nE_{\text{status}}$
- **n-year endowment**: $A = 1 - d\,\ddot{a}_{\overline{n}|}$
- **Pure endowment**: ${}_nE = v^{m+n} \cdot P(\text{status alive at } m+n)$

Deferral by m years is applied as $v^m \cdot P(\text{status alive at } m) \times$ (value at shifted ages).

The annuity-due used internally is computed via `annuity_multi`.

Value

A numeric scalar APV, or a one-row tibble if `tidy = TRUE`.

See Also

`insurance_xy` for the optimized two-life version, `insurance_x` for single-life insurance, `annuity_multi` for multi-life annuity APVs, `Var_insurance_x` for insurance variance.

Examples

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

# Joint-life (first-death) whole life insurance
insurance_multi(lt, ages = c(60, 62), i = 0.05,
                product = "whole", cohort = "first")

# Last-survivor 10-year term, deferred 5 years
insurance_multi(lt, ages = c(60, 62), i = 0.05,
                product = "term", cohort = "last",
                n = 10, m = 5)

# Endowment: verify decomposition A = A^1 + nE
A_endow <- insurance_multi(lt, ages = c(60, 62), i = 0.05,
                           product = "endowment", cohort = "first",
                           n = 10)
A_term  <- insurance_multi(lt, ages = c(60, 62), i = 0.05,
                           product = "term", cohort = "first",
                           n = 10)
A_pe    <- insurance_multi(lt, ages = c(60, 62), i = 0.05,
                           product = "pure_endowment",
                           cohort = "first", n = 10)
c(endowment = A_endow, term_plus_pe = A_term + A_pe)

# Three lives
insurance_multi(lt, ages = c(60, 65, 70), i = 0.05,
                product = "whole", cohort = "first")

# Tidy output
insurance_multi(lt, ages = c(60, 62), i = 0.05,
                product = "term", cohort = "first",
                n = 10, tidy = TRUE)
```

`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, `Var_insurance_x` for variance of level insurance.

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(lt, x = 60, i = 0.05, n = 5, 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 (derived from annuity_x)*

Description

Computes the actuarial present value (APV) of a discrete life insurance with benefit 1 payable at the end of the year of death. The implementation uses the standard identities that express insurance values as functions of the annuity-due value (Finan, Sections 37 and 41).

Usage

```
insurance_x(
  lt,
  x,
  i,
  n = NULL,
  m = 0L,
  type = c("whole", "term", "endowment"),
  tidy = FALSE
)
```

Arguments

`lt`	A life table data frame. Must contain columns `x` and `lx`.
`x`	Integer actuarial age.
`i`	Effective annual interest rate (must be `> -1`).
`n`	Integer term in years. Required for `type = "term"` or `type = "endowment"`. Ignored for `type = "whole"`.
`m`	Integer deferral in years (default `0`).
`type`	Contract type: `"whole"`, `"term"`, or `"endowment"`.
`tidy`	Logical. If `TRUE`, returns a one-row tibble.

Details

Supported contracts:

- **Whole life insurance** (Finan, Sec. 37.1): $A_x = 1 - d\,\ddot{a}_x$
- **n-year term insurance** (Finan, Sec. 27): $A^1_{x:\overline{n}|} = 1 - d\,\ddot{a}_{x:\overline{n}|} - v^n\,{}_np_x$
- **n-year endowment insurance** (Finan, Sec. 26.3.2): $A_{x:\overline{n}|} = 1 - d\,\ddot{a}_{x:\overline{n}|}$

The endowment decomposes as (Finan, Example 26.15):

$$A_{x:\overline{n}|} = A^1_{x:\overline{n}|} + {}_nE_x$$

Deferral by m years is handled as: $v^m\,{}_mp_x \times$ value at age $x + m$.

This function calls `annuity_x` internally using annual payments (`k = 1`) and `timing = "due"` (annuity-due). Fractional payments and Woolhouse approximations are not used here because the identities above are stated for annual discrete contracts.

The key identity connecting annuities and insurance is (Finan, Sec. 37.1):

$$\ddot{a}_x = \frac{1 - A_x}{d}, \quad \text{i.e., } A_x = 1 - d\,\ddot{a}_x$$

where $d = i/(1+i)$ is the annual effective discount rate.

For m-thly and continuous insurance APVs under UDD, use the adjustment factor $i/i^{(m)}$ or i/δ (Finan, Sec. 30).

Value

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

See Also

`annuity_x` for the annuity-due values used internally, `premium_x` for benefit premiums ($P = A/\ddot{a}$), `Var_insurance_x` for the variance, `t_Ex` for pure endowments, `insurance_xy` for two-life insurance.

Examples

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

# Whole life insurance: A_60 = 1 - d * \ddot{a}_60
insurance_x(lt, x = 60, i = 0.06, type = "whole")

# 5-year term insurance: A^1_{60:5} (Finan, Sec. 27)
insurance_x(lt, x = 60, i = 0.06, n = 5, type = "term")

# 5-year endowment insurance: A_{60:5} = A^1 + 5_E_60
insurance_x(lt, x = 60, i = 0.06, n = 5, type = "endowment")
```

```
# Verify endowment decomposition (Finan, Sec. 26.3.2):
# A_{x:n} = A^1_{x:n} + nEx
A_term  <- insurance_x(lt, x = 60, i = 0.06, n = 5, type = "term")
A_endow <- insurance_x(lt, x = 60, i = 0.06, n = 5, type = "endowment")
nEx     <- (1.06)^(-5) * lt$lx[lt$x == 65] / lt$lx[lt$x == 60]
c(endowment = A_endow, term_plus_nEx = A_term + nEx)  # should match

# 2-year deferred whole life
insurance_x(lt, x = 60, i = 0.06, m = 2, type = "whole")

# Tidy output
insurance_x(lt, x = 60, i = 0.06, n = 5, type = "term", tidy = TRUE)
```

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

Description

Computes the APV of a discrete two-life insurance with benefit 1 payable at the end of the year of the triggering death, assuming independent lives. The implementation uses standard identities that express insurance values in terms of the annuity-due value (generalizing Finan, Sections 27 and 37 to two-life statuses from Sections 58–59).

Usage

```
insurance_xy(
  lt,
  x,
  y,
  i,
  n = NULL,
  m = 0L,
  type = c("whole", "term", "endowment"),
  cohort = c("first", "last"),
  tidy = FALSE
)
```

Arguments

`lt`	A life table data frame with columns `x` and `lx`.
`x`	Integer actuarial age for life 1 at issue.
`y`	Integer actuarial age for life 2 at issue.
`i`	Annual effective interest rate (must be > `-1`).
`n`	Integer term in years. Required for `type = "term"` or `type = "endowment"`. Ignored for `type = "whole"`.
`m`	Integer deferral in years (default `0`).

type	Insurance type: "whole", "term", or "endowment".
cohort	Status cohort: "first" (joint-life) or "last" (last-survivor).
tidy	Logical. If TRUE, returns a one-row tibble.

Details

Supported contracts:

- **Whole life**: $A_{xy} = 1 - d\,\ddot{a}_{xy}$
- **n-year term**: $A^1_{xy:\overline{n}|} = 1 - d\,\ddot{a}_{xy:\overline{n}|} - v^n\,{}_np_{xy}$
- **n-year endowment**: $A_{xy:\overline{n}|} = 1 - d\,\ddot{a}_{xy:\overline{n}|}$

The two-life cohort determines the status:

- cohort = "first": joint-life (triggers on first death)
- cohort = "last": last-survivor (triggers on second death)

Deferral by m years is applied as: $v^m\,{}_mp_{xy} \times$ (value at ages $x+m, y+m$).

This function calls annuity_xy internally using annual payments (k = 1, timing = "due", woolhouse = "none").

For the term insurance adjustment $v^n\,{}_np$, the survival probability is computed using t_pxy at integer time n (fractional-age assumption is irrelevant at integer times).

The endowment decomposes as:

$$A_{xy:\overline{n}|} = A^1_{xy:\overline{n}|} + {}_nE_{xy}$$

where ${}_nE_{xy} = v^n\,{}_np_{xy}$ is the two-life pure endowment.

Value

A single numeric APV value (per unit benefit), or a one-row tibble if tidy = TRUE.

See Also

annuity_xy for two-life annuity APVs, insurance_x for single-life insurance, insurance_multi for N-life insurance, premium_xy for two-life benefit premiums, t_pxy for two-life survival.

Examples

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

# Whole life joint-life insurance (Finan, Sec. 58)
insurance_xy(lt, x = 60, y = 62, i = 0.06,
             type = "whole", cohort = "first")

# 4-year term, joint-life
insurance_xy(lt, x = 60, y = 62, i = 0.06,
             n = 4, type = "term", 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(), discount_factor_spot_tbl(), forward_rate_tbl(), standardize_interest(), yield_curve_tbl()

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()`, `future_value_tbl()`, `fv_flow()`, `irr_flow_multi()`, `plot_cash_flow()`, `present_value()`, `present_value_tbl()`, `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()`, `future_value_tbl()`, `fv_flow()`, `irr_flow()`, `plot_cash_flow()`, `present_value()`, `present_value_tbl()`, `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:

- $T_x = \sum_{k \geq x} L_k$: 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
```

`md_convert_rates` *Convert between dependent and independent decrement rates*

Description

Converts cause-specific decrement rates between dependent (observed in presence of all decrements) and independent (associated single decrement) forms, under a Uniform Distribution of Decrements (UDD) assumption (Finan, Sections 65 and 67).

Usage

```
md_convert_rates(
  rates,
  direction = c("indep_to_dep", "dep_to_indep"),
  assumption = c("UDD_single", "UDD_multi")
)
```

Arguments

`rates`	Numeric matrix or data frame of cause-specific rates, one row per age and one column per cause. Column names are used as cause labels.
`direction`	Conversion direction: `"indep_to_dep"` converts independent (absolute) rates $q_x'^{(j)}$ to dependent rates $q_x^{(j)}$ (default). `"dep_to_indep"` converts dependent rates $q_x^{(j)}$ to independent rates $q_x'^{(j)}$.
`assumption`	UDD assumption context: `"UDD_single"` (default) assumes UDD in each associated single decrement table (Finan, Sec. 67, second approach). `"UDD_multi"` assumes UDD in the multiple decrement table (Finan, Sec. 67, first approach).

Details

Independent to dependent (`"indep_to_dep"`):

Under UDD in each single decrement table (Finan, Sec. 67):

$$q_x^{(j)} = q_x'^{(j)} \int_0^1 \prod_{i \neq j} (1 - s \times q_x'^{(i)})\, ds$$

For 2 causes this simplifies to:

$$q_x^{(1)} = q_x'^{(1)}(1 - \frac{1}{2} q_x'^{(2)})$$

For 3 causes:

$$q_x^{(1)} = q_x'^{(1)}(1 - \frac{1}{2}(q_x'^{(2)} + q_x'^{(3)}) + \frac{1}{3} q_x'^{(2)} q_x'^{(3)})$$

Dependent to independent (`"dep_to_indep"`):

Under UDD in the multiple decrement table (Finan, Sec. 67):

$$q_x'^{(j)} = 1 - (1 - q_x^{(\tau)})^{q_x^{(j)} / q_x^{(\tau)}}$$

where $q_x^{(\tau)} = \sum_j q_x^{(j)}$.

Value

A matrix of the same dimensions as `rates`, with converted rates. Row and column names are preserved.

`multi_decrement_table` *Build a multiple decrement table*

Description

Constructs a multiple decrement table from either independent (absolute) rates or dependent (observed) rates, with automatic conversion between the two under a UDD assumption (Finan, Sections 65-67).

Usage

```
multi_decrement_table(
  x,
  q_prime = NULL,
  q_dep = NULL,
  radix = 1e+05,
  causes = NULL,
  assumption = c("UDD_single", "UDD_multi")
)
```

Arguments

x	Integer vector of ages.
q_prime	Optional numeric matrix of independent (absolute) rates $q_x'^{(j)}$, one row per age and one column per cause. Supply either q_prime or q_dep (not both).
q_dep	Optional numeric matrix of dependent rates $q_x^{(j)}$ in the same format. Supply either q_dep or q_prime.
radix	Starting population $l_a^{(\tau)}$ (default 100000).
causes	Character vector of cause names. If NULL, taken from column names of q_prime or q_dep, or defaults to "cause_1", "cause_2", etc.
assumption	UDD assumption for rate conversion: "UDD_single" (default, Finan Sec. 67 second approach) or "UDD_multi" (Finan Sec. 67 first approach). Only used when converting between rate types.

Details

The function builds the complete multiple decrement table containing:

- $l_x^{(\tau)}$: survivors under all decrements
- $d_x^{(j)}$: expected decrements due to cause j
- $q_x^{(j)}$: dependent probability of decrement due to cause j (Finan, Sec. 66)
- $q_x'^{(j)}$: independent (absolute) probability of decrement (Finan, Sec. 65)
- $q_x^{(\tau)}$: total probability of decrement
- $p_x^{(\tau)}$: total survival probability

The key relationships (Finan, Sec. 65):

$$p_x^{(\tau)} = \prod_{j=1}^{m} p_x'^{(j)} = \prod_{j=1}^{m} (1 - q_x'^{(j)})$$

$$q_x^{(\tau)} = \sum_{j=1}^{m} q_x^{(j)}$$

$$d_x^{(j)} = l_x^{(\tau)} \times q_x^{(j)}$$

Value

A tibble with columns x, lx_tau, q_tau, p_tau, and for each cause j: q_dep_j, q_prime_j, dx_j. The result has class "multi_decrement_table" in addition to "tbl_df".

`plot_cash_flow` *Plot a cash flow diagram*

Description

Creates a visual cash-flow diagram with arrows representing inflows (positive payments, upward) and outflows (negative payments, downward).

Usage

```
plot_cash_flow(
  payment,
  time = NULL,
  date = NULL,
  i = NULL,
  pv = NULL,
  title = NULL,
  financial = TRUE,
  col_inflow = "#1B9E77",
  col_outflow = "#D95F02",
  day_count = c("act/365", "act/360")
)
```

Arguments

`payment`	Numeric vector of cash flows.
`time`	Optional numeric vector of times.
`date`	Optional vector of dates. If supplied, the earliest date is treated as time 0 for present-value calculations.
`i`	Optional annual effective interest rate used to compute PV if `pv` is `NULL`.
`pv`	Optional numeric present value to display.
`title`	Optional character title for the plot.
`financial`	Logical. If `TRUE` (default), uses financial convention: inflows point up and outflows point down. If `FALSE`, all arrows point up with height proportional to absolute value.
`col_inflow`	Character color for inflow arrows.
`col_outflow`	Character color for outflow arrows.
`day_count`	Day-count convention used when `date` is supplied. One of `"act/365"` or `"act/360"`.

Details

The vertical height of each arrow is proportional to the absolute payment amount, normalized so the largest payment reaches unit height. Each arrow is labeled with the formatted payment amount.

If an interest rate `i` 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(), future_value_tbl(), fv_flow(), irr_flow(), irr_flow_multi(), present_value(), present_value_tbl(), pv_flow()

Examples

```
# Time-based diagram
plot_cash_flow(
  payment = c(-1000, 300, 400, 500),
  time = c(0, 1, 2, 3),
  i = 0.08
)

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

---

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_tbl

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_tbl(
  .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_tbl, bond_convexity, bond_duration

Other bonds: bond_book_value(), bond_book_value_tbl(), bond_callable_price(), bond_callable_price_tbl(), bond_convexity(), bond_duration(), bond_price(), bond_ytm(), portfolio_duration_tbl()

Examples

```
# Simple example: one portfolio
portfolio_convexity_tbl(
  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_tbl(
  positions,
  col_portfolio = "portfolio_id",
  col_market_value = "market_value",
  col_convexity = "convexity"
)
```

---

portfolio_duration_tbl

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_tbl(
  .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_tbl, bond_duration, bond_convexity

Other bonds: bond_book_value(), bond_book_value_tbl(), bond_callable_price(), bond_callable_price_tbl(), bond_convexity(), bond_duration(), bond_price(), bond_ytm(), portfolio_convexity_tbl()

Examples

```
# Simple example: one portfolio
portfolio_duration_tbl(
  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_tbl(
  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 (commercial) premium via the extended equivalence principle (Finan, Sections 70–71).

Usage

```
premium_gross(prem, alpha = 0, beta = 0, gamma = 0, tidy = FALSE)
```

Arguments

prem	A one-row tibble returned by premium_x(..., tidy = TRUE) or premium_xy(..., tidy = TRUE). Must contain columns premium (net premium per payment) and apv_premiums (APV of the premium annuity).
alpha	Numeric ≥ 0. Initial (acquisition) expense as a multiple of one gross premium payment G. The initial expense is $\alpha \cdot G$, paid once at issue. Default 0.
beta	Numeric in $[0, 1)$. Proportional (collection) expense as a fraction of each gross premium payment. Each period the insurer incurs $\beta \cdot G$. Default 0.
gamma	Numeric ≥ 0. Fixed maintenance expense per premium payment period (in monetary units). Default 0.
tidy	Logical. If TRUE, returns a one-row tibble with the gross premium and expense breakdown.

Details

Designed to work directly with the output of premium_x or premium_xy when tidy = TRUE.

The extended equivalence principle (Finan, Section 70) equates the APV of gross premiums with the APV of benefits plus expenses:

$$G \cdot \ddot{a} = P_{\text{net}} \cdot \ddot{a} + \alpha \cdot G + \beta \cdot G \cdot \ddot{a} + \gamma \cdot \ddot{a}.$$

Solving for G:

$$G = \frac{P_{\text{net}} + \gamma}{(1 - \beta) - \alpha/\ddot{a}}$$

where $\ddot{a}$ is the APV of the premium annuity (apv_premiums column in prem).

The expense structure maps to common actuarial categories (Finan, Section 71):

- α: acquisition costs (agent commission, underwriting) - higher in year 1
- β: collection costs - proportional to premium
- γ: maintenance costs - fixed per period

For more complex expense structures (different first-year vs. renewal rates, per-policy vs. per-thousand, settlement expenses), the user should build the expense APV manually using annuity_x and apply the equivalence principle directly (see Finan, Examples 70.2, 71.1–71.3).

Value

A numeric gross premium per payment, or a one-row tibble if tidy = TRUE with columns gross_premium, net_premium, alpha, beta, gamma, loading_pct.

See Also

premium_x for single-life net premiums, premium_xy for two-life net premiums, annuity_x for building custom expense APVs.

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(lt, x = 60, i = 0.05,
                 product = "whole", benefit = 100000,
                 tidy = TRUE)
premium_gross(net, alpha = 0.5, beta = 0.05, gamma = 50)

# Finan Example 70.3 style: 10% of premium + $25/yr + $250/yr
# beta = 0.10, gamma = 275 (= 25 + 250), alpha = 0
premium_gross(net, alpha = 0, beta = 0.10, gamma = 275)

# Tidy output with expense breakdown
premium_gross(net, alpha = 0.5, beta = 0.05, gamma = 50,
              tidy = TRUE)

# Two-life workflow
net_xy <- premium_xy(lt, x = 60, y = 62, i = 0.05,
                     type = "whole", cohort = "first",
                     benefit = 100000, tidy = TRUE)
premium_gross(net_xy, alpha = 0.3, beta = 0.04, gamma = 30)

# 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 (Finan, Section 41):

$$P = \frac{\text{APV of benefits}}{\text{APV of premium annuity}}.$$

Usage

```
premium_x(
  lt,
  x,
  i,
  product = c("whole", "term", "endowment", "variable_k"),
  benefit,
  n = NULL,
```

```
  m = 0,
  k = 1,
  frac = c("UDD", "CF", "CML", "Balducci"),
  premium_timing = c("due", "immediate"),
  prem_start = c("issue", "deferred"),
  n_prem = NULL,
  woolhouse = c("none", "first", "second"),
  tidy = FALSE,
  check = TRUE
)
```

Arguments

lt	A life table data frame containing at least columns x and lx.
x	Integer actuarial age at issue.
i	Effective annual interest rate (must be > -1).
product	Type of insurance: "whole", "term", "endowment", or "variable_k".
benefit	Benefit amount. For standard products, a single numeric value. For "variable_k", a numeric vector or a function of time.
n	Optional insurance term in years. Required for "term" and "endowment".
m	Nonnegative integer deferral period in years (default 0).
k	Number of premium payments per year (default 1).
frac	Fractional-age assumption for survival probabilities: "UDD", "CF", "CML", or "Balducci".
premium_timing	Timing of premium payments: "due" (in advance, default) or "immediate" (in arrears).
prem_start	Start of premium payments: "issue" (start at issue, time 0) or "deferred" (start at time m).
n_prem	Optional premium-paying term in years, counted from prem_start. If NULL, defaults to whole-life to the end of the table (for "whole") or to the insurance term n (for temporary products).
woolhouse	Woolhouse order for the premium annuity when k > 1: "none" (exact UDD, default), "first", or "second". Passed to annuity_x.
tidy	Logical. If TRUE, returns a one-row tibble with details.
check	Logical. If TRUE, performs basic input validation.

Details

The premium returned corresponds to one premium payment (annual if k = 1, monthly if k = 12, etc.).

The benefit premium is the level payment that satisfies the equivalence principle: the APV of premiums equals the APV of benefits at issue.

For standard products, the APV of benefits is (Finan, Sections 27 and 41):

- Whole life: A_x (Finan, Sec. 41.1)
- n-year term: $A^1_{x:\overline{n}|}$ (Finan, Sec. 41.2)
- n-year endowment: $A_{x:\overline{n}|}$ (Finan, Sec. 41.4)

The APV of the premium annuity is computed via annuity_x, supporting k-thly payments and Woolhouse approximations.

Premium payment start (Finan, Sec. 41.5):

- prem_start = "issue": premiums start at age x (time 0). The premium annuity is $\ddot{a}_{x:\overline{n_p}|}$.
- prem_start = "deferred": premiums start at age $x + m$ (after the deferral period). The premium annuity, valued at time 0, is $v^m \cdot {}_mp_x \cdot \ddot{a}_{x+m:\overline{n_p}|}$ which equals a deferred annuity with m years of deferral.

Value

A numeric net premium per payment, or a one-row tibble if tidy = TRUE.

See Also

insurance_x for the APV of benefits, annuity_x for the APV of the premium annuity, premium_xy for two-life premiums, premium_gross for gross premiums with expenses.

Examples

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

# Whole life insurance, annual premium (Finan, Sec. 41.1):
# P(A_x) = A_x / \ddot{a}_x
premium_x(lt, x = 60, i = 0.05, product = "whole", benefit = 100000)

# Verify manually: P = A / \ddot{a}
A  <- insurance_x(lt, x = 60, i = 0.05, type = "whole")
ad <- annuity_x(lt, x = 60, i = 0.05, timing = "due")
100000 * A / ad

# 5-year term insurance, annual premium (Finan, Sec. 41.2):
# P(A^1_{x:n}) = A^1_{x:n} / \ddot{a}_{x:n}
premium_x(lt, x = 60, i = 0.05, product = "term", n = 5, benefit = 100000)

# 5-year endowment insurance (Finan, Sec. 41.4):
# P(A_{x:n}) = A_{x:n} / \ddot{a}_{x:n}
premium_x(lt, x = 60, i = 0.05, product = "endowment", n = 5,
          benefit = 100000)

# Deferred whole life: premiums start after deferral (Finan, Sec. 41.5)
premium_x(lt, x = 60, i = 0.05, product = "whole", benefit = 100000,
          m = 2, prem_start = "deferred")
```

```
# t-payment whole life: 3 years of premiums for whole life coverage
premium_x(lt, x = 60, i = 0.05, product = "whole", benefit = 100000,
          n_prem = 3)

# Monthly premiums with Woolhouse
premium_x(lt, x = 60, i = 0.05, product = "whole", benefit = 100000,
          k = 12, woolhouse = "first")

# Tidy output with all details
premium_x(lt, x = 60, i = 0.05, product = "term", n = 5,
          benefit = 100000, tidy = TRUE)
```

`premium_xy` *Net premium for two-life insurance by the equivalence principle*

Description

Computes the net benefit premium of a two-life insurance contract (independent lives) using the equivalence principle (Finan, Sections 41 and 58–59):

$$P = \frac{\text{APV of benefits}}{\text{APV of premium annuity}}.$$

Usage

```
premium_xy(
  lt,
  x,
  y,
  i,
  type = c("whole", "term", "endowment"),
  cohort = c("first", "last"),
  benefit,
  n = NULL,
  m = 0,
  k = 1,
  premium_timing = c("due", "immediate"),
  prem_start = c("issue", "deferred"),
  n_prem = NULL,
  woolhouse = c("none", "first", "second"),
  tidy = FALSE,
  check = TRUE
)
```

Arguments

`lt`	A life table data frame with columns `x` and `lx`.

x	Integer actuarial age for life 1 at issue.
y	Integer actuarial age for life 2 at issue.
i	Annual effective interest rate (must be > -1).
type	Insurance type: "whole", "term", or "endowment".
cohort	Status cohort (benefit trigger): "first" (joint-life) or "last" (last-survivor).
benefit	Benefit amount (single nonnegative numeric value).
n	Optional insurance term in years. Required for "term" and "endowment".
m	Nonnegative integer deferral period in years (default 0).
k	Number of premium payments per year (default 1).
premium_timing	Timing of premium payments: "due" (in advance, default) or "immediate" (in arrears).
prem_start	Start of premium payments: "issue" (start at time 0) or "deferred" (start at time m).
n_prem	Optional premium-paying term in years, counted from prem_start. If NULL, defaults to whole-life for "whole" or insurance term n for temporary products.
woolhouse	Woolhouse order for the premium annuity when k > 1: "none" (exact k-thly, default), "first", or "second". Passed to annuity_xy.
tidy	Logical. If TRUE, returns a one-row tibble.
check	Logical. If TRUE, performs basic input validation.

Details

The premium returned corresponds to one premium payment (annual if k = 1, monthly if k = 12, etc.).

The APV of benefits is computed via insurance_xy, valued at issue (time 0, age x).

The APV of the premium annuity is computed via annuity_xy, also valued at time 0. Both APVs are at the same valuation point, ensuring the equivalence principle is applied correctly.

Premium payment start:

- prem_start = "issue": premiums start at time 0 (ages x, y). The premium annuity is $\ddot{a}_{xy:\overline{n_p}|}$.
- prem_start = "deferred": premiums start at time m (ages x+m, y+m). The premium annuity, valued at time 0, is a deferred annuity with m years of deferral: $v^m \cdot {}_mp_{xy} \cdot \ddot{a}_{x+m,y+m:\overline{n_p}|}$.

The premium annuity uses the same two-life status as the benefit.

Value

A numeric net premium per payment, or a one-row tibble if tidy = TRUE.

See Also

insurance_xy for the APV of two-life benefits, annuity_xy for the APV of the premium annuity, premium_x for single-life premiums, premium_gross for gross premiums with expenses.

Examples

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

# Joint-life whole insurance, annual premium from issue
premium_xy(lt, x = 60, y = 62, i = 0.05,
           type = "whole", cohort = "first",
           benefit = 100000)

# 4-year term, last-survivor
premium_xy(lt, x = 60, y = 62, i = 0.05,
           type = "term", cohort = "last",
           n = 4, benefit = 100000)
```

`present_value` *Present value of a single payment*

Description

Computes the present value of a future payment `C` due at time `t`, using the annual effective interest rate implied by the supplied rate specification.

Usage

```
present_value(C, rate, type = "effective", m = 1, t)
```

Arguments

`C`	Numeric vector of future payment amounts.
`rate`	Numeric vector of rate values.
`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"`.
`t`	Numeric vector of times in years until payment.

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 i,

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

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

This is the core numeric version of the calculation. It is designed to work naturally with vectors and with dplyr::mutate().

Input vectors must have length 1 or a common length. Standard recycling is supported only under that rule.

Missing values are propagated. This function does not accept dates. If you need a tabular output with actuarial fields, use present_value_tbl.

Value

Numeric vector of present values.

See Also

standardize_interest, future_value, present_value_tbl

Other time-value: future_value(), future_value_tbl(), fv_flow(), irr_flow(), irr_flow_multi(), plot_cash_flow(), present_value_tbl(), pv_flow()

Examples

```
# Simple scalar example
present_value(C = 1000, rate = 0.08, type = "effective", t = 3)

# Medium vectorized example
present_value(
  C = c(1000, 2500, 4000),
  rate = c(0.08, 0.10, 0.12),
  type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  t = c(3, 5, 2)
)

# Use inside a data pipeline
if (requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  cashflows <- tibble::tibble(
    amount = c(1000, 1500, 2000),
    rate   = c(0.08, 0.12, 0.09),
    type   = c("effective", "force", "nominal_interest"),
    m      = c(1, 1, 4),
    t      = c(2, 3, 5)
  )

  dplyr::mutate(
    cashflows,
```

```
    pv = present_value(C = amount, rate = rate, type = type, m = m, t = t)
  )
}
```

`present_value_tbl` *Present value details in tibble form*

Description

Computes the present value of a future payment and returns a tibble containing both the inputs and the actuarial quantities used in the calculation.

Usage

```
present_value_tbl(C, rate, type = "effective", m = 1, t)
```

Arguments

`C`	Numeric vector of future payment amounts.
`rate`	Numeric vector of rate values.
`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"`.
`t`	Numeric vector of times in years until payment.

Details

This is a reporting wrapper around `present_value`. It is useful when you want a tabular summary including the input rate specification, the annual effective rate, the annual discount factor, and the final present value.

This function follows the same recycling and validation rules as `present_value`. Input vectors must have length 1 or a common length.

Missing values are propagated.

Value

A tibble with columns:

C Future payment amount.

t Time in years until payment.

rate_input Original supplied rate.

rate_type Type of supplied rate.

m Compounding frequency.

i_effective Equivalent annual effective interest rate.

v Annual discount factor $(1+i)^{-1}$.

present_value Computed present value.

See Also

present_value, standardize_interest, future_value

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

Examples

```
# Simple scalar example
present_value_tbl(C = 1000, rate = 0.08, type = "effective", t = 3)

# Medium vectorized example
present_value_tbl(
  C = c(1000, 2500, 4000),
  rate = c(0.08, 0.10, 0.12),
  type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  t = c(3, 5, 2)
)

# Combine with dplyr
if (requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  cashflows <- tibble::tibble(
    amount = c(1000, 1500, 2000),
    rate   = c(0.08, 0.12, 0.09),
    type   = c("effective", "force", "nominal_interest"),
    m      = c(1, 1, 4),
    t      = c(2, 3, 5)
  )

  present_value_tbl(
    C = cashflows$amount,
    rate = cashflows$rate,
    type = cashflows$type,
    m = cashflows$m,
    t = cashflows$t
  )
}
```

---

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()`, `future_value_tbl()`, `fv_flow()`, `irr_flow()`, `irr_flow_multi()`, `plot_cash_flow()`, `present_value()`, `present_value_tbl()`

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 the terminal benefit reserve ${}_kV$ at one or more policy durations for a fully discrete single-life insurance contract, using the prospective or recursive method (Finan, Sections 47 and 52).

Usage

```
reserve_x(
  lt,
  x,
  i,
  type = c("whole", "term", "endowment"),
  n = NULL,
  benefit = 1,
  premium = NULL,
  h = NULL,
  at = NULL,
  method = c("prospective", "recursive"),
  tidy = TRUE
)
```

Arguments

`lt`	A life table data frame with columns `x` and `lx`.
`x`	Integer actuarial age at issue.
`i`	Annual effective interest rate (must be > -1).
`type`	Insurance type: `"whole"`, `"term"`, or `"endowment"`.
`n`	Integer insurance term in years. Required for `"term"` and `"endowment"`. For `"whole"`, determined from the life table.
`benefit`	Numeric benefit amount (default 1).
`premium`	Net premium per payment. If `NULL` (default), computed internally via the equivalence principle using `premium_x`.
`h`	Integer premium-paying term in years. If `NULL`, premiums are payable for the full duration of the contract (i.e., `h = n` for temporary products, whole life for whole). Set `h < n` for limited-payment policies (Finan, Sec. 47.3).

`at`	Integer vector of policy durations at which to compute the reserve. Default `NULL` computes for all integer durations `0, 1, ..., n` (or to end of table for whole life).
`method`	Computation method: `"prospective"` (default, Finan Sec. 47) or `"recursive"` (Finan Sec. 52).
`tidy`	Logical. If `TRUE` (default), returns a tibble schedule. If `FALSE`, returns a named numeric vector.

Details

Prospective method (Finan, Sections 47.1–47.3):

$$ {}_kV = \text{APV(future benefits at } x+k) - P \cdot \text{APV(future premiums at } x+k) $$

For each product type:

- **Whole life** (Sec. 47.1): ${}_kV(A_x) = A_{x+k} - P\,\ddot{a}_{x+k}$
- **Term** (Sec. 47.2), $k < n$: ${}_kV(A^1_{x:\overline{n}|}) = A^1_{x+k:\overline{n-k}|} - P\,\ddot{a}_{x+k:\overline{n_p-k}|}$ where $n_p = \min(n, h)$.
- **Endowment** (Sec. 47.3), $k < n$: ${}_kV(A_{x:\overline{n}|}) = A_{x+k:\overline{n-k}|} - P\,\ddot{a}_{x+k:\overline{n_p-k}|}$
- Endowment at $k = n$: ${}_nV =$ benefit.

For limited-payment policies (`h < n`), premiums cease after year h. For $k \geq h$, the premium annuity term is zero.

Recursive method (Finan, Section 52):

$$ {}_{k+1}V = \frac{({}_kV + \pi_k)(1+i) - b_{k+1} \cdot q_{x+k}}{p_{x+k}} $$

starting from ${}_0V = 0$ (equivalence principle).

Value

If `tidy = TRUE`, a tibble with columns `k` (duration), `age` ($x+k$), `reserve`, `premium_paid` (premium at start of year $k+1$, 0 after limited payment), and `benefit_due` (death benefit in year $k+1$). If `tidy = FALSE`, a named numeric vector of reserves.

See Also

`premium_x` for benefit premiums, `insurance_x` for insurance APVs, `annuity_x` for annuity APVs, `t_px` for survival probabilities.

Examples

```
lt <- data.frame(
  x  = 60:70,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000,
         86000, 81000, 75000, 68000, 60000)
)
```

```
# Whole life reserve schedule (Finan, Sec. 47.1)
reserve_x(lt, x = 60, i = 0.06, type = "whole")

# 5-year term (Finan, Sec. 47.2)
reserve_x(lt, x = 60, i = 0.06, type = "term", n = 5)

# 5-year endowment (Finan, Sec. 47.3)
reserve_x(lt, x = 60, i = 0.06, type = "endowment", n = 5)

# Limited payment: 3-payment, 5-year endowment
reserve_x(lt, x = 60, i = 0.06, type = "endowment",
          n = 5, h = 3)

# Recursive method (Finan, Sec. 52)
reserve_x(lt, x = 60, i = 0.06, type = "endowment",
          n = 5, method = "recursive")

# Verify: prospective = recursive
r_pro <- reserve_x(lt, x = 60, i = 0.06, type = "endowment",
                   n = 5, method = "prospective")
r_rec <- reserve_x(lt, x = 60, i = 0.06, type = "endowment",
                   n = 5, method = "recursive")
all.equal(r_pro$reserve, r_rec$reserve)

# Specific durations only
reserve_x(lt, x = 60, i = 0.06, type = "endowment",
          n = 5, at = c(0, 3, 5))

# Benefit of $100,000
reserve_x(lt, x = 60, i = 0.06, type = "whole",
          benefit = 100000)

# Custom premium (e.g., loaded)
reserve_x(lt, x = 60, i = 0.06, type = "endowment",
          n = 5, benefit = 100000, premium = 22000)

# As named vector
reserve_x(lt, x = 60, i = 0.06, type = "endowment",
          n = 5, tidy = FALSE)
```

`reserve_xy` *Benefit reserve schedule for two-life insurance*

Description

Computes the terminal benefit reserve ${}_kV$ at one or more policy durations for a fully discrete two-life insurance contract (independent lives), using the prospective or recursive method. Generalizes Finan, Sections 47 and 52 to the joint-life (Sec. 58) and last-survivor (Sec. 59) statuses.

Usage

```
reserve_xy(
  lt,
  x,
  y,
  i,
  type = c("whole", "term", "endowment"),
  cohort = c("first", "last"),
  n = NULL,
  benefit = 1,
  premium = NULL,
  h = NULL,
  at = NULL,
  method = c("prospective", "recursive"),
  tidy = TRUE
)
```

Arguments

`lt`	A life table data frame with columns `x` and `lx`.
`x`	Integer actuarial age for life 1 at issue.
`y`	Integer actuarial age for life 2 at issue.
`i`	Annual effective interest rate (must be `> -1`).
`type`	Insurance type: `"whole"`, `"term"`, or `"endowment"`.
`cohort`	Status cohort: `"first"` (joint-life, first death) or `"last"` (last-survivor, second death).
`n`	Integer insurance term in years. Required for `"term"` and `"endowment"`. For `"whole"`, determined from the life table.
`benefit`	Numeric benefit amount (default `1`).
`premium`	Net premium per payment. If `NULL` (default), computed internally via the equivalence principle using `premium_xy`.
`h`	Integer premium-paying term in years. If `NULL`, premiums are payable for the full duration of the contract. Set `h < n` for limited-payment policies.
`at`	Integer vector of policy durations at which to compute the reserve. Default `NULL` computes for all integer durations `0, 1, ..., n`.
`method`	Computation method: `"prospective"` (default) or `"recursive"`.
`tidy`	Logical. If `TRUE` (default), returns a tibble schedule. If `FALSE`, returns a named numeric vector.

Details

Prospective method (generalizing Finan, Sec. 47 to two lives):

$${}_kV = \text{APV(future benefits at ages } x+k, y+k) - P \cdot \text{APV(future premiums at ages } x+k, y+k)$$

The APV of future benefits uses `insurance_xy` at shifted ages $(x+k, y+k)$, and the APV of future premiums uses `annuity_xy` on the same status.

For endowment insurance at $k = n$: ${}_nV =$ benefit.

Recursive method (generalizing Finan, Sec. 52):

$$ {}_{k+1}V = \frac{({}_kV + \pi_k)(1+i) - b_{k+1} \cdot q_{\text{status}}(x+k, y+k)}{p_{\text{status}}(x+k, y+k)} $$

where q_{status} and p_{status} are the one-year death and survival probabilities of the two-life status, computed via `t_pxy`.

Value

If `tidy = TRUE`, a tibble with columns `k`, `age_x`, `age_y`, `reserve`, `premium_paid`, `benefit_due`. If `tidy = FALSE`, a named numeric vector of reserves.

See Also

`reserve_x` for single-life reserves, `premium_xy` for two-life premiums, `insurance_xy` for two-life insurance APVs, `annuity_xy` for two-life annuity APVs, `t_pxy` for two-life survival.

Examples

```
lt <- data.frame(
  x  = 60:70,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000,
         86000, 81000, 75000, 68000, 60000)
)

# Joint-life whole life reserve
reserve_xy(lt, x = 60, y = 62, i = 0.06,
           type = "whole", cohort = "first")

# Last-survivor 5-year endowment
reserve_xy(lt, x = 60, y = 62, i = 0.06,
           type = "endowment", cohort = "last", n = 5)

# Joint-life 4-year term
reserve_xy(lt, x = 60, y = 62, i = 0.06,
           type = "term", cohort = "first", n = 4)

# Limited payment: 3-payment, 5-year endowment, joint
reserve_xy(lt, x = 60, y = 62, i = 0.06,
           type = "endowment", cohort = "first",
           n = 5, h = 3)

# Verify: prospective = recursive
r_pro <- reserve_xy(lt, x = 60, y = 62, i = 0.06,
                    type = "endowment", cohort = "first",
                    n = 5, method = "prospective")
r_rec <- reserve_xy(lt, x = 60, y = 62, i = 0.06,
```

```
                    type = "endowment", cohort = "first",
                    n = 5, method = "recursive")
all.equal(r_pro$reserve, r_rec$reserve)

# Benefit of $100,000
reserve_xy(lt, x = 60, y = 62, i = 0.06,
           type = "whole", cohort = "first",
           benefit = 100000)

# Specific durations
reserve_xy(lt, x = 60, y = 62, i = 0.06,
           type = "endowment", cohort = "first",
           n = 5, at = c(0, 3, 5))
```

`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
)
```

`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()`, `discount_factor_spot_tbl()`, `forward_rate_tbl()`, `interest_equivalents()`, `yield_curve_tbl()`

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
    )
  )
}
```

`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,
  type = "effective",
  m = 1L,
```

```
  deferral_years = 0,
  timing = "immediate"
)
```

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.
`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"`.

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 only, 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:

$$s_n = \frac{(1+i)^n - 1}{i}$$

For due annuities:

$$\ddot{s}_n = (1+i)s_n$$

For continuous annuities:

$$\bar{s}_n = \frac{e^{\delta n} - 1}{\delta}$$

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

Value

Numeric vector of accumulation factors.

See Also

s_angle_tbl, a_angle, standardize_interest

Other annuities: Da_angle(), Da_angle_tbl(), Ia_angle(), Ia_angle_tbl(), a_angle(), a_angle_tbl(), arithmetic_annuity_av_tbl(), arithmetic_annuity_pv_tbl(), ga_angle(), ga_angle_tbl(), gs_angle(), gs_angle_tbl(), s_angle_tbl()

Examples

```
# Simple scalar examples
s_angle(n_years = 10, rate = 0.05, type = "effective")
s_angle(n_years = 10, rate = 0.06, type = "nominal_interest", m = 12,
        payments_per_year = 12)
s_angle(n_years = 15, rate = 0.04, type = "force", timing = "continuous")

# Medium 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),
  type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  deferral_years = c(0, 2, 3),
  timing = c("immediate", "due", "continuous")
)

# Use inside a data pipeline
if (requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  contracts <- tibble::tibble(
    n_years = c(10, 15, 20),
    rate = c(0.05, 0.06, 0.04),
    type = c("effective", "nominal_interest", "force"),
    m = c(1, 12, 1),
    payments_per_year = c(1, 12, NA),
    deferral_years = c(0, 2, 3),
    timing = c("immediate", "due", "continuous")
  )

  dplyr::mutate(
    contracts,
    factor = s_angle(
      n_years = n_years,
      payments_per_year = dplyr::coalesce(payments_per_year, 1L),
      rate = rate,
      type = type,
      m = m,
      deferral_years = deferral_years,
```

```
      timing = timing
    )
  )
}
```

`s_angle_tbl` *Level annuity accumulation details in tibble form*

Description

Computes the actuarial accumulation factor for a level annuity and returns a tibble with the main input values, implied rates, accumulation factor, payment amount, and future value.

Usage

```
s_angle_tbl(
  n_years,
  payments_per_year = 1L,
  rate,
  type = "effective",
  m = 1L,
  deferral_years = 0,
  timing = "immediate",
  payment = 1
)
```

Arguments

`n_years`	Numeric vector of payment durations in years.
`payments_per_year`	Positive integer vector giving the number of discrete payments per year. Ignored for continuous annuities.
`rate`	Numeric vector of rate values.
`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.
`timing`	Character vector. One of `"immediate"`, `"due"`, or `"continuous"`.
`payment`	Numeric vector of level payment amounts.

Details

This is a reporting wrapper around s_angle. The accumulation factor assumes unit payments. The future value is then computed as

$$FV = R \times s$$

where R is the payment amount and s is the accumulation factor.

Under the adopted horizon convention, deferral_years is metadata only and does not affect the accumulation factor.

Value

A tibble with columns:

n_years Payment duration in years.

payments_per_year Number of payments per year.

deferral_years Deferral period in years.

timing Payment timing convention.

rate_input Original supplied rate.

rate_type Type of supplied rate.

m Compounding frequency for nominal rates.

i_effective Equivalent annual effective rate.

delta Equivalent force of interest.

i_period Equivalent per-payment effective rate for discrete annuities.

v_period Equivalent per-payment discount factor for discrete annuities.

n_periods Number of payment periods for discrete annuities.

deferral_periods Number of deferred periods for discrete annuities.

accumulation_factor Computed accumulation factor.

payment Level payment amount.

future_value Future value of the annuity.

See Also

s_angle, a_angle_tbl, standardize_interest, future_value

Other annuities: Da_angle(), Da_angle_tbl(), Ia_angle(), Ia_angle_tbl(), a_angle(), a_angle_tbl(), arithmetic_annuity_av_tbl(), arithmetic_annuity_pv_tbl(), ga_angle(), ga_angle_tbl(), gs_angle(), gs_angle_tbl(), s_angle()

Examples

```
# Simple scalar example
s_angle_tbl(n_years = 10, rate = 0.05, payment = 1000)

# Medium vectorized example
s_angle_tbl(
  n_years = c(10, 15, 20),
```

```
  payments_per_year = c(1, 12, 1),
  rate = c(0.05, 0.06, 0.04),
  type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  deferral_years = c(0, 2, 3),
  timing = c("immediate", "due", "continuous"),
  payment = c(1000, 200, 5000)
)
```

`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`	A life table data frame. 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 `lt` carries a `frac` attribute, that value is used. 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")

# 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)
    )
}
```

`Var_annuity_x` *Variance of the actuarial present value of a life annuity*

Description

Computes the variance of the present value random variable of a discrete life annuity with k-thly payments, using the exact second moment under the UDD (Uniform Distribution of Deaths) assumption.

Usage

```
Var_annuity_x(
  lt,
  x,
  i,
  n = NULL,
  m = 0L,
  k = 1L,
  timing = c("immediate", "due"),
  tidy = FALSE
)
```

Arguments

lt	A life table data frame containing columns x and lx. Requires lx to apply exact UDD at fractional ages.
x	Integer actuarial age.
i	Effective annual interest rate (must be > -1).
n	Integer term in years. If NULL, whole life to end of table.
m	Integer deferment in years (default 0).
k	Integer payments per year (default 1). Example: k = 12 for monthly.
timing	Payment timing: "immediate" (arrears) or "due" (advance).
tidy	Logical. If TRUE, returns a one-row tibble with details.

Details

Let payments be of amount $1/k$ at times $m + u_j$, where $u_j = j/k$ with $j = 0, \ldots, kn - 1$ (due) or $j = 1, \ldots, kn$ (immediate).

Define $Z = \sum_j \frac{1}{k} v^{m+u_j} I(T_{x+m} > u_j)$. Then:

$$\mathrm{Var}(Z) = E[Z^2] - (E[Z])^2$$

with

$$E[Z^2] = \sum_j \sum_\ell \frac{1}{k^2} v^{2m+u_j+u_\ell} \cdot {}_{\max(u_j,u_\ell)}p_{x+m}.$$

This uses the fact that payments j and ℓ are both made if and only if the annuitant survives to $\max(u_j, u_\ell)$. The double sum is computed via a vectorized matrix product (outer) for efficiency.

For the special case of annual payments (k = 1) without deferral, the variance satisfies the classical identity (Finan, Section 37.1, Example 37.5):

$$\mathrm{Var}(\ddot{Y}_{x:\overline{n}|}) = \frac{{}^2A_{x:\overline{n}|} - (A_{x:\overline{n}|})^2}{d^2}$$

where $d = i/(1+i)$ and ${}^2A$ is the second moment of the endowment insurance (Finan, Section 27). This identity can be used to cross-validate results.

Fractional survival probabilities are exact under UDD (Finan, Section 24.1).

Value

A numeric variance, or a one-row tibble if tidy = TRUE with columns x, n, m, k, timing, i, EZ, EZ2, variance.

See Also

annuity_x for the expected APV (first moment), Var_insurance_x for the variance of life insurance APV, insurance_x for life insurance APVs.

Examples

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

# Annual variance, annuity-due
Var_annuity_x(lt, x = 60, i = 0.06, timing = "due")

# Monthly (k=12) exact under UDD for a 10-year term
Var_annuity_x(lt, x = 60, i = 0.06, n = 10, k = 12, timing = "immediate")

# Cross-validate with Finan identity (Sec. 37.1):
# Var(\ddot{Y}_x) = (2A_x - A_x^2) / d^2
v_out <- Var_annuity_x(lt, x = 60, i = 0.06, timing = "due", tidy = TRUE)
v_out

# 5-year temporary with tidy output
Var_annuity_x(lt, x = 60, i = 0.06, n = 5, k = 4,
              timing = "due", tidy = TRUE)

# Deferred annuity variance
Var_annuity_x(lt, x = 60, i = 0.06, m = 5, timing = "due")
```

Var_insurance_x *Variance of the actuarial present value of a life insurance*

Description

Computes the variance of the present value random variable of a discrete life insurance under the exact UDD assumption on a $1/k$-year grid.

Usage

```
Var_insurance_x(
  lt,
  x,
  i,
  product = c("whole", "term", "endowment", "pure_endowment"),
  benefit = 1,
  n = NULL,
  m = 0L,
  k = 1L,
  tidy = FALSE
)
```

Arguments

`lt`	Life table data frame containing columns `x` and `lx`. Requires `lx` to apply exact UDD at fractional ages.

x	Integer actuarial age at issue.
i	Effective annual interest rate (must be > -1).
product	Insurance type: "whole", "term", "endowment", or "pure_endowment".
benefit	Numeric scalar benefit amount (default 1).
n	Integer term in years after deferment. Required for "term", "endowment", and "pure_endowment".
m	Integer deferment in years (default 0).
k	Integer grid frequency per year (default 1). Example: k = 12 for monthly benefit timing.
tidy	Logical. If TRUE, returns a one-row tibble with details.

Details

The benefit is paid at the end of the $1/k$-year subperiod in which death occurs (or at time $m+n$ for the pure endowment).

Under UDD (Finan, Section 24.1), fractional survival is computed by linear interpolation of ℓ_{x+s} within each year. The death-in-subinterval probability is:

$$\Pr(T \in (u_{r-1}, u_r]) = S(u_{r-1}) - S(u_r)$$

where $S(u) = {}_up_{x+m}$.

Whole life insurance (Finan, Section 27):

$$\mathrm{Var}(Z_x) = {}^2A_x - (A_x)^2$$

where ${}^2A_x = \sum_{k=0}^{\infty} v^{2(k+1)} \cdot {}_kp_x \cdot q_{x+k}$.

Term insurance (Finan, Section 27):

$$\mathrm{Var}(Z^1_{x:\overline{n}|}) = {}^2A^1_{x:\overline{n}|} - (A^1_{x:\overline{n}|})^2.$$

Pure endowment (Finan, Section 26.3.1):

$$\mathrm{Var}(\bar{Z}_{\ x:\overline{n}|}) = v^{2n} \cdot {}_np_x \cdot {}_nq_x.$$

Endowment insurance (Finan, Example 26.15 and Section 26.3.2): Since the term component and the pure endowment are mutually exclusive ($Z^1_{x:\overline{n}|} \cdot \bar{Z}_{\ x:\overline{n}|} = 0$), the covariance is $-A^1_{x:\overline{n}|} \cdot A_{\ x:\overline{n}|}$ and therefore:

$$\mathrm{Var}(Z_{x:\overline{n}|}) = {}^2A^1_{x:\overline{n}|} + {}^2A_{\ x:\overline{n}|} - (A^1_{x:\overline{n}|} + A_{\ x:\overline{n}|})^2.$$

For the k-thly case, the sums run over all $1/k$-year subperiods, giving the exact UDD result.

Value

Numeric variance, or a one-row tibble if tidy = TRUE with columns x, m, n, k, product, i, benefit, EZ, EZ2, variance.

See Also

insurance_x for the expected APV (first moment), Var_annuity_x for the variance of life annuity APV, annuity_x for life annuity APVs.

Examples

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

# Whole life, annual
Var_insurance_x(lt, x = 60, i = 0.06, product = "whole")

# 10-year term, monthly (k=12)
Var_insurance_x(lt, x = 60, i = 0.06, product = "term", n = 10, k = 12)

# Finan Section 27 style: Var(Z) = 2A - A^2
# Verify by comparing tidy output EZ2 - EZ^2
Var_insurance_x(lt, x = 60, i = 0.06, product = "whole", tidy = TRUE)

# 10-year term with 5-year deferral, monthly
Var_insurance_x(lt, x = 60, i = 0.06, product = "term",
                n = 10, m = 5, k = 12)

# Pure endowment variance (Finan, Sec. 26.3.1):
# Var = v^{2n} * n_p_x * n_q_x
Var_insurance_x(lt, x = 60, i = 0.06, product = "pure_endowment",
                n = 10, tidy = TRUE)

# Endowment = term + pure endowment (Finan, Sec. 26.3.2)
Var_insurance_x(lt, x = 60, i = 0.06, product = "endowment",
                n = 10, tidy = TRUE)

# Benefit of $100,000
Var_insurance_x(lt, x = 60, i = 0.06, product = "whole",
                benefit = 100000)
```

---

yield_curve_tbl *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_tbl(
  .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_tbl, discount_factor_spot, standardize_interest

Other interest: discount_factor_spot(), discount_factor_spot_tbl(), forward_rate_tbl(), interest_equivalents(), standardize_interest()

Examples

```
# Simple example
res <- yield_curve_tbl(
  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_tbl(
  curves,
  col_term = "term",
  col_spot = "spot",
  plot = TRUE,
  .out = "v",
  .out_plot = "curve_plot"
)

res2$curve_plot[[2]]
```

Index