Package: tidyactuarial (via r-universe)

October 25, 2026

Type Package

Title Tidy Tools for Actuarial Mathematics and Life Contingencies

Version 0.2.0

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

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

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

Suggests knitr, purrr, rmarkdown, testthat (>= 3.0.0)

LazyData true

LazyDataCompression xz

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/JulianFajardo1908/tidyactuarial, https://julianfajardo1908.github.io/tidyactuarial/

BugReports https://github.com/JulianFajardo1908/tidyactuarial/issues

NeedsCompilation no

Config/roxygen2/version 8.1.0

Repository https://julianfajardo1908.r-universe.dev

Date/Publication 2026-10-25 12:27:20 UTC

RemoteUrl https://github.com/julianfajardo1908/tidyactuarial

RemoteRef HEAD

RemoteSha 2466fc143b89ba15020de6f8070a54cb6896eab7

Contents

a_angle . . . 4
accrued_interest . . . 7
accumulation_factor . . . 10
add_insurance . . . 13
add_premium_schedule . . . 14
aggregate_distribution . . . 15
aggregate_moments . . . 19
amort_schedule . . . 21
amort_schedule_general . . . 25
annuity_arith . . . 28
annuity_geom . . . 31
annuity_multi . . . 34
annuity_x . . . 37
annuity_xy . . . 41
apply_coverage . . . 44
apv_life_flow . . . 46
bond_book_value . . . 49
bond_callable_price . . . 52
bond_cash_flows . . . 55
bond_convexity . . . 56
bond_duration . . . 58
bond_price . . . 61
bond_ytm . . . 63
bonds_sample . . . 66
bootstrap_spot_curve . . . 67
cash_flows_sample . . . 69
collective_risk_spec . . . 70
commutation_table . . . 72
convexity_cash_flow . . . 74
coverage_spec . . . 76
discount_factor . . . 78
discount_factor_spot . . . 81
discretize_severity . . . 83
distribution_convolution . . . 86
distribution_moments . . . 89
duration_cash_flow . . . 91
e_x . . . 93
e_xy . . . 95
empirical_frequency . . . 97
empirical_severity . . . 99
forward_curve . . . 101
forward_rate . . . 103
frequency_spec . . . 106
future_value . . . 107
fv_flow . . . 109
immunize_duration . . . 111

immunize_duration_convexity . . . 114
insurance_variable_k . . . 117
insurance_x . . . 120
insurance_xj . . . 124
insurance_xy . . . 126
interest_equivalents . . . 130
irr_flow . . . 132
irr_flow_multi . . . 134
km_lifetable . . . 136
life_contract . . . 139
lifetable . . . 141
loans_sample . . . 144
lt_tau . . . 146
mc_annuity . . . 147
mc_insurance . . . 152
mc_loss . . . 156
mc_multilife_status . . . 159
mc_premium . . . 161
mc_reserve . . . 164
md_table . . . 168
mortality_colombia_tables . . . 170
mortality_law_table . . . 171
mortality_world_sample_2015_2023 . . . 175
mortality_world_sample_2023 . . . 176
multiple_decrement_sample . . . 178
par_rate . . . 179
plot_cash_flow . . . 181
plot_contingent_cash_flow . . . 184
plot_immunization_gap . . . 187
plot_km . . . 189
portfolio_convexity . . . 190
portfolio_duration . . . 193
premium_gross . . . 195
premium_x . . . 198
premium_xy . . . 202
present_value . . . 205
print.tidyact_aggregate_distribution . . . 207
print.tidyact_collective_risk_spec . . . 208
print.tidyact_convolution_spec . . . 208
print.tidyact_frequency_spec . . . 209
print.tidyact_payment_spec . . . 209
print.tidyact_severity_spec . . . 210
pv_flow . . . 210
reserve_x . . . 212
reserve_xy . . . 216
s_angle . . . 218
severity_spec . . . 221
simulate_aggregate . . . 224

simulate_annuity_x . . . 225
simulate_frequency . . . 227
simulate_insurance_x . . . 228
simulate_lifetime . . . 229
simulate_lifetimes . . . 231
simulate_severity . . . 233
sinking_fund_schedule . . . 234
soa08lt . . . 237
solve_value . . . 238
standardize_interest . . . 241
stop_loss_premium . . . 243
summary_mc . . . 244
t_Ex . . . 246
t_px . . . 249
t_pxy . . . 250
t_qx . . . 252
t_qxj . . . 254
tail_value_at_risk . . . 255
to_payment_basis . . . 256
value_at_risk . . . 258
yield_curve . . . 259

Index **262**

`a_angle` *Level annuity factor a-angle-n*

Description

Computes the actuarial present value factor for a level annuity using compact actuarial notation.

Usage

```
a_angle(
  n = NULL,
  k = 1L,
  i,
  i_type = "effective",
  m = 1L,
  h = 0,
  timing = "immediate",
  perpetuity = FALSE,
  payment = 1,
  tidy = FALSE
)
```

Arguments

n	Numeric vector of payment durations in years. Ignored when perpetuity = TRUE. If perpetuity = FALSE, each value must be positive and finite.
k	Positive integer vector giving the number of discrete payments per year. Ignored for continuous annuities.
i	Numeric vector of interest-rate values.
i_type	Character vector indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer vector giving the conversion frequency for nominal rates. Ignored for "effective" and "force".
h	Numeric vector of deferment times in years. Must be greater than or equal to 0.
timing	Character vector. One of "immediate", "due", or "continuous".
perpetuity	Logical vector. If TRUE, computes the perpetuity factor.
payment	Numeric vector of level payment amounts. Used only when tidy = TRUE to report the corresponding present value. The annuity factor itself is always computed for unit payments.
tidy	Logical scalar. If FALSE, returns a numeric annuity factor. If TRUE, returns a tibble with intermediate calculations.

Details

Supported timing conventions:

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

For discrete annuities, k is the number of payments per year, so payments are made every $1/k$ year. The function returns the annuity factor, assuming a unit payment at each payment time.

Deferment is supported through h. For discrete annuities, the deferment must align with the payment grid, that is, hk must be an integer.

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

This function follows the compact actuarial notation used throughout tidyactuarial:

- n: annuity term;
- k: payment frequency;
- i: interest rate;
- i_type: interest-rate type;
- m: conversion frequency for nominal rates;
- h: deferment period.

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:

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

More generally, let $N = nk$, let $b = 0$ for immediate payments and $b = 1$ for due payments. The deferred discrete factor is

$$\sum_{j=1}^{N} \exp\{-\delta[h + (j - b)/k]\}.$$

Each discrete payment is one unit, whereas a continuous annuity pays one unit per year. The continuous deferred factor is $\int_h^{h+n} e^{-\delta t}\, dt$.

The original log accumulation rate is retained even if its annual effective equivalent rounds to -1 or overflows. Geometric sums and continuous integrals use stable logarithmic formulas with exact zero-rate limits, rather than replacing a small nonzero rate by zero. Perpetuities require a strictly positive rate, without a numerical lower cutoff. Monetary values combine payments with log factors before declaring range loss; a reported annuity factor or equivalent rate may still be non-finite.

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

Value

If `tidy = FALSE`, a numeric vector of annuity factors.

If `tidy = TRUE`, a tibble with input values, equivalent rates, annuity factors, payment amounts, and present values.

See Also

s_angle, standardize_interest, present_value

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

Examples

```
# Numeric annuity factor
a_angle(n = 10, i = 0.05)

# Nominal interest converted monthly, with monthly payments
a_angle(
  n = 10,
  i = 0.06,
  i_type = "nominal_interest",
  m = 12,
  k = 12
```

```
)

# Continuous annuity
a_angle(
  n = 15,
  i = 0.04,
  i_type = "force",
  timing = "continuous"
)

# Tibble output for teaching or auditing
a_angle(
  n = 10,
  i = 0.05,
  payment = 1000,
  tidy = TRUE
)

# Vectorized example
a_angle(
  n = c(5, 10, 20),
  k = c(1, 12, 1),
  i = c(0.05, 0.06, 0.04),
  i_type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  h = c(0, 0, 2),
  timing = c("immediate", "immediate", "continuous"),
  perpetuity = c(FALSE, FALSE, FALSE)
)
```

`accrued_interest` *Accrued coupon interest and clean/dirty bond prices*

Description

Computes the coupon interest accrued between the previous coupon date and settlement, and optionally reconciles a quoted clean or dirty bond price. This is a day-count calculation, not a yield-based bond valuation.

Usage

```
accrued_interest(
  .data = NULL,
  face = NULL,
  c = NULL,
  settlement = NULL,
  previous_coupon = NULL,
  next_coupon = NULL,
```

```
  k = 1L,
  day_count = "act/act",
  clean_price = NULL,
  dirty_price = NULL,
  tol = 1e-10
)
```

Arguments

`.data`	Optional data frame containing one bond per row. Required columns are `face`, `c`, `settlement`, `previous_coupon`, and `next_coupon`. Optional columns are `k`, `day_count`, `clean_price`, and `dirty_price`. Other columns are preserved. When `.data` is supplied, `face`, `c`, and the date arguments must come from its columns.
`face`	Positive face value of the bond. Numeric scalar or vector in direct-input mode; otherwise the corresponding column of `.data`.
`c`	Nonnegative annual nominal coupon rate, as a proportion (0.06 for 6 percent). Numeric scalar or vector.
`settlement`	`Date` of settlement. The accrued-interest convention excludes this date from the elapsed interval.
`previous_coupon`	`Date` of the immediately preceding coupon date.
`next_coupon`	`Date` of the immediately following coupon date. Settlement must satisfy `previous_coupon <= settleme` At a coupon date, supply that date as `previous_coupon` to obtain zero accrued interest.
`k`	Number of regular coupon payments per year, a positive integer; defaults to 1. For data-frame input, the `k` column, if present, takes precedence over this argument.
`day_count`	Day-count convention, one of `"act/act"`, `"act/360"`, `"act/365"`, or `"30e/360"`. The default is `"act/act"`, defined as actual days divided by actual days in the coupon period, with annual coupon fraction `1/k`. For data-frame input, a `day_count` column, if present, takes precedence over this argument. `"30e/360"` denotes the European 30E/360 convention with both day-of-month values capped at 30.
`clean_price`	Optional quoted clean price, in the same currency units and for the same face value as `face`.
`dirty_price`	Optional dirty price, on the same basis. If either price is provided, the other is determined by the clean/dirty price identity. If both are supplied, their consistency is checked.
`tol`	Positive scalar tolerance for reconciling prices when both are supplied; finite and used as a scaled absolute tolerance. The bound is `tol * max(1, abs(clean_price), abs(dirty_price), abs(accrued_interest))` for each row. The unit floor is part of this price-reconciliation rule; it is not a yield-solver tolerance.

Details

Let F be face value, c the nominal annual coupon rate, and k the coupon frequency. The regular coupon is C = F * c / k.

For "act/act", with d actual elapsed calendar days and D actual days in the full coupon period, the year fraction and accrued interest are alpha = d / (k * D) and AI = F * c * alpha = C * d / D.

For "act/360" and "act/365", alpha equals actual elapsed calendar days divided by 360 or 365 respectively; for "30e/360", it equals the number of 30E/360 days divided by 360. These conventions accrue at the annual coupon rate and therefore need not reach exactly one regular coupon at the end of every actual calendar coupon period.

The price identity is dirty_price = clean_price + accrued_interest. A price supplied to this function is *not* computed from a yield curve. Calculations assume a fixed-rate bond with regular coupon periods, no stubs, no ex-coupon adjustment, no withholding tax, no amortizing principal, and no transaction costs. The caller supplies the correct bracketing coupon dates: a full coupon schedule is not inferred.

All direct-input arguments of length greater than one must have the same length; length-one values are recycled. Calendar dates must have class Date, not bare character strings or numeric date serials. Finite coupon and accrued amounts are recovered from overflowing intermediate products when possible; nonrepresentable final amounts cause an error.

Value

A tibble. For direct input, it contains the bond parameters, settlement/coupon dates, the chosen convention, coupon_per_period, days_elapsed, days_in_period, year_fraction, coupon_fraction, accrued_interest, clean_price, and dirty_price. For data-frame input, original columns are retained and the audit columns added. When neither price is supplied, both price columns contain NA_real_.

See Also

bond_cash_flows(), bond_price(), bond_ytm()

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

Examples

```
accrued_interest(
  face = 1000, c = 0.06, k = 2,
  previous_coupon = as.Date("2026-01-01"),
  settlement = as.Date("2026-04-01"),
  next_coupon = as.Date("2026-07-01"),
  clean_price = 1005
)

# The same position under two market day-count conventions.
accrued_interest(
  face = 1000, c = 0.06, k = 2,
  previous_coupon = as.Date("2026-01-01"),
  settlement = as.Date("2026-04-01"),
```

```
  next_coupon = as.Date("2026-07-01"),
  day_count = "30e/360"
)

# Pipe-friendly portfolio input: each row is one bond position.
bonds <- tibble::tibble(
  bond = c("A", "B"), face = c(1000, 500), c = c(0.06, 0.04),
  k = c(2L, 2L),
  previous_coupon = as.Date(c("2026-01-01", "2026-01-01")),
  settlement = as.Date(c("2026-04-01", "2026-04-01")),
  next_coupon = as.Date(c("2026-07-01", "2026-07-01"))
)
bonds |> accrued_interest()
```

`accumulation_factor` *Accumulation factor under conventional or time-varying interest*

Description

Computes the accumulation factor from time `s` to time `t` using exactly one of three interest specifications:

- a conventional interest rate `i`;
- an accumulation function `a(t)`;
- a time-varying force of interest `delta(t)`.

Usage

```
accumulation_factor(
  s = 0,
  t,
  i = NULL,
  i_type = "effective",
  m = 1,
  a = NULL,
  delta = NULL,
  subdivisions = 100L,
  rel.tol = 1e-08,
  tidy = FALSE
)
```

Arguments

`s`	Numeric vector of starting times. Defaults to 0.
`t`	Numeric vector of ending times.
`i`	Optional numeric vector of conventional interest-rate values.

`i_type`	Character vector indicating the conventional interest-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`. Used only when `i` is supplied.
`m`	Positive integer vector giving the conversion frequency for nominal rates. Used only when `i` is supplied.
`a`	Optional accumulation function of one numeric time argument. The function must return one positive finite numeric value for each supplied time.
`delta`	Optional force-of-interest function of one numeric time argument. Numerical integration is performed separately over each interval $[s, t]$.
`subdivisions`	Positive integer giving the maximum number of subintervals used by `stats::integrate()` when `delta` is supplied.
`rel.tol`	Positive relative tolerance passed to `stats::integrate()` when `delta` is supplied.
`tidy`	Logical scalar. If `FALSE`, returns a numeric vector. If `TRUE`, returns a tibble with the inputs and intermediate quantities.

Details

The accumulation factor is

$$A(s,t) = (1+i)^{t-s}$$

for a constant annual effective rate,

$$A(s,t) = \frac{a(t)}{a(s)}$$

for an accumulation function, and

$$A(s,t) = \exp\left(\int_s^t \delta(u)\,du\right)$$

for a time-varying force of interest.

Exactly one of `i`, `a`, or `delta` must be supplied.

Numeric arguments must have length 1 or a common length. Scalars are recycled over the remaining scenarios, making the function suitable for use inside `dplyr::mutate()` pipelines.

Times must satisfy $0 \le s \le t$. Missing times propagate as missing accumulation factors. Conventional inputs are valued using their original log accumulation rate. In tidy output, the annual equivalent may round to -1 or overflow to Inf even when the requested interval factor is positive and finite.

Value

If `tidy = FALSE`, a numeric vector of accumulation factors.

If `tidy = TRUE`, a tibble containing the interval, model, relevant intermediate quantities, and accumulation factor.

See Also

standardize_interest, present_value, future_value

Other interest: bootstrap_spot_curve(), discount_factor(), discount_factor_spot(), forward_curve(), forward_rate(), interest_equivalents(), par_rate(), standardize_interest(), yield_curve()

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

Examples

```
# Constant annual effective interest
accumulation_factor(t = 5, i = 0.07)

# The same financial law in three representations
c(
  rate = accumulation_factor(t = 5, i = 0.03, i_type = "force"),
  accumulation = accumulation_factor(t = 5, a = function(t) exp(0.03 * t)),
  varying_force = accumulation_factor(t = 5, delta = function(t) 0.03)
)

# One rate recycled over several times
accumulation_factor(
  t = c(1, 2, 5),
  i = 0.07
)

# Accumulation function
a_fun <- function(t) {
  exp(0.03 * t + 0.002 * t^2)
}

accumulation_factor(
  s = 2,
  t = 5,
  a = a_fun
)

# Time-varying force of interest
delta_fun <- function(t) {
  0.03 + 0.004 * t
}

accumulation_factor(
  s = 2,
  t = 5,
  delta = delta_fun
)

# Pipe-friendly use with a small tibble
if (requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  scenarios <- tibble::tibble(
```

```
    i = 0.07,
    t = c(1, 2, 3, 5)
  )

  scenarios |>
    dplyr::mutate(
      accumulation = accumulation_factor(
        t = t,
        i = i
      )
    )
}
```

`add_insurance` *Add an insurance benefit specification to a life contract*

Description

Adds the benefit side of a single-life or two-life insurance contract to an existing [life_contract()] object. This function stores the specification only; it does not calculate an actuarial present value.

Usage

```
add_insurance(
  contract,
  type = c("whole", "term", "endowment", "pure_endowment", "variable_k"),
  benefit = 1,
  n = Inf,
  h = 0L,
  k = 1L,
  frac = c("UDD", "CF", "CML", "Balducci")
)
```

Arguments

`contract`	A ‘tidyact_life_contract‘ object.
`type`	Insurance type. Single-life contracts support ‘"whole"‘, ‘"term"‘, ‘"endowment"‘, and ‘"variable_k"‘. Two-life contracts support ‘"whole"‘, ‘"term"‘, ‘"endowment"‘, and ‘"pure_endowment"‘.
`benefit`	Benefit amount. For standard products, a single nonnegative numeric value. For single-life ‘type = "variable_k"‘, a numeric vector or a function of time may be supplied.
`n`	Insurance term in years. Use ‘Inf‘ for whole-life insurance.
`h`	Nonnegative integer deferment period in years.
`k`	Positive integer. Benefit frequency for single-life ‘type = "variable_k"‘.
`frac`	Fractional-age assumption: ‘"UDD"‘, ‘"CF"‘, ‘"CML"‘, or ‘"Balducci"‘.

Details

For two-life contracts, the insurance status is inferred from ‘contract$lives‘: ‘"joint"‘ becomes joint-life status and ‘"last_survivor"‘ becomes last-survivor status.

Components may be added in either order and unrelated contract fields are retained. Standard finite terms must round to at least one year. Large integer-valued terms, deferments and frequencies are stored as doubles when outside R’s integer range; feasibility of a valuation horizon is checked by the consumer. Variable-benefit functions are stored without being evaluated.

Value

The original contract with an ‘insurance‘ component.

See Also

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

`add_premium_schedule` *Add a contingent premium-payment schedule to a life contract*

Description

Adds the premium-payment side of a single-life or two-life insurance contract. The schedule defines when premiums are payable while the selected life status is in force. This function stores the specification only.

Usage

```
add_premium_schedule(
  contract,
  k = 1L,
  timing = c("due", "immediate"),
  premium_start = c("issue", "deferred"),
  n_prem = NULL,
  woolhouse = c("none", "first", "second"),
  frac = NULL
)
```

Arguments

`contract`	A ‘tidyact_life_contract‘ object.
`k`	Positive integer. Number of premium payments per year.
`timing`	Timing of premium payments: ‘"due"‘ or ‘"immediate"‘.
`premium_start`	Start of premium payments: ‘"issue"‘ or ‘"deferred"‘.

`n_prem`	Premium-paying term in years. Use ‘NULL‘ to infer the term from the insurance component. A fractional value is allowed when ‘n_prem * k‘ is an integer.
`woolhouse`	Woolhouse approximation order: ‘"none"‘, ‘"first"‘, or ‘"second"‘.
`frac`	Fractional-age assumption. If ‘NULL‘, the insurance component’s assumption is used when available; otherwise ‘"UDD"‘.

Details

The fractional-age assumption is resolved when this component is added: later replacement of the insurance component does not change this stored assumption. ‘n_prem = NULL‘ remains unresolved until valuation. Finite schedules must contain a finite, positive whole number of payments. Large integer-valued frequencies are stored as doubles when needed.

Value

The original contract with a ‘premium_schedule‘ component.

See Also

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

`aggregate_distribution` *Compute the aggregate loss distribution*

Description

Evaluates the distribution of the collective aggregate loss

$$S = \sum_{i=1}^{N} X_i$$

from a `tidyact_collective_risk_spec` object.

The function separates the actuarial model from the computational algorithm. It never fits a distribution, never simulates the aggregate loss, and never discretizes a continuous severity implicitly.

Usage

```
aggregate_distribution(
  x,
  method = c("auto", "analytic", "convolution", "panjer", "fft"),
  tol = 1e-08,
  max_loss = NULL,
  ...
)
```

Arguments

x	A tidyact_collective_risk_spec object.
method	Computational method. One of "auto", "analytic", "convolution", "panjer", or "fft".
tol	Positive finite probability tolerance smaller than one. It is used when an infinite claim-count distribution must be truncated numerically or when Panjer recursion is allowed to stop after capturing at least 1 - tol of the aggregate probability.
max_loss	Optional nonnegative finite numeric scalar giving the largest aggregate-loss value to retain in the returned grid. Probability above the retained grid is not renormalized; it is reported separately as tail_probability, together with any omitted claim-count probability.
...	Reserved for future computational controls. Additional arguments are currently rejected.

Details

aggregate_distribution() operates only on an already specified collective model. The probabilistic model is never changed in order to make an algorithm applicable.

In particular, if the severity distribution is continuous and no supported analytic form is available, the function stops and asks the user to apply discretize_severity() explicitly before using Panjer recursion or FFT.

Automatic method selection:

With method = "auto", the function uses the following hierarchy:

1. an implemented analytic representation for deterministic aggregate structures;
2. Panjer recursion when the severity is on a lattice and the frequency distribution supplies valid Panjer parameters;
3. exact finite convolution when both frequency and severity have finite discrete support;
4. FFT when the severity is on a lattice and the frequency distribution supplies the probability operations required to truncate its count support.

If none of these conditions is satisfied, an informative error is returned.

Analytic method:

The initial analytic engine handles aggregate losses that are degenerate at zero and deterministic severities. If $X = c$ almost surely, then

$$S = cN$$

and the aggregate probabilities are obtained directly from the count distribution. Infinite count support is truncated according to tol unless max_loss supplies a tighter explicit cutoff.

Additional analytic compound families may be added later without changing the public interface.

Convolution method:

The convolution engine is exact for finite claim-count support and finite discrete severity support. It evaluates

$$P(S \in A) = \sum_n P(N = n) P(X_1 + \cdots + X_n \in A)$$

by repeated finite convolution. If `max_loss` is supplied, the full finite distribution is evaluated first and probability above the requested cutoff is reported as tail probability.

Panjer method:

For a frequency distribution in the $(a, b, 0)$ class,

$$p_n = \left(a + \frac{b}{n}\right) p_{n-1},$$

and a lattice severity with probabilities f_j, the aggregate probabilities satisfy

$$g_0 = G_N(f_0)$$

and, for $k \geq 1$,

$$g_k = \frac{\sum_{j=1}^{k} \left(a + \frac{bj}{k}\right) f_j g_{k-j}}{1 - a f_0}.$$

Only severity probabilities available on the lattice are used. No discretization is performed by this engine.

FFT method:

FFT evaluates a finite polynomial approximation to the compound distribution. For an infinite claim-count distribution, the count support is first truncated at a quantile whose omitted probability is no greater than `tol`. This omitted count probability is carried explicitly into `tail_probability`; it is not wrapped into the retained FFT grid.

The FFT engine therefore uses the truncated count polynomial

$$\sum_{n=0}^{n_{\max}} p_n z^n$$

rather than the full PGF. This avoids silently treating circular aliasing as aggregate probability.

Returned probabilities:

The returned `grid` contains:

- `loss`: aggregate loss amount;
- `probability`: probability mass assigned to that amount;
- `cdf`: cumulative probability captured on the retained grid.

The grid is intentionally not renormalized when a positive tail remains. Consequently,

$$\sum p_s + p_{\text{omitted}} = 1$$

up to numerical tolerance. With loss-grid truncation alone, omitted mass is above the grid. FFT count truncation also omits events with large claim counts but potentially small aggregate losses (especially when zero severities are possible). Thus `tail_probability` is total unrepresented mass, not necessarily `P(S > max(grid$loss))`. VaR checks whether omitted count mass could change its selected quantile; TVaR and stop-loss reject unresolved mass.

Value

An object with classes "tidyact_aggregate_distribution" and "tidyact_risk_distribution". It contains:

schema_version Internal schema version.

method The computational method actually used.

grid A tibble with loss, probability, and cdf.

support Support information for the retained grid.

tail_probability Probability not represented on the retained grid.

model The source collective risk specification.

label The model label, if any.

metadata Audit information about method selection, truncation, exactness, and numerical controls.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*, 2nd ed. Society of Actuaries.

Klugman, S. A., Panjer, H. H., and Willmot, G. E. (2019). *Loss Models: From Data to Decisions*, 5th ed. Wiley.

See Also

collective_risk_spec(), discretize_severity(), aggregate_moments()

Examples

```
frequency <- frequency_spec(
  "poisson",
  lambda = 4
)

severity <- severity_spec(
  "gamma",
  shape = 2,
  rate = 0.001
)

model <- collective_risk_spec(
  frequency,
  severity
)

# Continuous severities are never discretized implicitly.
## Not run:
aggregate_distribution(model)

## End(Not run)
```

```
lattice_severity <- discretize_severity(
  severity,
  step = 100
)

lattice_model <- collective_risk_spec(
  frequency,
  lattice_severity
)

aggregate_distribution(
  lattice_model,
  method = "panjer",
  tol = 1e-8
)

aggregate_distribution(
  lattice_model,
  method = "fft",
  tol = 1e-8
)
```

`aggregate_moments` *Calculate moments of a collective aggregate loss*

Description

Calculates analytical raw moments, central moments, or cumulants of a collective risk model

$$S = \sum_{i=1}^{N} X_i,$$

using the moments of the claim count distribution and the individual severity distribution stored in a `tidyact_collective_risk_spec` object.

No simulation, numerical integration, discretization, or approximation is performed by this function.

Usage

```
aggregate_moments(x, order = 1:4, type = "raw")
```

Arguments

`x`	A `tidyact_collective_risk_spec` object created by `collective_risk_spec()`.
`order`	Nonempty numeric vector of nonnegative integer moment orders. Cumulant orders must be positive.
`type`	One of `"raw"` (default), `"central"`, or `"cumulant"`.

Details

Let

$$m_j = E[X^j]$$

denote the raw severity moments and

$$\nu_k = E[(N)_k]$$

the factorial moments of the claim count. For positive integer r, the raw aggregate moment is evaluated through the random-sum identity

$$E[S^r] = \sum_{k=1}^{r} \nu_k B_{r,k}(m_1, \ldots, m_{r-k+1}),$$

where $B_{r,k}$ is a partial exponential Bell polynomial.

This formulation includes the familiar identities

$$E[S] = E[N]E[X]$$

and

$$\mathrm{Var}(S) = E[N]\,\mathrm{Var}(X) + \mathrm{Var}(N)(E[X])^2.$$

Central moments and cumulants are derived algebraically from aggregate raw moments. For a compound Poisson model, cumulants use the direct identity

$$\kappa_r(S) = \lambda E[X^r]$$

whenever the corresponding severity moment is finite.

The function respects the status information returned by `distribution_moments()`. Results are classified as:

- `"finite"`: a finite numerical result;
- `"infinite"`: a mathematically divergent raw or central moment;
- `"undefined"`: the requested quantity cannot be defined from the available moments;
- `"overflow"`: a theoretically finite calculation exceeded floating-point range.

Because both frequency and severity are nonnegative in this module, a finite first moment equal to zero identifies a degenerate-at-zero component. If either $N = 0$ almost surely or $X = 0$ almost surely, then the aggregate loss is degenerate at zero and all positive-order aggregate moments and cumulants are zero.

Value

A `tibble::tibble()` with one row per requested order and columns: `type`, `order`, `value`, and `status`.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*, 2nd ed. Society of Actuaries.

Klugman, S. A., Panjer, H. H., and Willmot, G. E. (2019). *Loss Models: From Data to Decisions*, 5th ed. Wiley.

See Also

`collective_risk_spec()`, `distribution_moments()`

Examples

```
frequency <- frequency_spec(
  "poisson",
  lambda = 4
)

severity <- severity_spec(
  "gamma",
  shape = 2,
  rate = 0.001
)

model <- collective_risk_spec(
  frequency,
  severity
)

aggregate_moments(model)

aggregate_moments(
  model,
  order = 1:4,
  type = "central"
)

aggregate_moments(
  model,
  order = 1:4,
  type = "cumulant"
)
```

`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,
  i,
  i_type = "effective",
  m = 1L,
  k = 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.
`i`	Numeric scalar. Annual interest-rate input.
`i_type`	Character string indicating the annual interest-rate type: `"effective"`, `"nominal_interest"`, `"nominal_discount"`, or `"force"`.
`m`	Positive integer. Conversion frequency for nominal annual rates.
`k`	Positive integer. Number of amortization 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`	Finite positive numeric tolerance in monetary units for zero-balance detection.

Details

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

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.

The original log accumulation rate determines the effective period rate and the level payment, avoiding cancellation near zero and saturation of the annual equivalent. The periodic rate must remain finite and greater than -1. The balance tolerance is in monetary units and does not classify nonzero interest rates as zero. Here n counts periods, so the contractual term in years is n / k.

This function follows the compact actuarial notation used throughout tidyactuarial: i is the interest rate, i_type is the interest-rate type, m is the conversion frequency for nominal annual rates, and k is the number of amortization periods per year.

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.

Named extra payments must identify exact integer periods; repeated names are summed. On a payoff row, requested extra principal has priority and the regular payment is reduced to the remaining amount due. Interest still follows the selected timing, and no rows follow full payoff. The general schedule uses a different allocation priority; see amort_schedule_general.

With adjust = "payment", the payment is recalculated after every realized period for the remaining contractual periods, even if no extra payment occurred. A supplied payment is the initial payment. The "none" and "term" policies keep the regular amount; both stop when the debt is repaid and may realize fewer than n rows. A requested loan below tol may produce an empty schedule. Calculated payments for active balances must be representable and positive; non-finite monetary ledger entries cause an explicit error. This does not require a representable displayed annual effective equivalent. The schedule allocates up to n rows; feasible terms are limited by integer indexing and available memory.

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.

k Schedule frequency.

timing Payment timing convention.

adjust Adjustment rule used.

See Also

a_angle, present_value, pv_flow, standardize_interest

Other amortization: amort_schedule_general(), sinking_fund_schedule()

Examples

```
amort_schedule(
  principal = 100000,
  n = 12,
  i = 0.12,
  i_type = "nominal_interest",
  m = 12,
  k = 12
)

amort_schedule(
  principal = 100000,
  n = 24,
  i = 0.12,
  i_type = "nominal_interest",
  m = 12,
  k = 12,
  extra_principal = c("6" = 5000, "12" = 3000),
  adjust = "term"
)

amort_schedule(
  principal = 100000,
  n = 24,
  i = 0.12,
  i_type = "nominal_interest",
  m = 12,
  k = 12,
  extra_principal = c("6" = 5000, "12" = 3000),
  adjust = "payment"
)
```

amort_schedule_general *General amortization schedule with variable rates and payments*

Description

Builds a loan amortization schedule allowing the effective interest-rate pattern, regular-payment pattern, and extra-principal pattern to vary by period.

Usage

```
amort_schedule_general(
  principal,
  n,
  i,
  i_type = "effective",
  m = 1L,
  k = 1L,
  timing = c("immediate", "due"),
  payment = NULL,
  extra_principal = 0,
  output = c("schedule", "summary"),
  tol = 1e-08
)
```

Arguments

principal	Positive numeric scalar. Initial outstanding balance.
n	Positive integer scalar. Maximum number of amortization periods.
i	Numeric vector of interest-rate inputs. It must have length 1 or n. A scalar is recycled over all periods.
i_type	Character vector indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force". It must have length 1 or n.
m	Positive integer vector giving the conversion frequency for nominal rates. It must have length 1 or n.
k	Positive integer scalar. Number of amortization periods per year.
timing	Character scalar. Either "immediate" for payments at the end of each period or "due" for payments at the beginning.
payment	Payment specification. It may be: • NULL, in which case a level payment is calculated; • a nonnegative numeric scalar; • a nonnegative numeric vector of length n;

- a function with arguments `period`, `ob_start`, and `i_effective_period`, returning one nonnegative payment.

`extra_principal`
: Optional nonnegative extra-principal payments. It may be a scalar or a numeric vector of length `n`. Defaults to zero.

`output`
: Character scalar. Either `"schedule"` for the complete amortization table or `"summary"` for a compact actuarial summary.

`tol`
: Finite positive numeric scalar in monetary units used for zero-balance detection and the negative-amortization threshold.

Details

The function extends the fixed-pattern workflow of `amort_schedule` without modifying that function. It supports:

- a scalar interest rate recycled over all periods;
- a vector of interest rates, one for each period;
- a level payment computed automatically;
- a scalar or period-specific payment vector;
- a payment function depending on the current loan state.

Original log accumulation rates are converted directly to period rates. Level-payment discount factors use cumulative log growth, rather than products of rounded accumulation factors. Period rates must remain finite and greater than -1; a displayed annual equivalent may overflow even when the period rate is representable. The term is `n / k` years.

Each value of `i` is interpreted as an annual rate under its associated `i_type` and `m`. It is first converted to an annual effective rate and then to the effective rate for one amortization period:

$$i_j^{(p)} = (1 + i_j)^{1/k} - 1.$$

When `payment = NULL`, the level payment is determined from the period-specific discount factors. For payments in arrears,

$$P = \frac{L}{\sum_{j=1}^{n} \prod_{h=1}^{j} (1 + i_h^{(p)})^{-1}}.$$

For payments in advance, the discount factors correspond to payment times $0, 1, \ldots, n-1$.

A payment smaller than the interest charged may produce negative amortization. The schedule identifies such periods explicitly.

Negative amortization means that interest exceeds the total payment, including extras, by more than `tol`. This condition applies to both timings, including interest accrued after an advance payment. Under due timing, the principal columns continue to record payments before interest; the negative-amortization flag refers to the net debt movement.

On a payoff row the regular payment is applied first and extras are capped at the remaining debt; no callback is evaluated after full payoff. This allocation differs from `amort_schedule`, which preserves the requested extra first. Both cap total cash flow at the amount due. Automatically calculated level payments are determined before extras and are not recast by this function. The

discount sum is evaluated in log space before combining it with the principal. Active balances require a positive, representable calculated payment; non-finite monetary ledgers cause errors. A principal below `tol` may produce an empty schedule, and small residual balances may be treated as zero within that absolute tolerance. Storage grows with `n`; feasible terms are limited by integer indexing and available memory.

Value

With `output = "schedule"`, a tibble with one row per realized period.

With `output = "summary"`, a one-row tibble containing the initial principal, number of realized periods, total interest, total paid, ending balance, and an indicator of negative amortization.

See Also

`amort_schedule`, `a_angle`, `standardize_interest`

Other amortization: `amort_schedule()`, `sinking_fund_schedule()`

Examples

```
# Level payments and a constant rate
amort_schedule_general(
  principal = 100000,
  n = 12,
  i = 0.12,
  i_type = "nominal_interest",
  m = 12,
  k = 12
)

# Increasing payments
amort_schedule_general(
  principal = 10000,
  n = 5,
  i = 0.06,
  payment = c(1800, 1900, 2000, 2200, 3000)
)

# Period-specific annual effective rates
amort_schedule_general(
  principal = 10000,
  n = 4,
  i = c(0.04, 0.05, 0.06, 0.07),
  payment = NULL
)

# Payment determined from the current balance
payment_rule <- function(
  period,
  ob_start,
  i_effective_period
) {
```

```
  1500 + 100 * (period - 1)
}

amort_schedule_general(
  principal = 8000,
  n = 6,
  i = 0.05,
  payment = payment_rule
)

# Compact summary
amort_schedule_general(
  principal = 100000,
  n = 60,
  i = 0.08,
  k = 12,
  output = "summary"
)
```

---

annuity_arith *Arithmetic annuity factor*

Description

Computes the actuarial present value or accumulated value factor for an arithmetic annuity, using compact actuarial notation.

Usage

```
annuity_arith(
  n,
  k = 1L,
  i,
  i_type = "effective",
  m = 1L,
  h = 0,
  timing = "immediate",
  pattern = "increasing",
  P1 = 1,
  g = 1,
  valuation = c("present", "accumulated"),
  tidy = FALSE
)
```

Arguments

n	Numeric vector of payment durations in years. Each value must be positive and finite.

k	Positive integer vector giving the number of payments per year.
i	Numeric vector of interest-rate values.
i_type	Character vector indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer vector giving the conversion frequency for nominal rates. Observed values are validated for every rate family, although only nominal conversions use this frequency.
h	Numeric vector of deferment times in years. Must be greater than or equal to 0.
timing	Character vector. One of "immediate" or "due".
pattern	Character vector. One of "increasing", "decreasing", or "custom".
P1	Numeric vector. First payment for pattern = "custom". Ignored for "increasing" and "decreasing".
g	Numeric vector. Arithmetic increment for pattern = "custom". Ignored for "increasing" and "decreasing".
valuation	Character string. Use "present" for present value or "accumulated" for accumulated value at the end of the term.
tidy	Logical scalar. If FALSE, returns a numeric factor. If TRUE, returns a tibble with intermediate quantities.

Details

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

Supported payment patterns:

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

Supported timing conventions:

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

This function follows the compact actuarial notation used throughout tidyactuarial: n is the term, k is the payment frequency, i is the interest-rate input, i_type is the interest-rate type, m is the conversion frequency for nominal rates, and h is the deferment period.

Rate validation uses standardize_interest. Valuation retains the original log accumulation rate, even if the displayed annual effective equivalent rounds to -1 or overflows.

For each scenario, the total number of payments is

$$N = nk.$$

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

Let $b = 0$ for immediate payments and $b = 1$ for due payments, $r = \delta/k$, and $H = hk$. The present value is $\sum_{j=1}^{N} P_j \exp[-r(H + j - b)]$; the value at $h + n$ years is $\sum_{j=1}^{N} P_j \exp[r(N - j + b)]$. In a custom stream, g is an increment in currency per payment, not an annual growth rate. Signed and zero payments are supported.

Payments are enumerated explicitly, so memory grows with N. Capital-factor products are evaluated stably using the shared cash-flow rules. Those rules do not recover arbitrary extreme signed cancellation across different factors, or payments already outside the numeric range.

Value

If `tidy = FALSE`, a numeric vector of arithmetic annuity factors.

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

See Also

a_angle, s_angle, standardize_interest

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

Examples

```
# Increasing arithmetic annuity
annuity_arith(n = 10, i = 0.05, pattern = "increasing")

# Decreasing arithmetic annuity
annuity_arith(n = 10, i = 0.05, pattern = "decreasing")

# Custom arithmetic annuity
annuity_arith(
  n = 10,
  i = 0.05,
  pattern = "custom",
  P1 = 100,
  g = 25
)

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

# Tibble output
annuity_arith(
  n = 10,
  i = 0.05,
  pattern = "decreasing",
  tidy = TRUE
)
```

annuity_geom *Geometric annuity factor*

Description

Computes the actuarial present value or accumulated value factor for a geometric annuity, using compact actuarial notation.

Usage

```
annuity_geom(
  n = NULL,
  k = 1L,
  i,
  i_type = "effective",
  m = 1L,
  g = 0,
  g_type = "effective",
  g_m = 1L,
  h = 0,
  timing = "immediate",
  P1 = 1,
  perpetuity = FALSE,
  valuation = c("present", "accumulated"),
  tidy = FALSE
)
```

Arguments

n	Numeric vector of payment durations in years. Ignored only when `perpetuity = TRUE` and `valuation = "present"`. Otherwise, each value must be positive and finite.
k	Positive integer vector giving the number of payments per year.
i	Numeric vector of interest-rate values.
i_type	Character vector indicating the interest-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
m	Positive integer vector giving the conversion frequency for nominal interest-rate inputs. Observed values are validated for every rate family, although only nominal conversions use this frequency.
g	Numeric vector of annual growth-rate values for the payments.
g_type	Character vector indicating the growth-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.

g_m	Positive integer vector giving the conversion frequency for nominal growth-rate inputs. Observed values are validated for every rate family, although only nominal conversions use this frequency.
h	Numeric vector of deferment times in years. Must be greater than or equal to 0. For present values, the deferment discounts the payment block. For accumulated values, it is recorded but does not change the factor under the adopted terminal-horizon convention.
timing	Character vector. One of "immediate" or "due".
P1	Numeric vector giving the first payment of the geometric sequence.
perpetuity	Logical vector. If TRUE, computes the geometric perpetuity present value. Perpetuities are not supported for valuation = "accumulated".
valuation	Character string. Use "present" for present value or "accumulated" for accumulated value at the end of the term.
tidy	Logical scalar. If FALSE, returns a numeric factor. If TRUE, returns a tibble with intermediate quantities.

Details

This function covers finite geometric annuities and geometric perpetuities.

Supported timing conventions:

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

This function follows the compact actuarial notation used throughout tidyactuarial: n is the term, k is the payment frequency, i is the interest-rate input, i_type is the interest-rate type, m is the conversion frequency for nominal interest rates, g is the growth-rate input, g_type is the growth-rate type, g_m is the conversion frequency for nominal growth rates, h is the deferment period, and P1 is the first payment.

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

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

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

If $i_p \neq g_p$, then

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

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

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

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

Valuation retains the original annual log rates δ_i and δ_g. Let $r = \delta_i/k$, $s = \delta_g/k$, $H = hk$, and $b = 0$ for immediate or $b = 1$ for due payments. With $G_N(z) = \sum_{j=0}^{N-1} \exp(zj)$, the finite present value is

$$P_1 \exp[-r(H + 1 - b)]G_N(s - r).$$

The accumulated value at $h + n$ years is

$$P_1 \exp[br + (N - 1)\max(r, s)]G_N(-|s - r|).$$

Stable logarithmic sums avoid subtracting nearly equal effective rates or accumulation levels. The spread is computed before period conversion as $(\delta_g - \delta_i)/k$; no nonzero spread is replaced by zero. For matching effective rates or nominal rates with equal conversion frequencies, log-ratio identities preserve close input differences before taking logarithms. Widely separated ratios that lose range fall back to the original log rates. Different rate families retain the accuracy limits of their separate conversions.

A perpetuity requires $\delta_i > \delta_g$, without an arbitrary numerical gap. Its value is $P_1 \exp[-r(H + 1 - b)]/[1 - \exp(s - r)]$. Missing finite terms propagate missing values. Signed and zero first payments retain their meaning. Standalone audit rates may saturate, while monetary values combine the first payment with log factors before declaring range loss. No arbitrary precision or universal extreme-range guarantee is implied. Unlike g in `annuity_arith`, this g is an annual growth-rate specification and not a monetary increment per payment.

Value

If `tidy = FALSE`, a numeric vector.

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

See Also

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

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

Examples

```
# Present value of a geometric annuity
annuity_geom(
  n = 10,
  i = 0.05,
  g = 0.02,
  valuation = "present"
)

# Accumulated value of a geometric annuity
annuity_geom(
  n = 10,
  i = 0.05,
  g = 0.02,
  valuation = "accumulated"
```

```
)

# Nominal interest and nominal growth
annuity_geom(
  n = 10,
  k = 12,
  i = 0.06,
  i_type = "nominal_interest",
  m = 12,
  g = 0.024,
  g_type = "nominal_interest",
  g_m = 12
)

# Tibble output
annuity_geom(
  n = 10,
  i = 0.05,
  g = 0.02,
  tidy = TRUE
)
```

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

Description

Computes the APV of a discrete annuity contingent on multiple independent lives using compact actuarial notation.

Usage

```
annuity_multi(
  lt,
  ages,
  i,
  i_type = "effective",
  m = 1L,
  n = NULL,
  h = 0L,
  k = 1L,
  annuity = c("cohort", "reversionary"),
  cohort = c("first", "last"),
  alpha = NULL,
  timing = c("immediate", "due"),
  woolhouse = c("none", "first", "second"),
  frac = c("UDD", "CF", "CML", "Balducci")
)
```

Arguments

lt	A life table object (data frame or tibble) with column x and at least one of lx, px, or qx. For different mortality assumptions by life, pass a list of life tables of the same length as ages, for example list(lt_male, lt_female).
ages	Integer vector of actuarial ages for the lives at issue. Must have length 1, 2, or 3.
i	Numeric scalar. Annual interest-rate input.
i_type	Character string indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer. Conversion frequency for nominal rates. Ignored for i_type = "effective" and i_type = "force". In tidyactuarial, m is reserved for interest conversion frequency, not deferment.
n	Integer term in years after deferment. If NULL, runs to the end of the available table horizon, conservatively based on the smallest omega across life tables.
h	Integer deferment period in years.
k	Integer payments per year. If k > 1, Woolhouse approximations may be applied.
annuity	Type of annuity logic: "cohort" uses the status defined in cohort; "reversionary" uses the α fractional reduction.
cohort	Survival status: "first" for joint-life, paying while all lives are alive, or "last" for last-survivor, paying while at least one life 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.
timing	"immediate" or "due".
woolhouse	"none", "first", or "second".
frac	Fractional-age assumption for exact payments: "UDD" (default), "CF", "CML" (alias of CF), or "Balducci".

Details

This implementation supports up to three lives, which covers the most common practical multi-life arrangements.

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.

This function follows the compact actuarial notation used throughout tidyactuarial: i is the interest rate, i_type is the interest-rate type, m is the conversion frequency for nominal rates, n is the term, h is the deferment period, and k is the payment frequency.

Under the assumption of independent future lifetimes, 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:

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

All values are measured at issue, conditional on the stated lives being alive at issue. Ages identify lives within one policy, not policy scenarios. A k-thly payment is $1/k$ of an annual unit: exact valuation sums $F(h + j/k)/k$, with $F(t) = e^{-\delta t}w(t)$ and the expected payment weight $w(t)$ defined above. Due uses $j = 0, \ldots, nk - 1$; immediate uses $j = 1, \ldots, nk$. The historical name `"reversionary"` means joint-and-survivor payments; it does not mean payments only after a death.

For `k > 1`, `woolhouse = "none"` uses the actual fractional payment grid. First-order Woolhouse subtracts $(1 - 1/k)[F(h) - F(h + n)]/2$ from the annual due value. Second-order also subtracts $(1 - 1/k^2)[(\delta + \eta_h)F(h) - (\delta + \eta_{h+n})F(h + n)]/12$. Here $\eta_t = -\log(w(t + 1)/w(t))$ is an annual forward estimate of the logarithmic decay of the payment weight, not an exact instantaneous force. Positive supported weights at both borders and the following year are required for second-order approximation. This is a smoothness approximation, not exact fractional valuation under `frac`; negative approximations produce a warning. Immediate subtracts $[F(h) - F(h + n)]/k$ from due. Exact valuation requires consecutive annual ages and retains the documented conservative common-age horizon; it does not extrapolate unobserved lives.

Value

A single numeric value representing the APV.

See Also

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

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

Examples

```
lt <- data.frame(
  x = 60:90,
  lx = seq(100000, 0, length.out = 31)
)

annuity_multi(
  lt = lt,
  ages = c(60, 62),
  i = 0.05,
  n = 5,
  cohort = "first",
  timing = "due"
)

# Half-yearly payments, each equal to half an annual unit, under UDD
annuity_multi(lt, ages = c(60, 62), i = 0.05, n = 5, k = 2,
              timing = "due", frac = "UDD")
```

```
annuity_multi(
  lt = lt,
  ages = c(60, 62),
  i = 0.05,
  n = 5,
  cohort = "last",
  timing = "due"
)
```

---

annuity_x *Actuarial present value of a life annuity*

Description

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

Usage

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

Arguments

lt	A life table as produced by lifetable, or a tidyact_life_contract object created by life_contract. A life table must contain columns x and lx.
x	Integer actuarial age, or a vector of scenario ages. Optional when lt is a single-life tidyact_life_contract.
i	Numeric annual interest-rate input, scalar or by scenario. Optional when lt is a single-life tidyact_life_contract.
i_type	Character string indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer. Conversion frequency for nominal interest rates. Ignored for i_type = "effective" and i_type = "force". In tidyactuarial, m is reserved for interest conversion frequency, not deferment.

n	Numeric term in years. Use NULL or Inf for a whole-life annuity. For exact k-thly valuation (woolhouse = "none"), fractional terms are allowed when n * k is an integer. Woolhouse approximations require an integer number of years.
h	Integer deferment period in years.
k	Positive integer. Number of annuity payments per year. For example, use k = 12 for monthly payments. The annuity is normalized to an annual payment rate of 1, so each individual payment has amount 1 / k.
timing	Character string. Either "immediate" for payments at the end of each payment period or "due" for payments at the beginning of each payment period.
woolhouse	Character string. For k > 1, use "none" for exact fractional-age computation, "first" for the first-order Woolhouse approximation, or "second" for the second-order Woolhouse approximation.
frac	Character string. Fractional-age assumption used when k > 1 and woolhouse = "none". Allowed values are "UDD", "CF", "CML", and "Balducci". If NULL, the frac attribute of lt is used when available; otherwise "UDD" is used.
tidy	Logical. If FALSE, returns a numeric APV. If TRUE, returns one tibble row per scenario with intermediate quantities.

Details

The function supports:

- whole-life annuities,
- temporary annuities,
- integer deferment,
- annual or k-thly payments,
- exact fractional survival for k-thly payments,
- first- and second-order Woolhouse approximations.

This function follows the compact actuarial notation used throughout tidyactuarial:

- lt: life table;
- x: actuarial age;
- i: interest rate;
- i_type: interest-rate type;
- m: interest conversion frequency;
- n: annuity term;
- h: deferment period;
- k: payment frequency.

For annual annuities-due,

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

For annual annuities-immediate,

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

Deferment is handled through

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

where h is the deferment period.

For k-thly payments with `woolhouse = "none"`, fractional survival is computed under the selected fractional-age assumption. The payment stream is normalized to an annual payment rate of 1:

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

Consequently, each k-thly installment has amount $1/k$. This normalization is essential when the function is used as the premium annuity in the equivalence principle: the resulting premium is annualized, and the amount paid at each installment is the annualized premium divided by k.

Scenarios and tidy workflows. All arguments except `lt` and `tidy` may have a common positive length; only length-one arguments recycle. `n = NULL` and `frac = NULL` keep their scalar default meanings. Scenarios are paired by position, not expanded to a Cartesian product, and each describes one life under the shared table. Invalid or missing inputs are rejected with the scenario number; no partial result is returned. Use `dplyr::mutate()` with `tidy = FALSE` for portfolios, or `tidy = TRUE` for an auditable scenario table. Different mortality tables require separate calls, for example in a list-column workflow.

Model and numerical limits. Discounting retains the original log accumulation rate; exact payments are valued from the issue age, avoiding products of separately rounded deferment and conditional values. Intermediate diagnostic values can be zero or infinite while the APV remains finite. Tables must have an annual age grid and nonincreasing finite `lx`, with a positive count at issue. For compatibility, a count of zero is assumed at one year beyond the last recorded age, even for an open input table. Known extinct tails contribute zero. Woolhouse is an approximation using annual survival estimates of boundary forces, not an exact integral; its second order requires positive survival where a boundary force is needed. Negative approximations generate a warning and are not clamped. Prefer `woolhouse = "none"` for exact payments under the chosen fractional assumption. At `n = 0`, APV and the annuity value at start are zero, while the zero-term pure endowment factor is one.

Value

If `tidy = FALSE`, a numeric vector of actuarial present values, of length one for a single scenario.

If `tidy = TRUE`, a tibble with one row per scenario, the main input values, equivalent interest rate, deferment factor, pure endowment factor, and APV.

See Also

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

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

Examples

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

# Annual annuity-immediate
annuity_x(
  lt = lt,
  x = 60,
  i = 0.06,
  timing = "immediate"
)

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

# Temporary annuity
annuity_x(
  lt = lt,
  x = 60,
  i = 0.06,
  n = 3,
  timing = "due"
)

# Deferred annuity
annuity_x(
  lt = lt,
  x = 60,
  i = 0.06,
  h = 2,
  timing = "due"
)

# Tidy output
annuity_x(
  lt = lt,
  x = 60,
  i = 0.06,
  n = 3,
  timing = "due",
  tidy = TRUE
)

# Pipe workflow with a life contract
life_contract(lt = lt, lives = "single", x = 60, i = 0.06) |>
```

```
  annuity_x(n = 3, timing = "due")

# Parallel single-life scenarios, suitable for mutate()
policies <- tibble::tibble(age = c(60, 61, 62), rate = c(0.03, 0.04, 0.05))
dplyr::mutate(policies,
  apv = annuity_x(lt, x = age, i = rate, n = 2, k = 12, timing = "due"))
annuity_x(lt, x = c(60, 61), i = 0.04, n = c(2, 3), tidy = TRUE)
```

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

Description

Computes the actuarial present value of a discrete annuity contingent on two independent lives, using compact actuarial notation.

Usage

```
annuity_xy(
  lt,
  x = NULL,
  y = NULL,
  i = NULL,
  i_type = "effective",
  m = 1L,
  status = c("joint", "last"),
  benefit = NULL,
  n = Inf,
  h = 0L,
  k = 1L,
  timing = c("immediate", "due"),
  woolhouse = c("none", "first", "second"),
  frac,
  tidy = FALSE,
  ...
)
```

Arguments

`lt`	A life table, a list of two life tables `list(lt_x, lt_y)`, or a `tidyact_life_contract` object created by `life_contract`. Each life table must contain columns `x` and `lx`.
`x`	Integer actuarial age for the first life.
`y`	Integer actuarial age for the second life.
`i`	Numeric scalar. Annual interest-rate input.

i_type	Character string indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer. Conversion frequency for nominal rates. Ignored for i_type = "effective" and i_type = "force".
status	Character string. Use "joint" for joint-life payments while both lives are alive, or "last" for last-survivor payments while at least one life is alive. Ignored when benefit is supplied.
benefit	Optional list with numeric scalar weights both, x_only, and y_only. These weights define the payment rate according to the survival state of the two lives. Weights may be signed; an expected payment is then a benefit weight, not necessarily a probability.
n	Term in years. Use Inf for the maximum horizon allowed by the life tables. For exact k-thly valuation (woolhouse = "none"), fractional terms are allowed when n * k is an integer. Woolhouse approximations require an integer number of years.
h	Nonnegative integer deferment period in years.
k	Positive integer. Number of payments per year. The annuity is normalized to an annual payment rate of 1, so each individual payment has amount 1 / k.
timing	Payment timing. Use "immediate" or "due".
woolhouse	Woolhouse approximation for k > 1: "none", "first", or "second".
frac	Fractional-age assumption for exact k-thly computation: "UDD", "CF", "CML", or "Balducci".
tidy	Logical scalar. If FALSE, returns a numeric APV. If TRUE, returns a one-row tibble.
...	Transitional compatibility for older calls using mortality_table, age_x, age_y, rate, rate_type, cohort, term_years, deferment_years, payments_per_year, and output.

Details

The function supports joint-life, last-survivor, and state-based reversionary-style payments. The life table input may be either one common table for both lives or a list of two life tables, one for each life.

This function follows the compact actuarial notation used throughout tidyactuarial: lt is the life table input, x and y are the two actuarial ages, i is the interest-rate input, i_type is the interest-rate type, m is the conversion frequency for nominal rates, n is the term, h is the deferment period, and k is the payment frequency.

The function assumes independent future lifetimes. For state-based benefits, the expected payment at time t is

$$b_{\text{both}}\,{}_tp_x\,{}_tp_y + b_{x\text{ only}}\,{}_tp_x(1 - {}_tp_y) + b_{y\text{ only}}\,{}_tp_y(1 - {}_tp_x).$$

When benefit = NULL, status = "joint" uses benefit = list(both = 1, x_only = 0, y_only = 0), while status = "last" uses benefit = list(both = 1, x_only = 1, y_only = 1).

For k-thly payments, the function values a payment rate of 1 per year, so each payment has size $1/k$. For exact valuation, a fractional term is admissible whenever it contains a whole number of payments:

$$nk \in \mathbb{N}.$$

Both lives are alive at issue; both, x_only and y_only refer to occupation at each payment date, without restarting both lives after a death. A designated reversion to y uses both=0, x_only=0, y_only=1; this differs from payments to either sole survivor and from a joint-and-survivor annuity paying while both are alive as well.

Exact valuation sums signed state payments on dates h+j/k, with each payment equal to its state rate divided by k. Cumulative death probabilities use direct survivor-count decrements and fractional interval masses, avoiding cancellation in 1-S. Original log interest, probabilities and monetary weights are combined before exponentiating when necessary.

Woolhouse approximations apply only to standard joint-life/last-survivor statuses. Put F(t)=exp(-g*t)w(t), where w is the status survival from issue and g is the input rate's annual log accumulation. First order subtracts (1-1/k)(F(h)-F(h+n))/2 from the annual due annuity. Second order also subtracts (1-1/k^2)((g+eta(h))F(h)-(g+eta(h+n))F(h+n))/12, with eta(t)=-log(w(t+1)/w(t)), a forward annual estimate, not an exact instantaneous force. Second order requires positive supported weights and the following year at both borders; otherwise use none or first. Immediate subtracts (F(h)-F(h+n))/k from the due approximation. A negative approximation is reported with a warning, without clipping.

Tables must satisfy the annual consistency requirements of `t_pxy`; issue ages must have positive lx for nonempty valuations. Payment grids must be finite, representable and no larger than R's integer range. One call values one contract; benefit vectors describe states, not scenarios. The existing table-horizon convention is retained: each marginal survival becomes zero beyond its last recorded age. On open tables this is truncation, not evidence of death, and may affect last-survivor or reversionary weights after the shorter horizon. Extend the tables for those interpretations. Infinite n means available horizon, not extrapolated lifetime coverage.

Value

If `tidy = FALSE`, a numeric scalar.

If `tidy = TRUE`, a one-row tibble with input values, standardized interest rate, term used, and APV.

See Also

`annuity_x`, `insurance_xy`, `premium_xy`, `t_pxy`, `t_px`

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

Examples

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

```
# Joint-life annuity-due
annuity_xy(
  lt = lt,
  x = 60,
  y = 62,
  i = 0.05,
  status = "joint",
  timing = "due"
)

# Last-survivor annuity-due
annuity_xy(
  lt = lt,
  x = 60,
  y = 62,
  i = 0.05,
  status = "last",
  timing = "due"
)

# Different life tables for the two lives
lt_m <- lt
lt_f <- data.frame(
  x  = 60:66,
  lx = c(100000, 99200, 98100, 96500, 94500, 92000, 89000)
)

annuity_xy(
  lt = list(lt_m, lt_f),
  x = 60,
  y = 62,
  i = 0.05,
  status = "joint",
  timing = "due"
)

# State-based reversionary-style payments
annuity_xy(
  lt = list(lt_m, lt_f),
  x = 60,
  y = 62,
  i = 0.05,
  benefit = list(both = 0, x_only = 1, y_only = 0),
  timing = "due"
)
```

---

apply_coverage *Apply an insurance contract to a loss severity specification*

Description

Creates a per-loss payment severity specification by combining an existing ground-up severity model with a contract created by `coverage_spec()`. The original model is retained and not discretized, fitted, or simulated.

Usage

```
apply_coverage(x, coverage, label = NULL)
```

Arguments

`x`	A severity specification inheriting from `tidyact_severity_spec`. It must represent nonnegative ground-up losses, not an existing payment specification.
`coverage`	A `tidyact_coverage_spec` object from `coverage_spec()`.
`label`	Optional nonempty character scalar. If `NULL`, the source severity label is inherited when present; otherwise the coverage label is used when present.

Details

The payment variable is defined on a **per-loss** basis, retaining every loss event even when the associated insurer payment is zero. This is not the conditional payment-per-payment severity distribution.

Let `X >= 0` be the ground-up severity, `f > 0` the inflation factor, `d_f` the indexed (or fixed) deductible, `L_f` the indexed (or fixed) layer limit, and `c` the insurer coinsurance fraction. The exact transformation is

$$Y_L = c \min\{(fX - d_f)_+, L_f\}$$

for an ordinary deductible and

$$Y_L = c \min\{fX, L_f\} 1(fX > d_f)$$

for a franchise deductible. The `limit` caps the covered layer **before** coinsurance; it is not a final insurer-payment limit.

A continuous ground-up distribution can become mixed: probability mass arises at zero and, when the limit is finite and binding, at the maximum payment `c * L_f`. Pre-existing atoms are preserved under the pushforward and may merge. When `coinsurance = 0` or `limit = 0`, the payment is identically zero. No smoothing or implicit discretization is performed.

The result is an executable per-loss payment distribution specification; its `source` and `coverage` fields contain everything necessary to recover the payment law from the underlying severity provider. The internal pure functions `.coverage_payment_amount()`, `.payment_cdf_from_loss()`, `.payment_boundary_atoms()`, and `.payment_discrete_pushforward()` implement the transformation exactly given loss values or exact source probabilities. Finite discrete sources are transformed exactly into a finite discrete distribution. Continuous or mixed sources use the existing custom-provider contract for CDF, quantiles, random generation and available exact moments. Moment orders without an exact registered formula are reported unavailable; no numerical integration or discretization is introduced silently.

Value

An S3 object with classes `tidyact_payment_spec`, `tidyact_severity_spec`, and `tidyact_distribution_spec`. The list stores `source`, `coverage`, `basis = "per_loss"`, `label`, and a normalized severity specification (`dist = "discrete"` for finite discrete losses or `dist = "custom"` for continuous and mixed losses). The support bound is not asserted to be the smallest possible bound under the original distribution.

References

Klugman, S. A., Panjer, H. H., and Willmot, G. E. (2012). *Loss Models: From Data to Decisions*, 4th ed., Chapter 8.

Frees, E. W., Lee, G., and Yang, L., *Loss Data Analytics*, chapters on claim severity and coverage modifications.

See Also

`coverage_spec()`

Examples

```
# Full, registered severity model.
ground_up <- severity_spec("gamma", shape = 2, rate = 0.001,
                           label = "Ground-up severity")
contract <- coverage_spec(deductible = 500, limit = 10000,
                          coinsurance = 0.8)
payments <- ground_up |> apply_coverage(contract)
inherits(payments, "tidyact_payment_spec")
payments$basis
payments$coverage$deductible
```

---

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 or by calendar dates.

Usage

```
apv_life_flow(
  lt,
  ages,
  t = NULL,
  date = NULL,
```

```
  date0 = NULL,
  cf,
  i,
  i_type = "effective",
  m = 1L,
  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. Use length 1 for a single life and length 2 or more for multiple lives.
`t`	Numeric vector of payment times in years, measured from time 0. Provide either `t` or `date`.
`date`	Optional vector of `Date` payment dates. Provide either `t` or `date`.
`date0`	Optional `Date` used as time 0 when `date` is provided. If missing, the minimum of `date` is used.
`cf`	Numeric vector of cash flows. Must have the same length as `t` or `date`.
`i`	Numeric scalar. Annual interest-rate input.
`i_type`	Character string indicating the interest-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Conversion frequency for nominal rates. Ignored for `i_type = "effective"` and `i_type = "force"`.
`status`	Survival status: `"single"`, `"first"`, `"last"`, or `"reversionary"`.
`alpha`	Reversionary fraction for `status = "reversionary"`. 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`.

This function follows the compact actuarial notation used throughout `tidyactuarial`: `t` denotes payment time, `cf` denotes cash flows, `i` denotes the interest rate, `i_type` denotes the interest-rate type, and `m` denotes the conversion frequency for nominal rates.

For each payment at time t, the APV contribution is

$$PV(t) = C(t)v^t P(\text{status alive at } t).$$

The survival probability depends on the `status`:

- `"single"`: ${}_tp_x$ for a single life.
- `"first"`: ${}_tp_{x_1} \cdot {}_tp_{x_2} \cdots$, so all lives must be alive.
- `"last"`: $1 - \prod_j (1 - {}_tp_{x_j})$, so at least one life must be alive.
- `"reversionary"`: full benefit while all lives are alive, and fraction α while at least one but not all lives are alive.

Fractional-year survival is computed under UDD within each year. Discounting uses the log accumulation rate from the original interest specification. A displayed discount factor may be zero or infinite even when its monetary APV contribution is representable.

Unknown mortality intervals produce missing contributions, unless survival has already reached zero. Missing numeric payment times and cash flows are retained as missing rows. Cumulative APV follows input row order and remains missing after a missing contribution. For compatibility, the `apv` attribute is the sum of nonmissing contributions (zero if none are known), so it is a subtotal when any contribution is missing, not a complete contract valuation. Calendar times use actual elapsed days divided by 365.25; all dates must be finite and nonmissing. Payments at the same time remain separate rows.

Value

A tibble with one row per payment and columns: `t`, `cf`, `surv_prob`, `discount`, `expected_cf`, `pv`, and `pv_cum`. If `date` was provided, a `date` column is included. The total APV is stored as `attr(result, "apv")`. The `contingency` attribute identifies `"survival"` or `"weighted"` (reversionary benefit fraction) for `plot_contingent_cash_flow()`.

Examples

```
lt <- data.frame(
  x = 40:100,
  lx = seq(100000, 0, length.out = 61)
)

apv_life_flow(
  lt = lt,
  ages = 40,
  t = c(1, 2, 3),
  cf = c(100, 100, 100),
  i = 0.05
)

apv_life_flow(
  lt = lt,
  ages = c(60, 58),
  t = c(1, 2, 3),
  cf = c(100, 100, 100),
  i = 0.05,
  status = "first"
)
```

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, using compact actuarial notation.

Usage

```
bond_book_value(
  face,
  c,
  n,
  t,
  k = 1L,
  y_effective_per_period = NULL,
  y = NULL,
  y_type = "effective",
  y_m = 1L,
  R = NULL,
  tol = 1e-10,
  check = TRUE,
  tidy = FALSE
)
```

Arguments

face	Numeric scalar. Face or par value of the bond.
c	Numeric scalar. Annual coupon rate as a proportion.
n	Numeric scalar. Final maturity in years.
t	Numeric vector. Valuation time(s) in years, measured from issue. Each value must lie between 0 and n and must align with coupon dates.
k	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	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. Conversion frequency for nominal annual yields.
R	Numeric scalar. Redemption value at final maturity. If NULL, defaults to face.
tol	Finite positive numeric scalar. Absolute tolerance for alignment of n * k and t * k with integer periods, not a monetary tolerance.

check	Logical scalar. If TRUE, performs input validation.
tidy	Logical scalar. If FALSE, returns a numeric vector of book values. If TRUE, returns a tibble with intermediate quantities.

Details

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

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

- at t = 0, the book value equals the bond price,
- at t = n, the book value is 0.

Assumptions:

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

Yield input conventions:

- If y_effective_per_period is supplied, it takes precedence over y, y_type, and y_m, and is interpreted as the effective yield per coupon period.
- Otherwise, y, y_type, and y_m define an annual yield specification. Valuation retains its original log accumulation rate even if an equivalent reported rate loses numeric range.

This function follows the compact bond notation used in tidyactuarial: P is price, face is the face value, c is the annual coupon rate, n is maturity, k is coupon frequency, y is the yield, R is redemption value, and t is valuation time.

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

$$BV_s = \sum_{r=s+1}^{N} C_r (1 + i_p)^{-(r-s)}.$$

This is the prospective book value on the bond's yield basis. Valuation times retain their supplied order and duplicates. At maturity the final coupon and redemption have both been paid, so the value is zero; the same convention gives zero for n = t = 0. Integer period counts must fit the cash-flow schedule, with sufficient memory for its rows.

Value

If tidy = FALSE, a numeric vector of book values, one for each t.

If tidy = TRUE, a tibble with valuation times, valuation periods, book values, yield information, and bond inputs.

See Also

bond_price, bond_cash_flows, bond_duration, bond_convexity

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

Examples

```
# Book value at time 0 equals price
bond_book_value(
  face = 100,
  c = 0.08,
  n = 5,
  t = 0,
  k = 2,
  y = 0.06,
  y_type = "effective"
)

# Book value at several coupon dates
bond_book_value(
  face = 100,
  c = 0.08,
  n = 5,
  t = c(0, 1, 2, 3, 4, 5),
  k = 1,
  y = 0.06,
  y_type = "effective"
)

# Tidy output
bond_book_value(
  face = 100,
  c = 0.08,
  n = 5,
  t = c(0, 1, 2, 3, 4, 5),
  k = 1,
  y = 0.06,
  y_type = "effective",
  tidy = TRUE
)

# Yield given directly per coupon period
bond_book_value(
  face = 1000,
  c = 0.05,
  n = 10,
  t = c(0, 2, 4, 6),
  k = 2,
  y_effective_per_period = 0.03
)
```

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

Description

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

Usage

```
bond_callable_price(
  face,
  c,
  n,
  k = 1L,
  call_t,
  call_R,
  y_effective_per_period = NULL,
  y = NULL,
  y_type = "effective",
  y_m = 1L,
  R = NULL,
  tol = 1e-10,
  check = TRUE,
  tidy = FALSE
)
```

Arguments

`face`	Numeric scalar. Face or par value of the bond.
`c`	Numeric scalar. Annual coupon rate as a proportion.
`n`	Numeric scalar. Final maturity in years. Must be strictly positive.
`k`	Positive integer. Number of coupon payments per year.
`call_t`	Numeric vector of callable times in years. Each value must be strictly between `0` and `n`, and must align with coupon dates.
`call_R`	Numeric vector of call prices corresponding to `call_t`.
`y_effective_per_period`	Optional numeric scalar. Effective yield per coupon period. If supplied, it is used directly.
`y`	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. Conversion frequency for nominal annual yields.
`R`	Numeric scalar. Redemption value at final maturity. If `NULL`, defaults to `face`.

tol	Finite positive numeric scalar. Absolute tolerance for alignment of n * k and call_t * k with integer periods. It is not a monetary tolerance and does not define ties between scenario prices.
check	Logical scalar. If TRUE, performs input validation.
tidy	Logical scalar. If FALSE, returns the worst-case callable-bond price. If TRUE, returns a tibble with all redemption scenarios.

Details

The bond is evaluated under each possible redemption scenario:

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

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

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

Assumptions:

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

Yield input conventions:

- If y_effective_per_period is supplied, it takes precedence over y, y_type, and y_m, and is interpreted as the effective yield per coupon period.
- Otherwise, y, y_type, and y_m define an annual yield specification. Scenario prices retain its original log accumulation rate even if a reported equivalent loses numeric range.

This function follows the compact bond notation used in tidyactuarial: face is the face value, c is the annual coupon rate, n is maturity, k is coupon frequency, y is the target yield, R is the final redemption value, call_t is the vector of call times, and call_R is the vector of call prices.

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

Then this function returns

$$\min_j P_j(y)$$

when tidy = FALSE.

This is the maximum price an investor can pay while still guaranteeing at least the target yield under the least favorable redemption scenario. A scenario pays all coupons through its redemption date, including the coupon on that date, plus its specified redemption value. There is no assumption of reinvestment at the target yield after early redemption and no stochastic model of interest rates or the issuer's exercise decision. Call dates must remain distinct and strictly inside maturity after alignment to integer coupon periods; two dates cannot represent the same coupon date. Rows are sorted by time and retain their original `scenario_id`.

Value

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

If `tidy = TRUE`, a tibble with one row per redemption scenario, including scenario prices and the worst-case indicator. `is_worst_case` is `TRUE` exactly when a computed scenario price equals the computed minimum, including all exact computed ties. Near ties are not marked.

See Also

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

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

Examples

```
# Callable bond with two possible call dates
bond_callable_price(
  face = 100,
  c = 0.08,
  n = 10,
  k = 2,
  call_t = c(5, 7),
  call_R = c(105, 102),
  y = 0.06,
  y_type = "effective"
)

# Tidy output with all redemption scenarios
bond_callable_price(
  face = 100,
  c = 0.08,
  n = 10,
  k = 2,
  call_t = c(5, 7),
  call_R = c(105, 102),
  y = 0.06,
  y_type = "effective",
  tidy = TRUE
)

# Target yield given directly per coupon period
bond_callable_price(
```

```
  face = 1000,
  c = 0.05,
  n = 12,
  k = 2,
  call_t = c(4, 8),
  call_R = 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, using compact actuarial notation.

Usage

```
bond_cash_flows(face, c, n, k = 1L, R = NULL, tol = 1e-10, check = TRUE)
```

Arguments

face	Numeric scalar. Face value of the bond.
c	Numeric scalar. Annual coupon rate.
n	Numeric scalar. Years to maturity.
k	Positive integer. Number of coupon payments per year.
R	Numeric scalar. Redemption value. If NULL, it is set equal to face.
tol	Finite positive numeric scalar. Absolute tolerance for integer alignment of k and n * k; it is not a monetary tolerance.
check	Logical scalar. Input validation.

Details

This function follows the compact bond notation used in tidyactuarial: face is the face value, c is the annual coupon rate, n is the term to maturity, k is the number of coupon payments per year, and R is the redemption value.

Stub periods are not supported; therefore, n * k must be an integer. The coupon is $face \cdot c/k$. Coupons and redemption remain separate rows at maturity. At n = 0, only immediate redemption is returned. The schedule requires finite coupons and an integer number of periods below .Machine$integer.max, with sufficient memory for the resulting rows.

Value

A tibble with the bond cash-flow schedule. The main actuarial columns are t for payment time and cf for cash flow.

Examples

```
bond_cash_flows(
  face = 1000,
  c = 0.05,
  n = 10,
  k = 2,
  R = 1000
)
```

---

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, using compact actuarial notation.

Usage

```
bond_convexity(
  face,
  c,
  n,
  k = 1L,
  y_effective_per_period = NULL,
  y = NULL,
  y_type = "effective",
  y_m = 1L,
  R = NULL,
  tol = 1e-10,
  check = TRUE
)
```

Arguments

face	Numeric scalar. Face value of the bond.
c	Numeric scalar. Annual coupon rate as a proportion.
n	Numeric scalar. Time to maturity in years.
k	Positive integer. Number of coupon payments per year.
y_effective_per_period	Optional numeric scalar. Effective yield per coupon period.
y	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. Conversion frequency for nominal annual yields.
R	Numeric scalar. Redemption value at maturity. If NULL, defaults to face.
tol	Finite positive numeric scalar. Tolerance used to check maturity alignment with integer coupon periods, not a monetary price threshold.
check	Logical scalar. If TRUE, performs input validation.

Details

Assumptions:

- Coupons are paid in arrears at regular intervals.
- n * k must be an integer.
- Stub periods are not supported.
- Valuation is at a coupon date with 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, y_type, and y_m define an annual yield specification. Calculations retain its original log growth; reported effective equivalents remain subject to numeric range.

This function follows the compact bond notation used in tidyactuarial: face is the face value, c is the annual coupon rate, n is the time to maturity, k is the coupon frequency, y is the annual yield input, and R is the redemption value.

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

$$C_p = \frac{1}{P} \frac{\sum_{r=1}^{N} C_r r(r+1) v^r}{(1+j)^2}.$$

Discrete convexity in years is C_p/k^2. This is sensitivity to the nominal annual yield $z = kj$; it is generally not the relative second derivative with respect to the annual effective yield $i = (1+j)^k - 1$. Before passing convexity to immunize_duration_convexity(), apply the chain rule described there or compute convexity_cash_flow() from annual cash-flow times.

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. Prices use original log yields and moments use relative present values, avoiding monetary numerator overflow. Nonzero-term bonds require a positive finite price and finite risk measures; changing currency alone does not create an undefined sensitivity. Reported equivalent yields may reach -1 or Inf by rounding while the original specification remains usable. At zero term, the existing convention returns immediate redemption and zero risk.

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.

k 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: accrued_interest(), bond_book_value(), bond_callable_price(), bond_duration(), bond_price(), bond_ytm(), portfolio_convexity(), portfolio_duration()

Examples

```
bond_convexity(
  face = 100,
  c = 0.08,
  n = 5,
  k = 2,
  y = 0.06,
  y_type = "effective"
)

bond_convexity(
  face = 1000,
  c = 0.05,
  n = 10,
  k = 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, using compact actuarial notation.

Usage

```
bond_duration(
  face,
  c,
  n,
  k = 1L,
  y_effective_per_period = NULL,
  y = NULL,
  y_type = "effective",
  y_m = 1L,
  R = NULL,
  tol = 1e-10,
  check = TRUE
)
```

Arguments

face	Numeric scalar. Face value of the bond.
c	Numeric scalar. Annual coupon rate as a proportion.
n	Numeric scalar. Time to maturity in years.
k	Positive integer. Number of coupon payments per year.
y_effective_per_period	Optional numeric scalar. Effective yield per coupon period.
y	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. Conversion frequency for nominal annual yields.
R	Numeric scalar. Redemption value at maturity. If NULL, defaults to face.
tol	Finite positive numeric scalar. Tolerance used to check maturity alignment with integer coupon periods, not a monetary price threshold.
check	Logical scalar. If TRUE, performs input validation.

Details

Assumptions:

- Coupons are paid in arrears at regular intervals.
- n * k must be an integer.
- Stub periods are not supported.
- Valuation is at a coupon date with 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`, `y_type`, and `y_m` define an annual yield specification. Calculations retain its original log growth; reported effective equivalents remain subject to numeric range.

This function follows the compact bond notation used in `tidyactuarial`: `face` is the face value, `c` is the annual coupon rate, `n` is the time to maturity, `k` is the coupon frequency, `y` is the annual yield input, and `R` is the redemption value.

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

Macaulay duration in coupon periods:

$$D_p = \frac{\sum_{r=1}^{N} r C_r v^r}{P}.$$

Macaulay duration in years is $D = D_p/k$.

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

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

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. The two coefficients must use the same yield coordinate: pair `modified_duration_periods_j` with the period convexity for changes in j. Annual effective duration requires annual effective convexity, obtained using the chain rule or `convexity_cash_flow()` on year times. Prices use original log yields and moments use relative present values. Nonzero-term bonds require a positive finite price and finite risk measures; calendar tolerance does not define a monetary zero. Reported equivalent yields may reach -1 or Inf by rounding while the specification remains usable. At zero term, immediate redemption and zero risk follow the existing convention.

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.

k 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: `accrued_interest()`, `bond_book_value()`, `bond_callable_price()`, `bond_convexity()`, `bond_price()`, `bond_ytm()`, `portfolio_convexity()`, `portfolio_duration()`

Examples

```
bond_duration(
  face = 100,
  c = 0.08,
  n = 5,
  k = 2,
  y = 0.06,
  y_type = "effective"
)

bond_duration(
  face = 1000,
  c = 0.05,
  n = 10,
  k = 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, using compact actuarial notation.

Usage

```
bond_price(
  face,
  c,
  n,
  k = 1L,
  y_effective_per_period = NULL,
  y = NULL,
  y_type = "effective",
  y_m = 1L,
  R = NULL,
  tol = 1e-10,
  check = TRUE
)
```

Arguments

face	Numeric scalar. Face value of the bond.
c	Numeric scalar. Annual coupon rate as a proportion.

n	Numeric scalar. Time to maturity in years.
k	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	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. Conversion frequency for nominal annual yields.
R	Numeric scalar. Redemption value at maturity. If NULL, defaults to face.
tol	Finite positive 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.
- n * k 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, y_type, and y_m.

Annual yield specifications are valued using their original log accumulation rate, avoiding loss of precision or range in intermediate annual equivalents.

This function follows the compact bond notation used in tidyactuarial: face is the face value, c is the annual coupon rate, n is the time to maturity, k is the coupon frequency, y is the annual yield input, and R is the redemption value.

Let j be the effective yield per coupon period, k the number of coupon payments per year, and let $N = nk$ be the total number of coupon periods. With coupon per period $C = face \cdot c/k$ and discount factor $v = 1/(1 + j)$, the price is:

$$P = \sum_{r=1}^{N} C_r v^r,$$

where the sum runs over all coupon and redemption cash flows indexed by coupon period r.

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: accrued_interest(), bond_book_value(), bond_callable_price(), bond_convexity(), bond_duration(), bond_ytm(), portfolio_convexity(), portfolio_duration()

Examples

```
# 5-year annual coupon bond, yield given as annual effective
bond_price(
  face = 100,
  c = 0.08,
  n = 5,
  k = 1,
  y = 0.06,
  y_type = "effective"
)

# 10-year semiannual bond, yield given directly per coupon period
bond_price(
  face = 1000,
  c = 0.05,
  n = 10,
  k = 2,
  y_effective_per_period = 0.03
)

# Semiannual coupons, nominal annual yield convertible quarterly
bond_price(
  face = 100,
  c = 0.08,
  n = 5,
  k = 2,
  y = 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, using compact actuarial notation.

Usage

```
bond_ytm(
  P,
  face,
  c,
  n,
  k = 1L,
  R = NULL,
  interval = NULL,
  tol = 1e-12,
  maxiter = 1000,
  check = TRUE
)
```

Arguments

P	Numeric scalar. Observed dirty price of the bond at time 0.
face	Numeric scalar. Face value of the bond.
c	Numeric scalar. Annual coupon rate as a proportion.
n	Numeric scalar. Time to maturity in years. Must be strictly positive.
k	Positive integer. Number of coupon payments per year.
R	Numeric scalar. Redemption value at maturity. If NULL, defaults to face.
interval	Optional finite numeric vector of length 2 giving an increasing bracket for the effective yield per coupon period, with both bounds above -1.
tol	Finite positive numeric scalar. Yield-coordinate tolerance passed to uniroot, not a monetary price tolerance.
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).
- n * k must be an integer.
- Stub periods are not supported.

This function follows the compact bond notation used in tidyactuarial: P is the observed dirty price, face is the face value, c is the annual coupon rate, n is the time to maturity, k is the coupon frequency, and R is the redemption value.

The effective yield per coupon period j is the solution to

$$P = \sum_{r=1}^{N} C_r (1+j)^{-r}.$$

With nonnegative payments and at least one positive future payment, price is strictly decreasing in $j > -1$, so the yield is unique. The root is found with `uniroot`, using the log of the discounted cash-flow sum relative to price to avoid monetary overflow or underflow. If no `interval` is supplied, the lower bound is the nearest double precision number above -1 and the upper bound starts at 0.10 and is widened until the root is bracketed. A yield outside the representable domain or an exhausted iteration limit causes an error. Tolerance applies to the yield coordinate according to `uniroot`; it does not guarantee a given price residual, particularly for yields near -1.

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

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

Value

A one-row tibble with columns:

price Input dirty price.

i_period Effective yield per coupon period.

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

i_effective_annual Annual effective yield.

k Coupon frequency.

See Also

`bond_price`, `bond_cash_flows`, `bond_duration`, `bond_convexity`, `bond_callable_price`

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

Examples

```
bond_ytm(
  P = 100,
  face = 100,
  c = 0.06,
  n = 5,
  k = 1
)

bond_ytm(
  P = 950,
  face = 1000,
  c = 0.05,
  n = 10,
  k = 2
```

```
)
```

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

Description

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

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

Usage

```
bonds_sample

bonds_sample
```

Format

A tibble with 5 rows and 8 variables:

bond_id Bond identifier.

face Face value of the bond.

c Annual coupon rate.

k Number of coupon payments per year.

n Maturity in years.

y Annual nominal yield rate consistent with coupon frequency.

bond_type Short bond type label.

P Theoretical bond price computed from the listed yield rate.

A tibble with 5 rows and 8 variables:

bond_id Bond identifier.

face Face value of the bond.

c Annual coupon rate.

k Number of coupon payments per year.

n Maturity in years.

y Annual nominal yield rate consistent with coupon frequency.

bond_type Short bond type label.

P Theoretical bond price computed from the listed yield rate.

Details

This dataset uses the compact bond notation adopted in `tidyactuarial`: `face` is the face value, `c` is the annual coupon rate, `k` is the coupon frequency, `n` is the maturity, `y` is the yield input, and `P` is the bond price.

This dataset uses the compact bond notation adopted in `tidyactuarial`: `face` is the face value, `c` is the annual coupon rate, `k` is the coupon frequency, `n` is the maturity, `y` is the yield input, and `P` is the bond price.

Source

Synthetic pedagogical data created for tidyactuarial examples.

Synthetic pedagogical data created for tidyactuarial examples.

Examples

```
data(bonds_sample)

bonds_sample |>
  dplyr::select(bond_id, face, c, k, n, y, P)
```

```
data(bonds_sample)

bonds_sample |>
  dplyr::select(bond_id, face, c, k, n, y, P)
```

---

bootstrap_spot_curve *Bootstrap a spot yield curve from instrument prices*

Description

Recovers annual effective spot rates and discount factors sequentially from observed dirty prices and dated cash flows. Each instrument supplies one new maturity; all its earlier positive cash flows must occur at maturities already bootstrapped.

Usage

```
bootstrap_spot_curve(
  .data,
  col_instrument = "instrument",
  col_price = "price",
  col_t = "t",
  col_cf = "cf"
)
```

Arguments

`.data`	Nonempty data frame with one row per priced instrument.
`col_instrument`	Character scalar naming the unique instrument-ID column.
`col_price`	Character scalar naming the observed dirty-price column.
`col_t`	Character scalar naming the list-column of positive payment times in years.
`col_cf`	Character scalar naming the list-column of nonnegative cash flows corresponding to `col_t`.

Details

For instrument j, with maturity t_j, dirty price P_j, and cash flows $CF_{j,k}$, the price identity is

$$P_j = \sum_{k<j} CF_{j,k} v(t_k) + CF_{j,j} v(t_j).$$

Hence the new discount factor is recovered exactly as

$$v(t_j) = \frac{P_j - \sum_{k<j} CF_{j,k} v(t_k)}{CF_{j,j}}.$$

The annual effective spot rate is

$$i_{t_j} = v(t_j)^{-1/t_j} - 1.$$

Each input row represents one instrument; `t` and `cf` are list-columns. Repeated payment times within an instrument are combined, allowing coupon and redemption flows produced by `bond_cash_flows()`. Instruments are processed by increasing final positive-payment time, not input row order. Zero cash flows are ignored. Duplicate final maturities, missing earlier discount-factor nodes, zero or negative residual prices, and nonpositive terminal cash flows are rejected.

This is exact triangular bootstrapping: it does **not** interpolate, extrapolate, fit a curve, or estimate a least-squares solution. An annual effective spot rate can be negative; its discount factor must be positive. Input prices and cash flows must use the same monetary units. Both the positive finite discount factor and its annual effective rate above -1 must be representable. Rounded input prices can make a small terminal residual ill-conditioned; no tolerance silently replaces a nonpositive residual. `repricing_error` reports the recomputed floating-point discrepancy, not an independent accuracy guarantee or a convergence test.

Value

A tibble ordered by maturity, with columns `instrument`, `t`, `price`, `known_pv`, `terminal_cf`, `v`, `i`, `repriced_price`, and `repricing_error`. The columns `v` and `i` contain the new maturity's discount factor and annual effective spot rate. `repricing_error` is the model price minus the observed dirty price, in monetary units.

See Also

`yield_curve()`, `discount_factor_spot()`, `forward_rate()`, `bond_cash_flows()`

Other interest: `accumulation_factor()`, `discount_factor()`, `discount_factor_spot()`, `forward_curve()`, `forward_rate()`, `interest_equivalents()`, `par_rate()`, `standardize_interest()`, `yield_curve()`

Examples

```
# Recover a known 1-, 2-, and 3-year annual effective spot curve.
t <- 1:3
i <- c(0.04, 0.045, 0.05)
v <- (1 + i)^(-t)
instruments <- tibble::tibble(
  instrument = c("Zero 1Y", "Coupon 2Y", "Coupon 3Y"),
  price = c(100 * v[1],
            5 * v[1] + 105 * v[2],
            6 * v[1] + 6 * v[2] + 106 * v[3]),
  t = list(1, c(1, 2), c(1, 2, 3)),
  cf = list(100, c(5, 105), c(6, 6, 106))
)
curve <- instruments |> bootstrap_spot_curve()
curve
yield_curve(t = curve$t, i = curve$i)
```

---

cash_flows_sample *Sample cash flows for interest theory examples*

Description

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

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

Usage

```
cash_flows_sample

cash_flows_sample
```

Format

A tibble with 19 rows and 5 variables:

scenario_id Scenario identifier.

t Payment time.

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

cashflow_type Type of cash flow.

description Short description of the cash flow.

A tibble with 19 rows and 5 variables:

scenario_id Scenario identifier.

t Payment time.

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

cashflow_type Type of cash flow.

description Short description of the cash flow.

Details

This dataset uses the compact financial-actuarial notation used throughout `tidyactuarial`: `t` denotes time and `C` denotes the cash-flow amount at that time.

This dataset uses the compact financial-actuarial notation used throughout `tidyactuarial`: `t` denotes time and `C` denotes the cash-flow amount at that time.

Source

Synthetic pedagogical data created for tidyactuarial examples.

Synthetic pedagogical data created for tidyactuarial examples.

Examples

```
data(cash_flows_sample)

cash_flows_sample |>
  dplyr::filter(scenario_id == "investment_project") |>
  dplyr::select(scenario_id, t, C, cashflow_type)

data(cash_flows_sample)

cash_flows_sample |>
  dplyr::filter(scenario_id == "investment_project") |>
  dplyr::select(scenario_id, t, C, cashflow_type)
```

`collective_risk_spec` *Specify a collective risk model*

Description

Creates a lightweight specification of the classical collective risk model

$$S = \sum_{i=1}^{N} X_i,$$

where N is a claim count random variable and $X_1, X_2, \ldots$ are identically distributed individual claim severities. Independence between the frequency variable and the severities, and mutual independence of the severities, are intrinsic assumptions of this specification.

The constructor records the probabilistic model only. It does not calculate aggregate moments, construct the distribution of S, discretize a severity distribution, simulate losses, or apply an approximation.

Usage

```
collective_risk_spec(frequency, severity, count_basis = NULL, label = NULL)
```

Arguments

frequency	A tidyact_frequency_spec object, typically created with frequency_spec() or empirical_frequency().
severity	A tidyact_severity_spec object, typically created with severity_spec() or empirical_severity().
count_basis	Either NULL, "loss", or "payment". The count basis indicates whether frequency counts underlying losses or positive payments. When NULL, the constructor uses basis information already carried by the component specifications when available; otherwise the default is "loss".
label	Optional nonempty character scalar identifying the model.

Details

The collective model is represented by the random sum

$$S = X_1 + \cdots + X_N,$$

with the convention $S = 0$ when $N = 0$.

collective_risk_spec() deliberately keeps the model specification separate from computational methods. In particular, no Panjer recursion, convolution, FFT calculation, Monte Carlo simulation, numerical discretization, or normal approximation is performed by this function.

The count_basis field is part of the model because frequency and severity must refer to the same event basis. A loss-basis specification counts underlying losses, including losses that may later produce a zero payment after coverage is applied. A payment-basis specification counts positive payments. Conversion between these bases is a separate actuarial operation and is not performed silently by this constructor.

The constructor does not expose an independence argument. Independence is part of the definition of this version of the collective risk model rather than an optional computational setting.

Value

An object with classes "tidyact_collective_risk_spec" and "tidyact_risk_spec". It contains the following fields:

- schema_version: internal schema version.
- frequency: the frequency specification.
- severity: the severity specification.
- count_basis: either "loss" or "payment".
- label: optional user label.
- metadata: reserved metadata list.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*, 2nd ed. Society of Actuaries.

See Also

`frequency_spec()`, `severity_spec()`, `empirical_frequency()`, `empirical_severity()`, `distribution_moments()`

Examples

```
frequency <- frequency_spec(
  "poisson",
  lambda = 4
)

severity <- severity_spec(
  "gamma",
  shape = 2,
  rate = 0.001
)

model <- collective_risk_spec(
  frequency,
  severity
)

model

# An explicit payment-basis model.
collective_risk_spec(
  frequency,
  severity,
  count_basis = "payment",
  label = "Illustrative payment model"
)

# Empirical components can be used without fitting a parametric model.
empirical_n <- empirical_frequency(c(0, 0, 1, 1, 2, 3))
empirical_x <- empirical_severity(c(500, 750, 1000, 1500, 3000))

collective_risk_spec(
  empirical_n,
  empirical_x
)
```

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

Description

The function builds annual commutation columns from `x`, `lx`, and an interest-rate specification. The interest rate is converted internally to an original log accumulation rate before forming discounted products.

Usage

```
commutation_table(lt, i, i_type = "effective", m = 1L, check = TRUE)
```

Arguments

`lt`	A life table object as produced by `lifetable`. It must contain columns `x` and `lx`. If available, `qx` or `px` may also be present, but `dx` is computed robustly from successive values of `lx`.
`i`	Numeric scalar. Annual interest-rate input.
`i_type`	Character string indicating the interest-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Conversion frequency for nominal rates. Ignored for `i_type = "effective"` and `i_type = "force"`.
`check`	Logical. If `TRUE`, performs basic input checks.

Details

Constructs classical annual commutation functions D_x, N_x, S_x, C_x, M_x, and R_x from a life table defined at integer ages, using compact actuarial notation.

This function follows the compact actuarial notation used throughout `tidyactuarial`: `lt` is the life table, `i` is the interest-rate input, `i_type` is the interest-rate type, and `m` is the conversion frequency for nominal rates.

The rate specification is validated through `standardize_interest`. If i_e denotes the annual effective rate, then

$$v = \frac{1}{1 + i_e}.$$

The annual deaths are computed as

$$d_x = l_x - l_{x+1},$$

closing the table with $l_{\omega+1} = 0$. The main commutation functions are then computed as

$$D_x = v^x l_x, \qquad C_x = v^{x+1} d_x,$$

Discounted products retain the original log accumulation rate to avoid intermediate factor overflow or underflow. This constructor always closes the terminal count, including when the input carries open-table metadata. The horizon is therefore different from the truncated integral in `e_x`. Reverse cumulative sums are used to obtain N_x, S_x, M_x, and R_x.

Value

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

Examples

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

commutation_table(
  lt = lt,
  i = 0.05
)

commutation_table(
  lt = lt,
  i = 0.06,
  i_type = "nominal_interest",
  m = 12
)
```

---

convexity_cash_flow *Convexity of a general cash-flow stream*

Description

Computes the yield convexity of a general cash-flow stream at a specified valuation time.

Usage

```
convexity_cash_flow(
  cf,
  t,
  rate,
  valuation_time = 0,
  output = c("value", "audit")
)
```

Arguments

cf	Numeric vector of cash-flow amounts.
t	Numeric vector of cash-flow times. Times must be expressed in the same units used by ‘rate‘.
rate	Effective yield per unit of time. It must be a finite numeric scalar greater than ‘-1‘.
valuation_time	Finite numeric scalar indicating the valuation time. Defaults to ‘0‘.
output	Character scalar indicating the returned object: ‘"value"‘ returns convexity as a numeric scalar; ‘"audit"‘ returns a detailed tibble.

Details

Let τ be the valuation time. Under an effective yield i, the value of the remaining cash-flow stream is

$$P_\tau(i) = \sum_{j:t_j>\tau} C_j(1+i)^{-(t_j-\tau)}.$$

The convexity returned by this function is the relative second derivative of value with respect to the effective yield:

$$\mathcal{C}_\tau = \frac{1}{P_\tau(i)} \frac{d^2 P_\tau(i)}{di^2}.$$

Equivalently,

$$\mathcal{C}_\tau = \frac{\sum_{j:t_j>\tau}(t_j-\tau)(t_j-\tau+1)C_j(1+i)^{-(t_j-\tau+2)}}{P_\tau(i)}.$$

Together with modified duration, this convexity gives the second-order approximation

$$\frac{\Delta P_\tau}{P_\tau} \approx -D_{\mathrm{mod},\tau}\Delta i + \frac{1}{2}\mathcal{C}_\tau(\Delta i)^2.$$

The function follows an ex-cash-flow convention: cash flows occurring exactly at ‘valuation_time‘ are excluded. Therefore, the result represents convexity immediately after any cash flow paid at that time.

For streams containing both positive and negative cash flows, convexity remains a yield-sensitivity measure, but it may be negative and may not have the usual interpretation associated with conventional bonds. Relative contributions retain log weights rather than first forming a monetary convexity numerator. Representable products can be recovered when a standalone discount factor underflows; its audit column can then be zero while present value is positive. Discount factors, present values, their total/absolute sum, and the contributions/result must remain finite. Signed streams with absolute total value at most ‘sqrt(.Machine$double.eps)‘ times the sum of absolute present values are rejected as numerically cancelled.

Value

If ‘output = "value"‘, a numeric scalar.

If ‘output = "audit"‘, a tibble with six columns: ‘time‘, ‘remaining_time‘, ‘cash_flow‘, ‘discount_factor‘, ‘present_value‘, and ‘convexity_contribution‘.

See Also

[duration_cash_flow()]

Examples

```
convexity_cash_flow(
  cf = c(100, 100, 1100),
  t = c(1, 2, 3),
  rate = 0.05
)

convexity_cash_flow(
  cf = c(100, 100, 1100),
  t = c(1, 2, 3),
  rate = 0.05,
  valuation_time = 1
)

convexity_cash_flow(
  cf = c(100, 100, 1100),
  t = c(1, 2, 3),
  rate = 0.05,
  output = "audit"
)
```

---

coverage_spec *Specify an insurance coverage contract*

Description

Creates a lightweight S3 specification for per-loss insurance coverage, including an ordinary or franchise deductible, a coverage limit, coinsurance, inflation, and optional indexation of contract terms.

Usage

```
coverage_spec(
  deductible = 0,
  limit = Inf,
  coinsurance = 1,
  inflation_factor = 1,
  deductible_type = c("ordinary", "franchise"),
  index_deductible = FALSE,
  index_limit = FALSE,
  label = NULL
)
```

Arguments

deductible	Nonnegative finite numeric scalar. The deductible applied to the inflated loss, possibly adjusted by index_deductible.

`limit` Nonnegative numeric scalar, or `Inf`. Maximum covered amount **after the deductible but before coinsurance**. This is not the maximum ground-up loss and not necessarily the insurer's final payment limit.

`coinsurance` Numeric scalar in `[0, 1]`. Fraction of the covered amount paid by the insurer, applied after the limit. Zero is allowed to represent a contract with no insurer participation.

`inflation_factor` Positive finite numeric scalar multiplying the original loss. This is a **factor**, not a rate: use `1.10` for 10 percent inflation.

`deductible_type` Either `"ordinary"` (default) or `"franchise"`. Under a franchise deductible, a loss exceeding the deductible is payable from the first monetary unit, subject to the limit and coinsurance.

`index_deductible` Logical scalar. If `TRUE`, multiply the deductible by `inflation_factor` before applying it to the inflated loss.

`index_limit` Logical scalar. If `TRUE`, multiply the limit by `inflation_factor` before applying it. An infinite limit remains infinite.

`label` Optional nonempty, non-whitespace character scalar identifying the contract.

Details

This function only records contractual terms. It does not accept a severity distribution, evaluate payments, simulate losses, or change a frequency model.

Let `X >= 0` denote the original loss, `f` the inflation factor, `d` the deductible, `L` the limit, and `c` the coinsurance fraction. Write `Z = f * X`, `d_f = d * f` if the deductible is indexed (otherwise `d`), and `L_f = L * f` if the limit is indexed (otherwise `L`). The specified processing order is inflation, deductible, limit, and coinsurance.

For an ordinary deductible, the intended payment per loss is

$$Y_L = c \min\{\max(Z - d_f, 0), L_f\}.$$

For a franchise deductible, the intended payment per loss is

$$Y_L = c \min\{Z, L_f\} \mathbf{1}_{\{Z > d_f\}}.$$

Thus a franchise loss exactly equal to its deductible produces no payment. With a finite limit, the maximum insurer payment is `c * L_f` (or a smaller achievable maximum if the loss distribution is bounded).

In standard actuarial notation, the *maximum covered loss* for an ordinary deductible with finite `d_f` and `L_f` is `u = d_f + L_f`; it is distinct from this function's `limit` argument.

All monetary amounts must use the same currency and reference monetary basis. Inputs are deliberately scalar: one object represents one contract.

Value

An object of class `tidyact_coverage_spec`, implemented as a named list containing `deductible`, `limit`, `coinsurance`, `inflation_factor`, `deductible_type`, `index_deductible`, `index_limit`, and `label`. No derived payments or distributional quantities are stored.

References

Klugman, S. A., Panjer, H. H., and Willmot, G. E. (2012). *Loss Models: From Data to Decisions*, 4th edition, Chapter 8.

Examples

```
# Default: full insurance of each loss, without inflation.
coverage_spec()

# Ordinary deductible with a capped covered layer and 80% coinsurance.
contract <- coverage_spec(
  deductible = 500,
  limit = 10000,
  coinsurance = 0.80,
  label = "Standard contract"
)
contract$limit

# A qualifying loss is covered from its first unit under a franchise.
coverage_spec(deductible = 1000, deductible_type = "franchise")

# Inflation with indexed deductible, but a fixed nominal limit.
coverage_spec(
  deductible = 500,
  limit = 10000,
  inflation_factor = 1.12,
  index_deductible = TRUE,
  index_limit = FALSE
)

# Completely indexed contract, including an unbounded limit.
coverage_spec(
  deductible = 500,
  limit = Inf,
  inflation_factor = 1.05,
  index_deductible = TRUE,
  index_limit = TRUE
)
```

---

discount_factor *Discount factor under conventional or time-varying interest*

Description

Computes the discount factor from time `t` back to time `s` using exactly one of three interest specifications:

- a conventional interest rate `i`;
- an accumulation function `a(t)`;
- a time-varying force of interest `delta(t)`.

Usage

```
discount_factor(
  s = 0,
  t,
  i = NULL,
  i_type = "effective",
  m = 1,
  a = NULL,
  delta = NULL,
  subdivisions = 100L,
  rel.tol = 1e-08,
  tidy = FALSE
)
```

Arguments

s	Numeric vector of valuation times. Defaults to 0.
t	Numeric vector of payment or accumulation times.
i	Optional numeric vector of conventional interest-rate values.
i_type	Character vector indicating the conventional interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force". Used only when i is supplied.
m	Positive integer vector giving the conversion frequency for nominal rates. Used only when i is supplied.
a	Optional accumulation function of one numeric time argument.
delta	Optional force-of-interest function of one numeric time argument.
subdivisions	Positive integer giving the maximum number of subintervals used by stats::integrate() when delta is supplied.
rel.tol	Positive relative tolerance passed to stats::integrate() when delta is supplied.
tidy	Logical scalar. If FALSE, returns a numeric vector. If TRUE, returns a tibble with the inputs, accumulation factor, and discount factor.

Details

The discount factor is the reciprocal of the corresponding accumulation factor:

$$v(s,t) = \frac{1}{A(s,t)}.$$

Therefore,

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

for a constant annual effective rate,

$$v(s,t) = \frac{a(s)}{a(t)}$$

for an accumulation function, and

$$v(s,t) = \exp\left(-\int_s^t \delta(u)\,du\right)$$

for a time-varying force of interest.

Exactly one of `i`, `a`, or `delta` must be supplied. The requested discount factor is evaluated from the log accumulation, without requiring its reciprocal to be representable. It must be positive and finite. In tidy output, an unrepresentable intermediate accumulation factor may be Inf or zero; the discount factor is computed independently.

Numeric arguments may have length 1 or a common length. Scalars are recycled over the remaining scenarios, making the function suitable for use inside `dplyr::mutate()` pipelines.

This function handles general discounting over an interval $[s,t]$. For discount factors obtained specifically from a spot-rate curve, see `discount_factor_spot`.

Value

If `tidy = FALSE`, a numeric vector of discount factors.

If `tidy = TRUE`, a tibble containing the interval, interest model, relevant intermediate quantities, accumulation factor, and discount factor.

See Also

`accumulation_factor`, `discount_factor_spot`, `present_value`, `pv_flow`

Other interest: `accumulation_factor()`, `bootstrap_spot_curve()`, `discount_factor_spot()`, `forward_curve()`, `forward_rate()`, `interest_equivalents()`, `par_rate()`, `standardize_interest()`, `yield_curve()`

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

Examples

```
# Constant annual effective interest
discount_factor(
  t = 5,
  i = 0.07
)

# One rate recycled over several times
discount_factor(
  t = c(1, 2, 5),
  i = 0.07
)

# Accumulation function
a_fun <- function(t) {
  exp(0.03 * t + 0.002 * t^2)
}
```

```
discount_factor(
  s = 2,
  t = 5,
  a = a_fun
)

# Time-varying force of interest
delta_fun <- function(t) {
  0.03 + 0.004 * t
}

discount_factor(
  s = 2,
  t = 5,
  delta = delta_fun
)

# Pipe-friendly use
if (requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  scenarios <- tibble::tibble(
    i = 0.07,
    t = c(1, 2, 3, 5)
  )

  scenarios |>
    dplyr::mutate(
      discount = discount_factor(
        t = t,
        i = i
      )
    )
}
```

`discount_factor_spot` *Spot discount factor*

Description

Computes the discount factor implied by a spot rate for a given time, using compact actuarial notation.

Usage

```
discount_factor_spot(t, i, i_type = "effective", m = 1L, tidy = FALSE)
```

Arguments

t	Numeric vector of times in years. Each value must be greater than or equal to 0.
i	Numeric vector of spot-rate values.
i_type	Character vector indicating the spot-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer vector giving the conversion frequency for nominal spot-rate inputs. Observed values are validated as positive integers for every family, even when the conversion itself does not use m.
tidy	Logical scalar. If FALSE, returns a numeric discount factor. If TRUE, returns a tibble with intermediate calculations.

Details

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

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

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

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

This function follows the compact actuarial notation used throughout tidyactuarial: t denotes time, i denotes the spot-rate input, i_type denotes the interest-rate type, and m denotes the conversion frequency for nominal rates.

If $t = 0$, the discount factor is 1. Valuation retains the original log accumulation rate. In tidy output, an extreme annual equivalent can round to -1 or overflow to Inf without invalidating a representable discount factor.

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

Value

If tidy = FALSE, a numeric vector of discount factors.

If tidy = TRUE, a tibble with input values, standardized rates, and discount factors.

See Also

standardize_interest, present_value, pv_flow

Other interest: accumulation_factor(), bootstrap_spot_curve(), discount_factor(), forward_curve(), forward_rate(), interest_equivalents(), par_rate(), standardize_interest(), yield_curve()

Examples

```
# Numeric discount factor
discount_factor_spot(
  t = 3,
  i = 0.05
)

# Vectorized example
discount_factor_spot(
  t = c(1, 2, 3),
  i = c(0.05, 0.055, 0.06)
)

# FM-style input with nominal annual interest
discount_factor_spot(
  t = 2,
  i = 0.08,
  i_type = "nominal_interest",
  m = 2
)

# Tibble output for teaching or auditing
discount_factor_spot(
  t = c(1, 2, 3),
  i = c(0.05, 0.055, 0.06),
  tidy = TRUE
)
```

---

discretize_severity *Discretize a continuous claim severity distribution*

Description

Converts a continuous nonnegative severity specification into an explicit lattice distribution. The transformation is intended for actuarial algorithms that require discrete severities, such as Panjer recursion or FFT-based aggregate-loss methods.

Discretization is always explicit: this function is never called silently by aggregate-loss functions.

Usage

```
discretize_severity(
  x,
  step,
  to = NULL,
  method = c("unbiased", "upper", "lower", "rounding"),
  tail_prob = 1e-08,
  label = NULL
)
```

Arguments

x	A continuous tidyact_severity_spec object.
step	Positive finite numeric scalar giving the lattice spacing.
to	Optional positive finite upper grid point. It must be an integer multiple of step. If NULL, the upper point is selected from the (1 - tail_prob) quantile and rounded upward to the next grid point.
method	Discretization method. One of "unbiased", "upper", "lower", or "rounding".
tail_prob	Probability used to select to when to = NULL. Must satisfy 0 < tail_prob < 1.
label	Optional nonempty character scalar. If NULL, the label of x is retained.

Details

Let F be the severity CDF, h the lattice step, and $b = \mathrm{to}$. The lattice is

$$0, h, 2h, \ldots, b.$$

The four methods follow standard actuarial discretization conventions.

Upper method:
For lattice points below to,

$$p_x = F(x + h) - F(x).$$

This assigns each interval to its lower lattice endpoint. Probability above to is assigned explicitly to the upper grid point.

Lower method:

$$p_0 = F(0), \qquad p_x = F(x) - F(x - h).$$

This assigns each interval to its upper lattice endpoint. Probability above to is added to the upper grid point.

Rounding method:
Probability is allocated according to the midpoint intervals around each lattice point. Thus the first cell extends from zero to step / 2, interior cells have width step, and the last cell contains everything above to - step / 2.

Unbiased method:
The unbiased method performs local first-moment matching. If $L(d) = E[\min(X, d)]$, the finite-grid probabilities before the final tail adjustment are

$$p_0 = 1 - F(0) - \frac{L(h) - L(0)}{h},$$

$$p_x = \frac{2L(x) - L(x - h) - L(x + h)}{h}, \qquad 0 < x < b,$$

and

$$p_b = \frac{L(b) - L(b-h)}{h} - 1 + F(b).$$

The residual tail probability above `to` is then assigned to `to`. Consequently, the resulting distribution is a proper probability distribution and its first moment matches $E[\min(X, b)]$, rather than silently discarding probability above the truncation point.

For the exponential, gamma, lognormal, Weibull, and Pareto II families, the first limited moment used by the unbiased method is evaluated analytically. For other continuous specifications with a CDF, including Burr XII, log-logistic, and eligible custom models, the limited mean is obtained by numerical integration of the survival function. The source used is recorded in `metadata$discretization$limited_mean_source`.

Only continuous severity specifications are accepted in this initial implementation. Finite discrete and empirical severities already have probability masses and should not be silently re-discretized. Support for mixed severities can be added later without changing this public interface.

The returned object uses the existing `"discrete"` severity operations but has `support$type = "lattice"`. Zero-probability grid points need not be stored explicitly in `params$values`; the lattice information is retained in `support`.

Value

A `tidyact_severity_spec` and `tidyact_distribution_spec` object with `dist = "discrete"`, lattice support, and discretization information in `metadata$discretization`.

The metadata records:

- `method`;
- `step`;
- `to`;
- whether `to` was supplied or selected from `tail_prob`;
- the requested `tail_prob` when applicable;
- the actual probability above `to`;
- the explicit tail treatment;
- the original distribution identifier; and
- the source of the limited mean for the unbiased method.

References

Klugman, S. A., Panjer, H. H., and Willmot, G. E. (2019). *Loss Models: From Data to Decisions*, 5th ed. Wiley.

See Also

`severity_spec()`, `distribution_convolution()`

Examples

```
x <- severity_spec(
  "gamma",
  shape = 2,
  rate = 0.001
)

# Automatic upper grid point from a small tail probability.
dx <- discretize_severity(
  x,
  step = 100
)
dx

# Inspect the resulting lattice probabilities.
tibble::tibble(
  amount = dx$params$values,
  probability = dx$params$probs
)

# Use a fixed truncation point and another discretization rule.
discretize_severity(
  x,
  step = 250,
  to = 10000,
  method = "rounding"
)

# The discretization choices remain auditable.
dx$metadata$discretization
```

`distribution_convolution` *Specify a convolution of probability distributions*

Description

Creates a lazy specification of the convolution of two independent distributions or of an iid sum of a single distribution.

If X and Y are independent, the pairwise form represents

$$Z = X + Y.$$

The iid form represents

$$Z_n = X_1 + \cdots + X_n,$$

where $X_1, \ldots, X_n$ are independent copies of the distribution specified by `x`.

The function records the convolution model but does not evaluate a density, mass function, distribution function, or numerical convolution.

Usage

```
distribution_convolution(x, y = NULL, n = NULL, label = NULL)
```

Arguments

`x`	A `tidyact_distribution_spec` object.
`y`	Optional second `tidyact_distribution_spec` object. Supply `y` for a pairwise convolution `x + y`.
`n`	Optional nonnegative integer scalar. Supply `n` for the iid convolution of `n` copies of `x`.
`label`	Optional nonempty character scalar identifying the convolution.

Details

Exactly one of `y` and `n` must be supplied.

The pairwise form

```
distribution_convolution(x, y)
```

represents the distribution of the sum of two independent random variables.

The iid form

```
distribution_convolution(x, n = 5)
```

represents the five-fold convolution of `x`.

The cases `n = 0` and `n = 1` follow the usual convolution conventions:

- `n = 0` represents the degenerate distribution at zero;
- `n = 1` returns the original distribution specification. If `label` is supplied, only the label of the returned copy is changed.

The operation is deliberately *lazy*. It does not choose a computational algorithm, discretize a continuous distribution, truncate an infinite support, or apply FFT or Panjer recursion. Those decisions belong to later computational functions.

Support bounds are propagated algebraically. For example, if `x` has support bounded below by a and `y` by b, the convolution is bounded below by $a + b$. Infinite upper bounds are preserved.

Support type is inferred conservatively:

- count + count remains count;
- iid lattice sums remain lattice;
- a continuous component convolved with a discrete component is continuous;

- any convolution involving a mixed distribution is classified as mixed;
- otherwise discrete supports are retained as discrete.

Independence is part of the meaning of a convolution specification and is therefore not stored as a separate flag.

Value

For a genuine convolution, an object with classes `"tidyact_convolution_spec"` and `"tidyact_distribution_spec"`.

The object follows the common distribution specification schema:

`schema_version` Internal schema version.

`dist` The identifier `"convolution"`.

`params` The lazy convolution definition.

`support` Inferred support information.

`data` Always `NULL`.

`custom` Always `NULL`.

`label` Optional user label.

`metadata` Reserved metadata list.

When `n = 1`, the original distribution specification is returned instead of wrapping it in a convolution object.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*, 2nd ed. Society of Actuaries.

Klugman, S. A., Panjer, H. H., and Willmot, G. E. (2019). *Loss Models: From Data to Decisions*, 5th ed. Wiley.

See Also

`frequency_spec()`, `severity_spec()`, `collective_risk_spec()`

Examples

```
x <- severity_spec(
  "gamma",
  shape = 2,
  rate = 0.001
)

y <- severity_spec(
  "lognormal",
  meanlog = 7,
  sdlog = 0.6
)
```

```
# Pairwise convolution: X + Y
distribution_convolution(x, y)

# Four iid claim amounts: X1 + ... + X4
distribution_convolution(
  x,
  n = 4
)

# The zero-fold convolution is degenerate at zero.
distribution_convolution(
  x,
  n = 0
)

# The one-fold convolution is the original specification.
identical(
  distribution_convolution(x, n = 1),
  x
)

# Count distributions can also be convolved.
n_claims <- frequency_spec(
  "poisson",
  lambda = 3
)

distribution_convolution(
  n_claims,
  n = 2
)
```

`distribution_moments` *Calculate moments of a frequency or severity distribution*

Description

Calculates raw, central, factorial, or cumulant moments from an existing frequency or severity specification. The calculation is deterministic: it does not fit a model, generate observations, or approximate moments using simulation or numerical integration.

Usage

```
distribution_moments(x, order = 1:4, type = "raw")
```

Arguments

`x`	A `tidyact_frequency_spec` or `tidyact_severity_spec` object.

`order`	Nonempty numeric vector of nonnegative integer moment orders. For cumulants, all orders must be positive.
`type`	One of `"raw"` (default), `"central"`, `"factorial"`, or `"cumulant"`. Factorial moments are available only for frequency specifications.

Details

For a random variable Z, the raw and central moments are

$$m_r = E[Z^r], \qquad \mu_r = E[(Z - E[Z])^r].$$

For a nonnegative integer-valued frequency N, the factorial moment is

$$E[(N)_r], \qquad (N)_r = N(N-1)\cdots(N-r+1).$$

Cumulants are calculated algebraically from raw moments. In particular, $\kappa_1 = E[Z]$, $\kappa_2 = \mathrm{Var}(Z)$, and $\kappa_3 = E[(Z - E[Z])^3]$ when the relevant moments are finite.

Moment order zero is one for raw, central, and factorial moments. Cumulants start at order one.

All built-in distributions use their registered analytical raw or factorial moment functions. A custom specification must explicitly supply `raw_moment` or (for a frequency) `factorial_moment` to provide a moment calculation; supplying only a PMF or PDF does not trigger numerical integration or summation. Raw and factorial frequency moments can be converted into one another using Stirling numbers when the necessary source moments are available.

The `status` column distinguishes:

- `"finite"`: a finite numerical result;
- `"infinite"`: a mathematically divergent moment;
- `"undefined"`: the moment is not defined or the specification does not provide sufficient information to evaluate it;
- `"overflow"`: a theoretically finite moment or an intermediate algebraic expression exceeds the available floating-point range.

For built-in Pareto II, Burr XII, and log-logistic severities, divergence is determined from the distribution's analytical moment-existence condition. For custom distributions, an explicitly returned positive `Inf` is interpreted as mathematical divergence; custom moment functions are responsible for this distinction.

A central moment of positive order requires a finite mean. When the mean exists but a higher-order moment diverges, the corresponding central moment is infinite. Higher-order cumulants are reported as undefined if the moments needed for their definition are infinite. The first cumulant follows the first raw moment.

Central moments and cumulants derived from raw moments may suffer from cancellation for extremely large parameter values. Such arithmetic is not silently replaced by a simulated or approximated answer.

Value

A `tibble::tibble()` with one row per requested order and columns: `type` (character), `order` (integer), `value` (double), and `status` (character). Input order and repeated orders are preserved. Undefined values are returned as `NA_real_`.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*, 2nd ed. Society of Actuaries.

See Also

`frequency_spec()`, `severity_spec()`

Examples

```
# Raw and central moments of a claim count distribution.
counts <- frequency_spec("poisson", lambda = 4)
distribution_moments(counts)
distribution_moments(counts, order = 1:4, type = "central")
distribution_moments(counts, order = 0:4, type = "factorial")
distribution_moments(counts, order = 1:4, type = "cumulant")

# Raw moments of a nonnegative claim amount.
loss <- severity_spec("gamma", shape = 2.5, rate = 0.001)
distribution_moments(loss, order = 1:4)

# The mean exists but the variance does not.
heavy_tail <- severity_spec("pareto", shape = 1.5, scale = 2000)
distribution_moments(heavy_tail, order = 1:4)
distribution_moments(heavy_tail, order = 1:4, type = "central")

# A nonrandom payment has zero central moments of positive order.
fixed <- severity_spec("degenerate", value = 1000)
distribution_moments(fixed, order = 1:4, type = "central")
```

---

duration_cash_flow *Duration of a general cash-flow stream*

Description

Computes the Macaulay or modified duration of a general cash-flow stream at a specified valuation time.

Usage

```
duration_cash_flow(
  cf,
  t,
  rate,
  valuation_time = 0,
  duration = c("macaulay", "modified"),
  output = c("value", "audit")
)
```

Arguments

`cf`	Numeric vector of cash-flow amounts.
`t`	Numeric vector of cash-flow times. Times must be expressed in the same units used by ‘rate‘.
`rate`	Effective yield per unit of time. It must be a finite numeric scalar greater than ‘-1‘.
`valuation_time`	Finite numeric scalar indicating the valuation time. Defaults to ‘0‘.
`duration`	Character scalar indicating the duration measure: ‘"macaulay"‘ or ‘"modified"‘.
`output`	Character scalar indicating the returned object: ‘"value"‘ returns the selected duration as a numeric scalar; ‘"audit"‘ returns a detailed tibble.

Details

Let τ be the valuation time. Under an effective yield i, the value of the remaining cash-flow stream is

$$P_\tau = \sum_{j:t_j>\tau} C_j(1+i)^{-(t_j-\tau)}.$$

The Macaulay duration measured from τ is

$$D_{\mathrm{Mac},\tau} = \frac{\sum_{j:t_j>\tau}(t_j-\tau)C_j(1+i)^{-(t_j-\tau)}}{P_\tau}.$$

The modified duration with respect to the effective yield is

$$D_{\mathrm{mod},\tau} = \frac{D_{\mathrm{Mac},\tau}}{1+i}.$$

The function follows an ex-cash-flow convention: cash flows occurring exactly at ‘valuation_time‘ are excluded. Therefore, the result represents duration immediately after any cash flow paid at that time.

For streams containing both positive and negative cash flows, duration remains a yield-sensitivity measure, but it may not have the usual interpretation as a weighted-average payment time. Relative contributions are computed without first forming a monetary duration numerator. Representable products can be recovered when a standalone discount factor underflows; its audit column can then be zero while present value is positive. Discount factors, present values, their total/absolute sum, and the selected contributions/result must remain finite. A signed stream is rejected when absolute total value is at most ‘sqrt(.Machine$double.eps)‘ times the sum of absolute present values.

Value

If ‘output = "value"‘, a numeric scalar.

If ‘output = "audit"‘, a tibble with six columns: ‘time‘, ‘remaining_time‘, ‘cash_flow‘, ‘discount_factor‘, ‘present_value‘, and ‘duration_contribution‘.

Examples

```
duration_cash_flow(
  cf = c(100, 100, 1100),
  t = c(1, 2, 3),
  rate = 0.05
)

duration_cash_flow(
  cf = c(100, 100, 1100),
  t = c(1, 2, 3),
  rate = 0.05,
  duration = "modified"
)

duration_cash_flow(
  cf = c(100, 100, 1100),
  t = c(1, 2, 3),
  rate = 0.05,
  valuation_time = 1,
  output = "audit"
)
```

---

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): $\int_0^s {}_up_y \, du = \frac{p_y}{q_y} \ln\left(\frac{p_y + q_y s}{p_y}\right)$

Integrals use stable exponential/logarithmic increments for small durations. Extinct cohorts contribute zero to later intervals, even if their conditional probabilities are undefined. The CF/Balducci zero-survival interval has integral zero; UDD retains its linear decline. Unknown survivor counts propagate as missing expectations. When annual probabilities must be derived from survivor counts, survival uses the direct count ratio rather than $1 - q_y$. Its logarithm retains the CF integral even when the annual ratio is below floating-point range.

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"`, as in `t_pxy`: `"UDD"`, `"CF"`, `"CML"`, or `"Balducci"`. If not specified and `lt` carries a `frac` attribute, that value is used. An inherited `"CML"` attribute is normalized to `"CF"`.
`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. Table consistency and scalar logical controls are always checked.
`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 and evaluated analytically under UDD, constant force, or Balducci, retaining small survival in log form. Observed annual `lx` ratios determine the fractional survival. Survival is always measured from the initial ages, conditional on both lives being alive at time zero. In particular, last-survivor survival at later times does not imply that both lives remain alive.

Horizons end at the last recorded age: the older life limits joint-life expectation and the younger life limits last-survivor expectation. Survival beyond that age is taken as zero, as in `t_px`; an open table therefore gives a truncated expectation. Missing initial ages or zero initial survivors give `NA`, including when `t = 0`. Tables must have unique consecutive annual integer ages and finite, nonnegative, nonincreasing `lx`. Observed `px`/`qx` must be probabilities consistent with annual `lx` ratios within `1e-10`; unknown probabilities may be `NA`. The age tolerance `tol` does not relax this mortality consistency threshold.

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)
```

`empirical_frequency` *Specify an empirical claim frequency distribution*

Description

Constructs the exact discrete distribution implied by observed claim counts or a weighted count table. No parametric distribution is fitted, no smoothing is applied, and unobserved counts receive probability zero.

Usage

```
empirical_frequency(data, count = NULL, weight = NULL, label = NULL)
```

Arguments

`data`	A nonempty numeric vector of observed nonnegative integer counts, or a data frame containing those counts.
`count`	For a data frame, an unquoted column containing the observed counts. Required for data-frame input and omitted for vector input.
`weight`	Optional nonnegative observation weights. For a data frame, provide an unquoted column; for a vector input, provide a numeric vector of the same length. When omitted, every row has weight one. Weights in a summarized frequency table may represent the number of policies having each claim count; they are **not** exposure offsets for fitting a rate.
`label`	Optional nonempty character scalar identifying the distribution.

Details

If the observed counts are $n_1, \ldots, n_m$ and their nonnegative weights are $w_1, \ldots, w_m$, the empirical probability mass function is

$$\widehat{p}_k = \frac{\sum_{i=1}^{m} w_i \, I(n_i = k)}{\sum_{i=1}^{m} w_i}.$$

At least one weight must be positive. Rows with zero weight do not contribute to the empirical support. Missing, negative, fractional, or nonfinite claim counts are rejected rather than silently discarded. Counts must not exceed 2^{53}, the largest consecutive integer representable exactly by an IEEE-754 double. Weights must be finite, nonnegative, and have a strictly positive finite total. Counts are kept in increasing order; no gaps in the observed set are filled with artificial positive probabilities.

This is the empirical distribution of the **claim count per observational unit** (such as a policy-year), not a table of individual claims. A vector of all ones describes a degenerate count distribution at one, not the total number of claims in a portfolio.

The result has the same lightweight S3 contract as `frequency_spec()`. Its `dist` field is `"empirical"`; `params` holds the finite count values and their exact empirical probabilities, and `data` contains the aggregated observed weights and probabilities. Only this compact frequency table is retained, not the original individual records.

The internal frequency registry provides the PMF, CDF, quantile function, RNG, raw and factorial moments, and probability generating function for this model. A general finite empirical frequency distribution is not assumed to belong to the Panjer $(a, b, 0)$ class; its `panjer` operation is therefore unavailable. Other methods can use the same specification without silently fitting or approximating its probabilities.

Value

An object of classes `"tidyact_frequency_spec"` and `"tidyact_distribution_spec"`, with the ordered fields `schema_version`, `dist`, `params`, `support`, `data`, `custom`, `label`, and `metadata`. The `data` field is a `tibble::tibble()` with columns `count`, `weight`, and `probability`, one row per count with positive empirical mass.

See Also

`frequency_spec()`, `distribution_moments()`

Examples

```
# One observation per policy-year: 0, 1, or 2 claims.
counts <- c(0, 0, 0, 1, 1, 2)
n <- empirical_frequency(counts)
n
n$data
distribution_moments(n, order = 1:3)
distribution_moments(n, order = 1:3, type = "factorial")

# The same empirical distribution as a summarized portfolio table.
portfolio <- tibble::tibble(
  claims = c(0, 1, 2),
```

```
  policies = c(3, 2, 1)
)
portfolio |>
  empirical_frequency(count = claims, weight = policies)

# Unweighted data-frame input and a descriptive label.
observations <- tibble::tibble(claim_count = c(0, 1, 0, 2, 1))
observations |>
  empirical_frequency(count = claim_count, label = "Observed policy counts")
```

`empirical_severity` *Specify an empirical claim severity distribution*

Description

Constructs the discrete severity distribution implied by observed individual claim amounts or a weighted table of claim amounts. No parametric family is fitted, observations are not rounded, and no smoothing or discretization is performed.

Usage

```
empirical_severity(data, amount = NULL, weight = NULL, label = NULL)
```

Arguments

`data`	A nonempty numeric vector of nonnegative observed claim amounts, or a data frame containing those amounts.
`amount`	For data-frame input, an unquoted column containing individual claim amounts. Required for data frames; omit for numeric vectors.
`weight`	Optional nonnegative observation weights. For data frames, supply an unquoted column; for vectors, supply a numeric vector of equal length. If omitted, all observations have weight one. In a summarized table, weights can represent the multiplicities of each observed amount. Weights are not exposure offsets or inputs to a fitted model.
`label`	Optional nonempty character scalar identifying the distribution.

Details

Given observed amounts $x_1, \ldots, x_n$ and nonnegative weights $w_1, \ldots, w_n$, the exact empirical probability assigned to amount x is

$$\widehat{p}(x) = \frac{\sum_{i=1}^{n} w_i I(x_i = x)}{\sum_{i=1}^{n} w_i}.$$

Only strictly positive-weight observations contribute to the distribution; at least one positive weight is required. Repeated amounts are combined and stored in increasing order. Every unobserved

amount has probability zero. Amounts must be finite, nonnegative numeric values. An observed amount of zero is valid and produces a genuine probability mass at zero; it is not removed or converted into a positive-payment severity.

The result is a *finite discrete* severity specification, even when the observed amounts are decimal or come from an underlying continuous loss process. It is **not** a continuous density estimate and has no PDF. Values are compared exactly as represented numerically: close but unequal amounts are not merged. The function also does not infer a loss/payment basis, deductibles, limits, or monetary rounding rules.

The S3 contract matches `severity_spec()`: `dist = "empirical"`, with `params = list(values = ..., probs = ...)` and a compact `data` table. The original individual observations are not retained. The internal severity registry makes the PMF, CDF, quantile, RNG, raw moments, limited moments, MGF, and Laplace transform available. No simulation occurs during construction.

For these finite observations all positive integer-order raw moments exist mathematically, although large powers can exceed floating-point range.

Value

An object with classes `"tidyact_severity_spec"` and `"tidyact_distribution_spec"`, containing `schema_version`, `dist`, `params`, `support`, `data`, `custom`, `label`, and `metadata`. The `data` field is a `tibble::tibble()` with `amount`, `weight`, and `probability` columns, sorted by amount. Only observed amounts with positive empirical probability are retained.

See Also

`severity_spec()`, `empirical_frequency()`, `distribution_moments()`

Examples

```
# One claim amount per observation, including a zero payment.
losses <- c(0, 500, 500, 1250.50, 3000)
x <- empirical_severity(losses)
x
x$data
distribution_moments(x, order = 1:4)

# Exactly the same empirical distribution from a summarized claim table.
claims <- tibble::tibble(
  payment = c(0, 500, 1250.50, 3000),
  claims = c(1, 2, 1, 1)
)
claims |>
  empirical_severity(amount = payment, weight = claims)

# A tidyverse pipeline with individual claim observations.
tibble::tibble(claim_amount = c(150, 150, 375, 900)) |>
  empirical_severity(amount = claim_amount, label = "Reported severities")
```

`forward_curve` *Construct a forward-rate curve from spot rates*

Description

Computes the annual effective forward rates for every consecutive interval implied by an observed spot yield curve. Unlike `forward_rate()`, which evaluates a selected interval, `forward_curve()` returns the complete adjacent-interval schedule in a tidy, long-format tibble.

Usage

```
forward_curve(
  .data = NULL,
  t = NULL,
  i = NULL,
  col_t = "t",
  col_i = "i",
  col_curve = NULL,
  i_type = "effective",
  m = 1L,
  include_origin = TRUE
)
```

Arguments

`.data`	Optional data frame or tibble containing spot maturities and rates, in numeric columns or corresponding list-columns.
`t`	Numeric vector of maturities in years if `.data = NULL`.
`i`	Numeric vector of spot rates if `.data = NULL`.
`col_t`	Name of the maturity column in `.data`.
`col_i`	Name of the spot-rate column in `.data`.
`col_curve`	Optional name of a curve-identifier column in `.data`. Required to distinguish multiple curves stored as numeric node rows; optional for list-column input.
`i_type`	Spot-rate convention: `"effective"`, `"nominal_interest"`, `"nominal_discount"` or `"force"`. Scalar for multiple curves, or scalar/nodewise for a single curve.
`m`	Conversion frequency for nominal rates. A positive integer; scalar for multiple curves, or scalar/nodewise for one curve.
`include_origin`	Logical scalar. If `TRUE` (default), include the initial interval from time zero when the first observed node is positive.

Details

Let $0 = t_0 < t_1 < \cdots < t_n$ denote maturities, and let $v(t_j)$ be their spot discount factors. For each consecutive interval,

$$(1 + f_{t_{j-1},t_j})^{t_j - t_{j-1}} = \frac{v(t_{j-1})}{v(t_j)},$$

so the annual effective forward rate is

$$f_{t_{j-1},t_j} = \left(\frac{v(t_{j-1})}{v(t_j)}\right)^{1/(t_j - t_{j-1})} - 1.$$

The function evaluates log discount-factor ratios from original log spot growth rates, retaining small rates even when reported discounts round to 1. Equal spot log rates are preserved across intervals with a large common origin. Discount factors are obtained from the existing `discount_factor_spot()` function, which standardizes the chosen interest-rate convention. Negative spot and forward rates are supported when the discount factors are positive and finite.

With `include_origin = TRUE`, the first interval starts at time zero and uses the exact identity $v(0) = 1$. A supplied zero-maturity node is retained without duplication. With `include_origin = FALSE`, only intervals between supplied nodes are returned.

Maturities are sorted within each curve. Repeated, nonfinite or negative maturities, missing or nonfinite rates, and invalid discount factors are rejected. No interpolation, extrapolation, curve fitting or forward-rate forecast is performed. Intervals may have unequal lengths. Spot-to-forward relationships represent no-arbitrage implications of the supplied discount factors, not predictions of realized future rates.

Accepted input shapes are: direct numeric `t` and `i` vectors; a data frame with numeric `t` and `i` columns (one node per row, optionally partitioned using `col_curve`); or a data frame with `t` and `i` list-columns (one complete curve per row), such as `yield_curve()` output. Each curve produces one or more rows. `curve` is a character identifier: an ID from `col_curve`, or the input curve's sequential number if no ID is supplied. If more than one curve is supplied, `i_type` and `m` must be scalars; for a single curve they can instead have one value per node.

Value

A tibble, with one row per forward interval and columns `curve`, `t_start`, `t_end`, `duration`, `v_start`, `v_end`, and `f`. `f` is an annual effective forward rate and `duration` is measured in years.

See Also

`forward_rate()`, `yield_curve()`, `discount_factor_spot()`, `bootstrap_spot_curve()`

Other interest: `accumulation_factor()`, `bootstrap_spot_curve()`, `discount_factor()`, `discount_factor_spot()`, `forward_rate()`, `interest_equivalents()`, `par_rate()`, `standardize_interest()`, `yield_curve()`

Examples

```
# A forward curve for consecutive annual spot maturities.
forward_curve(t = 1:3, i = c(0.04, 0.045, 0.05))

# The output of yield_curve() can be used directly in a pipe.
yield_curve(t = c(1, 2, 3), i = c(0.04, 0.045, 0.05)) |>
  forward_curve()

# Multiple curves, using tidyverse-friendly list-columns.
tibble::tibble(
  scenario = c("base", "stress"),
  t = list(c(1, 2, 3), c(1, 3, 5)),
```

```
  i = list(c(0.04, 0.045, 0.05), c(0.05, 0.055, 0.06))
) |>
  forward_curve(col_curve = "scenario")
```

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

Description

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

Usage

```
forward_rate(
  .data = NULL,
  t = NULL,
  i = NULL,
  t_start = NULL,
  t_end = NULL,
  col_t = "t",
  col_i = "i",
  col_t_start = "t_start",
  col_t_end = "t_end",
  i_type = "effective",
  m = 1L,
  method = c("exact", "linear"),
  plot = FALSE,
  .out = "f",
  .out_plot = "forward_rate_plot",
  .keep = c("all", "used", "none"),
  .na = c("propagate", "error", "drop")
)
```

Arguments

`.data`	A data.frame or tibble. If `NULL`, `t`, `i`, `t_start`, and `t_end` must be supplied.
`t`	Numeric vector of maturities in years when `.data = NULL`.
`i`	Numeric vector of spot-rate values 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_t`	Name of the list-column containing maturities.
`col_i`	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.
i_type	Character vector indicating the spot-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force". May have length 1 or the same length as each curve.
m	Positive integer vector giving the conversion frequency for nominal spot rates. May have length 1 or the same length as each curve.
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. For tibble input, col_t and col_i 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 standardized annual effective 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.

This function follows the compact actuarial notation used throughout tidyactuarial: t denotes maturity, i denotes the spot rate, i_type denotes the interest-rate type, and m denotes the conversion frequency for nominal spot rates. The output column f denotes the implied annual effective forward rate.

Exact extraction retains the original log accumulation rates and evaluates the spot-forward identity without forming powers or their ratio. Reported annual equivalents and the final forward rate can reach double precision limits. Linear extraction interpolates annual effective spot rates, not log discounts, and requires finite annual equivalents greater than -1. With .na = "propagate", an incomplete row keeps its position and has a NULL plot entry when plotting is requested.

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: accumulation_factor(), bootstrap_spot_curve(), discount_factor(), discount_factor_spot(), forward_curve(), interest_equivalents(), par_rate(), standardize_interest(), yield_curve()

Examples

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

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

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

# Nominal annual spot rates convertible semiannually
forward_rate(
  t = c(1, 2, 3),
  i = c(0.05, 0.055, 0.06),
  i_type = "nominal_interest",
  m = 2,
  t_start = 1,
  t_end = 3
)
```

`frequency_spec` *Specify a claim frequency distribution*

Description

Creates a lightweight specification of a claim count distribution for use in actuarial risk models.

Usage

```
frequency_spec(dist, ..., custom = NULL, label = NULL)
```

Arguments

`dist`	Character scalar identifying the frequency distribution. Currently supported values are ‘"poisson"‘, ‘"binomial"‘, ‘"negative_binomial"‘, ‘"geometric"‘, and ‘"custom"‘.
`...`	Named parameters defining the selected distribution.
`custom`	Optional named list defining a custom count distribution. Used only when ‘dist = "custom"‘. At minimum, the list must contain ‘pmf‘, a probability mass function.
`label`	Optional character scalar used to identify the specification.

Details

A frequency specification represents a nonnegative integer-valued random variable. The constructor stores the probabilistic model only; it does not simulate observations, estimate parameters, or calculate aggregate losses.

Built-in parameterizations follow ‘stats‘: Poisson uses ‘lambda‘, binomial uses ‘size‘ and ‘prob‘, negative binomial uses ‘size‘ and ‘prob‘, and geometric uses ‘prob‘ with support beginning at zero.

Value

An object with classes ‘"tidyact_frequency_spec"‘ and ‘"tidyact_distribution_spec"‘.

Examples

```
frequency_spec("poisson", lambda = 4)
frequency_spec("binomial", size = 20, prob = 0.15)
frequency_spec("negative_binomial", size = 2.5, prob = 0.4)
frequency_spec("geometric", prob = 0.25)
```

`future_value` *Future value of a single payment*

Description

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

Usage

```
future_value(C, i, i_type = "effective", m = 1, t, tidy = FALSE)
```

Arguments

`C`	Numeric vector of initial payment amounts or capitals.
`i`	Numeric vector of interest-rate values.
`i_type`	Character vector indicating the interest-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer vector giving the conversion frequency for nominal rates. Observed values are validated as positive integers for all rate families; only nominal conversions use their numeric value.
`t`	Numeric vector of times in years from valuation to accumulation.
`tidy`	Logical scalar. If `FALSE`, returns a numeric future value. If `TRUE`, returns a tibble with intermediate calculations.

Details

The future value is computed as

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

where i is the annual effective interest rate.

The input interest rate may be supplied as:

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

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

Valuation retains the log accumulation rate of the original specification. In tidy output, an extreme annual equivalent can round to -1 or overflow to Inf while the requested accumulation factor remains representable. Capital-factor products use log space when a separate factor loses range or precision. The product can be representable even when the factor is not.

This function follows the compact actuarial notation used throughout `tidyactuarial`: `C` denotes the initial payment amount or capital, `t` denotes time, `i` denotes the interest-rate input, `i_type` denotes the interest-rate type, and `m` denotes the conversion frequency for nominal rates.

Input vectors must have length 1 or a common length. Missing values are propagated. This function does not accept dates; use `fv_flow` for dated cash flows.

Value

If `tidy = FALSE`, a numeric vector of future values.

If `tidy = TRUE`, a tibble with input values, equivalent rates, accumulation factors, and future values.

See Also

`standardize_interest`, `present_value`, `fv_flow`

Other time-value: `accumulation_factor()`, `discount_factor()`, `fv_flow()`, `irr_flow()`, `irr_flow_multi()`, `plot_cash_flow()`, `present_value()`, `pv_flow()`, `solve_value()`

Examples

```
# Numeric future value
future_value(C = 1000, i = 0.08, t = 3)

# Nominal interest converted monthly
future_value(
  C = 1000,
  i = 0.12,
  i_type = "nominal_interest",
  m = 12,
  t = 5
)

# Tibble output for teaching or auditing
future_value(
  C = 1000,
  i = 0.08,
  t = 3,
  tidy = TRUE
)

# Vectorized example
future_value(
  C = c(1000, 2500, 4000),
  i = c(0.08, 0.10, 0.12),
  i_type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  t = c(3, 5, 2)
)
```

`fv_flow` *Future value of a general cash flow*

Description

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

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

Usage

```
fv_flow(
  cf,
  i,
  i_type = "effective",
  m = 1L,
  t = NULL,
  date = NULL,
  day_count = c("act/365", "act/360"),
  tol = 1e-10
)
```

Arguments

`cf`	Numeric vector of cash flows.
`i`	Numeric scalar or numeric vector of interest-rate values.
`i_type`	Character vector indicating the interest-rate type: `"effective"`, `"nominal_interest"`, `"nominal_discount"`, or `"force"`. May have length 1 or the same length as `cf`.
`m`	Conversion frequency for nominal rates. Must contain positive integers for nominal conversions and is ignored by other rate families. May have length 1 or the same length as `cf`.
`t`	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.
`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 `cf`. Its timing is supplied either through `t` (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 `i` has length 1, the same rate is used for all cash flows.
- If `i` has the same length as `cf`, each rate is interpreted as the annual effective spot rate associated with the corresponding cash-flow 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 `i` is a vector, the accumulation uses the implied discount-factor ratio under the spot-rate interpretation. The ratio is evaluated from log accumulation rates, avoiding overflow of its separate numerator and denominator. If a horizon annual equivalent rounds to -1 or overflows to Inf, tied horizon flows must have exactly equal log accumulation rates; otherwise the existing annual-rate tolerance applies.

Products are evaluated in log space when separate factors lose range or precision; original cash-flow signs and known zero amounts are preserved. If individual values overflow, amounts with exactly equal finite log factors are netted before valuation. Cancellation between unequal factors is not recovered by this rule and may still produce non-finite results.

This function follows the compact actuarial notation used throughout `tidyactuarial`: `cf` denotes cash flows, `t` denotes time, `i` denotes the interest rate, `i_type` denotes the interest-rate type, and `m` denotes the conversion frequency for nominal rates.

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

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

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 cash-flow time.

See Also

`pv_flow`, `future_value`, `standardize_interest`

Other time-value: `accumulation_factor()`, `discount_factor()`, `future_value()`, `irr_flow()`, `irr_flow_multi()`, `plot_cash_flow()`, `present_value()`, `pv_flow()`, `solve_value()`

Examples

```
# Constant annual effective rate
fv_flow(
  cf = c(100, 150, 200),
  i = 0.08,
  i_type = "effective",
  t = c(0, 1, 2)
)

# Spot rates, one per cash flow
fv_flow(
  cf = c(100, 150, 200),
  i = c(0.05, 0.055, 0.06),
  i_type = "effective",
  t = c(1, 2, 3)
)

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

# Nominal rates by cash flow
fv_flow(
  cf = c(100, 100, 100),
  i = c(0.12, 0.12, 0.12),
  i_type = "nominal_interest",
  m = c(12, 12, 12),
  t = c(1, 2, 3)
)
```

---

immunize_duration *Duration-based immunization with multiple assets*

Description

Computes asset weights that duration-immunize a stream of liabilities using two or more assets, using compact actuarial notation.

Usage

```
immunize_duration(L, t, P, D, i, i_type = "effective", m = 1L)
```

Arguments

`L`	Numeric vector with liability payments.
`t`	Numeric vector of the same length as `L`, giving the times at which each liability payment occurs.
`P`	Numeric vector with present values or prices of the immunizing assets, evaluated on the same yield basis.
`D`	Numeric vector with the Macaulay duration of each asset, expressed in the same time units as `t`.
`i`	Numeric scalar. Interest-rate input used to discount the liabilities.
`i_type`	Character string indicating the interest-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Conversion frequency for nominal rates. Ignored for `i_type = "effective"` and `i_type = "force"`.

Details

The method enforces:

1. present value of assets = present value of liabilities;
2. Macaulay duration of assets = Macaulay duration of liabilities.

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

This function follows the compact actuarial notation used throughout `tidyactuarial`: `L` denotes liabilities, `t` denotes payment times, `P` denotes asset prices or present values, `D` denotes asset durations, `i` denotes the interest rate, `i_type` denotes the interest-rate type, and `m` denotes the conversion frequency for nominal rates.

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

$$\sum_j w_j P_j = PV_L$$

and

$$\frac{\sum_j w_j P_j D_j}{\sum_j w_j P_j} = D_L.$$

For two assets, the closed-form solution is:

$$w_1 = \frac{PV_L(D_L - D_2)}{P_1(D_1 - D_2)}, \qquad w_2 = \frac{PV_L - w_1 P_1}{P_2}.$$

For three or more assets, the minimum-norm solution of the linear system $Aw = b$ is computed, where A is a $2 \times r$ matrix with rows

$$(P_1, \ldots, P_r)$$

and

$$(P_1 D_1, \ldots, P_r D_r),$$

and

$$b = (PV_L, PV_L D_L)^T.$$

The minimum Euclidean norm is taken over asset units `w`, not portfolio market-value fractions. It is evaluated by singular value decomposition of the row-scaled constraint matrix, without forming normal equations. Short positions are allowed. Matching present value and duration gives a zero value and first derivative of the surplus at the base yield, but does not ensure nonnegative surplus after rate shifts; the Redington convexity condition must be checked separately. Liability values retain the original log accumulation rate, and relative moments avoid monetary numerator overflow. Payment times include zero: a liability at time zero contributes value but no duration. Constraints are scaled by a common monetary magnitude and by moment magnitudes before products are formed. These are row transformations, preserving the norm over units; asset columns are not rescaled independently. Final values, weights and sensitivities must be finite, and nearly cancelled liability value is rejected. Rank and cancellation tests remain subject to double precision; reported portfolio moments should be inspected for ill-conditioned systems.

Value

A tibble with:

w Numeric vector of asset weights or units.

PV_L Present value of the liabilities.

D_L Macaulay duration of the liabilities.

PV_A Present value of the immunized asset portfolio.

D_A 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(
  L = c(5000, 8000),
  t = c(3, 7),
  P = c(100, 200),
  D = c(3, 7),
```

```
  i = 0.05
)

# Three-asset immunization: minimum-norm solution
immunize_duration(
  L = c(5000, 8000),
  t = c(3, 7),
  P = c(100, 150, 200),
  D = c(2, 5, 8),
  i = 0.05
)

# Nominal annual interest rate convertible monthly
immunize_duration(
  L = c(5000, 8000),
  t = c(3, 7),
  P = c(100, 200),
  D = c(3, 7),
  i = 0.06,
  i_type = "nominal_interest",
  m = 12
)
```

`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 = present value of liabilities;
2. Macaulay duration of assets = Macaulay duration of liabilities;
3. convexity of assets = convexity of liabilities.

Usage

```
immunize_duration_convexity(L, t, P, D, C, i, i_type = "effective", m = 1L)
```

Arguments

`L`	Numeric vector with liability payments.
`t`	Numeric vector of the same length as `L`, giving the times at which each liability payment occurs.
`P`	Numeric vector with present values or prices of the immunizing assets, evaluated on the same yield basis.

D	Numeric vector with the Macaulay duration of each asset, expressed in the same time units as t.
C	Numeric vector with each asset's relative second derivative $P_j''(i_e)/P_j(i_e)$ with respect to the annual effective yield i_e, using the same time units as t. A coupon-period convexity divided by squared coupon frequency generally uses a different yield variable and must be converted before use here.
i	Numeric scalar. Interest-rate input used to discount the liabilities.
i_type	Character string indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer. Conversion frequency for nominal rates. Ignored for i_type = "effective" and i_type = "force".

Details

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

This function follows the compact actuarial notation used throughout tidyactuarial: L denotes liabilities, t denotes payment times, P denotes asset prices or present values, D denotes asset durations, C denotes asset convexities, i denotes the interest rate, i_type denotes the interest-rate type, and m denotes the conversion frequency for nominal rates.

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 $v = 1/(1+i)$ after converting i to the equivalent effective rate.

The weights w_j satisfy the $3 \times r$ system $Aw = b$, where the rows of A are

$$(P_1, \ldots, P_r),$$

$$(P_1 D_1, \ldots, P_r D_r),$$

and

$$(P_1 C_1, \ldots, P_r C_r),$$

and

$$b = (PV_L,\ PV_L D_L,\ PV_L C_L)^T.$$

For three assets, the system is square and solved directly. For four or more assets, the minimum-norm solution

$$w = A^T (A A^T)^{-1} b$$

is computed. It is evaluated by singular value decomposition of the row-scaled constraint matrix, without forming normal equations. The norm is over asset units w, not market-value fractions; short positions are allowed.

All asset sensitivities must use the same annual effective yield as the liabilities, even when i_type specifies a nominal rate or a force. For coupon frequency k, let $j = (1+i_e)^{1/k} - 1$. If $C_j = P''(j)/P$ and $D_j^* = -P'(j)/P$, the required convexity is

$$C_{i_e} = C_j [j'(i_e)]^2 - D_j^* j''(i_e).$$

The outputs `discrete_convexity_periods` from `bond_convexity()` and `modified_duration_periods_j` from `bond_duration()` give C_j and D_j^*. Alternatively, use `convexity_cash_flow()` on the asset cash flows at the annual effective yield.

Matching convexity makes the surplus's second derivative zero at the base yield. This differs from Redington's strict positive-curvature condition and does not guarantee a nonnegative surplus after rate shifts. Liability values retain the original log accumulation rate; convexity is still with respect to the annual effective yield. Relative moments and row scaling before monetary products prevent avoidable overflow. Scaling uses common monetary and moment magnitudes, preserving the minimum norm over asset units, without rescaling asset columns independently. Time-zero liabilities contribute value but zero duration and convexity. Final values, weights and sensitivities must be finite; nearly cancelled liability values are rejected. Rank and cancellation tests remain subject to double precision; reported portfolio moments should be inspected for ill-conditioned systems.

Value

A tibble with:

w Numeric vector of asset weights or units.

PV_L Present value of the liabilities.

D_L Macaulay duration of the liabilities.

C_L Discrete convexity of the liabilities.

PV_A Present value of the asset portfolio.

D_A Macaulay duration of the asset portfolio.

C_A 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
immunize_duration_convexity(
  L = c(5000, 8000, 10000),
  t = c(3, 5, 7),
  P = c(100, 150, 200),
  D = c(2, 5, 8),
  C = c(6, 30, 72) / (1.05)^2,
  i = 0.05
)

# Four-asset immunization: minimum-norm solution
immunize_duration_convexity(
  L = c(5000, 8000, 10000),
  t = c(3, 5, 7),
  P = c(100, 120, 150, 200),
```

```
  D = c(2, 4, 6, 8),
  C = c(6, 20, 42, 72) / (1.05)^2,
  i = 0.05
)

# Nominal annual interest rate convertible monthly
immunize_duration_convexity(
  L = c(5000, 8000, 10000),
  t = c(3, 5, 7),
  P = c(100, 150, 200),
  D = c(2, 5, 8),
  C = c(6, 30, 72) / (1 + 0.06 / 12)^(2 * 12),
  i = 0.06,
  i_type = "nominal_interest",
  m = 12
)
```

`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 and is payable at the end of the subperiod of death, using compact actuarial notation.

Usage

```
insurance_variable_k(
  lt,
  x,
  i,
  benefit,
  n = NULL,
  h = 0,
  k = 12,
  i_type = "effective",
  m = 1L,
  frac,
  tidy = FALSE,
  check = TRUE,
  tol = 1e-10
)
```

Arguments

`lt`	A life table object or data frame containing at least columns `x` and `lx`.
`x`	Integer actuarial age at issue.

i	Annual interest-rate input.
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 benefit is numeric. If benefit is a function, n must be supplied.
h	Nonnegative deferment period in years. Default is 0.
k	Positive integer. Number of subperiods per year. Default is 12.
i_type	Character scalar indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer scalar giving the conversion frequency for nominal interest-rate inputs. Ignored for "effective" and "force".
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.
tol	Numeric tolerance used for integer-grid checks.

Details

This function is useful for level, increasing, decreasing, and credit-style life insurance benefits when benefits are specified at a subannual frequency.

This function follows the compact actuarial notation used throughout tidyactuarial: lt is the life table, x is the age at issue, i is the interest-rate input, i_type is the interest-rate type, m is the conversion frequency for nominal rates, n is the term, h is the deferment period, and k is the number of subperiods per year.

Let k be the number of subperiods per year and $N = nk$ the total number of subperiods in the insurance term. With deferment h, the actuarial present value at age x is:

$$APV = \sum_{j=1}^{N} v^{h+j/k} b_j \left({}_{h+(j-1)/k}p_x - {}_{h+j/k}p_x \right).$$

Here b_j is the benefit payable if death occurs in subperiod j, and $v = (1 + i_e)^{-1}$, where i_e is the annual effective interest rate equivalent to the input i, i_type, and m.

If benefit is numeric of length 1, it is recycled to all subperiods. If it is numeric with length greater than 1, its length must equal nk. If benefit is a function, it is evaluated at the end of each subperiod, at times $1/k, 2/k, \ldots, n$.

Fractional mortality is interpolated under the selected assumption. Death masses are evaluated directly from annual survivor decrements and stable fractional formulas, avoiding subtraction of rounded survival probabilities. The table must have consecutive annual ages and nonincreasing finite 'lx', with positive exposure at issue and support through 'x + h + n'. No extra terminal death year is invented. A zero annual survival under CF or Balducci uses the limiting mass at the beginning of that annual interval, consistently with the package's fractional survival convention.

This values one policy per call. A benefit vector is a sequence of payments, not a vector of policy scenarios; signed benefits are supported. Functional benefits are evaluated at times relative to the start of coverage, excluding ‘h‘; discounting and death probabilities use absolute times including ‘h‘. The payment is at the end of the subperiod of death, not at the exact death time. Full contractual amounts are used, without division by ‘k‘. ‘i_effective‘ is diagnostic and may round to -1 or Inf for an extreme valid original rate; valuation retains the original log accumulation rate.

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.

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

Examples

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

# Monthly insurance with increasing benefits
insurance_variable_k(
  lt = 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 = lt,
  x = 60,
  i = 0.05,
  benefit = balance,
  n = 1,
  k = 12
)

# Level benefit with annual payments
insurance_variable_k(
  lt = lt,
```

```
  x = 60,
  i = 0.05,
  benefit = 1,
  n = 5,
  k = 1
)

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

# Nominal annual interest convertible monthly
insurance_variable_k(
  lt = lt,
  x = 60,
  i = 0.06,
  i_type = "nominal_interest",
  m = 12,
  benefit = rep(1000, 12),
  n = 1,
  k = 12
)

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

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

Description

Computes the actuarial present value of a discrete single-life insurance using a life table and compact actuarial notation.

Usage

```
insurance_x(
  lt,
  x,
  i,
  i_type = "effective",
  m = 1L,
  n = Inf,
  h = 0L,
  type = c("whole", "term", "endowment"),
  benefit = 1,
  tidy = FALSE,
  ...
)
```

Arguments

lt	A life table as produced by lifetable. It must contain columns x and lx.
x	Integer actuarial age at issue, or a vector of scenario ages.
i	Numeric annual interest-rate input, scalar or by scenario.
i_type	Character string indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer. Conversion frequency for nominal rates. Ignored for i_type = "effective" and i_type = "force".
n	Integer insurance term in years. Required for type = "term" and type = "endowment". Use Inf only for whole-life insurance.
h	Integer deferment period in years.
type	Character string. One of "whole", "term", or "endowment".
benefit	Numeric benefit amount, scalar or by scenario. A vector represents separate contracts, not a benefit schedule within one contract.
tidy	Logical scalar. If FALSE, returns a numeric APV. If TRUE, returns one tibble row per scenario with intermediate quantities.
...	Transitional compatibility for older calls using mortality_table, age, rate, rate_type, term_years, deferral_years, insurance_type, and output.

Details

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

Supported contracts:

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

This function follows the compact actuarial notation used throughout `tidyactuarial`: `lt` is the life table, `x` is the age at issue, `i` is the interest-rate input, `i_type` is the interest-rate type, `m` is the conversion frequency for nominal rates, `n` is the insurance term, and `h` is the deferment period.

The function computes APVs directly from `lx`. For a deferred insurance, the value at age `x + h` is multiplied by

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

For whole-life insurance, the death benefit APV at the deferred starting age is computed over the available life-table horizon. For term and endowment insurance, the death benefit is computed over the first `n` years. Endowment insurance additionally includes the pure endowment component

$$v^n \, {}_np_x.$$

Scenarios. `x`, `i`, `i_type`, `m`, `n`, `h`, `type`, and `benefit` may have a common positive length; only length-one arguments recycle. The table and `tidy` control are shared. Scenarios are paired by position, not expanded to a Cartesian product. Missing or invalid cases are rejected with the scenario number. Use the numeric result in `dplyr::mutate()`, or request a tibble for intermediate values. Explicit interest arguments override contract defaults.

Valuation and scope. For annual deaths in interval $[r, r+1)$, the unconditional payment probability is $(l_{x+r} - l_{x+r+1})/l_x$. The benefit is discounted at $r + 1$, not at the exact moment of death. Decrements are formed before division, and log ratios and original interest rates retain representable APVs when intermediate conditional values or discount factors overflow or underflow. Intermediate diagnostics may therefore be zero or infinite while APV is finite. Tables require an annual age grid, nonincreasing finite `lx`, and a positive count at issue. For compatibility the terminal count one year beyond the table is zero, even for an open input table. Extinct tails contribute zero. A zero-term endowment pays the benefit at time `h` on survival; zero-term death insurance pays nothing.

Use `insurance_variable_k` for benefits varying by subperiod, including level benefits paid at the end of the subperiod of death, and `t_Ex` for pure endowments. Continuous death-time insurance, fractional issue ages and time-varying interest curves require additional model specifications and are not implied by this annual contract.

Value

If `tidy = FALSE`, a numeric vector of actuarial present values, of length one for a single scenario.

If `tidy = TRUE`, a tibble with one row per scenario, the main input values, equivalent interest rate, deferral factor, pure endowment factor, annuity-due value used in the standard identities, and APV.

See Also

`annuity_x`, `premium_x`, `reserve_x`, `t_Ex`, `insurance_xy`

Other life-contingencies: `add_insurance()`, `add_premium_schedule()`, `annuity_multi()`, `annuity_x()`, `annuity_xy()`, `insurance_variable_k()`, `insurance_xy()`, `life_contract()`, `premium_gross()`, `premium_x()`, `premium_xy()`, `reserve_x()`, `reserve_xy()`, `simulate_annuity_x()`, `simulate_insurance_x()`

Examples

```
lt <- data.frame(
  x  = 60:65,
```

```
  lx = c(100000, 99000, 97500, 95500, 93000, 90000)
)

# Whole-life insurance
insurance_x(
  lt = lt,
  x = 60,
  i = 0.06,
  type = "whole"
)

# 5-year term insurance
insurance_x(
  lt = lt,
  x = 60,
  i = 0.06,
  n = 5,
  type = "term"
)

# 5-year endowment insurance
insurance_x(
  lt = lt,
  x = 60,
  i = 0.06,
  n = 5,
  type = "endowment"
)

# Deferred whole-life insurance
insurance_x(
  lt = lt,
  x = 60,
  i = 0.06,
  h = 2,
  type = "whole"
)

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

# Parallel single-life scenarios with a shared mortality table
insurance_x(lt, x = c(60, 61), i = 0.04, n = c(2, 3),
            type = "term", benefit = c(10000, 20000), tidy = TRUE)
```

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

Description

This function evaluates cause-specific insurance benefits under a multiple decrement model. It supports whole-life and term insurance, integer deferment, vectorized issue ages and interest-rate assumptions, and compact actuarial notation.

Usage

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

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

Arguments

`md`	A multiple decrement table produced by `md_table`. Must contain columns `x`, `p_total`, and the requested `cause`.

x	Integer age(s) at issue.
i	Finite annual interest-rate input in the valid domain of its original rate specification. Extreme annual equivalents may round to -1 or overflow, while the original log accumulation remains usable.
cause	Character scalar. Name of the cause column in md, for example "q_death".
type	Character scalar. Insurance type: "whole" or "term".
benefit	Numeric benefit amount payable at the end of the year of decrement by the specified cause. Default is 1.
n	Integer term length in years. Required when type = "term".
h	Integer deferment period in years. Default is 0.
i_type	Character vector indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer vector giving the conversion frequency for nominal rates. Ignored for "effective" and "force".
tidy	Logical. If TRUE, returns a tibble with inputs and insurance_xj.
check	Logical. If TRUE, performs input validation.
tol	Numeric tolerance for integer checks.

Details

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

Let $q_{x+k}^{(j)}$ be the one-year decrement probability for cause j at age $x+k$, and let $p_{x+r}^{(\tau)}$ be the one-year total survival probability against all decrements at age $x+r$.

For a product with deferment h and term n, with benefit payable at the end of the year of decrement by cause j, the APV is:

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

Here $v = (1+i_e)^{-1}$, where i_e is the annual effective interest rate obtained from i, i_type, and m.

If type = "whole", the function sets n to the remaining length of the table after deferment, that is, whole life over the available ages. With an open table this is a truncated value, not a complete whole-life APV; completeness requires that no active survivors remain beyond the last row. Cause probabilities are dependent probabilities of exit from the active state in a competing-decrement model, not independent/net decrement rates. Survival to each year's start uses all decrements. The payment is at year end, not the moment of exit; the cause-specific probability is not multiplied by survival to year end. Values are conditional on active status at issue.

The selected probabilities must be finite, in [0,1], with cause probability no greater than total decrement. Ages must be consecutive annual ages. Optional 'q_total' is checked against 'p_total'; redundant cohort counts are not the valuation basis. 'q_total' itself can be selected to value all exits. Original log accumulation and cumulative log survival retain representable values despite overflowing annual equivalent rates or underflowing intermediate survival. Scalar inputs recycle to a common positive scenario length; 'cause' and 'type' select one model per call.

This function follows the compact actuarial notation used throughout `tidyactuarial`: `md` is a multiple decrement table, `x` is age at issue, `i` is the interest rate, `i_type` is the interest-rate type, `m` is the conversion frequency for nominal rates, `n` is the term, and `h` is the deferment period.

Value

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

See Also

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

Examples

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

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

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

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

Description

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

Usage

```
insurance_xy(
  lt,
  x = NULL,
  y = NULL,
  i = NULL,
  i_type = "effective",
  m = 1L,
  type = c("whole", "term", "endowment"),
  status = c("joint", "last"),
  n = Inf,
  h = 0L,
  benefit = 1,
  frac,
  tidy = FALSE,
  ...
)
```

Arguments

`lt`	A life table, a list of two life tables `list(lt_x, lt_y)`, or a `tidyact_life_contract` object created by `life_contract`. Each life table must contain columns `x` and `lx`.
`x`	Integer actuarial age for the first life.
`y`	Integer actuarial age for the second life.
`i`	Numeric scalar. Annual interest-rate input.
`i_type`	Character string indicating the interest-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Conversion frequency for nominal rates. Ignored for `i_type = "effective"` and `i_type = "force"`.
`type`	Character string. One of `"whole"`, `"term"`, or `"endowment"`.
`status`	Character string. Use `"joint"` for first-death insurance or `"last"` for second-death insurance.
`n`	Term in years. Required as finite for term and endowment insurance. Use `Inf` for whole-life insurance.
`h`	Nonnegative integer deferment period in years.
`benefit`	Numeric scalar. Insurance benefit.
`frac`	Fractional-age assumption used in two-life survival probabilities. One of `"UDD"`, `"CF"`, `"CML"`, or `"Balducci"`.

tidy	Logical scalar. If FALSE, returns a numeric APV. If TRUE, returns a one-row tibble.
...	Transitional compatibility for older calls using mortality_table, age_x, age_y, rate, rate_type, insurance_type, cohort, term_years, deferment_years, and output.

Details

Supported contracts:

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

The status argument determines the two-life status:

- status = "joint": first-death insurance, based on the joint-life status.
- status = "last": second-death insurance, based on the last-survivor status.

This function follows the compact actuarial notation used throughout tidyactuarial: lt is the life table input, x and y are the two actuarial ages, i is the interest-rate input, i_type is the interest-rate type, m is the conversion frequency for nominal rates, n is the insurance term, and h is the deferment period.

Both lives are alive at issue (state 11). Internal states 10 and 01 mean only x or only y is alive; 00 is absorbing. The API does not take an observed single-survivor initial state. In a joint-life contract, leaving 11 triggers the first-death benefit once. A last-survivor contract continues through 10 or 01 and pays on entering 00. Both deaths within one annual interval trigger one benefit, payable at the end of that year.

Valuation sums discounted annual transition masses from the issue ages; it does not subtract rounded annuity values. With marginal survival Sx, Sy, cumulative death Qx, Qy, and interval death masses Dx, Dy, the first-death mass in (t,t+1] is Dx(t) Sy(t) + Sx(t+1) Dy(t); the second-death mass is Dx(t) Qy(t) + Qx(t) Dy(t) + Dx(t) Dy(t). Counts supply the decrements directly. Original log interest and monetary amounts are combined before exponentiating when necessary. The annual payment model is independent of fractional interpolation; frac is retained as an audit assumption.

The following fully discrete identities remain checks on the result:

For a whole-life contract:

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

For an n-year term insurance:

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

For an n-year endowment insurance:

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

Whole-life valuation requires observed mortality through extinction of the selected status: at least one marginal must reach zero within the joint horizon, and both must reach zero for last-survivor insurance. An open table cannot imply a terminal death or maturity benefit. Use an explicit term

or endowment for the observed horizon, or extend the table. No missing annual interval of an open marginal table is extrapolated. A zero-term endowment pays benefit at h if the status survives; a zero-term term insurance has no death coverage and pays zero. Annual tables must satisfy the consistency requirements of t_pxy, and issue ages must have positive lx. One call values one contract, not scenarios.

A deferred insurance retains survival probabilities from the issue ages. Its continuation value conditional on the selected status surviving to h is multiplied by the deferment factor

$$v^h \, {}_hp_{xy}$$

for the selected two-life status. For last-survivor insurance, this conditioning includes paths with only one survivor at h; it does not restart both lives at deferred ages. In tidy output, `annuity_due_value` is the annuity at time h conditional on that status surviving to h, and `survival_to_maturity`, when reported, uses the same conditioning.

Value

If `tidy = FALSE`, a numeric scalar.

If `tidy = TRUE`, a one-row tibble with input values, standardized interest rate, deferment factor, unit APV, and APV.

See Also

annuity_xy, premium_xy, insurance_x, t_pxy

Other life-contingencies: add_insurance(), add_premium_schedule(), annuity_multi(), annuity_x(), annuity_xy(), insurance_variable_k(), insurance_x(), life_contract(), premium_gross(), premium_x(), premium_xy(), reserve_x(), reserve_xy(), simulate_annuity_x(), simulate_insurance_x()

Examples

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

insurance_xy(
  lt = lt,
  x = 60,
  y = 62,
  i = 0.06,
  type = "whole",
  status = "joint"
)

insurance_xy(
  lt = lt,
  x = 60,
  y = 62,
  i = 0.06,
  type = "term",
```

```
  status = "last",
  n = 10,
  tidy = TRUE
)

lt |>
  life_contract(lives = "joint", x = 60, y = 62, i = 0.06) |>
  insurance_xy(
    type = "term",
    n = 4,
    status = "joint"
  )
```

`interest_equivalents` *Equivalent interest rates in FM actuarial notation*

Description

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

Usage

```
interest_equivalents(
  i_type = c("effective", "nominal_interest", "nominal_discount", "force"),
  i,
  m = 1L
)
```

Arguments

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

Details

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

This function follows the compact actuarial notation used throughout `tidyactuarial`: `i` denotes the interest-rate value, `i_type` denotes the type of interest rate, and `m` denotes the conversion frequency for nominal rates.

Calculations retain the log accumulation rate of the original input. An extreme annual effective equivalent may round to -1 or overflow to Inf while other equivalent quantities remain representable.

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

notation Actuarial notation for the equivalent rate.

m Conversion frequency used for nominal rates.

description Human-readable description.

value Equivalent rate value.

See Also

`standardize_interest`, `discount_factor_spot`

Other interest: `accumulation_factor()`, `bootstrap_spot_curve()`, `discount_factor()`, `discount_factor_spot()`, `forward_curve()`, `forward_rate()`, `par_rate()`, `standardize_interest()`, `yield_curve()`

Examples

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

# Batch use with purrr
if (requireNamespace("purrr", quietly = TRUE) &&
    requireNamespace("tibble", quietly = TRUE)) {
  cases <- tibble::tibble(
    i_type = c("effective", "force"),
    i = 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, using compact actuarial notation.

Usage

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

Arguments

cf	Numeric vector of cash flows.
t	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.
m	Positive integer used only to report an equivalent nominal annual interest rate convertible 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 cf. Its timing is given either through t (in years) or date (calendar dates). If date is supplied, the earliest date is treated as time 0.

The IRR returned is interpreted as an annual effective rate.

Root finding scales the cash flows by their largest absolute amount; this preserves roots across monetary units. The reported NPV remains in the original currency. Scaling that would erase a nonzero amount through underflow is rejected. The tolerance controls the rate-coordinate solve.

This function follows the compact actuarial notation used throughout `tidyactuarial`: `cf` denotes cash flows, `t` denotes time, and `m` denotes the conversion frequency used to report the equivalent nominal annual interest 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. If there is exactly one sign change, the IRR is usually unique under the usual ordered cash-flow setting.

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 `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 `cf`.

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: `accumulation_factor()`, `discount_factor()`, `future_value()`, `fv_flow()`, `irr_flow_multi()`, `plot_cash_flow()`, `present_value()`, `pv_flow()`, `solve_value()`

Examples

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

irr_flow(
  cf = 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, using compact actuarial notation.

Usage

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

Arguments

`cf`	Numeric vector of cash flows.
`t`	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.
`m`	Positive integer used only to report an equivalent nominal annual interest rate convertible `m` times per year.
`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.
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 scaled NPV evaluates to zero), and applies uniroot to each candidate interval.

The IRRs returned are interpreted as annual effective rates.

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

Cash flows are divided by their largest absolute amount during the search; reported NPVs remain in the original currency. Scaling that would erase a nonzero amount through underflow is rejected. Only exact computed grid zeros and sign-changing brackets generate candidates: an absolutely small monetary NPV is not by itself evidence of a root. Grid detection remains heuristic, including for tangential or very close roots.

This function follows the compact actuarial notation used throughout tidyactuarial: cf denotes cash flows, t denotes time, and m denotes the conversion frequency used to report the equivalent nominal annual interest rate.

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 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 `cf`.

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: `accumulation_factor()`, `discount_factor()`, `future_value()`, `fv_flow()`, `irr_flow()`, `plot_cash_flow()`, `present_value()`, `pv_flow()`, `solve_value()`

Examples

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

# A cash flow with multiple sign changes
irr_flow_multi(
  cf = c(-1000, 5000, -4500, 200),
  t = c(0, 1, 2, 3),
  search_interval = c(-0.99, 5),
  grid_points = 5000
)

# Date-based version
irr_flow_multi(
  cf = 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).
- $a_x = (L_x - \ell_{x+\Delta}\Delta)/d_x$: mean time lived within the interval by those who die, in years (not a fraction when interval width differs from one). Undefined if there are no deaths.

Nonfinite time/entry rows are omitted with a warning; the remaining sample must be nonempty. Breaks are sorted and deduplicated. Beyond the last observation, the KM step is held constant (no tail extrapolation is fitted). CF/Balducci exposure at an interval ending at zero survival uses its zero limit; positive deaths with zero exposure give infinite central mortality. Confidence intervals remain undefined at survival zero or one. Standard errors and intervals use Greenwood sums directly, preserving small positive survival estimates without a fixed survival floor. For delayed entry the risk set uses the existing inclusive convention `entry <= time`; this is a deliberate convention at tied entries/events. The lifetable component may have arbitrary interval widths: annual consumers such as `t_px` require conversion to an annual grid with the needed endpoint survivors; its class alone does not certify that conversion.

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")
```

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

Description

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

Usage

```
life_contract(
  lt,
  lives = c("single", "joint", "last_survivor"),
  x = NULL,
  y = NULL,
  i,
  i_type = "effective",
  m = 1L,
  ...
)
```

Arguments

`lt`	A life table or a list of two life tables. For single-life contracts, provide one data frame or tibble. For two-life contracts, provide either one data frame used for both lives, or ‘list(lt_x, lt_y)‘ with one table for each life. Each life table must contain columns ‘x‘ and ‘lx‘.
`lives`	Character string. Use ‘"single"‘ for a single-life contract, ‘"joint"‘ for a joint-life two-life contract, or ‘"last_survivor"‘ for a last-survivor two-life contract.
`x`	Numeric scalar. Age of the single life, or age of the first life in a two-life contract.
`y`	Numeric scalar. Age of the second life in a two-life contract. Required when ‘lives‘ is ‘"joint"‘ or ‘"last_survivor"‘.

i	Numeric scalar. Annual interest-rate input.
i_type	Character string indicating the interest-rate type. Allowed values are '"effective"', '"nominal_interest"', '"nominal_discount"', and '"force"'.
m	Positive integer. Conversion frequency for nominal rates. Ignored for 'i_type = "effective"' and 'i_type = "force"'.
...	Reserved for future extensions. Deprecated argument names such as 'mortality_table', 'age', 'age_x', 'age_y', 'rate', and 'rate_type' are not accepted.

Details

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

'life_contract()' follows the compact actuarial notation used throughout 'tidyactuarial':

* 'lt': life table; * 'x': age of the first or single life; * 'y': age of the second life; * 'i': interest rate; * 'i_type': type of interest rate; * 'm': conversion frequency for nominal rates.

The object stores actuarial fields using the compact names above. During the 0.1.4 API transition, it also stores internal compatibility fields so that functions not yet migrated can continue to read the contract. These internal fields are not part of the preferred user-facing notation.

The original rate and table are retained, without converting the stored rate to an annual effective rate or altering the table. Rate domains require 'i > -1' for effective rates, '1 + i/m > 0' for nominal interest and '1 - i/m > 0' for nominal discount; any finite force is allowed. Table validation here checks structure and numeric 'x'/'lx' columns. Valuation functions additionally check mortality consistency, supported ages, horizons and payment grids for their particular models. Integer-valued specifications outside R's integer range are stored as doubles, never 'NA'.

Value

An object of class '"tidyact_life_contract"'.

See Also

Other life-contingencies: add_insurance(), add_premium_schedule(), annuity_multi(), annuity_x(), annuity_xy(), insurance_variable_k(), insurance_x(), insurance_xy(), premium_gross(), premium_x(), premium_xy(), reserve_x(), reserve_xy(), simulate_annuity_x(), simulate_insurance_x()

Examples

```
lt <- data.frame(
  x = 40:90,
  lx = round(100000 * exp(-0.018 * (0:50)^1.35))
)
lt$lx[nrow(lt)] <- 0

life_contract(
  lt = lt,
  lives = "single",
```

```
  x = 40,
  i = 0.05
)

life_contract(
  lt = lt,
  lives = "joint",
  x = 60,
  y = 58,
  i = 0.05
)
```

`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). With FALSE and only lx, the last dx/qx/px/mx are unknown (NA); a supplied terminal probability is used when available.
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 $1/a_x$, which is 2 for ax = 0.5, 1 for ax = 1, and Inf for ax = 0. 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, including px = 1 - qx when both are provided. Validated ages within tolerance of integers are rounded before truncation.
tol	Numeric tolerance for integer checks and consistency checks.

Details

The table can be built from any 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):

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

With a_x the mean fraction of the year lived by those who die, exposure is $L_x = \ell_{x+1} + a_x d_x$ and $m_x = d_x/L_x$. Consequently,

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

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 $d_\omega = \ell_\omega$. For positive terminal survivors, $L_\omega = a_x \ell_\omega$ and $m_\omega = 1/a_x$, which is 2 under UDD (`ax = 0.5`). At `ax = 0`, the zero-exposure limit is $m_\omega = \infty$; at `ax = 1`, $m_\omega = 1$. Infinite `mx` inputs remain unsupported; this limit can be obtained from `lx` or `qx`.

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 $1/a_x$ and is `Inf` if `ax = 0`.

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
```

`loans_sample` *Sample loan contracts for amortization examples*

Description

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

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

Usage

```
loans_sample

loans_sample
```

Format

A tibble with 4 rows and 7 variables:

loan_id Loan identifier.

L Initial loan principal.

i Annual effective interest rate.

n_months Loan term in months.

k Number of payments per year.

loan_type Short loan type label.

R Level payment amount per payment period.

A tibble with 4 rows and 7 variables:

loan_id Loan identifier.

L Initial loan principal.

i Annual effective interest rate.

n_months Loan term in months.

k Number of payments per year.

loan_type Short loan type label.

R Level payment amount per payment period.

Details

This dataset uses compact loan notation: `L` is the initial loan principal, `i` is the annual effective interest rate, `n_months` is the loan term in months, `k` is the payment frequency, and `R` is the level payment.

This dataset uses compact loan notation: `L` is the initial loan principal, `i` is the annual effective interest rate, `n_months` is the loan term in months, `k` is the payment frequency, and `R` is the level payment.

Source

Synthetic pedagogical data created for tidyactuarial examples.

Synthetic pedagogical data created for tidyactuarial examples.

Examples

```
data(loans_sample)

loans_sample |>
  dplyr::select(loan_id, L, i, n_months, k, R)

data(loans_sample)

loans_sample |>
  dplyr::select(loan_id, L, i, n_months, k, R)
```

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

Description

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

Usage

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

Arguments

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

Details

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

Value

A lifetable object as produced by `lifetable`.

Examples

```
qx_df <- tibble::tibble(
  x = 30:35,
  q_death = c(0.001, 0.0012, 0.0014, 0.0017, 0.0020, 1.0000),
  q_disability = c(0.002, 0.0021, 0.0022, 0.0023, 0.0024, 0.0000)
)
md <- md_table(qx_df, radix = 1e5, close = TRUE)
lt <- lt_tau(md, radix = 1e5, close = TRUE, frac = "UDD")
t_px(lt, x = 30, t = 5)
```

mc_annuity *Compute simulated present values for life annuities*

Description

Computes Monte Carlo simulated present values of life annuity payments from simulated future lifetimes, using compact actuarial notation.

Usage

```
mc_annuity(
  .data = NULL,
  i,
  payment = 1,
  k = 1L,
  type = c("whole", "temporary", "deferred", "deferred_temporary", "certain",
    "guaranteed", "whole_life"),
  n = NULL,
  h = 0,
  n_guar = NULL,
  timing = c("immediate", "due"),
 i_type = c("effective", "nominal_interest", "nominal_discount", "force", "nominal"),
  m = 1,
  col_K = "Kx",
  col_T = "Tx",
  col_pv = "pv_annuity",
  ...
)
```

Arguments

.data	A data frame or tibble containing simulated future lifetimes, typically returned by simulate_lifetime or by mc_multilife_status when working with multiple-life statuses. For life-contingent calculations, observed curtate lifetimes must be finite nonnegative integers and observed complete lifetimes finite and nonnegative. Paired observations must satisfy $K_x \leq T_x < K_x + 1$.
i	Numeric scalar. Interest-rate input used for discounting.
payment	Numeric scalar. Amount paid at each annuity payment date. Default is 1. This is a per-payment amount, not an annualized payment rate.
k	Positive integer. Number of annuity payments per year. Default is 1, corresponding to annual payments.
type	Character string specifying the annuity type. Canonical options are "whole", "temporary", "deferred", "deferred_temporary", "certain", and "guaranteed". The transitional alias "whole_life" is also accepted and mapped to "whole".
n	Numeric scalar. Term of the annuity in years. Required for "temporary", "deferred_temporary", and "certain" annuities.

h	Numeric scalar. Deferral period in years. Default is 0. Required to be positive for "deferred" and "deferred_temporary" annuities. For fractional payments, the calendar is anchored at h, which need not be a multiple of 1 / k.
n_guar	Numeric scalar. Guaranteed payment period in years. Required for type = "guaranteed".
timing	Character string specifying the annuity payment timing. Available options are "immediate" and "due".
i_type	Character string specifying the interest-rate convention. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force". The transitional value "nominal" is accepted and treated as "nominal_interest".
m	Positive integer. Number of interest conversion periods per year for nominal annual rates. Default is 1. This argument controls the interest-rate conversion frequency only. It does not represent annuity payment frequency.
col_K	Character string. Name of the column containing simulated curtate future lifetimes. Default is "Kx".
col_T	Character string. Name of the column containing simulated complete future lifetimes. Default is "Tx". This column is required when k > 1, except for annuities certain.
col_pv	Character string. Name of the output column containing simulated present values of annuity payments. Default is "pv_annuity".
...	Transitional compatibility for older calls using data, rate, payments_per_year, annuity, term, deferral_years, guarantee_years, interest_type, k_col, tx_col, and annuity_col.

Details

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

This function follows the compact actuarial notation used throughout tidyactuarial: i is the interest-rate input, i_type is the interest-rate convention, m is the nominal conversion frequency, k is the annuity payment frequency, n is the annuity term, and h is the deferral period.

The arguments m and k have different meanings:

- m is used only for nominal interest-rate conversion.
- k controls how frequently annuity payments are made.

The argument payment represents the amount of each annuity payment. Thus, for a monthly annuity with total annual payment equal to 1, use payment = 1 / 12 and k = 12. The output makes the units explicit through payment_per_payment = payment and payment_annualized = k * payment.

For annual payments, k = 1, the function works directly with K_x. For fractional payments, such as monthly, quarterly, or semiannual payments, the function uses T_x to determine whether the life is alive at each fractional payment time. Missing lifetimes produce missing life-contingent present values and payment counts; certain annuities remain independent of lifetime data. Fractional due payments use dates strictly before death; fractional immediate payments include a date exactly

equal to the complete lifetime. Guaranteed payments remain payable throughout the guarantee period. Discounting retains the log accumulation rate from the original interest convention and combines monetary amounts with discount factors before losing their numerical range. At extreme rates, the displayed effective rate or annual discount factor may be infinite or rounded to a boundary; these metadata are not used to value the payments.

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

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

where c is the amount of each payment.

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

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

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

Value

A tibble with the original simulation columns and additional columns:

i Original interest-rate input.

i_type Interest-rate convention.

m Interest conversion frequency.

i_effective Equivalent annual effective interest rate.

v Annual discount factor.

type Canonical annuity type.

payment Legacy-compatible amount of each annuity payment.

payment_per_payment Amount paid at each payment date.

payment_annualized Payment amount multiplied by the number of payments per year.

k Annuity payment frequency.

n Annuity term, if applicable.

h Deferral period.

n_guar Guaranteed period, if applicable.

timing Annuity payment timing.

n_payments Number of payments made in the simulated scenario.

first_payment_time First payment time in the simulated scenario.

last_payment_time Last payment time in the simulated scenario.

pv_annuity Simulated present value of annuity payments, or another name supplied through `col_pv`.

For transition, the output also includes legacy columns such as `rate`, `interest_type`, `effective_rate`, `discount_factor`, `annuity`, `payments_per_year`, `term`, `deferral_years`, and `guarantee_years`.

References

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

See Also

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

Other monte-carlo: mc_insurance(), mc_loss(), mc_multilife_status(), mc_premium(), mc_reserve(), simulate_lifetime(), simulate_lifetimes(), summary_mc()

Examples

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

# Annual whole-life annuity-due
lt |>
  simulate_lifetime(
    x = 40,
    n_sim = 25,
    seed = 123
  ) |>
  mc_annuity(
    i = 0.05,
    type = "whole",
    payment = 1,
    k = 1,
    timing = "due"
  )

# Monthly whole-life annuity-due with total annual payment equal to 1
lt |>
  simulate_lifetime(
    x = 40,
    n_sim = 25,
    frac = "udd",
    seed = 123
  ) |>
  mc_annuity(
    i = 0.05,
    type = "whole",
    payment = 1 / 12,
    k = 12,
    timing = "due"
  )

# Quarterly temporary life annuity-immediate
lt |>
```

```
  simulate_lifetime(
    x = 40,
    n_sim = 25,
    frac = "udd",
    seed = 123
  ) |>
  mc_annuity(
    i = 0.05,
    type = "temporary",
    n = 20,
    payment = 1 / 4,
    k = 4,
    timing = "immediate"
  )

# Nominal interest convertible monthly, with quarterly payments
lt |>
  simulate_lifetime(
    x = 40,
    n_sim = 25,
    frac = "udd",
    seed = 123
  ) |>
  mc_annuity(
    i = 0.06,
    i_type = "nominal_interest",
    m = 12,
    type = "whole",
    payment = 1 / 4,
    k = 4,
    timing = "due"
  )

# Multiple-life status workflow
lt |>
  simulate_lifetimes(
    x = c(60, 58),
    n_sim = 25,
    frac = "udd",
    seed = 123
  ) |>
  mc_multilife_status(status = "joint") |>
  mc_annuity(
    i = 0.04,
    type = "whole",
    payment = 1,
    k = 1,
    timing = "due",
    col_K = "K_status",
    col_T = "T_status"
  )
```

`mc_insurance` *Compute simulated present values for life insurance benefits*

Description

Computes Monte Carlo simulated present values of life insurance benefits from simulated future lifetimes, using compact actuarial notation.

Usage

```
mc_insurance(
  .data = NULL,
  i = NULL,
  benefit = 1,
  type = c("whole", "term", "deferred", "deferred_term", "pure_endowment", "endowment",
    "whole_life", "deferred_temporary"),
  n = NULL,
  h = 0,
  timing = c("end_of_year", "moment_of_death"),
  i_type = c("effective", "nominal_interest", "nominal_discount", "force", "nominal"),
  m = 1,
  col_K = "Kx",
  col_T = "Tx",
  col_pv = "pv_benefit",
  ...
)
```

Arguments

`.data`	A data frame or tibble containing simulated future lifetimes, typically returned by `simulate_lifetime` or by `mc_multilife_status` when working with multiple-life statuses. Observed curtate lifetimes must be finite nonnegative integers and observed complete lifetimes finite and nonnegative. Paired observations must satisfy $K_x \leq T_x < K_x + 1$; missing values remain missing.
`i`	Numeric scalar. Interest-rate input used for discounting.
`benefit`	Numeric scalar. Monetary amount paid when the insured benefit is triggered. Default is `1`. Unlike an annuity payment, this is a single benefit amount and has no annualized-versus-per-payment interpretation.
`type`	Character string specifying the insurance type. Canonical options are `"whole"`, `"term"`, `"deferred"`, `"deferred_term"`, `"pure_endowment"`, and `"endowment"`. Transitional aliases `"whole_life"` and `"deferred_temporary"` are also accepted.
`n`	Numeric scalar. Insurance term in years. Required for `"term"`, `"deferred_term"`, `"pure_endowment"`, and `"endowment"`.
`h`	Numeric scalar. Deferral period in years. Default is `0`. Required to be positive for `"deferred"` and `"deferred_term"` insurance.

timing	Character string specifying when death benefits are paid. Available options are "end_of_year" and "moment_of_death". Default is "end_of_year".
i_type	Character string specifying the interest-rate convention. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force". The transitional value "nominal" is accepted and treated as "nominal_interest".
m	Positive integer. Number of interest conversion periods per year for nominal annual rates. Default is 1. This argument controls the interest-rate conversion frequency only. It does not represent benefit frequency or premium payment frequency.
col_K	Character string. Name of the column containing simulated curtate future lifetimes. Default is "Kx".
col_T	Character string. Name of the column containing simulated complete future lifetimes. Required when timing = "moment_of_death". Default is "Tx".
col_pv	Character string. Name of the output column containing simulated present values of benefits. Default is "pv_benefit".
...	Transitional compatibility for older calls using data, rate, insurance, term, deferral_years, payment_timing, interest_type, k_col, tx_col, and benefit_col.

Details

This function is designed to be used after simulate_lifetime, simulate_lifetimes, or mc_multilife_status. It takes simulated values of the curtate future lifetime K_x, and when needed the complete future lifetime T_x, and evaluates the present value random variable associated with classical life insurance benefits.

Discounting uses the log accumulation rate from the original interest convention and combines the benefit amount with the discount factor. The displayed annual effective rate and discount factor may lose range at extreme rates; they are metadata rather than the valuation basis. A zero benefit remains zero for an observed eligible event, while a missing required lifetime continues to give a missing present value.

This function follows the compact actuarial notation used throughout tidyactuarial: i is the interest-rate input, i_type is the interest-rate convention, m is the nominal conversion frequency, n is the insurance term, and h is the deferral period.

The arguments m and col_K have deliberately different roles:

- m is used only for nominal interest-rate conversion.
- col_K identifies the simulated curtate future lifetime column.

If timing = "end_of_year", death benefits are discounted using $K_x + 1$. Under this discrete convention, finite terms and deferral periods must be whole numbers of years because eligibility is determined from the curtate lifetime K_x. If timing = "moment_of_death", death benefits are discounted using T_x, and fractional terms or deferral periods are allowed.

The following insurance types are supported:

- "whole": benefit is paid whenever death occurs.
- "term": benefit is paid if death occurs within n years.
- "deferred": benefit is paid if death occurs after the deferral period h.

- "deferred_term": benefit is paid if death occurs after h and within the following n years.
- "pure_endowment": benefit is paid at time n if the life survives to that time.
- "endowment": death benefit is paid if death occurs within n years; otherwise, a survival benefit is paid at time n.

With timing = "moment_of_death", death at exactly n belongs to the term component. A pure endowment requires $T_x > n$, so term and pure-endowment benefits are disjoint and sum to the endowment benefit on every observed trajectory. The annual convention continues to partition the events using $K_x < n$ and $K_x \geq n$.

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

Value

A tibble with the original simulation columns and additional columns:

i Original interest-rate input.

i_type Interest-rate convention.

m Interest conversion frequency.

i_effective Equivalent annual effective interest rate.

v Annual discount factor.

type Canonical insurance type.

benefit Benefit amount.

n Insurance term, if applicable.

h Deferral period.

timing Timing used for death benefits.

benefit_time Simulated payment time of the benefit.

benefit_indicator Indicator that the benefit is paid.

pv_benefit Simulated present value of the benefit, or another name supplied through col_pv.

For transition, the output also includes legacy columns such as rate, interest_type, effective_rate, discount_factor, insurance, term, deferral_years, and payment_timing.

References

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

See Also

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

Other monte-carlo: mc_annuity(), mc_loss(), mc_multilife_status(), mc_premium(), mc_reserve(), simulate_lifetime(), simulate_lifetimes(), summary_mc()

Examples

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

# Whole-life insurance payable at the end of the year of death
lt |>
  simulate_lifetime(
    x = 40,
    n_sim = 25,
    seed = 123
  ) |>
  mc_insurance(
    i = 0.05,
    type = "whole",
    benefit = 1
  )

# 20-year term insurance
lt |>
  simulate_lifetime(
    x = 40,
    n_sim = 25,
    seed = 123
  ) |>
  mc_insurance(
    i = 0.05,
    type = "term",
    n = 20,
    benefit = 100000
  )

# 10-year deferred whole-life insurance
lt |>
  simulate_lifetime(
    x = 40,
    n_sim = 25,
    seed = 123
  ) |>
  mc_insurance(
    i = 0.05,
    type = "deferred",
    h = 10,
    benefit = 1
  )

# Endowment insurance payable at the moment of death if death occurs
lt |>
  simulate_lifetime(
    x = 40,
    n_sim = 25,
```

```
    frac = "udd",
    seed = 123
  ) |>
  mc_insurance(
    i = 0.05,
    type = "endowment",
    n = 20,
    timing = "moment_of_death",
    benefit = 1
  )

# Nominal interest rate convertible monthly
lt |>
  simulate_lifetime(
    x = 40,
    n_sim = 25,
    seed = 123
  ) |>
  mc_insurance(
    i = 0.06,
    i_type = "nominal_interest",
    m = 12,
    type = "whole",
    benefit = 1
  )

# First-death insurance using a multiple-life status
lt |>
  simulate_lifetimes(
    x = c(60, 58),
    n_sim = 25,
    frac = "udd",
    seed = 123
  ) |>
  mc_multilife_status(status = "joint") |>
  mc_insurance(
    i = 0.04,
    type = "whole",
    benefit = 100000,
    col_K = "K_status",
    col_T = "T_status"
  )
```

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

Description

Constructs simulated actuarial losses from present values of benefits and premium-payment annuities while keeping annualized premiums distinct from amounts paid at each premium date.

Usage

```
mc_loss(
  .data = NULL,
  col_Z = "pv_benefit",
  col_Y = "pv_annuity",
  col_P = "P",
  col_L = "L",
  P = NULL,
  premium_unit = c("auto", "annualized", "per_payment", "annuity_scale"),
  k = NULL,
  annuity_payment = NULL,
  tol = 1e-10,
  ...
)
```

Arguments

.data	A data frame or tibble containing simulated present values of benefits and premium annuities.
col_Z	Character scalar naming the simulated present value of benefits. Default is "pv_benefit".
col_Y	Character scalar naming the simulated present value of the premium annuity. Default is "pv_annuity".
col_P	Character scalar naming the premium input column. When omitted, the function searches, in order, for premium_annualized, premium_per_payment, P, and premium.
col_L	Character scalar naming the simulated loss output. Default is "L".
P	Optional nonnegative numeric scalar supplied directly instead of reading a premium column.
premium_unit	Unit of P or the selected premium column: "annualized", "per_payment", or "annuity_scale". The latter is the coefficient that directly multiplies col_Y. With "auto", explicit modern column names determine the unit, while P, premium, and a direct P argument retain their historical annuity-scale interpretation.
k	Optional positive integer number of premium payments per year. By default it is inferred from payments_per_year or k in .data; if neither exists, k = 1 is used.
annuity_payment	Optional positive numeric scalar giving the payment amount used to construct col_Y. It is normally inferred from payment_per_payment, payment, or payment_annualized in the output of mc_annuity.
tol	Nonnegative numeric tolerance for consistency checks.
...	Transitional compatibility for older calls using data, benefit_col, annuity_col, premium_col, loss_col, and premium.

Details

Let Z be the present value random variable of benefits. Suppose `col_Y` contains

$$Y_c = c \sum_j v^{t_j},$$

where c is the amount used in `mc_annuity` at each of the k premium dates per year.

If $P^{(k)}$ is the annualized premium, the actual installment is $P^{(k)}/k$, and the simulated present value of premiums is

$$\Pi = \frac{P^{(k)}}{kc} Y_c.$$

Hence the loss at issue is

$$L = Z - \Pi.$$

The recommended normalized premium annuity uses `payment = 1 / k` in `mc_annuity`. Then $c = 1/k$, $Y_c = Y^{(k)}$, and the expression simplifies to

$$L = Z - P^{(k)} Y^{(k)}.$$

This convention agrees with `premium_x`, `premium_xy`, `reserve_x`, and `reserve_xy`: the premium is annualized, while the amount collected at each date is the annualized premium divided by `k`.

For backward compatibility, legacy columns `P` and `premium` are interpreted as coefficients that multiply the supplied annuity present value directly. If such a coefficient is q, then

$$P_{\text{per payment}} = qc, \qquad P^{(k)} = kqc.$$

For annualized or per-payment inputs, monetary income retains the original premium amount and frequency until it is combined with the annuity present value. This avoids losing a representable income when an intermediate annuity-scale coefficient underflows or overflows. Displayed converted amounts may still be zero or infinite if they themselves do not fit in double precision. An explicitly supplied annuity-scale coefficient is used as supplied; a rounded zero or nonfinite coefficient cannot reconstruct information that has already been lost. Select a representable monetary premium column when passing extreme results from `mc_premium`. Positive infinite annualized annuity-basis metadata are accepted only when they agree with the product of the finite per-payment basis and frequency; they cannot be used alone to infer the payment basis.

Value

A data frame or tibble containing the original simulation and standardized columns:

premium_annualized Annualized premium $P^{(k)}$.

premium_per_payment Amount collected at each premium date.

payments_per_year Premium payment frequency `k`.

premium_annuity_scale Coefficient multiplying `col_Y`.

pv_premiums Simulated present value of premium income.

L Simulated loss, or the name supplied through `col_L`.

A compatibility column `loss` is synchronized with `col_L`.

References

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

See Also

mc_annuity, mc_insurance, mc_premium, mc_reserve, premium_x, premium_xy

Other monte-carlo: mc_annuity(), mc_insurance(), mc_multilife_status(), mc_premium(), mc_reserve(), simulate_lifetime(), simulate_lifetimes(), summary_mc()

Examples

```
simulated <- tibble::tibble(
  pv_benefit = c(900, 700, 1100),
  pv_annuity = c(8, 7, 9),
  payment_per_payment = 1 / 12,
  payment_annualized = 1,
  payments_per_year = 12,
  premium_annualized = 120
)

simulated |>
  mc_loss()

# A premium supplied directly as an annualized amount
simulated |>
  mc_loss(
    P = 120,
    premium_unit = "annualized"
  )

# Historical coefficient multiplying the annuity PV directly
tibble::tibble(
  pv_benefit = c(900, 700),
  pv_annuity = c(8, 7),
  P = 100
) |>
  mc_loss()
```

---

mc_multilife_status *Compute multiple-life simulated status variables*

Description

Combines simulated future lifetimes for several lives into one simulated multiple-life status. A joint-life status terminates at the first death, whereas a last-survivor status terminates at the last death.

Usage

```
mc_multilife_status(
  data,
  status = c("joint", "last_survivor"),
  col_sim = "sim_id",
  col_life = "life_id",
  col_K = "Kx",
  col_T = "Tx",
  tol = 1e-10
)
```

Arguments

`data`	A data frame or tibble, typically returned by `simulate_lifetimes`.
`status`	Multiple-life status. Canonical values are `"joint"` and `"last_survivor"`. The aliases `"joint_life"`, `"last"`, and `"last_survivor_life"` are accepted.
`col_sim`	Character scalar naming the simulation identifier column. It must not collide with the output fields `K_status`, `T_status`, `n_lives`, or `status`.
`col_life`	Character scalar naming the life identifier column. The default is `"life_id"`. Supply `NULL` only when no life identifier exists; in that case the function can validate equal row counts across simulations but cannot detect duplicated or substituted lives.
`col_K`	Character scalar naming the curtate future lifetime column.
`col_T`	Character scalar naming the complete future lifetime column.
`tol`	Nonnegative numeric tolerance for checking whole-number curtate lifetimes. It does not relax the interval $K \leq T < K + 1$.

Details

For simulated complete future lifetimes $T_1, \ldots, T_r$, the multiple-life complete future lifetime is

$$T_{\text{joint}} = \min(T_1, \ldots, T_r)$$

for the joint-life status and

$$T_{\text{last}} = \max(T_1, \ldots, T_r)$$

for the last-survivor status.

The corresponding curtate future lifetime is obtained with the same minimum or maximum operation on $K_1, \ldots, K_r$. Because the function validates $K_j \leq T_j < K_j + 1$, the returned values satisfy the same curtate-complete relationship. Each complete lifetime must satisfy this interval exactly after the existing whole-number check and rounding of `K`. Incompatible complete lifetimes are rejected, not rounded or repaired. Ordinary curtate outputs use integer storage; larger values retain numeric storage.

When `col_life` is available, every simulation must contain exactly one row for every life and the same set of lives must appear in every simulation. This prevents a missing or duplicated life from being silently treated as a different multiple-life contract.

Value

A tibble with one row per simulation and:

sim_id Standard simulation identifier.

sim Compatibility alias for the simulation identifier.

K_status Curtate future lifetime of the status.

T_status Complete future lifetime of the status.

n_lives Number of lives represented in each simulation.

status Canonical status: `"joint"` or `"last_survivor"`.

If `col_sim` is neither `"sim_id"` nor `"sim"`, that original identifier column is also retained.

See Also

`simulate_lifetimes`, `mc_insurance`, `mc_annuity`, `mc_reserve`

Other monte-carlo: `mc_annuity()`, `mc_insurance()`, `mc_loss()`, `mc_premium()`, `mc_reserve()`, `simulate_lifetime()`, `simulate_lifetimes()`, `summary_mc()`

Examples

```
simulated <- tibble::tibble(
  sim_id = rep(1:2, each = 2),
  life_id = rep(c("x", "y"), times = 2),
  Kx = c(2, 5, 4, 1),
  Tx = c(2.4, 5.2, 4.5, 1.3)
)

simulated |>
  mc_multilife_status(status = "joint")

simulated |>
  mc_multilife_status(status = "last_survivor")
```

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

Description

Estimates net premiums from simulated present values of benefits and premium annuities, while distinguishing the annualized premium from the amount paid at each premium date.

Usage

```
mc_premium(
  .data = NULL,
  col_Z = "pv_benefit",
  col_Y = "pv_annuity",
  col_P = "P",
  by = NULL,
  na_rm = TRUE,
  k = NULL,
  annuity_payment = NULL,
  tol = 1e-10,
  ...
)
```

Arguments

.data	A data frame or tibble containing simulated present values. Usually obtained after applying mc_insurance and mc_annuity to the same simulated lifetime sample.
col_Z	Character scalar naming the simulated present value of benefits. Default is "pv_benefit".
col_Y	Character scalar naming the simulated present value of the premium annuity. Default is "pv_annuity".
col_P	Character scalar naming the compatibility output containing the coefficient that directly multiplies col_Y. Default is "P". For a normalized k-thly premium annuity constructed with payment = 1 / k, this coefficient is the annualized premium.
by	Optional character vector of grouping columns. If supplied, the premium is estimated separately within each group. If by = NULL and .data is already grouped, the current grouping is used.
na_rm	Logical scalar. If TRUE, simulations with a missing benefit or annuity present value are removed as complete pairs. The same simulation rows are therefore used in both means.
k	Optional positive integer number of premium payments per year. By default it is inferred from payments_per_year or k in .data; if neither exists, k = 1 is used.
annuity_payment	Optional positive numeric scalar giving the amount paid at each date when col_Y was constructed. It is normally inferred from payment_per_payment, payment, or payment_annualized in the output of mc_annuity.
tol	Nonnegative numeric tolerance for consistency checks.
...	Transitional compatibility for older calls using data, benefit_col, annuity_col, and premium_col.

Details

Let Z denote the simulated present value of benefits. Suppose the simulated premium annuity is

$$Y_c = c \sum_j v^{t_j},$$

where c is the amount used at each of the k premium dates per year. The Monte Carlo coefficient that directly multiplies this annuity is

$$\widehat{q} = \frac{\overline{Z}}{\overline{Y_c}}.$$

The corresponding premium amount at each payment date is

$$\widehat{P}_{\text{per payment}} = \widehat{q}c,$$

and the annualized premium is

$$\widehat{P}^{(k)} = k\widehat{q}c.$$

The recommended premium-annuity normalization uses `payment = 1 / k` in `mc_annuity`. In that case,

$$\widehat{q} = \widehat{P}^{(k)}, \qquad \widehat{P}_{\text{per payment}} = \frac{\widehat{P}^{(k)}}{k}.$$

If instead `payment = 1`, then $\widehat{q}$ is the amount paid at each premium date, while the annualized premium is $k\widehat{q}$.

When `na_rm = TRUE`, missing values are removed jointly: a simulation contributes only if both `col_Z` and `col_Y` are observed. This prevents the numerator and denominator from being estimated from different simulated samples. Monetary premium amounts retain the ratio before intermediate overflow or underflow. A displayed annuity-scale coefficient may still be zero or infinite when the coefficient itself is outside double precision, while the per-payment and annualized amounts remain representable. The equivalence residual uses the ratio before that loss of range. Positive subnormal observations are retained using scaled means when an ordinary mean rounds to zero. A genuinely zero annuity exposure is rejected. A positive infinite `payment_annualized` metadata value is accepted only when it agrees with the product of a finite payment basis and finite frequency. The payment basis cannot be inferred from that infinite value alone; retain `payment_per_payment` or supply `annuity_payment`.

Value

A data frame or tibble containing the original simulations and standardized premium columns:

premium_annualized Estimated annualized premium $\widehat{P}^{(k)}$.

premium_per_payment Amount collected at each premium date.

payments_per_year Premium payment frequency `k`.

premium_annuity_scale Coefficient $\widehat{q}$ multiplying `col_Y`.

mc_equivalence_residual Difference $\overline{Z} - \widehat{q}\overline{Y_c}$ within the estimation group.

The column selected by `col_P` stores `premium_annuity_scale` for backward compatibility. Unless `col_P = "premium"`, a compatibility column `premium` is also created with the same scale coefficient. Existing grouping is used for estimation and removed from the result.

References

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

See Also

mc_insurance, mc_annuity, mc_loss, mc_reserve, premium_x, premium_xy

Other monte-carlo: mc_annuity(), mc_insurance(), mc_loss(), mc_multilife_status(), mc_reserve(), simulate_lifetime(), simulate_lifetimes(), summary_mc()

Examples

```
simulated <- tibble::tibble(
  pv_benefit = c(800, 1200),
  pv_annuity = c(8, 12),
  payment_per_payment = 1 / 12,
  payment_annualized = 1,
  payments_per_year = 12
)

simulated |>
  mc_premium()

# Here P is annualized because the annuity uses payment = 1 / 12.

# With payment = 1 at each monthly date, P is the monthly amount,
# while premium_annualized is twelve times P.
simulated_unit_payments <- tibble::tibble(
  pv_benefit = c(800, 1200),
  pv_annuity = c(96, 144),
  payment_per_payment = 1,
  payment_annualized = 12,
  payments_per_year = 12
)

simulated_unit_payments |>
  mc_premium()
```

---

mc_reserve *Compute Monte Carlo prospective reserves for life contingencies*

Description

Computes simulated prospective loss random variables at one or more policy durations. Future benefits and future premiums are revalued from each requested duration.

Usage

```
mc_reserve(
  .data = NULL,
  t = 0,
  i = NULL,
  P = NULL,
  col_P = "P",
  premium_unit = c("auto", "annualized", "per_payment", "annuity_scale"),
  benefit = 1,
  payment = 1,
  k = 1L,
 type = c("whole", "term", "deferred", "deferred_term", "pure_endowment", "endowment",
    "whole_life", "deferred_temporary"),
 annuity_type = c("whole", "temporary", "deferred", "deferred_temporary", "certain",
    "guaranteed", "whole_life"),
  n = NULL,
  h = 0,
  n_guar = NULL,
  timing = c("end_of_year", "moment_of_death"),
  premium_timing = c("due", "immediate"),
  reserve_timing = c("before_payment", "after_payment"),
 i_type = c("effective", "nominal_interest", "nominal_discount", "force", "nominal"),
  m = 1,
  col_K = "Kx",
  col_T = "Tx",
  in_force_basis = c("auto", "complete", "curtate"),
  not_in_force = c("na", "zero"),
  col_L = "L_t",
  tol = 1e-10,
  premium = NULL,
  ...
)
```

Arguments

`.data`	A data frame or tibble containing simulated curtate future lifetimes and, when required, complete future lifetimes. Observed curtate lifetimes must be finite nonnegative integers and observed complete lifetimes finite and nonnegative. Paired observations must satisfy $K_x \leq T_x < K_x + 1$; missing values remain missing.
`t`	Nonnegative numeric vector of valuation durations.
`i`	Numeric scalar. Interest-rate input used for discounting.
`P`	Optional nonnegative numeric scalar supplied directly as the premium input.
`col_P`	Character scalar naming a premium column in `.data`. The default is `"P"`. When it is not supplied explicitly, modern standardized premium columns are preferred.

premium_unit	Unit of P or the selected premium column: "annualized", "per_payment", or "annuity_scale". With "auto", premium_annualized, premium_per_payment, and premium_annuity_scale determine their own units, while legacy P and premium retain their historical annuity-scale interpretation.
benefit	Nonnegative numeric scalar. Insurance benefit amount.
payment	Positive numeric scalar. Amount used at each premium-annuity payment date. For the normalized k-thly convention, use payment = 1 / k.
k	Positive integer number of premium payments per year.
type	Insurance type. Supported values are "whole", "term", "deferred", "deferred_term", "pure_endowment", and "endowment".
annuity_type	Premium-annuity type. Supported values are "whole", "temporary", "deferred", "deferred_temporary", "certain", and "guaranteed".
n	Optional contract or premium term in years.
h	Nonnegative deferral period in years.
n_guar	Optional guaranteed premium-payment period in years.
timing	Death-benefit timing: "end_of_year" or "moment_of_death".
premium_timing	Premium-annuity timing: "due" or "immediate".
reserve_timing	Whether cash flows exactly at the valuation duration are included: "before_payment" or "after_payment".
i_type	Interest-rate convention.
m	Positive integer nominal conversion frequency.
col_K	Curtate future lifetime column.
col_T	Complete future lifetime column.
in_force_basis	Basis used to determine whether a simulated policy is in force: "auto", "complete", or "curtate".
not_in_force	Output for known scenarios that are not in force: "na" or "zero". Missing lifetime information remains missing under either option.
col_L	Output column containing the prospective loss.
tol	Nonnegative numeric tolerance for consistency checks.
premium	Deprecated explicit alias for P. It remains a formal argument because otherwise R partially matches it against premium_unit and premium_timing.
...	Transitional compatibility for older argument names.

Details

Let Z_t be the present value at duration t of future benefits. Suppose the future premium annuity is constructed with amount c at each of the k payment dates per year:

$$Y_{t,c} = c \sum_{j:t_j \geq t} v^{t_j - t}.$$

If $P^{(k)}$ is the annualized premium, then the installment is $P^{(k)}/k$ and the coefficient multiplying $Y_{t,c}$ is

$$q = \frac{P^{(k)}}{kc}.$$

Therefore the simulated prospective loss is

$$L_t = Z_t - qY_{t,c}.$$

Under the recommended normalization `payment = 1 / k`, $q = P^{(k)}$ and

$$L_t = Z_t - P^{(k)}Y_t^{(k)}.$$

If no premium is supplied, the function estimates the issue scale coefficient from complete benefit-annuity simulation pairs:

$$\widehat{q} = \frac{\overline{Z}_0}{\overline{Y}_{0,c}}.$$

This makes the sample mean issue loss equal to zero, up to numerical error, when `reserve_timing = "before_payment"`. In this case `premium_unit` does not reinterpret the estimated coefficient: the selected unit only describes a supplied premium input. Missing issue annuities are excluded jointly with missing benefits.

The column `Y_t` is the present value of the premium-annuity basis, including the amount supplied through `payment`. The column `future_pv_premiums` is the actual simulated present value of premium income, equal to `premium_annuity_scale * Y_t`. This identity is mathematical: the actual cash flows are valued directly using the original log interest rate, so a rounded or overflowing displayed coefficient or annuity basis need not destroy representable premium income. A supplied annuity-scale coefficient is interpreted as received; lost information is not inferred from an explicitly rounded coefficient.

For `type = "pure_endowment"` and `timing = "moment_of_death"`, survival requires $T_x > n$. Death at exactly `n` belongs to the term component, so term and pure-endowment benefits form a disjoint partition of the endowment. The annual convention retains $K_x < n$ and $K_x \geq n$.

These are scenario-specific prospective losses, not conditional expected reserves. The existing in-force filter takes precedence over payment-date inclusion: complete lifetimes require $T_x > t$, and curtate lifetimes require $K_x \geq t$. Thus a complete lifetime dying exactly at `t` is not in force even for a before-payment calculation. The chosen `not_in_force` policy applies afterwards; unknown lifetimes stay missing. Known scenarios without a covered benefit have zero benefit value.

At extreme rates, effective-rate and discount-factor metadata can lose range. The function retains the original rate and monetary amounts for valuation. Exactly coincident finite monetary flows can cancel even if their separate present values overflow. General cancellation between unrepresentable flows is not guaranteed. Internal premium estimation requires representable observed issue present values. Nonfinite converted premium metadata are accepted only when they match a conversion from the resolved premium; directly supplied premiums must remain finite. Positive infinite annualized annuity-basis metadata are likewise accepted only if they agree with the product of a finite payment basis and frequency. That infinite annualized value alone cannot identify the payment basis.

Value

A tibble with one row per simulation and requested duration. Important standardized columns include:

Z_t Future benefit present value at duration `t`.

Y_t Future premium-annuity basis present value.

future_pv_premiums Actual future premium present value.

premium_annualized Annualized premium $P^{(k)}$.

premium_per_payment Premium paid at each date.

premium_annuity_scale Coefficient multiplying `Y_t`.

L_t Prospective loss, or the name supplied through `col_L`.

See Also

`mc_annuity`, `mc_insurance`, `mc_premium`, `mc_loss`, `reserve_x`, `reserve_xy`

Other monte-carlo: `mc_annuity()`, `mc_insurance()`, `mc_loss()`, `mc_multilife_status()`, `mc_premium()`, `simulate_lifetime()`, `simulate_lifetimes()`, `summary_mc()`

Examples

```
simulated <- tibble::tibble(
  Kx = c(2, 4),
  Tx = c(2.4, 4.7)
)

simulated |>
  mc_reserve(
    t = c(0, 1),
    i = 0.05,
    type = "whole",
    annuity_type = "whole",
    payment = 1 / 12,
    k = 12
  )
```

---

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

Description

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

Usage

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

Arguments

qx_df	A data.frame/tibble with an age column (default x) and one or more cause columns containing annual probabilities in $[0, 1]$. Recommended naming convention: cause columns start with "q_" (e.g., q_death, q_disability).
age_col	Character. Name of the age column (default "x").
cause_cols	Character vector. Names of the cause columns. If NULL (default), all columns other than age_col are treated as causes. Cause names must not collide with reserved totals or generated death columns.
radix	Numeric. Starting cohort size at the first age (default 1e5).
close	Logical. If TRUE, requires $q_{\omega}^{(\tau)} = 1$ at the last age (default TRUE).
check	Logical. If TRUE, performs input validation (default TRUE).
tol	Numeric tolerance used in checks (default 1e-10).

Details

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

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

Value

A tibble with columns:

- x: integer ages.
- lx: cohort ℓ_x.
- q_total: total decrement probability $q_x^{(\tau)}$.
- p_total: total survival probability $p_x^{(\tau)}$.
- d_total: total decrements $d_x^{(\tau)} = \ell_x q_x^{(\tau)}$.
- cause columns (as provided).
- cause-specific decrements d_* with $d_x^{(j)} = \ell_x q_x^{(j)}$.

Examples

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

---

mortality_colombia_tables

Colombian mortality tables

Description

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

Usage

```
mortality_colombia_tables
```

Format

A tibble with 12 variables:

table_id Mortality table identifier.

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

x Integer actuarial age.

lx Number of survivors at exact age `x`.

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

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

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

mu_x Force of mortality at age `x`, when available.

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

source Source identifier or reference.

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

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

Details

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

The variable names follow the compact actuarial notation used throughout `tidyactuarial`: `x` denotes age, `lx` the number of lives, `dx` the number of deaths, `qx` the one-year death probability, `px` the one-year survival probability, `mu_x` the force of mortality, and `ex` the life expectancy at age `x`.

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

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

Source

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

Examples

```
data(mortality_colombia_tables)

head(mortality_colombia_tables)

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

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

head(rv08_male)
```

---

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

Description

Creates a tidy life table with one row per integer age from a parametric mortality law, using compact actuarial notation.

Usage

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

```
  x_max,
  ...,
  params = NULL,
  frac = c("CF", "UDD", "Balducci", "CML"),
  l0 = 1e+05,
  close = TRUE,
  a_x = 0.5,
  check = TRUE,
  tol = 1e-10,
  radix = NULL,
  ax = NULL
)
```

Arguments

law	Character. Mortality law. One of "Exponential", "Gompertz", "Makeham", "Weibull", "Logistic", "DeMoivre", "Beta", or "HeligmanPollard".
x_min	Integer. Minimum age, inclusive.
x_max	Integer. Maximum age, inclusive. Must satisfy x_min < x_max.
...	Named law parameters. These values override params. Placing ... before arguments such as close and check avoids partial-matching conflicts with the Gompertz and Makeham parameter c.
params	Named list of law parameters, or NULL.
frac	Character. Within-year interpolation label, independent of the annual probability calculation. One of "CF", "UDD", "Balducci", or "CML". "CML" is treated as "CF".
l0	Numeric scalar. Initial radix, that is, the starting value l_{x_min}.
close	Logical. If TRUE, forces the last age to close.
a_x	Numeric scalar. Average fraction of the year lived by those dying between age x and $x + 1$. Default is 0.5. Exposure uses $L_x = l_{x+1} + a_x d_x$; this is an approximation controlled by a_x, not an integration of continuous survival.
check	Logical. If TRUE, performs strict input validation.
tol	Numeric tolerance used in checks.
radix	Deprecated. Use l0.
ax	Deprecated. Use a_x.

Details

The output follows tidyactuarial conventions and includes columns such as x, qx, px, lx, dx, Lx, Tx, ex, and mx.

This function follows the compact actuarial notation used throughout tidyactuarial: x denotes age, l0 denotes the starting radix, a_x denotes the average fraction of the year lived by those dying during the age interval, qx denotes the one-year death probability, and px = 1 - qx. Ratios use their actual denominators: mx = dx / Lx and ex = Tx / lx. Positive deaths with zero exposure give infinite mx; extinct cohorts with zero numerators and denominators give NaN for these undefined ratios.

Heligman-Pollard follows the eight-parameter odds specification in Heligman and Pollard (1980), section 2.1, equation (1), doi:10.1017/S0020268100040257. Earlier seven-parameter calls must be respecified: the previous implementation omitted the childhood exponent and assigned different meanings to `C` through `G`.

The parameter `F_hp` is used for the Heligman-Pollard law instead of `F`, because `F` is historically associated with `FALSE` in R and should not be promoted in a CRAN-facing API. The old name `F` is still accepted as a transitional alias.

Value

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

Supported laws

- **Exponential**: $\mu_x = \lambda$.
- **Gompertz**: $\mu_x = Bc^x$.
- **Makeham**: $\mu_x = A + Bc^x$.
- **Weibull**: $\mu_x = (shape/scale)(x/scale)^{shape-1}$.
- **Logistic**: $\mu_x = (A + Bc^x)/(1 + Cc^x)$.
- **DeMoivre**: finite lifetime law with $q_x = 1/(\omega - x)$ for $x < \omega$.
- **Beta**: scaled lifetime model $X/\omega \sim Beta(\alpha, \beta)$.
- **HeligmanPollard**: eight-parameter odds model $q_x/p_x = A^{(x+B)^C} + D\exp(-E(\log x - \log F)^2) + GH^x$, with `F_hp` denoting F.

Law parameters

Parameters for the selected law are supplied either through `...` or through the named list `params`. Direct parameters supplied through `...` override values in `params`.

Required parameters:

- `"Exponential"`: `lambda`.
- `"Gompertz"`: `B`, `c`.
- `"Makeham"`: `A`, `B`, `c`.
- `"Weibull"`: `shape`, `scale`. Transitional aliases `k` and `lambda` are accepted.
- `"Logistic"`: `A`, `B`, `c`, `C`.
- `"DeMoivre"`: `omega`.
- `"Beta"`: `alpha`, `beta`, `omega`.
- `"HeligmanPollard"`: `A`, `B`, `C`, `D`, `E`, `F_hp`, `G`, and `H`. `A`, `B`, `D`, and `G` must be nonnegative; `C`, `E`, `F_hp`, and `H` must be positive. At least one of `A`, `D`, or `G` must be positive. Ages must be at least 1. Transitional alias `F` is accepted and mapped to `F_hp`.

Converting mu(x) to qx

For laws defined by a force of mortality μ_x, the one-year death probability is $q_x = 1 - \exp(-\int_x^{x+1} \mu(s)\, ds)$, independently of `frac`. The column `mu_x` contains the force evaluated at age x, not its annual integral. Exponential, Gompertz, Makeham and Weibull use analytic integrals; Logistic uses numerical integration with relative tolerance `1e-10`. The `frac` label records a within-year interpolation convention; it does not replace integration of the continuous law. Force products and Logistic ratios use logarithmic evaluation to avoid intermediate power overflow; Weibull annual increments use `log1p/expm1` instead of subtracting nearly equal cumulative hazards.

Direct qx laws

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

Closure

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

See Also

`lifetable`, `t_px`, `e_xy`

Examples

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

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

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

mortality_law_table("Weibull", 1, 110, shape = 2.5, scale = 90)

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

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

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

mortality_law_table(
  "HeligmanPollard",
  1, 110,
  A = 0.0002,
  B = 0.1,
```

```
  C = 0.3,
  D = 0.03,
  E = 10,
  F_hp = 20,
  G = 0.00005,
  H = 1.08
)

# Transitional compatibility with older names
mortality_law_table("Weibull", 1, 110, k = 2.5, lambda = 90)

mortality_law_table(
  "HeligmanPollard",
  1, 110,
  A = 0.0002,
  B = 0.1,
  C = 0.3,
  D = 0.03,
  E = 10,
  F = 20,
  G = 0.00005,
  H = 1.08
)
```

`mortality_world_sample_2015_2023`

World mortality sample panel, 2015–2023

Description

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

Usage

```
mortality_world_sample_2015_2023
```

Format

A tibble with 35,451 rows and 14 variables:

country Country name.

country_code Numeric ISO country code.

continent Continent.

region Geographic region.

year Calendar year, from 2015 to 2023.

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

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

x Integer actuarial age, from 0 to 100.

mx Central death rate at age x.

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

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

lx Number of survivors at exact age x, based on l0 = 100000.

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

source Data source.

Details

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

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

with $a_x = 0.5$. The last available age is closed by setting qx = 1. Survivors lx and expected deaths dx are then reconstructed recursively from l0 = 100000.

This dataset is a selected-country panel, not a complete world mortality database.

Source

United Nations, World Population Prospects 2024.

Examples

```
data(mortality_world_sample_2015_2023)

mortality_world_sample_2015_2023 |>
  dplyr::filter(country == "Colombia", sex == "both", x == 70) |>
  dplyr::select(year, pandemic_period, qx, lx)
```

---

mortality_world_sample_2023

World mortality sample, 2023

Description

A compact international sample of period life tables for selected countries in 2023. The dataset is intended for recent and simple examples involving central death rates, one-year death probabilities, survival probabilities, and life-table calculations.

Usage

```
mortality_world_sample_2023
```

Format

A tibble with 3,939 rows and 13 variables:

country Country name.

country_code Numeric ISO country code.

continent Continent.

region Geographic region.

year Calendar year.

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

x Integer actuarial age, from 0 to 100.

mx Central death rate at age `x`.

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

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

lx Number of survivors at exact age `x`, based on `l0 = 100000`.

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

source Data source.

Details

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

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

with $a_x = 0.5$. The last available age is closed by setting `qx = 1`. Survivors `lx` and expected deaths `dx` are then reconstructed recursively from `l0 = 100000`.

This dataset is a selected-country sample, not a complete world mortality database.

Source

United Nations, World Population Prospects 2024.

Examples

```
data(mortality_world_sample_2023)

mortality_world_sample_2023 |>
  dplyr::filter(country == "Colombia", sex == "both") |>
  dplyr::select(x, mx, qx, px, lx, dx)
```

`multiple_decrement_sample` *Sample multiple decrement probabilities*

Description

A small pedagogical annual multiple decrement dataset with three causes: death, disability, and withdrawal. It is intended for examples involving multiple decrement tables, total-decrement life tables, cause-specific decrement probabilities, and cause-specific insurance benefits.

A small pedagogical annual multiple decrement dataset with three causes: death, disability, and withdrawal. It is intended for examples involving multiple decrement tables, total-decrement life tables, cause-specific decrement probabilities, and cause-specific insurance benefits.

Usage

```
multiple_decrement_sample

multiple_decrement_sample
```

Format

A tibble with 7 rows and 6 variables:

x Integer actuarial age.

q_death One-year death decrement probability.

q_disability One-year disability decrement probability.

q_withdrawal One-year withdrawal decrement probability.

q_total Total one-year decrement probability.

p_total Total one-year survival probability.

A tibble with 7 rows and 6 variables:

x Integer actuarial age.

q_death One-year death decrement probability.

q_disability One-year disability decrement probability.

q_withdrawal One-year withdrawal decrement probability.

q_total Total one-year decrement probability.

p_total Total one-year survival probability.

Details

This dataset already follows the compact actuarial convention `x` for age and `q` for decrement probabilities.

This dataset already follows the compact actuarial convention `x` for age and `q` for decrement probabilities.

Source

Synthetic pedagogical data created for tidyactuarial examples.

Synthetic pedagogical data created for tidyactuarial examples.

Examples

```
data(multiple_decrement_sample)

multiple_decrement_sample |>
  dplyr::select(x, q_death, q_disability, q_withdrawal, q_total, p_total)
```


```
data(multiple_decrement_sample)

multiple_decrement_sample |>
  dplyr::select(x, q_death, q_disability, q_withdrawal, q_total, p_total)
```


---

par_rate *Par coupon rate implied by a spot yield curve*

Description

Computes the level annual coupon rate that makes a bond's time-zero dirty price equal to its face value, using the spot discount factors at each coupon date. Unlike `bond_ytm()`, it does not solve for a flat yield.

Usage

```
par_rate(
  .data = NULL,
  t = NULL,
  i = NULL,
  n = NULL,
  face = 100,
  k = 1L,
  R = NULL,
  i_type = "effective",
  m = 1L,
  col_t = "t",
  col_i = "i",
  tol = 1e-10
)
```

Arguments

`.data`	Optional data frame containing a spot curve as numeric columns, or multiple curves as list-columns. For pipe-friendly usage.

t	Numeric vector of curve maturities (years) when .data = NULL.
i	Numeric vector of observed spot rates when .data = NULL.
n	Bond maturity in years. Defaults to the maximum maturity of each curve; may also be a scalar or one value per curve.
face	Strictly positive face value; scalar or one value per curve.
k	Positive integer coupon frequency per year; scalar or one value per curve. Coupon dates are seq_len(n * k) / k.
R	Redemption amount, scalar or one value per curve; defaults to face.
i_type	Spot rate type ("effective", "nominal_interest", "nominal_discount", or "force"). Passed to discount_factor_spot(). Must have length one or one value per curve node.
m	Conversion frequency for nominal spot rates, passed to discount_factor_spot(). Must have length one or one per curve node.
col_t	Name of the spot maturity column in .data.
col_i	Name of the spot rate column in .data.
tol	Positive numerical tolerance for aligning coupon dates and curve maturities. Does not authorize interpolation.

Details

For a bond with face value $F > 0$, redemption R, term n, and k coupon payments per year, the price equation is

$$P = (Fc/k)\sum_{j=1}^{nk} v(j/k) + Rv(n).$$

Setting the dirty price $P = F$ and solving for the nominal annual coupon rate gives

$$c_{par} = k[F - Rv(n)]/[F\sum_{j=1}^{nk} v(j/k)].$$

When redemption equals face value, this reduces to

$$c_{par} = k[1 - v(n)]/\sum_{j=1}^{nk} v(j/k).$$

Spot inputs use annual effective rates by default, with $v(t) = (1 + i_t)^{-t}$. Other supported interest conventions are converted using discount_factor_spot(). The calculation uses only observed nodes: every coupon date must match one curve maturity; there is no interpolation or extrapolation. Small floating-point discrepancies within tol are accepted as identical dates.

Input can be (1) numeric vectors t and i, (2) a data frame with numeric t and i columns, such as bootstrap_spot_curve() output, or (3) a tibble with t and i list-columns, such as yield_curve() output. In the third form each row describes one curve, and the input columns are retained. n, face, k, and R may be scalars or have one value per curve. For a single curve, n = NULL uses the largest observed maturity. A coupon rate can be negative when the implied financial terms require it; the function does not impose a contractual zero floor.

This model assumes regular coupons paid in arrears, no stub periods, no accrued interest, and a deterministic redemption at maturity. Calculation separates the dimensionless rate from monetary amounts and uses the original log spot growth at maturity to retain small coupon rates. Final amounts and the positive coupon discount sum must be representable. `repricing_error` reports the floating-point price discrepancy; extreme cancellation, especially with negative coupons, can still limit precision.

Value

One tibble row per curve, containing the supplied curve columns and `n`, `face`, `k`, `R`, `annuity_factor`, `v_n`, `c_par`, `coupon_per_period`, `repriced_price`, and `repricing_error`. The `c_par` value is the nominal annual coupon rate as a proportion; `coupon_per_period` is its monetary coupon amount. `repricing_error` is the reconstructed dirty price minus the face value.

See Also

`bootstrap_spot_curve()`, `yield_curve()`, `bond_cash_flows()`, `bond_price()`, `discount_factor_spot()`

Other interest: `accumulation_factor()`, `bootstrap_spot_curve()`, `discount_factor()`, `discount_factor_spot()`, `forward_curve()`, `forward_rate()`, `interest_equivalents()`, `standardize_interest()`, `yield_curve()`

Examples

```
# An annual-pay bond whose spot curve was obtained from market prices.
curve <- tibble::tibble(
  t = c(1, 2, 3),
  i = c(0.04, 0.045, 0.05)
)
curve |> par_rate(face = 100, k = 1)

# A two-year bond with semiannual coupons: spot nodes every six months.
par_rate(
  t = c(0.5, 1, 1.5, 2),
  i = rep(0.04, 4),
  n = 2,
  k = 2
)

# A one-row curve returned by yield_curve() is also accepted.
yield_curve(t = 1:3, i = c(0.04, 0.045, 0.05)) |>
  par_rate()
```

`plot_cash_flow` *Plot a cash-flow diagram*

Description

Creates a professional cash-flow diagram with arrows representing inflows and outflows over time, using compact actuarial notation.

Usage

```
plot_cash_flow(
  .data = NULL,
  C,
  t = NULL,
  date = NULL,
  i = NULL,
  i_type = "effective",
  m = 1L,
  PV = NULL,
  payment = NULL,
  time = NULL,
  rate = NULL,
  pv = NULL,
  title = NULL,
  subtitle = NULL,
  x_label = NULL,
  amount_label = "Cash flow",
  financial = TRUE,
  normalize = FALSE,
  aggregate = TRUE,
  show_labels = TRUE,
  label_size = 3.5,
  arrow_size = 0.8,
  timeline_size = 0.7,
  label_digits = 2L,
  label_format = c("auto", "full", "compact"),
  currency = "",
  col_inflow = "#1B9E77",
  col_outflow = "#D95F02",
  date_labels = "%Y-%m-%d",
  day_count = c("act/365", "act/360"),
  ...
)
```

Arguments

.data	Optional data.frame or tibble containing cash-flow columns.
C	Numeric vector of cash flows, or a column name when .data is supplied.
t	Optional numeric vector of times in years, or a column name when .data is supplied.
date	Optional date vector, or a column name when .data is supplied.
i	Optional annual interest-rate input used to compute present value if PV is NULL.
i_type	Character string indicating the interest-rate type.
m	Positive integer. Conversion frequency for nominal rates.
PV	Optional numeric present value to display.

payment	Deprecated. Use C.
time	Deprecated. Use t. Kept explicit to avoid partial matching with timeline_size.
rate	Deprecated. Use i.
pv	Deprecated. Use PV.
title	Optional plot title.
subtitle	Optional plot subtitle.
x_label	Optional x-axis label.
amount_label	Optional y-axis label when normalize = FALSE.
financial	Logical. If TRUE, positive cash flows point upward and negative cash flows point downward.
normalize	Logical. If TRUE, arrow heights are normalized.
aggregate	Logical. If TRUE, cash flows occurring at the same time or date are summed before plotting.
show_labels	Logical. If TRUE, labels are shown next to cash-flow arrows.
label_size	Numeric text size for labels.
arrow_size	Numeric line width for arrows.
timeline_size	Numeric line width for the time axis.
label_digits	Integer between 0 and 50: decimal digits for cash-flow labels.
label_format	Character string controlling label notation. Use "auto" to abbreviate values from one million onward, "full" for unabridged values, or "compact" for abbreviated values such as 1.25M.
currency	Optional currency or unit prefix, such as $.
col_inflow	Character color for inflow arrows.
col_outflow	Character color for outflow arrows.
date_labels	Character date-label format passed to ggplot2::scale_x_date().
day_count	Day-count convention used when date is supplied.
...	Reserved for future extensions.

Details

Present values use the original interest specification and combine capital and discount factors stably. Opposite amounts with exactly equal finite log factors are netted if individual present values overflow. Numeric timelines may include negative times; the valuation origin remains zero. Dated timelines use the earliest date as origin. Non-finite inferred present values or aggregated amounts are rejected. Extreme monetary magnitudes use scaled graphical coordinates, with the power of ten stated on the axis; raw amounts remain in plot$data$C. Automatic/compact labels use scientific notation beyond the available suffixes or when fixed decimals would show a nonzero amount as zero. label_format = "full" retains explicit fixed-decimal formatting. Overlapping text labels may be omitted at rendering time; every arrow remains. For dense streams, use show_labels = FALSE. Simultaneous flows are netted by default; aggregate = FALSE keeps their exact positions, so arrows at identical times can overlap. No dates are jittered. PV is a supplied annotation when present, not a recalculation or check. Date origins and day counts differ from apv_life_flow() and are not silently unified. The returned ggplot remains editable with additional layers.

This function uses compact actuarial notation: C denotes cash-flow amounts, t denotes times in years, i denotes the interest-rate input, and PV denotes present value.

Value

A ggplot2 object.

See Also

pv_flow, fv_flow, irr_flow, standardize_interest

Other time-value: accumulation_factor(), discount_factor(), future_value(), fv_flow(), irr_flow(), irr_flow_multi(), present_value(), pv_flow(), solve_value()

Examples

```
plot_cash_flow(
  C = c(-1000, 300, 400, 500),
  t = c(0, 1, 2, 3),
  i = 0.08,
  currency = "$"
)

cashflows <- tibble::tibble(
  t = c(0, 1, 2, 3),
  C = c(-1000, 300, 400, 500)
)

cashflows |>
  plot_cash_flow(C = C, t = t, i = 0.08)

dated_flows <- tibble::tibble(
  date = as.Date(c("2026-01-01", "2026-07-01", "2027-01-01")),
  C = c(-1000, 450, 700)
)

dated_flows |>
  plot_cash_flow(C = C, date = date, i = 0.08)

plot_cash_flow(
  payment = c(-1000, 300, 400, 500),
  time = c(0, 1, 2, 3),
  rate = 0.08,
  currency = "$"
)
```

---

plot_contingent_cash_flow

Plot discrete life-contingent cash flows

Description

Separates contractual amounts, unconditional payment probabilities, expected cash flows and actuarial present-value contributions in editable panels. Consumes the payment-level output of `apv_life_flow()` without recomputing mortality, dates or interest, or an explicitly specified event table.

Usage

```
plot_contingent_cash_flow(
  .data,
  view = c("all", "contractual", "probability", "expected", "pv"),
  title = NULL,
  subtitle = NULL,
  x_label = NULL,
  show_labels = TRUE,
  label_size = 3.2,
  label_digits = 2L,
  label_format = c("auto", "full", "compact"),
  currency = "",
  col_inflow = "#1B9E77",
  col_outflow = "#D95F02",
  date_labels = "%Y-%m-%d"
)
```

Arguments

`.data`	A nonempty data frame or tibble. Supply an `apv_life_flow()` result, or columns `t`, `cf`, `probability`, `discount` and `event`. Optional `date` supplies the horizontal calendar axis. Optional `expected_cf` and `pv` are reused after consistency checks.
`view`	One of `"all"`, `"contractual"`, `"probability"`, `"expected"`, or `"pv"`. The default shows four panels with separate vertical scales.
`title, subtitle`	Optional plot title and subtitle.
`x_label`	Optional horizontal axis label.
`show_labels`	Logical. Show amount/probability labels, omitting text that overlaps at rendering time. All payment rows remain in the plot.
`label_size`	Positive text size.
`label_digits`	Integer between 0 and 50 for monetary labels.
`label_format`	Monetary notation: `"auto"`, `"full"`, or `"compact"`, as in `plot_cash_flow()`. Probabilities/weights use separate unitless labels.
`currency`	Monetary prefix; never applied to probabilities or weights.
`col_inflow, col_outflow`	Colors for positive and negative monetary flows.
`date_labels`	Calendar label format for `ggplot2::scale_x_date()`.

Details

For a binary payment event E_j, contractual signed amount C_j, payment date t_j and deterministic discount d_j, the panels show

$$C_j, \quad p_j = P(E_j), \quad C_j p_j, \quad C_j p_j d_j.$$

Probabilities are unconditional relative to the common initial actuarial state, not conditional annual mortality rates. For survival payments, $p_j = {}_{t_j}p_x$. For death in $(a, b]$, $p_j = {}_{a}p_x - {}_{b}p_x = {}_{a}p_x \, {}_{b-a}q_{x+a}$. A benefit payable at the end of that interval is discounted to its payment date, not the exact time of death. Continuous death-time benefits require a different model.

In explicit event tables, `event` contains `"survival"`, `"death_interval"` or `"certain"` (the last requires probability one). Interval probabilities must be supplied; the plot never differences arbitrary rows of a stream. Rows may represent different events at the same date; probabilities are never added or netted. Coincident stems can overlap; points identify their endpoints and no payment date is moved. Negative amounts denote outflows.

For new `apv_life_flow()` results, contingency metadata distinguishes binary survival from a reversionary expected benefit fraction. For the latter, `surv_prob` is $P(\text{all alive}) + \alpha P(\text{some but not all alive})$, not generally a probability: the second panel is labeled **Expected benefit fraction**. Results lacking metadata use the conservative label **Payment weight**; no event is inferred from column names alone. Explicit tables can use `event = "weighted"` for such finite multipliers as well.

`life_contract()` stores a specification, and `insurance_x()`/`annuity_x()` return aggregate values rather than payment schedules. These objects cannot be plotted directly: supply an explicit payment table instead. The plot does not expand contracts or reevaluate actuarial assumptions.

All required numeric values must be finite, with nonnegative `t` and `discount`, and probabilities in $[0, 1]$ for binary events. Missing or inconsistent rows are rejected, never silently removed. Supplied expected amounts and PVs must agree with the identities to relative tolerance 1e-10; they are retained, not replaced. The plot ignores cached APV/cumulative attributes and sums the displayed contributions. Calendar dates and stored `t` are retained; in particular `apv_life_flow()` uses actual days/365.25. All rows must share a valuation origin and monetary unit.

Value

An editable ggplot2 object. Its tibble `data` has one row per payment per selected measure: `row`, `t`, `x`, `event`, `measure`, `value`, `plot_value` and `label`. `value` retains original units; extreme monetary panels use scaled `plot_value` with the power of ten in the panel title.

See Also

`plot_cash_flow()`, `apv_life_flow()`, `life_contract()`, `insurance_x()`, `annuity_x()`

Examples

```
lt <- tibble::tibble(x = 60:64, lx = c(1000, 800, 600, 300, 0))
apv_life_flow(lt, ages = 60, t = 1:3, cf = rep(1200, 3), i = 0.04) |>
  plot_contingent_cash_flow(currency = "$")

# Three-year term insurance, paid at each year-end of death.
# P(j-1 < T <= j) = (l[x+j-1] - l[x+j]) / l[x], not q[x+j-1].
```

```
death_flows <- tibble::tibble(
  t = 1:3, cf = rep(10000, 3),
  probability = c(200, 200, 300) / 1000,
  discount = 1.04^-(1:3), event = "death_interval"
)
death_flows |> plot_contingent_cash_flow(currency = "$")
death_flows |> plot_contingent_cash_flow(view = "pv") +
  ggplot2::labs(title = "Term insurance: contributions to APV")
```

`plot_immunization_gap` *Plot immunization performance under interest-rate shifts*

Description

Computes and plots asset present value minus liability present value for an immunized portfolio under small interest rate changes. This allows visual evaluation of duration or duration-convexity immunization quality.

Usage

```
plot_immunization_gap(
  L,
  t,
  asset_cashflows,
  w,
  i,
  i_type = "effective",
  m = 1L,
  delta = 0.01,
  n_grid = 200L
)
```

Arguments

`L`	Numeric vector of liability payments.
`t`	Numeric vector of times of each liability payment.
`asset_cashflows`	A list where each element is a list with components `$cf` and `$t`, defining each asset's cash flow.
`w`	Numeric vector of portfolio weights or units. Must have the same length as `asset_cashflows`.
`i`	Base interest-rate input.
`i_type`	Character string indicating the interest-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Conversion frequency for nominal rates. Ignored for `i_type = "effective"` and `i_type = "force"`.

`delta`	A numeric value defining the range of annual effective rates: from `i_effective - delta` to `i_effective + delta`, where `i_effective` is the annual effective rate equivalent to `i`, `i_type`, and `m`.
`n_grid`	Number of rate values to evaluate. Integer from 2 through `.Machine$integer.max`, with sufficient memory for the requested grid.

Details

This function follows the compact actuarial notation used throughout `tidyactuarial`: `L` denotes liabilities, `t` denotes payment times, `w` denotes asset weights, `cf` denotes cash flows, `i` denotes the interest-rate input, `i_type` denotes the interest-rate type, and `m` denotes the conversion frequency for nominal rates.

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 the base rate and non-negative nearby if the convexity condition is also met. The grid is expressed in annual effective yields and clipped, when needed, to the nearest representable value strictly above -1. Its base annual effective equivalent and grid endpoints must be finite and representable. Gap values combine signed weighted cash flows using log discounts, rather than subtracting separately overflowed portfolio values. Zero-weight assets contribute zero after validation. A nonfinite gap causes an explicit error. Recovery is limited by double precision and by cancellation already lost in input data; arbitrary extreme signed cancellation is not guaranteed. The graph reprices the supplied cash flows. Their consistency with the prices and sensitivities used to solve for weights remains the caller's responsibility. Matching convexity gives zero second derivative at the base yield and does not by itself ensure a positive surplus on either side.

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(
  L = c(5000, 8000),
```

```
  t = c(3, 7),
  asset_cashflows = list(
    list(cf = c(0, 0, 100), t = c(1, 2, 3)),
    list(cf = c(0, 0, 0, 0, 0, 0, 200), t = 1:7)
  ),
  w = c(5, 2.5),
  i = 0.05,
  delta = 0.02
)
```

`plot_km` *Plot a Kaplan–Meier survival curve*

Description

Creates a step-function plot of the Kaplan–Meier survival estimate $\hat{S}(t)$ with optional pointwise confidence bands. Designed to work directly with the output of `km_lifetable`.

Usage

```
plot_km(
  km,
  time_col = "time",
  surv_col = "S",
  lower_col = "ci_low",
  upper_col = "ci_high",
  conf_int = TRUE,
  title = NULL
)
```

Arguments

`km`	A data frame or tibble with at least columns for time and survival. Can also be the full list returned by `km_lifetable`, in which case the `$km` component is extracted automatically.
`time_col`	Character. Name of the time column. Default `"time"`.
`surv_col`	Character. Name of the survival column. Default `"S"` (matching `km_lifetable` output).
`lower_col`	Character. Name of the lower CI column. Default `"ci_low"` (matching `km_lifetable` output).
`upper_col`	Character. Name of the upper CI column. Default `"ci_high"` (matching `km_lifetable` output).
`conf_int`	Logical. If `TRUE` (default) and CI columns exist in `km`, plot a step-wise confidence ribbon.
`title`	Optional character string for the plot title.

Details

Both the survival curve and the confidence band are rendered as step functions (using geom_step), which is the correct representation for the KM estimator - a right-continuous step function that drops at each observed event time.

The confidence band uses geom_stepribbon logic: the data is internally expanded so that a ribbon-fill follows the step pattern rather than interpolating linearly between event times.

Value

A ggplot object that can be further customised with additional ggplot2 layers.

See Also

km_lifetable for fitting the KM estimator and building the empirical life table.

Examples

```
set.seed(42)
n <- 150
time   <- rexp(n, rate = 0.05)
status <- rbinom(n, 1, prob = 0.7)

# Fit KM and plot directly
out <- km_lifetable(time, status, breaks = 0:30)

# Pass the full list - $km is extracted automatically
plot_km(out)

# Or pass just the km tibble
plot_km(out$km)

# Without confidence band
plot_km(out, conf_int = FALSE, title = "KM Survival Curve")

# Customise with ggplot2 layers
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot_km(out) +
    ggplot2::geom_hline(yintercept = 0.5, linetype = "dashed") +
    ggplot2::labs(subtitle = "Dashed line = median survival")
}
```

---

portfolio_convexity *Compute portfolio convexity as a market-value-weighted average*

Description

Computes portfolio convexity from individual position convexities using present values or market values as weights, using compact actuarial notation.

Usage

```
portfolio_convexity(
  .data = NULL,
  portfolio_id = NULL,
  P = NULL,
  C = NULL,
  col_portfolio = "portfolio_id",
  col_P = "P",
  col_C = "C",
  .out = "C_P",
  .out_value = "P_total",
  .out_n = "n_positions",
  .na = c("propagate", "error", "drop"),
  ...
)
```

Arguments

.data	A data.frame or tibble. If NULL, P and C 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.
P	Numeric vector of present values, prices, or market values when .data = NULL.
C	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_P	Name of the numeric column containing present values, prices, or market values.
col_C	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 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".
...	Transitional compatibility for older calls using market_value, convexity, col_market_value, and col_convexity. These names are mapped to P, C, col_P, and col_C.

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:

$$C_P = \frac{\sum_{j=1}^{r} P_j C_j}{\sum_{j=1}^{r} P_j}$$

where P_j is the present value or market value of position j, and C_j is its convexity.

This function follows the compact actuarial notation used throughout `tidyactuarial`: `P` denotes price, present value, or market value, and `C` denotes convexity.

The function is deliberately agnostic about the convexity convention, but all individual convexities must be expressed on the same basis within each portfolio. For example, do not mix convexities measured in coupon periods with convexities measured in years. All convexities must also use the same yield coordinate; nominal and annual effective yield convexities generally differ even when both use years. Values and convexities are finite and nonnegative, and each portfolio requires a finite positive total value. Zero-value positions remain in the position count. Relative weights are combined with convexities without forming monetary convexity products, recovering representable weighted contributions when separate products or weights lose range. Under `.na = "propagate"`, incomplete known portfolios have missing results. Positions with missing portfolio identifiers are retained as one final row with an `NA` identifier and missing convexity, total value and position count. They are not assigned to any known portfolio. `"drop"` removes incomplete positions with a warning; `"error"` rejects them.

Value

A tibble with one row per portfolio and columns for portfolio convexity, total portfolio value, and number of positions used.

References

Marcel B. Finan, *A Basic Course in the Theory of Interest and Derivatives Markets: A Preparation for the Actuarial Exam FM/2*, Section 55: Redington Immunization and Convexity.

Kellison, S. G. *The Theory of Interest*, Chapter 11: Duration, Convexity and Immunization.

See Also

`portfolio_duration`, `bond_convexity`, `bond_duration`

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

Examples

```
# Simple example: one portfolio
portfolio_convexity(
  P = c(1000, 2000, 500),
  C = c(20, 12, 35)
)

# Medium example: two portfolios
positions <- tibble::tibble(
  portfolio_id = c("A", "A", "B", "B"),
  P = c(1000, 2000, 1000 / 1.08^2, 1000 / 1.08^4),
  C = c(20, 12, 6, 18)
)

portfolio_convexity(
```

```
  positions,
  col_portfolio = "portfolio_id",
  col_P = "P",
  col_C = "C"
)
```

`portfolio_duration` *Compute portfolio duration as a market-value-weighted average*

Description

Computes portfolio duration from individual position durations using present values, prices, or market values as weights, using compact actuarial notation.

Usage

```
portfolio_duration(
  .data = NULL,
  portfolio_id = NULL,
  P = NULL,
  D = NULL,
  col_portfolio = "portfolio_id",
  col_P = "P",
  col_D = "D",
  .out = "D_P",
  .out_value = "P_total",
  .out_n = "n_positions",
  .na = c("propagate", "error", "drop"),
  ...
)
```

Arguments

`.data`	A data.frame or tibble. If `NULL`, `P` and `D` 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.
`P`	Numeric vector of present values, prices, or market values when `.data = NULL`.
`D`	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_P`	Name of the numeric column containing present values, prices, or market values.
`col_D`	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 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"`.
`...`	Transitional compatibility for older calls using `market_value`, `duration`, `col_market_value`, and `col_duration`. These names are mapped to `P`, `D`, `col_P`, and `col_D`.

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:

$$D_P = \frac{\sum_{j=1}^{r} P_j D_j}{\sum_{j=1}^{r} P_j}$$

where P_j is the present value, price, or market value of position j, and D_j is its duration.

This function follows the compact actuarial notation used throughout `tidyactuarial`: `P` denotes price, present value, or market value, and `D` denotes duration.

The function is deliberately agnostic about the duration convention, but all individual durations must be expressed on the same basis within each portfolio. For example, do not mix Macaulay durations in years with durations measured in coupon periods. All measures must refer to the same yield coordinate as well as the same time units for an interpretation as price sensitivity to a common shift. Values and durations are finite and nonnegative, and each portfolio requires a finite positive total value. Zero-value positions remain in the position count. Relative weights are combined with durations without forming monetary duration products, recovering representable weighted contributions when separate products or weights lose range. Under `.na = "propagate"`, incomplete known portfolios have missing results. Positions with missing portfolio identifiers are retained as one final row with an `NA` identifier and missing duration, total value and position count. They are not assigned to any known portfolio. `"drop"` removes incomplete positions with a warning; `"error"` rejects them.

Value

A tibble with one row per portfolio and columns for portfolio duration, total portfolio value, and number of positions used.

References

Marcel B. Finan, *A Basic Course in the Theory of Interest and Derivatives Markets: A Preparation for the Actuarial Exam FM/2*, Section 54: Macaulay and Modified Durations.

Kellison, S. G. *The Theory of Interest*, Chapter 11: Duration, Convexity and Immunization.

See Also

`portfolio_convexity`, `bond_duration`, `bond_convexity`

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

Examples

```
# Simple example: one portfolio
portfolio_duration(
  P = c(1000, 2000, 500),
  D = c(7, 5, 10)
)

# Medium example: two portfolios
positions <- tibble::tibble(
  portfolio_id = c("A", "A", "B", "B"),
  P = c(1000, 2000, 1000 / 1.08^2, 1000 / 1.08^4),
  D = c(7, 5, 2, 4)
)

portfolio_duration(
  positions,
  col_portfolio = "portfolio_id",
  col_P = "P",
  col_D = "D"
)
```

---

premium_gross *Gross premium under a simple expense-loaded equivalence principle*

Description

Converts a net-premium result from premium_x or premium_xy into an expense-loaded gross premium.

Usage

```
premium_gross(
  prem,
  alpha = 0,
  beta = 0,
  gamma = 0,
  k = NULL,
  output = c("value", "summary", "audit", "table"),
  tidy = NULL,
  check = TRUE,
  tol = 1e-10,
  ...
)
```

Arguments

prem	A one-row data frame or tibble. The preferred input is `premium_x(..., output = "summary")` or `premium_xy(..., output = "summary")`. It must contain an annualized net premium and the APV of the normalized premium annuity.
alpha	Nonnegative numeric scalar. Initial acquisition expense as a multiple of one gross premium installment. The expense at issue is $\alpha G^{(k)}/k$.
beta	Numeric scalar in $[0, 1)$. Proportional expense charged on every gross premium installment.
gamma	Nonnegative numeric scalar. Fixed monetary expense incurred at every premium-payment date. Its APV is $\gamma k a^{(k)}$, because `a_premiums` is normalized with payments of $1/k$.
k	Optional positive integer payment frequency. By default it is inferred from `payments_per_year` or `k` in `prem`; if neither is present, `k = 1` is used.
output	Output level. `"value"` returns the annualized gross premium, `"summary"` returns a compact one-row tibble, and `"audit"` returns the extended-equivalence components in long format. `"table"` is accepted as a compatibility alias for `"summary"`.
tidy	Deprecated compatibility argument. `TRUE` maps to `output = "summary"` and `FALSE` to `output = "value"`.
check	Logical scalar. If `TRUE`, validates the inputs.
tol	Nonnegative finite numeric tolerance. Monetary consistency is checked relatively to the compared amounts, without a one-currency-unit floor; frequency and integer checks use absolute tolerance.
...	Transitional compatibility. The deprecated argument `payments_per_year` is mapped to `k`.

Details

The function preserves the package-wide distinction between an annualized premium and the amount paid at each of the `k` payment dates.

Let $a^{(k)}$ denote the APV of the premium annuity returned by `premium_x` or `premium_xy`. That annuity has k payments per year, each of amount $1/k$; hence it represents an annual payment rate of 1.

If $P^{(k)}$ is the annualized net premium and $G^{(k)}$ is the annualized gross premium, the extended equivalence equation implemented is

$$G^{(k)} a^{(k)} = P^{(k)} a^{(k)} + \alpha \frac{G^{(k)}}{k} + \beta G^{(k)} a^{(k)} + \gamma k a^{(k)}.$$

Therefore,

$$G^{(k)} = \frac{P^{(k)} + k\gamma}{(1-\beta) - \alpha/(k a^{(k)})}.$$

The actual premium installment is

$$G_{\text{per payment}} = \frac{G^{(k)}}{k}.$$

For k = 1, this reduces to the previous annual formula:

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

This intentionally simple model assumes that gamma is incurred at the same dates and under the same contingency as premium payments. Expenses with a different timing or contingency require their own APV and are not represented by gamma.

Preferred column names in prem are premium_annualized, premium_per_payment, payments_per_year, and apv_premium_annuity. Legacy annual tables using P, premium, P_net, a_premiums, or apv_premiums remain supported.

Legacy premium columns are accepted automatically only when k = 1, because their unit is ambiguous for subannual premiums.

When annualized and per-payment premiums are both supplied, both must be valid and agree in their units. A supplied payment frequency is validated even if k is explicit. Acquisition shares and expense products retain representable values without overflowing $ka^{(k)}$ first. The loading is evaluated directly from the expense equation; a small positive loading can be reported even when the rounded gross and net premiums coincide. The equivalence residual uses the same equation factored in annuity units, avoiding subtraction of overflowing APVs where possible; it is not forced to zero. Individual audit APVs may still be infinite. No finite result is promised when the annualized premium itself is outside numeric range.

Value

For output = "value", a numeric scalar containing the annualized gross premium $G^{(k)}$.

For output = "summary", a one-row tibble with six columns: annualized gross premium, gross premium per payment, annualized net premium, annualized loading, payment frequency, and equivalence residual.

For output = "audit", a long-format tibble with the APVs of gross premiums, benefits, and each expense component.

See Also

premium_x, premium_xy, annuity_x, annuity_xy

Other life-contingencies: add_insurance(), add_premium_schedule(), annuity_multi(), annuity_x(), annuity_xy(), insurance_variable_k(), insurance_x(), insurance_xy(), life_contract(), premium_x(), premium_xy(), reserve_x(), reserve_xy(), simulate_annuity_x(), simulate_insurance_x()

Examples

```
net <- tibble::tibble(
  premium_annualized = 1200,
  premium_per_payment = 100,
  payments_per_year = 12,
  apv_premium_annuity = 10
)

premium_gross(
```

```
  net,
  alpha = 0.5,
  beta = 0.05,
  gamma = 20,
  output = "summary"
)

# Pipeline from a net-premium calculation
## Not run:
life_contract(
  lt = lt,
  lives = "single",
  x = 40,
  i = 0.05
) |>
  add_insurance(
    type = "term",
    benefit = 100000,
    n = 20
  ) |>
  add_premium_schedule(
    k = 12,
    n_prem = 10
  ) |>
  premium_x(output = "summary") |>
  premium_gross(
    alpha = 0.5,
    beta = 0.05,
    gamma = 20,
    output = "summary"
  )

## End(Not run)
```

`premium_x` *Net premium for single-life insurance by the equivalence principle*

Description

Computes the net benefit premium of a single-life insurance contract from the equation of equivalence:

$$\mathrm{APV}(\text{premiums}) = \mathrm{APV}(\text{benefits}).$$

Usage

```
premium_x(
  lt,
  x,
  i,
```

```
  i_type = "effective",
  m = 1L,
  type = c("whole", "term", "endowment", "variable_k"),
  benefit = 1,
  n = Inf,
  h = 0L,
  k = 1L,
  frac = c("UDD", "CF", "CML", "Balducci"),
  timing = c("due", "immediate"),
  premium_start = c("issue", "deferred"),
  n_prem = NULL,
  woolhouse = c("none", "first", "second"),
  output = c("value", "summary", "audit"),
  tidy = NULL,
  check = TRUE,
  ...
)
```

Arguments

`lt`	A life table containing at least columns `x` and `lx`, or a single-life contract created with `life_contract`.
`x`	Integer actuarial age at issue. Optional when `lt` is a single-life contract.
`i`	Numeric scalar. Annual interest-rate input. Optional when `lt` is a single-life contract.
`i_type`	Character string indicating the interest-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer. Conversion frequency for nominal interest-rate inputs.
`type`	Insurance type. One of `"whole"`, `"term"`, `"endowment"`, or `"variable_k"`.
`benefit`	Benefit amount. For standard products, a single nonnegative numeric value. For `type = "variable_k"`, a numeric vector or a function of time may be supplied.
`n`	Insurance term in years. Use `Inf` for whole-life insurance. For `type = "variable_k"`, fractional terms aligned with the benefit frequency are permitted; the default `Inf` infers a finite term from a numeric benefit schedule.
`h`	Nonnegative integer deferment period in years.
`k`	Positive integer. Number of premium payments per year.
`frac`	Fractional-age assumption. One of `"UDD"`, `"CF"`, `"CML"`, or `"Balducci"`.
`timing`	Premium timing: `"due"` or `"immediate"`.
`premium_start`	Start of premium payments: `"issue"` or `"deferred"`.
`n_prem`	Premium-paying term in years. If finite, `n_prem * k` must be an integer.
`woolhouse`	Woolhouse approximation order for the premium annuity: `"none"`, `"first"`, or `"second"`.
`output`	Output level. `"value"` returns the annualized premium as a numeric scalar; `"summary"` returns a compact one-row executive summary; `"audit"` returns the components of the equivalence calculation in long format.

tidy	Deprecated compatibility argument. TRUE maps to output = "summary" and FALSE maps to output = "value" when output is not supplied.
check	Logical scalar. If TRUE, performs input validation.
...	Transitional compatibility for older calls using mortality_table, age, rate, rate_type, insurance_type, term_years, deferral_years, payments_per_year, premium_timing, and premium_term_years.

Details

For premiums payable k times per year, the function distinguishes between the annualized premium $P^{(k)}$ and the amount paid at each installment, $P^{(k)}/k$.

Let Z be the present-value random variable of benefits and let $Y^{(k)}$ be the present-value random variable of a premium annuity normalized to an annual payment rate of 1. The loss at issue is

$$L_0 = Z - P^{(k)} Y^{(k)}.$$

The equivalence principle, $\mathrm{E}[L_0] = 0$, gives

$$P^{(k)} = \frac{\mathrm{APV}(\text{benefits})}{\mathrm{APV}(\text{premium annuity})}.$$

Because annuity_x values a k-thly annuity with payments of $1/k$, this quotient is the annualized premium. The actual installment paid each fraction of the year is

$$P_{\text{per payment}} = \frac{P^{(k)}}{k}.$$

A contract assembled with pipes may be valued directly:

```
life_contract(...) |>
  add_insurance(...) |>
  add_premium_schedule(...) |>
  premium_x(output = "summary")
```

Explicit arguments supplied to premium_x() override values stored in the contract components.

For variable benefits the actual coverage term, including a term inferred from the benefit vector, bounds the premium-paying period and is reported in audit output. Benefit frequency inherited from an insurance component can differ from premium frequency. The benefit vector describes payments within one contract, not independent scenarios. This function retains its scalar-contract interface; use separate calls for different contracts. The premium annuity must have a positive finite APV. An undefined or nonfinite premium-annuity factor is rejected, rather than treating an indeterminate ratio of two infinite APVs as a premium.

Value

For output = "value", a numeric scalar containing the annualized premium $P^{(k)}$.

For output = "summary", a one-row tibble with six columns: annualized premium, premium per payment, payment frequency, APV of benefits, APV of the premium annuity, and equivalence residual.

For output = "audit", a long-format tibble containing the principal inputs and calculated components.

See Also

life_contract, add_insurance, add_premium_schedule, insurance_x, insurance_variable_k, annuity_x, premium_xy, premium_gross

Other life-contingencies: add_insurance(), add_premium_schedule(), annuity_multi(), annuity_x(), annuity_xy(), insurance_variable_k(), insurance_x(), insurance_xy(), life_contract(), premium_gross(), premium_xy(), reserve_x(), reserve_xy(), simulate_annuity_x(), simulate_insurance_x()

Examples

```
lt <- data.frame(
  x = 40:90,
  lx = round(100000 * exp(-0.018 * (0:50)^1.35))
)
lt$lx[nrow(lt)] <- 0

# Direct calculation: annualized premium
premium_x(
  lt = lt,
  x = 40,
  i = 0.05,
  type = "term",
  benefit = 100000,
  n = 20,
  k = 12,
  n_prem = 10
)

# Executive result with annualized and monthly premiums
premium_x(
  lt = lt,
  x = 40,
  i = 0.05,
  type = "term",
  benefit = 100000,
  n = 20,
  k = 12,
  n_prem = 10,
  output = "summary"
)

# Contract assembled with pipes
life_contract(
  lt = lt,
  lives = "single",
  x = 40,
  i = 0.05
) |>
  add_insurance(
    type = "term",
    benefit = 100000,
    n = 20
```

```
) |>
add_premium_schedule(
  k = 12,
  n_prem = 10,
  timing = "due"
) |>
premium_x(output = "summary")
```

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

Description

Computes the net benefit premium for a joint-life or last-survivor insurance contract.

Usage

```
premium_xy(
  lt,
  x = NULL,
  y = NULL,
  i = NULL,
  i_type = "effective",
  m = 1L,
  type = c("whole", "term", "endowment", "pure_endowment"),
  benefit = 1,
  n = Inf,
  h = 0L,
  k = 1L,
  frac = c("UDD", "CF", "CML", "Balducci"),
  timing = c("due", "immediate"),
  premium_start = c("issue", "deferred"),
  n_prem = NULL,
  status = c("joint", "last"),
  woolhouse = c("none", "first", "second"),
  output = c("value", "summary", "audit", "table"),
  tidy = NULL,
  check = TRUE,
  tol = 1e-10,
  ...
)
```

Arguments

`lt`	A life table, a list of two life tables `list(lt_x, lt_y)`, or a two-life contract created with `life_contract`.
`x`	Integer actuarial age for the first life. Optional for a `life_contract`.

y	Integer actuarial age for the second life. Optional for a life_contract.
i	Numeric scalar. Annual interest-rate input. Optional for a life_contract.
i_type	Interest-rate type: "effective", "nominal_interest", "nominal_discount", or "force".
m	Positive integer. Conversion frequency for nominal rates.
type	Insurance type: "whole", "term", "endowment", or "pure_endowment".
benefit	Nonnegative insurance benefit.
n	Insurance term in years after deferment. Use Inf for whole-life insurance.
h	Nonnegative integer deferment period in years.
k	Positive integer. Number of premium payments per year.
frac	Fractional-age assumption: "UDD", "CF", "CML", or "Balducci".
timing	Premium timing: "due" or "immediate".
premium_start	Start of premiums: "issue" or "deferred".
n_prem	Premium-paying term in years. A fractional value is permitted when n_prem * k is an integer.
status	Two-life status: "joint" or "last". For a life_contract, the value is inferred from lives unless supplied explicitly.
woolhouse	Woolhouse approximation order for the premium annuity: "none", "first", or "second".
output	Output level. "value" returns the annualized premium; "summary" returns a compact one-row result; "audit" returns the equivalence components in long format. "table" is accepted as a deprecated alias for "summary".
tidy	Deprecated compatibility argument. TRUE maps to output = "summary" and FALSE to output = "value".
check	Logical scalar. If TRUE, validates inputs.
tol	Numeric tolerance for integer-grid checks.
...	Transitional compatibility for older calls using mortality_table, age_x, age_y, rate, rate_type, insurance_type, term_years, deferment_years, payments_per_year, premium_timing, premium_term_years, and cohort.

Details

For premiums payable k times per year, the function distinguishes between the annualized premium $P^{(k)}$ and the amount paid at each installment, $P^{(k)}/k$.

Let Z denote the present-value random variable of the two-life insurance benefit and let $Y^{(k)}$ denote the present value of the contingent premium annuity normalized to an annual payment rate of 1. The loss at issue is

$$L_0 = Z - P^{(k)} Y^{(k)}.$$

The equivalence principle gives

$$P^{(k)} = \frac{\mathrm{APV}(\text{benefits})}{\mathrm{APV}(\text{premium annuity})}.$$

Since annuity_xy assigns amount $1/k$ to each k-thly payment, this quotient is the annualized premium. The installment is

$$P_{\text{per payment}} = \frac{P^{(k)}}{k}.$$

Standard whole-life, term, and endowment benefits are valued through insurance_xy. A pure endowment is valued directly as the discounted benefit times status survival at maturity, retaining the issue-age conditioning, including for deferred last-survivor benefits. The identity endowment minus term remains a mathematical check, rather than the numerical calculation. Tables and horizons are validated without inventing terminal mortality. The premium annuity must have a positive finite representable APV; the output is one contract, not parallel scenarios.

A contract assembled with pipes may be valued directly:

```
life_contract(...) |>
  add_insurance(...) |>
  add_premium_schedule(...) |>
  premium_xy(output = "summary")
```

Explicit arguments supplied to `premium_xy()` override values stored in the contract components.

Value

For `output = "value"`, a numeric scalar containing the annualized premium $P^{(k)}$.

For `output = "summary"`, a one-row tibble with six columns: annualized premium, premium per payment, payment frequency, APV of benefits, APV of the premium annuity, and equivalence residual.

For `output = "audit"`, a long-format tibble.

See Also

premium_x, insurance_xy, annuity_xy, reserve_xy, life_contract, add_insurance, add_premium_schedule

Other life-contingencies: add_insurance(), add_premium_schedule(), annuity_multi(), annuity_x(), annuity_xy(), insurance_variable_k(), insurance_x(), insurance_xy(), life_contract(), premium_gross(), premium_x(), reserve_x(), reserve_xy(), simulate_annuity_x(), simulate_insurance_x()

Examples

```
lt <- data.frame(
  x = 40:100,
  lx = round(100000 * exp(-0.012 * (0:60)^1.35))
)
lt$lx[nrow(lt)] <- 0

premium_xy(
  lt = lt,
  x = 60,
  y = 62,
  i = 0.05,
  type = "term",
```

```
  status = "joint",
  benefit = 100000,
  n = 20,
  k = 12,
  n_prem = 10,
  output = "summary"
)

life_contract(
  lt = lt,
  lives = "joint",
  x = 60,
  y = 62,
  i = 0.05
) |>
  add_insurance(
    type = "term",
    benefit = 100000,
    n = 20
  ) |>
  add_premium_schedule(
    k = 12,
    n_prem = 10
  ) |>
  premium_xy(output = "summary")
```

`present_value` *Present value of a single payment*

Description

Computes the present value of a future payment due at a given time, using the annual effective interest rate implied by the supplied interest-rate specification and compact actuarial notation.

Usage

```
present_value(C, i, i_type = "effective", m = 1, t, tidy = FALSE)
```

Arguments

`C`	Numeric vector of future payment amounts or capitals.
`i`	Numeric vector of interest-rate values.
`i_type`	Character vector indicating the interest-rate type. Allowed values are `"effective"`, `"nominal_interest"`, `"nominal_discount"`, and `"force"`.
`m`	Positive integer vector giving the conversion frequency for nominal rates. Observed values are validated as positive integers for all rate families; only nominal conversions use their numeric value.

t	Numeric vector of times in years until payment.
tidy	Logical scalar. If FALSE, returns a numeric present value. If TRUE, returns a tibble with intermediate calculations.

Details

The present value is computed as

$$PV = Cv^t = \frac{C}{(1+i)^t}$$

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

The input interest rate may be supplied as:

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

Internally, all rate specifications are first converted to the equivalent annual effective interest rate using standardize_interest. Valuation retains the log accumulation rate of the original specification instead of reconstructing it from a rounded annual equivalent. In tidy output, extreme annual equivalents may round to -1 or overflow to Inf; the annual discount factor and present value are calculated independently. Products with extreme factors are combined in log space when necessary, so an unrepresentable factor need not invalidate a representable value.

This function follows the compact actuarial notation used throughout tidyactuarial: C denotes the future payment amount or capital, t denotes time, i denotes the interest-rate input, i_type denotes the interest-rate type, and m denotes the conversion frequency for nominal rates.

Input vectors must have length 1 or a common length. Missing values are propagated. This function does not accept dates; use pv_flow for dated cash flows.

Value

If tidy = FALSE, a numeric vector of present values.

If tidy = TRUE, a tibble with input values, equivalent rates, discount factors, and present values.

See Also

standardize_interest, future_value, pv_flow

Other time-value: accumulation_factor(), discount_factor(), future_value(), fv_flow(), irr_flow(), irr_flow_multi(), plot_cash_flow(), pv_flow(), solve_value()

Examples

```
# Numeric present value
present_value(C = 1000, i = 0.08, t = 3)

# Nominal interest converted monthly
```

```
present_value(
  C = 1000,
  i = 0.12,
  i_type = "nominal_interest",
  m = 12,
  t = 5
)

# Tibble output for teaching or auditing
present_value(
  C = 1000,
  i = 0.08,
  t = 3,
  tidy = TRUE
)

# Vectorized example
present_value(
  C = c(1000, 2500, 4000),
  i = c(0.08, 0.10, 0.12),
  i_type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  t = c(3, 5, 2)
)
```

`print.tidyact_aggregate_distribution` *Print an aggregate loss distribution*

Description

Print an aggregate loss distribution

Usage

```
## S3 method for class 'tidyact_aggregate_distribution'
print(x, ...)
```

Arguments

`x`	A ‘tidyact_aggregate_distribution‘ object.
`...`	Additional arguments. Currently ignored.

Value

‘x‘, invisibly.

`print.tidyact_collective_risk_spec`
Print a collective risk model specification

Description

Print a collective risk model specification

Usage

```
## S3 method for class 'tidyact_collective_risk_spec'
print(x, ...)
```

Arguments

`x`	A ‘tidyact_collective_risk_spec‘ object.
`...`	Additional arguments. Currently ignored.

Value

‘x‘, invisibly.

`print.tidyact_convolution_spec`
Print a convolution specification

Description

Print a convolution specification

Usage

```
## S3 method for class 'tidyact_convolution_spec'
print(x, ...)
```

Arguments

`x`	A ‘tidyact_convolution_spec‘ object.
`...`	Additional arguments. Currently ignored.

Value

The input object, invisibly.

print.tidyact_frequency_spec

Print a frequency distribution specification

Description

Print a frequency distribution specification

Usage

```
## S3 method for class 'tidyact_frequency_spec'
print(x, ...)
```

Arguments

x	A ‘tidyact_frequency_spec‘ object.
...	Additional arguments. Currently ignored.

Value

The input object, invisibly.

print.tidyact_payment_spec

Print a payment severity specification

Description

Print a payment severity specification

Usage

```
## S3 method for class 'tidyact_payment_spec'
print(x, ...)
```

Arguments

x	A ‘tidyact_payment_spec‘ object.
...	Unused.

Value

‘x‘, invisibly.

print.tidyact_severity_spec *Print a severity distribution specification*

Description

Print a severity distribution specification

Usage

```
## S3 method for class 'tidyact_severity_spec'
print(x, ...)
```

Arguments

x	A ‘tidyact_severity_spec‘ object.
...	Unused.

Value

‘x‘, invisibly.

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(
  cf,
  i,
  i_type = "effective",
  m = 1L,
  t = NULL,
  date = NULL,
  day_count = c("act/365", "act/360")
)
```

Arguments

cf	Numeric vector of cash flows.
i	Numeric scalar or numeric vector of interest-rate values.
i_type	Character vector indicating the interest-rate type: "effective", "nominal_interest", "nominal_discount", or "force". May have length 1 or the same length as cf.
m	Positive integer vector giving the conversion frequency for nominal rates. May have length 1 or the same length as cf.
t	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.
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 cf. Its timing is supplied either through t (in years) or date (calendar dates). If date is supplied, the earliest date is taken as time 0.

Interest-rate input:

- If i has length 1, the same rate is used for all cash flows.
- If i has the same length as cf, each rate is interpreted as the spot rate associated with the corresponding cash-flow 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. Discounting retains the original log accumulation rates, including when annual effective equivalents round to -1 or overflow to Inf.

Products are evaluated in log space when separate factors lose range or precision; original cash-flow signs and known zero amounts are preserved. If individual values overflow, amounts with exactly equal finite log factors are netted before valuation. Cancellation between unequal factors is not recovered by this rule and may still produce non-finite results.

This function follows the compact actuarial notation used throughout tidyactuarial: cf denotes cash flows, t denotes time, i denotes the interest rate, i_type denotes the interest-rate type, and m denotes the conversion frequency for nominal rates.

When i 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 cash flow 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: accumulation_factor(), discount_factor(), future_value(), fv_flow(), irr_flow(), irr_flow_multi(), plot_cash_flow(), present_value(), solve_value()

Examples

```
# Constant annual effective rate
pv_flow(
  cf = c(100, 150, 200),
  i = 0.08,
  i_type = "effective",
  t = c(0, 1, 2)
)

# Spot rates, one per cash flow
pv_flow(
  cf = c(100, 150, 200),
  i = c(0.05, 0.055, 0.06),
  i_type = "effective",
  t = c(1, 2, 3)
)

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

# Nominal rates by cash flow
pv_flow(
  cf = c(100, 100, 100),
  i = c(0.12, 0.12, 0.12),
  i_type = "nominal_interest",
  m = c(12, 12, 12),
  t = c(1, 2, 3)
)
```

---

reserve_x *Benefit reserves for single-life insurance*

Description

Computes terminal benefit reserves at selected integer policy durations by the prospective or recursive method.

Usage

```
reserve_x(
  lt,
  x,
  i,
  i_type = "effective",
  m = 1L,
  type = c("whole", "term", "endowment"),
  n = Inf,
  h = 0L,
  benefit = 1,
  P = NULL,
  k = 1L,
  frac = c("UDD", "CF", "CML", "Balducci"),
  timing = c("due", "immediate"),
  premium_start = c("issue", "deferred"),
  n_prem = NULL,
  woolhouse = c("none", "first", "second"),
  t = NULL,
  method = c("prospective", "recursive"),
  output = c("summary", "value", "audit", "table"),
  tidy = NULL,
  check = TRUE,
  ...
)
```

Arguments

`lt`	A life table containing columns `x` and `lx`, or a single-life contract created with `life_contract`.
`x`	Integer actuarial age at issue. Optional for a life contract.
`i`	Numeric scalar. Annual interest-rate input. Optional for a life contract.
`i_type`	Interest-rate type: `"effective"`, `"nominal_interest"`, `"nominal_discount"`, or `"force"`.
`m`	Positive integer. Conversion frequency for nominal interest rates.
`type`	Insurance type: `"whole"`, `"term"`, or `"endowment"`.
`n`	Insurance term in years. Use `Inf` for whole-life insurance.
`h`	Nonnegative integer deferment period in years.
`benefit`	Positive insurance benefit.
`P`	Optional annualized premium $P^{(k)}$. If `NULL`, it is calculated with `premium_x` by the equivalence principle.

k	Positive integer. Number of premium payments per year.
frac	Fractional-age assumption for exact k-thly premium annuities.
timing	Premium timing. Currently only "due" is supported, which gives reserves immediately before any premium payable at duration t.
premium_start	Start of premium payments: "issue" or "deferred".
n_prem	Premium-paying term in years. A fractional term is permitted when n_prem * k is an integer.
woolhouse	Woolhouse approximation order for the premium annuity: "none", "first", or "second".
t	Optional integer vector of policy durations. If NULL, all supported integer durations are returned.
method	Reserve method: "prospective" or "recursive". The recursive method currently supports annual premiums only (k = 1).
output	Output level. "summary" returns a compact reserve schedule, "value" returns a named numeric vector, and "audit" returns the reserve components in long format. "table" is accepted as a deprecated alias for "summary".
tidy	Deprecated compatibility argument. TRUE maps to output = "summary" and FALSE to output = "value".
check	Logical scalar. If TRUE, validates inputs.
...	Transitional compatibility for older calls using mortality_table, age, rate, rate_type, insurance_type, term_years, premium, premium_term_years, and durations.

Details

The prospective method supports annual and true k-thly premiums payable in advance. The premium input P is always interpreted as the annualized premium $P^{(k)}$. The amount paid at each premium date is $P^{(k)}/k$.

At integer duration t, conditional on survival to age $x+t$, the prospective reserve is

$$ {}_tV_x = \mathrm{APV}_t(\text{future benefits}) - P^{(k)}\,\mathrm{APV}_t(\text{future premium annuity}). $$

The premium annuity is normalized to an annual payment rate of 1, so each installment has amount $1/k$. Therefore, the multiplier in the reserve formula is the annualized premium $P^{(k)}$, not the installment $P^{(k)}/k$.

Reserves are measured immediately before any premium payable at duration t. This is the standard fully discrete terminal-reserve convention for premiums payable in advance.

For annual premiums, the recursive method uses

$$ {}_{t+1}V_x = \frac{({}_tV_x + P_t)(1+i) - b_{t+1}q_{x+t}}{p_{x+t}}. $$

The implementation evaluates its algebraically equivalent backward form

$$ {}_tV_x = v(p_{x+t}\,{}_{t+1}V_x + q_{x+t}b_{t+1}) - P_t. $$

This avoids amplification by repeated division by small survival probabilities. The boundary value is the prospective value at the last supported positive cohort within the contract horizon; it is not generally zero for whole-life insurance. Already defined maturity payments are retained. Unrequested extinct conditioning states are not traversed. A requested intermediate reserve conditional on an extinct cohort remains undefined.

Tables must have nonincreasing finite `lx` on an annual age grid. The recursive calculation retains the original log accumulation rate and forms death decrements before dividing survivor counts. Positive survival is obtained directly from counts, not as $1 - q_x$. The whole-life output horizon remains the last recorded age with positive `lx`; finite coverage must be supported by recorded table ages. These local conventions are retained separately from the implicit final interval in insurance APVs. `t` indexes durations of one contract, not independent policy scenarios. No clipping or zeroing of small/negative reserves is applied. Nonfinite future APV components do not in general identify their difference; exact recovery of arbitrary cancellations between infinite components is not promised.

Value

For `output = "summary"`, a tibble with at most six columns: duration, attained age, reserve, annualized premium, premium per payment, and method.

For `output = "value"`, a named numeric vector.

For `output = "audit"`, a long-format tibble containing future benefit APV, future premium APV, reserve, and premium amounts at each duration.

See Also

`premium_x`, `insurance_x`, `annuity_x`, `life_contract`, `add_insurance`, `add_premium_schedule`

Other life-contingencies: `add_insurance()`, `add_premium_schedule()`, `annuity_multi()`, `annuity_x()`, `annuity_xy()`, `insurance_variable_k()`, `insurance_x()`, `insurance_xy()`, `life_contract()`, `premium_gross()`, `premium_x()`, `premium_xy()`, `reserve_xy()`, `simulate_annuity_x()`, `simulate_insurance_x()`

Examples

```
lt <- data.frame(
  x = 40:90,
  lx = round(100000 * exp(-0.018 * (0:50)^1.35))
)
lt$lx[nrow(lt)] <- 0

reserve_x(
  lt = lt,
  x = 40,
  i = 0.05,
  type = "term",
  n = 20,
  benefit = 100000,
  k = 12,
  n_prem = 10,
  t = c(0, 5, 10, 15, 20)
)
```

```
life_contract(
  lt = lt,
  lives = "single",
  x = 40,
  i = 0.05
) |>
  add_insurance(
    type = "term",
    benefit = 100000,
    n = 20
  ) |>
  add_premium_schedule(
    k = 12,
    n_prem = 10
  ) |>
  reserve_x(
    t = c(0, 5, 10, 15, 20),
    output = "summary"
  )
```

`reserve_xy` *Benefit reserves for two-life insurance*

Description

Computes prospective or recursive terminal reserves for joint-life and last-survivor insurance. For k-thly premiums, 'P' is the annualized premium and each installment equals 'P / k'.

Usage

```
reserve_xy(
  lt,
  x = NULL,
  y = NULL,
  i = NULL,
  i_type = "effective",
  m = 1L,
  type = c("whole", "term", "endowment", "pure_endowment"),
  status = c("joint", "last"),
  n = Inf,
  h = 0L,
  benefit = 1,
  P = NULL,
  n_prem = NULL,
  k = 1L,
  timing = c("due", "immediate"),
  premium_start = c("issue", "deferred"),
```

```
  frac = c("UDD", "CF", "CML", "Balducci"),
  woolhouse = c("none", "first", "second"),
  t = NULL,
  method = c("prospective", "recursive"),
  output = c("summary", "value", "audit", "table"),
  tidy = NULL,
  check = TRUE,
  tol = 1e-10,
  ...
)
```

Arguments

lt	One life table, ‘list(lt_x, lt_y)‘, or a two-life [life_contract()] object.
x, y	Integer ages at issue. Optional for a life contract.
i	Annual interest-rate input. Optional for a life contract.
i_type	Interest-rate type.
m	Conversion frequency for nominal rates.
type	One of ‘"whole"‘, ‘"term"‘, ‘"endowment"‘, or ‘"pure_endowment"‘.
status	‘"joint"‘ or ‘"last"‘.
n	Insurance term in years after deferment.
h	Nonnegative integer deferment in years.
benefit	Positive benefit amount.
P	Optional annualized premium. If ‘NULL‘, [premium_xy()] is used.
n_prem	Premium-paying term. Fractional values require ‘n_prem * k‘ to be an integer.
k	Number of premium payments per year.
timing	‘"due"‘ or ‘"immediate"‘.
premium_start	‘"issue"‘ or ‘"deferred"‘.
frac	Fractional-age assumption.
woolhouse	‘"none"‘, ‘"first"‘, or ‘"second"‘.
t	Integer policy durations.
method	‘"prospective"‘ or ‘"recursive"‘. Recursion currently requires ‘h = 0‘, ‘k = 1‘, due premiums, and premiums starting at issue.
output	‘"summary"‘, ‘"value"‘, or ‘"audit"‘. ‘"table"‘ is a compatibility alias for ‘"summary"‘.
tidy	Deprecated logical output selector.
check	Logical input-check switch.
tol	Numeric tolerance.
...	Deprecated argument aliases.

Details

The prospective reserve is

$$ {}_tV = \mathrm{APV}_t(\text{future benefits}) - P^{(k)}\,\mathrm{APV}_t(\text{future premium annuity}). $$

Because [annuity_xy()] assigns amount ‘1 / k‘ to each payment, ‘P‘ is annualized and the actual installment is ‘P / k‘.

Reserves are measured immediately before a premium payable at duration ‘t‘. With immediate premiums this includes the installment scheduled exactly at ‘t‘, including a last installment at maturity. Future installments use the selected annuity method; the current installment is an exact ‘P / k‘. Pure endowments use direct discounted survival to maturity, avoiding a subtraction of nearly equal insurance values. Both lives are assumed alive at the valuation duration, under independent mortality. In particular, ‘status = "last"‘ selects the insured event; it does not condition on an unspecified surviving life. Last-survivor recursion tracks the both-alive and each single-survivor state separately. Both recursive statuses use backward conditional transition identities, direct annual ‘lx‘ decrements and the original log interest basis. This avoids forward cancellation and loss from an effective-rate conversion. Unreachable transitions contribute exactly zero. The recursion’s existing timing and deferment restrictions are retained. Conditional states whose reserves themselves overflow double precision are not generally recoverable.

Value

A compact tibble, a named numeric vector, or a long audit tibble.

See Also

[premium_xy()], [annuity_xy()], [insurance_xy()], [reserve_x()]

Other life-contingencies: `add_insurance()`, `add_premium_schedule()`, `annuity_multi()`, `annuity_x()`, `annuity_xy()`, `insurance_variable_k()`, `insurance_x()`, `insurance_xy()`, `life_contract()`, `premium_gross()`, `premium_x()`, `premium_xy()`, `reserve_x()`, `simulate_annuity_x()`, `simulate_insurance_x()`

---

s_angle *Level annuity accumulation factor s-angle-n*

Description

Computes the actuarial accumulation factor for a level annuity using compact actuarial notation.

Usage

```
s_angle(
  n,
  k = 1L,
  i,
  i_type = "effective",
  m = 1L,
  h = 0,
```

```
  timing = "immediate",
  payment = 1,
  tidy = FALSE
)
```

Arguments

n	Numeric vector of payment durations in years. Each value must be positive and finite.
k	Positive integer vector giving the number of discrete payments per year. Ignored for continuous annuities.
i	Numeric vector of interest-rate values.
i_type	Character vector indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer vector giving the conversion frequency for nominal rates. Ignored for "effective" and "force".
h	Numeric vector of deferment times in years. Must be greater than or equal to 0. Under the adopted horizon convention, this is metadata only for accumulation factors.
timing	Character vector. One of "immediate", "due", or "continuous".
payment	Numeric vector of level payment amounts. Used only when tidy = TRUE to report the corresponding future value. The accumulation factor itself is always computed for unit payments.
tidy	Logical scalar. If FALSE, returns a numeric accumulation factor. If TRUE, returns a tibble with intermediate calculations.

Details

Supported timing conventions:

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

For discrete annuities, k is the number of payments per year, so payments are made every $1/k$ year. The function returns the accumulation factor, assuming a unit payment at each payment time.

Horizon convention: the future value is measured at the end of the payment term, $h + n$. This is the last payment time for immediate annuities and one payment interval after the last payment for due annuities. Under this convention, a pure deferment that shifts the entire payment block forward in time does not change the accumulation factor when the payment pattern is otherwise unchanged. Therefore, h is recorded and validated, but it does not modify the factor.

Only finite payment terms are supported. A perpetuity has no finite terminal date at which this function defines an accumulated value.

This function follows the compact actuarial notation used throughout tidyactuarial:

- n: annuity term;

- k: payment frequency;
- i: interest rate;
- i_type: interest-rate type;
- m: conversion frequency for nominal rates;
- h: deferment period.

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

For finite discrete annuities:

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

For due annuities:

$$\ddot{s}_{\overline{n|}} = (1+i)s_{\overline{n|}}$$

For continuous annuities:

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

For $N = nk$, let $b = 0$ for immediate payments and $b = 1$ for due payments. The discrete factor at horizon $h + n$ is

$$\sum_{j=1}^{N} \exp\{\delta[n - (j-b)/k]\}.$$

Discrete payments are one unit each; continuous payments are one unit per year. The latter factor is $\int_0^n e^{\delta u}\, du$. Original log accumulation rates and stable logarithmic sums/integrals preserve small nonzero rates and saturated annual effective equivalents. Monetary values combine payments and log factors stably; standalone audit rates or factors may still overflow or underflow.

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

Value

If tidy = FALSE, a numeric vector of accumulation factors.

If tidy = TRUE, a tibble with input values, equivalent rates, accumulation factors, payment amounts, and future values.

See Also

a_angle, standardize_interest, future_value

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

Examples

```
# Numeric accumulation factor
s_angle(n = 10, i = 0.05)

# Nominal interest converted monthly, with monthly payments
s_angle(
  n = 10,
```

```
  i = 0.06,
  i_type = "nominal_interest",
  m = 12,
  k = 12
)

# Continuous annuity
s_angle(
  n = 15,
  i = 0.04,
  i_type = "force",
  timing = "continuous"
)

# Tibble output for teaching or auditing
s_angle(
  n = 10,
  i = 0.05,
  payment = 1000,
  tidy = TRUE
)

# Vectorized example
s_angle(
  n = c(5, 10, 20),
  k = c(1, 12, 1),
  i = c(0.05, 0.06, 0.04),
  i_type = c("effective", "nominal_interest", "force"),
  m = c(1, 12, 1),
  h = c(0, 2, 3),
  timing = c("immediate", "due", "continuous")
)
```

`severity_spec` *Specify an individual claim severity distribution*

Description

Creates a lightweight specification of a nonnegative claim amount X. A specification records the probabilistic model; it does not fit data, draw samples, discretize amounts, or calculate aggregate losses.

Usage

```
severity_spec(dist, ..., custom = NULL, label = NULL)
```

Arguments

dist	One of "exponential", "gamma", "lognormal", "weibull", "pareto", "burr", "loglogistic", "degenerate", "discrete", or "custom".
...	Named distribution parameters, as described in *Details*.
custom	Named list of computational functions and support information when dist = "custom"; NULL for built-in families.
label	Optional nonempty character scalar identifying the model.

Details

All severities satisfy $X \geq 0$. The supported built-in models are:

- **Exponential:** rate > 0, matching stats::dexp().
- **Gamma:** shape > 0, rate > 0, matching stats::dgamma().
- **Lognormal:** meanlog is finite and sdlog > 0, matching stats::dlnorm(). A point mass is specified with "degenerate".
- **Weibull:** shape > 0, scale > 0, matching stats::dweibull().
- **Pareto II (Lomax):** shape > 0, scale > 0; the survival function is $P(X > x) = (1 + x/\text{scale})^{-\text{shape}}$ for $x \geq 0$. This is *not* the Pareto I distribution with positive lower endpoint.
- **Burr XII:** shape1 > 0, shape2 > 0, scale > 0, with survival function $P(X > x) = [1 + (x/\text{scale})^{\text{shape1}}]^{-\text{shape2}}$.
- **Log-logistic:** shape > 0, scale > 0, the Burr XII special case shape1 = shape, shape2 = 1.
- **Degenerate:** value >= 0 is a finite amount with probability one.
- **Finite discrete:** values is a strictly increasing vector of distinct finite nonnegative amounts; probs contains strictly positive probabilities, with the same length and summing to one (to numerical tolerance). Amounts need not be integer multiples of a common unit.

Unlike a frequency distribution, an individual claim severity may have an atom at zero and need not be absolutely continuous. The "degenerate" and "discrete" models are represented as discrete specifications rather than being forced into a continuous family.

For dist = "custom", custom must be a named list with support and at least one of pdf or pmf. The support list must contain type, lower, and upper, with type in "continuous", "discrete", "lattice", or "mixed"; lattice support additionally requires a positive step. The support component may also be a function of params returning that list. A mixed specification requires pdf, pmf, and cdf. Custom operations, when provided, follow these interfaces: pdf(x, params, log = FALSE), pmf(x, params, log = FALSE), cdf(q, params, lower.tail = TRUE, log.p = FALSE), quantile(p, params, lower.tail = TRUE, log.p = FALSE), rng(n, params), raw_moment(order, params), limited_moment(order, limit, params), mgf(t, params), and laplace(t, params). The argument params is the named list collected from .... As with frequency_spec(), the constructor checks the *shape* of a custom definition but cannot certify its mathematical validity.

The internal registry supplies probability functions and raw moments for built-in families. Some advanced operations (for example limited moments, the MGF of heavy-tailed distributions, and numerical Laplace transforms) are intentionally left unavailable until their public interfaces are set. In particular, no numerical approximation is performed by this constructor.

Value

An object with classes "tidyact_severity_spec" and "tidyact_distribution_spec". The ordered fields are schema_version, dist, params, support, data, custom, label, and metadata. Its data field is NULL; empirical severities are handled separately.

References

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*, 2nd ed. Society of Actuaries.

See Also

frequency_spec()

Examples

```
severity_spec("exponential", rate = 0.002)
severity_spec("gamma", shape = 2.5, rate = 0.001)
severity_spec("lognormal", meanlog = 8, sdlog = 0.8)
severity_spec("weibull", shape = 1.7, scale = 5000)

# Pareto II and Burr XII use severity distributions starting at zero.
severity_spec("pareto", shape = 3, scale = 2000)
severity_spec("burr", shape1 = 1.5, shape2 = 2.5, scale = 1500)
severity_spec("loglogistic", shape = 2.5, scale = 1500)

# A fixed benefit amount, including a point mass at zero.
severity_spec("degenerate", value = 1000)
severity_spec("degenerate", value = 0)

# A claim amount distribution with a genuine atom at zero.
severity_spec(
  "discrete",
  values = c(0, 500, 2500),
  probs = c(0.1, 0.6, 0.3),
  label = "Illustrative claim payments"
)

# An explicitly defined custom continuous model.
severity_spec(
  "custom",
  rate = 0.002,
  custom = list(
    pdf = function(x, params, log = FALSE) {
      stats::dexp(x, rate = params$rate, log = log)
    },
    support = list(type = "continuous", lower = 0, upper = Inf)
  )
)
```

`simulate_aggregate` *Simulate a collective risk model*

Description

Samples independent realizations of S = X[1] + ... + X[N], drawing frequencies first and then independent severities conditional on those counts. A zero-frequency realization has aggregate loss zero. The model assumes iid severities independent of frequency. Whenever the moments exist, E[S] = E[N] E[X], and Var(S) = E[N] Var(X) + Var(N) E[X]^2.

Usage

```
simulate_aggregate(x, n_sim = 1000L, seed = NULL, max_claims = 1000000L)
```

Arguments

`x`	A [collective_risk_spec()] object. Zero-payment specifications produced by [to_payment_basis()] are also accepted.
`n_sim`	Positive integer number of independent realizations.
`seed`	Optional nonnegative integer random seed, restored after use.
`max_claims`	Maximum total number of individual claim draws in one invocation. Exceeding this protective allocation limit is an error, not a truncation of the frequency distribution.

Details

The resulting aggregate distribution follows the frequency and severity specifications on their explicitly declared count basis. All individual severities are iid and independent of frequency.

Value

A tibble with ‘simulation‘, ‘claim_count‘, and ‘aggregate_loss‘.

Examples

```
model <- collective_risk_spec(frequency_spec("poisson", lambda = 2),
                              severity_spec("degenerate", value = 100))
simulate_aggregate(model, n_sim = 5, seed = 30)
```

`simulate_annuity_x` *Monte Carlo simulation of a life annuity*

Description

Simulates present values of a discrete single-life annuity from a life table. Annual and subannual payments are supported. For subannual payments, a complete future lifetime is generated inside the year of death under a fractional-age assumption.

Usage

```
simulate_annuity_x(
  lt,
  x = NULL,
  i = NULL,
  i_type = NULL,
  m = NULL,
  n = Inf,
  k = 1L,
  timing = c("due", "immediate"),
  payment = 1,
  frac = c("udd", "cml", "balducci"),
  method = c("inverse"),
  n_sim = 10000L,
  seed = NULL,
  output = c("simulations", "summary"),
  ...
)
```

Arguments

`lt`	A life table or a `tidyact_life_contract`. A life table must contain columns `x` and `lx`.
`x`	Integer actuarial age. Optional for a life contract.
`i`	Numeric scalar interest-rate input.
`i_type`	Interest-rate convention.
`m`	Positive integer nominal conversion frequency.
`n`	Positive integer term in years, or `Inf` for whole life.
`k`	Positive integer number of annuity payments per year.
`timing`	Payment timing: `"due"` or `"immediate"`.
`payment`	Nonnegative amount paid at each payment date. To simulate an annualized payment rate of 1 with `k` payments per year, use `payment = 1 / k`.
`frac`	Fractional-age assumption used to simulate the complete lifetime within the year of death: `"udd"`, `"cml"`, or `"balducci"`.

method	Simulation method. Currently only "inverse".
n_sim	Positive integer number of simulations.
seed	Optional nonnegative integer seed.
output	"simulations" or "summary".
...	Transitional compatibility for mortality_table, age, rate, rate_type, term_years, and payments_per_year.

Details

The amount payment is a per-payment amount. If N payments are made at times t_j, the simulated present value is

$$\sum_{j=1}^{N} \text{payment}\, v^{t_j}.$$

For k > 1, the curtate lifetime alone is insufficient because payments may occur during the year of death. The function therefore simulates a complete lifetime $T_x = K_x + S$ under frac. Fractional survival is retained in log form, including when an annual lx ratio underflows. Cash flows use the original log interest rate and combine the monetary amount before exponentiating. The displayed annual effective rate may overflow or round to -1 at extreme forces. With a finite horizon, surviving paths have Kx = n, missing Tx, and died_within_horizon = FALSE; their future lifetime is censored, not observed to equal n.

Value

A tibble with one row per simulation, or a summary from summary_mc. Simulation output includes payment_per_payment, payment_annualized, Kx, Tx, n_payments, present_value, and pv_annuity.

See Also

Other simulation: simulate_insurance_x()

Other life-contingencies: add_insurance(), add_premium_schedule(), annuity_multi(), annuity_x(), annuity_xy(), insurance_variable_k(), insurance_x(), insurance_xy(), life_contract(), premium_gross(), premium_x(), premium_xy(), reserve_x(), reserve_xy(), simulate_insurance_x()

Examples

```
lt <- data.frame(
  x = 60:66,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000, 0)
)

simulate_annuity_x(
  lt = lt,
  x = 60,
  i = 0.05,
  n_sim = 25,
  seed = 123
```

```
)

simulate_annuity_x(
  lt = lt,
  x = 60,
  i = 0.05,
  n = 5,
  k = 12,
  payment = 1 / 12,
  frac = "udd",
  n_sim = 25,
  seed = 123
)
```

`simulate_frequency` *Simulate claim counts from a frequency model*

Description

Draws independent realizations of the claim frequency random variable N using the RNG registered for its distribution. A draw represents the number of claims on one exposure or observation unit, not the portfolio total unless the model itself is specified at portfolio level.

Usage

```
simulate_frequency(x, n_sim = 1000L, seed = NULL)
```

Arguments

`x`	A [frequency_spec()] or [empirical_frequency()] specification.
`n_sim`	Positive integer number of simulations.
`seed`	Optional nonnegative integer seed. When supplied, the caller's random-number state is restored after the call.

Details

A missing RNG provider is an error, not a reason to silently approximate the model. For an empirical frequency, the draws follow the observed mass function exactly.

Value

A tibble with ‘simulation‘ and ‘claim_count‘ columns.

Examples

```
simulate_frequency(frequency_spec("poisson", lambda = 3), n_sim = 5, seed = 10)
```

simulate_insurance_x *Monte Carlo simulation of a life insurance*

Description

Simulates present values of a discrete single-life insurance payable at the end of the year of death, or at maturity for an endowment insurance. Monetary values retain the original log interest rate, even when the displayed annual effective rate loses range. Finite-horizon survivors are censored: `Kx = n` is a placeholder and `died_within_horizon = FALSE`, not an observed death at that duration.

Usage

```
simulate_insurance_x(
  lt,
  x = NULL,
  i = NULL,
  i_type = NULL,
  m = NULL,
  type = c("whole", "term", "endowment"),
  n = Inf,
  benefit = 1,
  method = c("inverse"),
  n_sim = 10000L,
  seed = NULL,
  output = c("simulations", "summary"),
  ...
)
```

Arguments

`lt`	A life table or a `tidyact_life_contract`.
`x`	Integer actuarial age.
`i`	Numeric scalar interest-rate input.
`i_type`	Interest-rate convention.
`m`	Positive integer nominal conversion frequency.
`type`	`"whole"`, `"term"`, or `"endowment"`.
`n`	Positive integer term, or `Inf` for whole life.
`benefit`	Nonnegative insurance benefit.
`method`	Simulation method. Currently only `"inverse"`.
`n_sim`	Positive integer number of simulations.
`seed`	Optional nonnegative integer seed.
`output`	`"simulations"` or `"summary"`.
`...`	Transitional compatibility for `mortality_table`, `age`, `rate`, `rate_type`, `insurance_type`, and `term_years`.

Value

A tibble with one row per simulation, or a summary. Simulation output includes `benefit_indicator`, `benefit_time`, `present_value`, and `pv_benefit`.

See Also

Other simulation: `simulate_annuity_x()`

Other life-contingencies: `add_insurance()`, `add_premium_schedule()`, `annuity_multi()`, `annuity_x()`, `annuity_xy()`, `insurance_variable_k()`, `insurance_x()`, `insurance_xy()`, `life_contract()`, `premium_gross()`, `premium_x()`, `premium_xy()`, `reserve_x()`, `reserve_xy()`, `simulate_annuity_x()`

Examples

```
lt <- data.frame(
  x = 60:66,
  lx = c(100000, 99000, 97500, 95500, 93000, 90000, 0)
)

simulate_insurance_x(
  lt = lt,
  x = 60,
  i = 0.05,
  type = "term",
  n = 5,
  benefit = 100000,
  n_sim = 25,
  seed = 123
)
```

---

simulate_lifetime *Simulate future lifetimes from a life table*

Description

Simulates curtate and, optionally, complete future lifetimes from a life table containing one-year death probabilities.

Usage

```
simulate_lifetime(
  lt,
  x,
  n_sim = 10000L,
  x_col = "x",
  qx_col = "qx",
  method = c("inverse", "multinomial", "antithetic"),
  frac = c("udd", "cml", "balducci", "none", "constant_force"),
```

```
  seed = NULL,
  include_distribution = FALSE,
  truncation = c("conditional", "error"),
  tol = 1e-10
)
```

Arguments

`lt`	A data frame or tibble containing the life table.
`x`	Nonnegative integer initial actuarial age. The age must appear explicitly in the column selected by `x_col`.
`n_sim`	Positive integer number of simulations.
`x_col`	Character scalar naming the age column.
`qx_col`	Character scalar naming the one-year death-probability column.
`method`	Simulation method for the curtate future lifetime: `"inverse"`, `"multinomial"`, or `"antithetic"`.
`frac`	Fractional-age assumption used inside the year of death: `"udd"`, `"cml"`, `"balducci"`, or `"none"`. The legacy value `"constant_force"` is accepted as an alias for `"cml"`.
`seed`	Optional nonnegative integer seed. When supplied, the caller's random-number state is restored after simulation.
`include_distribution`	Logical scalar. If `TRUE`, the probability distribution used for simulation is attached as a list-column.
`truncation`	Treatment of a life table whose available death probabilities do not exhaust the lifetime distribution. With `"conditional"`, the historical behavior is retained and the probabilities are normalized conditional on death within the available ages, with a warning. With `"error"`, the simulation stops.
`tol`	Nonnegative numeric tolerance used for age-grid and probability checks.

Details

For a life aged x, the curtate future lifetime has probability mass function

$$P(K_x = k) = {}_kp_x q_{x+k}.$$

The selected life-table ages must therefore start at x and proceed in consecutive one-year steps. Missing or duplicated ages are rejected because the survival recursion would otherwise no longer represent ${}_kp_x$.

Under UDD, the fractional part of T_x is uniform conditional on the year of death. Under CML, it follows the conditional constant-force distribution. Under Balducci, it follows the corresponding conditional Balducci distribution. Positive annual survival is retained even below `tol`; only exact zero survival gives the established zero fractional lifetime under CML and Balducci. Tolerances do not replace a positive probability by zero.

Value

A tibble with one row per simulation and standardized columns including `sim_id`, `Kx`, `curtate_lifetime`, `Tx`, `complete_lifetime`, `death_age`, `method`, and `frac`. The logical column `distribution_conditioned` indicates whether a truncated table was normalized conditionally.

References

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

See Also

`simulate_lifetimes`, `mc_insurance`, `mc_annuity`, `mc_premium`, `mc_loss`, `mc_reserve`

Other monte-carlo: `mc_annuity()`, `mc_insurance()`, `mc_loss()`, `mc_multilife_status()`, `mc_premium()`, `mc_reserve()`, `simulate_lifetimes()`, `summary_mc()`

Examples

```
lt <- tibble::tibble(
  x = 40:43,
  qx = c(0.10, 0.20, 0.30, 1)
)

lt |>
  simulate_lifetime(
    x = 40,
    n_sim = 25,
    method = "inverse",
    frac = "udd",
    seed = 123
  )

lt |>
  simulate_lifetime(
    x = 40,
    n_sim = 25,
    method = "antithetic",
    frac = "cml",
    seed = 123
  )
```

`simulate_lifetimes` *Simulate future lifetimes for multiple lives*

Description

Simulates independent curtate and complete future lifetimes for several initial ages from one life table. The function uses simulate_lifetime as its single-life simulation engine so that age validation, truncation treatment, fractional-age assumptions, and random-number conventions remain consistent across the Monte Carlo module.

Usage

```
simulate_lifetimes(
  data,
  x,
  n_sim = 1000L,
  frac = c("udd", "cml", "balducci", "constant", "cfm", "constant_force"),
  method = c("inverse", "multinomial", "antithetic"),
  seed = NULL,
  truncation = c("conditional", "error"),
  tol = 1e-10
)
```

Arguments

data	A data frame or tibble containing a life table.
x	Nonempty numeric vector of nonnegative integer initial ages. Each position represents one life; repeated ages are allowed.
n_sim	Positive integer number of simulations per life.
frac	Fractional-age assumption. Canonical values are "udd", "cml", and "balducci". The historical values "constant", "cfm", and "constant_force" are accepted as aliases for "cml".
method	Simulation method passed to simulate_lifetime: "inverse", "multinomial", or "antithetic".
seed	Optional nonnegative integer seed. When supplied, one random stream is initialized for the complete multiple-life simulation and the caller's previous random-number state is restored afterward.
truncation	Treatment of a mortality table that does not exhaust the lifetime distribution: "conditional" retains the historical conditional simulation with one warning, while "error" stops.
tol	Nonnegative numeric tolerance used for age and mortality checks.

Details

The life table must contain an age column named x, age, or Age, and at least one mortality basis among qx, px, or lx. The precedence is qx, then px, then lx.

Unlike the previous implementation, invalid mortality values are not silently replaced by zero and the last death probability is not forcibly overwritten. Such repairs can materially change the simulated lifetime distribution.

If `lx` is supplied, one-year death probabilities are derived as

$$q_y = 1 - \frac{l_{y+1}}{l_y}$$

for ages with positive exposure. A terminal zero in `lx` therefore generates the appropriate terminal `qx = 1`. If the final available `lx` remains positive, the resulting distribution is recognized as truncated and handled according to `truncation`. Positive exposure means `lx > 0`, independently of the table radix. Death probabilities use the direct decrement $(l_y - l_{y+1})/l_y$, avoiding loss from subtracting a rounded survival ratio from one.

Conditional on the supplied life table, the simulated lives are independent. A single seed is set once for the entire call; the seed is not reset for each life, which avoids introducing artificial perfect dependence between lives of the same age.

Value

A tibble with one row per simulation and life. Standard columns include `sim_id`, `simulation_id`, `life_id`, `x`, `Kx`, `curtate_lifetime`, `Tx`, `complete_lifetime`, `death_age`, `method`, and `frac`. Compatibility aliases `sim`, `life`, `K`, and `T` are retained.

See Also

`simulate_lifetime`, `mc_multilife_status`, `mc_insurance`, `mc_annuity`, `mc_reserve`

Other monte-carlo: `mc_annuity()`, `mc_insurance()`, `mc_loss()`, `mc_multilife_status()`, `mc_premium()`, `mc_reserve()`, `simulate_lifetime()`, `summary_mc()`

Examples

```
lt <- tibble::tibble(
  x = 40:43,
  qx = c(0.10, 0.20, 0.30, 1)
)

simulate_lifetimes(
  data = lt,
  x = c(40, 41),
  n_sim = 25,
  frac = "udd",
  seed = 123
)
```

`simulate_severity` *Simulate individual claim severities*

Description

Samples independent nonnegative claim amounts from an actuarial severity specification, including atoms at zero when present. A realization is one individual claim amount, not an aggregate or a fitted severity estimate.

Usage

```
simulate_severity(x, n_sim = 1000L, seed = NULL)
```

Arguments

x	A [severity_spec()], [empirical_severity()], or compatible payment severity specification.
n_sim	Positive integer number of simulations.
seed	Optional nonnegative integer seed; preserves the caller's RNG state when supplied.

Details

A missing RNG provider produces an explicit error. Zero severities are retained; no implicit conversion from per-loss to per-payment basis takes place.

Value

A tibble with ‘simulation‘ and ‘claim_amount‘ columns.

Examples

```
simulate_severity(severity_spec("exponential", rate = 0.002),
                  n_sim = 5, seed = 20)
```

---

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, using compact actuarial notation.

Usage

```
sinking_fund_schedule(
  principal,
  n,
  i,
  j,
  k = 1L,
  i_type = "effective",
  j_type = "effective",
  m = 1L,
  j_m = 1L,
  deposit = NULL,
  tol = 1e-08
)
```

Arguments

principal	Numeric scalar. Initial loan amount.
n	Positive integer. Number of schedule periods.
i	Numeric scalar. Annual interest-rate input for the loan.
j	Numeric scalar. Annual accumulation-rate input for the sinking fund.
k	Positive integer. Number of schedule periods per year.
i_type	Character string indicating the loan interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
j_type	Character string indicating the sinking-fund accumulation-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer. Conversion frequency for nominal loan rates. Validated for every rate family, although it affects conversion only for nominal rates.
j_m	Positive integer. Conversion frequency for nominal sinking-fund rates. Validated for every rate family, although it affects conversion only for nominal rates.
deposit	Optional numeric scalar. Level sinking-fund deposit per period. If NULL, it is computed so that the fund accumulates to principal at the end of period n (n / k years).
tol	Finite positive numeric tolerance in monetary units used for 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.

This function follows the compact actuarial notation used throughout tidyactuarial: i is the loan interest rate, j is the sinking-fund accumulation rate, k is the number of schedule periods per year, m is the conversion frequency for the loan rate, and j_m is the conversion frequency for the sinking-fund rate.

The supplied annual rate specifications are converted to effective annual rates and then to effective rates per schedule period:

$$i_p = (1 + i_e)^{1/k} - 1, \qquad j_p = (1 + j_e)^{1/k} - 1.$$

If deposit is NULL, the unit-payment accumulation sum is evaluated stably from the original log fund rate. Only an exactly zero rate uses the zero-rate limit principal / n. The monetary balance tolerance does not determine whether a rate is zero. The existing requirement of finite annual and period equivalents is retained. Non-finite monetary ledger entries cause an explicit error. A user-supplied deposit is not adjusted to force exact redemption: any uncovered loan balance or fund surplus remains in the corresponding final column and triggers the existing mismatch warning when it exceeds tol. Redemption from the fund is an internal transfer and is not counted again in the borrower's external cash flow. Storage grows with n; feasible terms are limited by integer indexing and available memory.

Otherwise, the standard sinking-fund formula is used:

$$deposit = \frac{principal}{s_{\overline{n}|j_p}}$$

where

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

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.

i_effective_loan_annual Equivalent annual effective loan rate.

i_effective_fund_annual Equivalent annual effective fund rate.

i_loan_period Effective loan rate per schedule period.

i_fund_period Effective fund rate per schedule period.

k Schedule frequency.

See Also

amort_schedule, s_angle

Other amortization: amort_schedule(), amort_schedule_general()

Examples

```
sinking_fund_schedule(
  principal = 100000,
  n = 12,
  i = 0.12,
  j = 0.096,
  i_type = "nominal_interest",
  j_type = "nominal_interest",
  m = 12,
  j_m = 12,
```

```
  k = 12
)

sinking_fund_schedule(
  principal = 50000,
  n = 10,
  i = 0.02,
  j = 0,
  deposit = NULL
)
```

`soa08lt` *SOA Illustrative Life Table*

Description

This dataset is intended for reproducible examples, internal validation, and benchmark tests for life-contingency functions such as `annuity_x`, `insurance_x`, `premium_x`, `reserve_x`, `annuity_xy`, `insurance_xy`, `premium_xy`, and `reserve_xy`.

Usage

```
soa08lt
```

Format

A tibble with one row per integer age and the following columns:

x Integer actuarial age.

lx Number of lives surviving to exact age `x`.

dx Number of deaths between exact ages `x` and `x + 1`.

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

px One-year probability of survival from exact age `x` to `x + 1`.

Details

A tidy version of the Society of Actuaries illustrative life table commonly used in life-contingencies examples and benchmark calculations.

The table is included as a convenient benchmark table for actuarial calculations. The build script in `data-raw/soa08lt.R` constructs this tidy dataset from the SOA illustrative actuarial table object distributed in the `lifecontingencies` package.

The column names already follow the compact actuarial notation used in `tidyactuarial`: `x` for age, `lx` for lives, `dx` for deaths, `qx` for one-year death probability, and `px` for one-year survival probability.

Source

Society of Actuaries illustrative life table, commonly referenced in Bowers et al. (1997), *Actuarial Mathematics*, Appendix 2A.

References

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

Spedicato, G. A. (2013). The `lifecontingencies` package: performing financial and actuarial mathematics calculations in R.

Examples

```
data(soa08lt)
head(soa08lt)

annuity_x(
  lt = soa08lt,
  x = 65,
  i = 0.06,
  timing = "due"
)
```

`solve_value` *Solve a scalar equation of value*

Description

Solves for one unknown scalar argument of an existing actuarial or financial function. The remaining arguments are supplied through ‘args‘, and the unknown is chosen with ‘solve_for‘.

Usage

```
solve_value(
  fn,
  solve_for,
  target,
  args = list(),
  result = NULL,
  interval = NULL,
  start = NULL,
  derivative = NULL,
  method = c("auto", "uniroot", "newton"),
  multiple = c("error", "all", "first"),
  tol = 1e-10,
  maxiter = 1000L,
```

```
  scan_points = 401L,
  max_expand = 6L,
  output = c("value", "summary", "audit"),
  residual_tol = tol
)
```

Arguments

fn	A function, or the name of a function, whose scalar result is to be matched to ‘target‘.
solve_for	Character scalar. Name of the scalar argument to solve for.
target	Finite numeric scalar. Desired value of the selected result.
args	Named list containing all known arguments passed to ‘fn‘. Do not supply a non-‘NULL‘ value for the argument named in ‘solve_for‘.
result	Optional result extractor. Use ‘NULL‘ when ‘fn‘ returns a numeric scalar; a character scalar naming a column or list element; or a function that extracts a numeric scalar from the object returned by ‘fn‘.
interval	Optional finite numeric vector of length 2. It defines the admissible search interval. When omitted, a conservative interval is inferred from ‘solve_for‘, ‘target‘, and the numeric values in ‘args‘.
start	Optional finite numeric scalar used as the initial value for Newton–Raphson. It is also used as a fallback starting point when ‘method = "auto"‘ cannot bracket a root.
derivative	Optional function of one argument returning the derivative of the equation residual with respect to the unknown. When omitted under Newton–Raphson, a central finite-difference derivative is used.
method	Character scalar. One of ‘"auto"‘, ‘"uniroot"‘, or ‘"newton"‘. The default first searches for bracketed roots and uses Newton–Raphson only when no bracket is found and ‘start‘ is available.
multiple	Character scalar controlling multiple roots found inside the interval: ‘"error"‘, ‘"all"‘, or ‘"first"‘.
tol	Positive finite numeric scalar controlling unknown-coordinate precision: passed to ‘uniroot‘, used relatively for Newton corrections, and used when identifying duplicate roots.
maxiter	Positive integer. Maximum number of iterations.
scan_points	Positive integer. Number of points used to detect sign changes and possible multiple roots.
max_expand	Nonnegative integer. Maximum number of automatic interval expansions when an interval was inferred and a bracket is not found initially. An interval supplied explicitly by the user is never expanded.
output	Character scalar. One of ‘"value"‘, ‘"summary"‘, or ‘"audit"‘.
residual_tol	Positive finite numeric scalar. Absolute residual tolerance in the units of the selected result of ‘fn‘. Defaults to ‘tol‘.

Details

The equation solved is

$$g(x) = f(\ldots, x, \ldots) - \text{target} = 0,$$

where ‘f‘ is the function supplied in ‘fn‘ and ‘x‘ is the argument named in ‘solve_for‘.

This function is intended to reuse the stable calculation functions already available in tidyactuarial. It does not reproduce their actuarial formulas.

Newton iteration accepts an exact zero immediately. Otherwise, convergence requires both a small residual and a small Newton correction relative to the unknown: `abs(dx) <= tol * max(1, abs(x))`. The residual must satisfy `abs(achieved - target) <= residual_tol`. A nonzero derivative is not rejected merely because its units make its absolute magnitude small. A step clipped to an interval boundary does not by itself demonstrate convergence.

The audit field ‘converged‘ requires successful solver termination and the explicit residual tolerance, without a hidden numerical floor. Exhausting ‘uniroot‘ iterations is an error. A bracketed approximation can satisfy coordinate precision while failing residual precision; ‘"value"‘ and ‘"summary"‘ still return that candidate, so use ‘"audit"‘ to inspect this distinction. Newton retains its error when it cannot meet both criteria.

When changing the units of ‘fn‘ and ‘target‘ by a positive factor, scale ‘residual_tol‘ by the same factor and retain ‘tol‘. For example, a root in years with a value in currency has a temporal coordinate tolerance and a monetary residual tolerance. No automatic scale is inferred from ‘target‘.

During interval scanning, only exact computed zeros are accepted directly as grid roots. Other candidates require a sign-changing bracket and are refined numerically. A small residual alone does not establish a root; tangential roots may require an appropriate Newton starting value.

‘solve_value()‘ is restricted to one scalar unknown. It is appropriate for quantities such as an effective interest rate, a payment, a principal, a present value, or another continuous scalar parameter.

Bracketed root finding is preferred because it is generally more stable than an unconstrained Newton iteration. The interval is scanned before solving; therefore, possible multiple roots can be detected. This is important for non-monotone equations of value.

Discrete unknowns, such as an integer number of payments, should not be solved by pretending they are continuous. A dedicated discrete solver or a model-specific wrapper should be used for those cases.

Value

Depending on ‘output‘:

- ‘"value"‘: numeric scalar or vector containing the solution(s).
- ‘"summary"‘: compact tibble with the unknown, solution, target, achieved value, residual, and method.
- ‘"audit"‘: detailed tibble with convergence diagnostics and search information.

See Also

[stats::uniroot()], [irr_flow()], [irr_flow_multi()], [bond_ytm()], [present_value()], [a_angle()]

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

Examples

```
# Solve for the annual effective rate
solve_value(
  fn = present_value,
  solve_for = "i",
  target = 8000,
  args = list(
    C = 10000,
    t = 5,
    i_type = "effective"
  ),
  interval = c(0, 0.20)
)

# Solve for a level annuity payment. Because a_angle(tidy = TRUE)
# returns a tibble, select the present_value column explicitly.
solve_value(
  fn = a_angle,
  solve_for = "payment",
  target = 100000,
  args = list(
    n = 10,
    i = 0.05,
    timing = "immediate",
    tidy = TRUE
  ),
  result = "present_value",
  interval = c(0, 20000),
  output = "summary"
)

# Newton--Raphson for a simple custom equation
solve_value(
  fn = function(x) x^2,
  solve_for = "x",
  target = 2,
  start = 1,
  interval = c(0, 2),
  method = "newton"
)

# A term in years, allowing one cent of monetary residual
solve_value(
  fn = function(t) 1000 * 1.05^t,
  solve_for = "t", target = 1500, interval = c(0, 20),
  tol = 1e-10, residual_tol = 0.01, output = "audit"
)
```

`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`, using compact actuarial notation.

Usage

```
standardize_interest(i_type = "effective", i, m = 1, ...)
```

Arguments

`i_type`	Character vector indicating the interest-rate type. Must be one of `"effective"`, `"nominal_interest"`, `"nominal_discount"`, or `"force"`.
`i`	Numeric vector of interest-rate values.
`m`	Positive integer vector giving the conversion frequency for nominal rates. Ignored for `"effective"` and `"force"`.
`...`	Transitional compatibility for older internal calls using `type =` and `rate =`. These names are accepted and mapped to `i_type` and `i`.

Details

This function follows the compact actuarial notation used throughout `tidyactuarial`: `i` denotes the interest-rate value, `i_type` denotes the interest-rate type, and `m` denotes the conversion frequency for nominal rates.

The conversion formulas are:

effective: $i = i$ (identity)

nominal_interest: $i_e = (1 + j^{(m)}/m)^m - 1$

nominal_discount: $i_e = (1 - d^{(m)}/m)^{-m} - 1$

force: $i_e = e^{\delta} - 1$

Input vectors must have length 1 or a common length. Conversions use log1p and expm1 to retain precision near zero. A missing rate type produces a missing equivalent rate. At extreme inputs, an annual equivalent can round to -1 or overflow to Inf. This is a limit of its numeric representation; interval factors and basic cash-flow valuations retain the original log accumulation rate.

Value

Numeric vector of annual effective rates. Missing values are propagated.

See Also

`interest_equivalents`, `discount_factor_spot`

Other interest: `accumulation_factor()`, `bootstrap_spot_curve()`, `discount_factor()`, `discount_factor_spot()`, `forward_curve()`, `forward_rate()`, `interest_equivalents()`, `par_rate()`, `yield_curve()`

Examples

```
# Scalar cases
standardize_interest(i_type = "nominal_interest", i = 0.18, m = 4)
standardize_interest(i_type = "nominal_discount", i = 0.10, m = 12)
standardize_interest(i_type = "force", i = 0.12)

# Vectorized case
standardize_interest(
  i_type = c("nominal_interest", "force", "effective"),
  i = 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,
    i = c(0.05, 0.08, 0.10),
    i_type = c("force", "nominal_interest", "nominal_discount"),
    m = c(1, 4, 12)
  )

  dplyr::mutate(
    portfolio,
    i_effective = standardize_interest(
      i_type = i_type,
      i = i,
      m = m
    )
  )
}
```

---

stop_loss_premium *Net stop-loss premium*

Description

Computes E[(X-d)+], the expected excess loss over retention d, with no loading or discounting. This is a per-risk expectation for X. Equivalently, the premium is the integral of the survival function from the retention to infinity, provided the expectation is finite.

Usage

```
stop_loss_premium(x, retention = 0, value = NULL)
```

Arguments

x	A nonnegative loss sample or simulation tibble, a supported severity specification, or an aggregate-distribution result.
retention	Finite nonnegative retention amount(s).
value	Optional name of a loss column when ‘x‘ is a data frame.

Details

Closed forms are available for exponential, gamma, lognormal, Weibull, and Pareto II severity. Burr XII, log-logistic, and bounded custom distributions with a survival provider use explicitly identified numerical integration of the survival function. Finite discrete models and observations are summed exactly over their mass distributions. A missing aggregate tail forbids an exact stop-loss expectation; no renormalization or zero assumption is made. The Pareto II premium is infinite when shape <= 1. Per-loss exponential severities transformed by [apply_coverage()] use closed forms for ordinary and franchise deductibles, including unlimited coverage. This avoids numerical integration over very large limits.

Value

A tibble with ‘retention‘, ‘premium‘, and ‘method‘.

Examples

```
stop_loss_premium(c(0, 100, 200, 500), retention = c(0, 150))
stop_loss_premium(severity_spec("exponential", rate = 0.01), retention = 50)
```

---

summary_mc *Summarise Monte Carlo simulation output*

Description

Computes tidy summary statistics from simulated actuarial values, using a column-oriented interface consistent with the Monte Carlo functions in `tidyactuarial`.

Usage

```
summary_mc(
  .data = NULL,
  col_value = "present_value",
  by = NULL,
  probs = c(0.025, 0.5, 0.975),
  var_probs = c(0.95, 0.99),
  na_rm = TRUE,
  ...
)
```

Arguments

.data	A data.frame or tibble containing simulation output.
col_value	Character string. Name of the numeric column to summarise. Defaults to "present_value".
by	Optional character vector of grouping columns. If supplied, the summary is computed separately within each group. If by = NULL and .data is already grouped with dplyr::group_by(), the current grouping structure is used.
probs	Numeric vector of probabilities for quantiles. Use numeric() to omit these columns.
var_probs	Numeric vector of probabilities for VaR and TVaR. Use numeric() to omit these columns.
na_rm	Logical scalar. If TRUE, missing values are removed before computing statistics.
...	Transitional compatibility for older calls using data, value_col, and group_cols.

Details

This function is intentionally generic. It can summarise simulated present values from annuities, insurances, premiums, reserves, losses, or any other numeric actuarial indicator stored in a tidy simulation table.

This function follows the compact column-naming convention used throughout the Monte Carlo layer of tidyactuarial: column arguments are prefixed with col_. Thus, col_value identifies the simulated actuarial value to summarise.

The function computes the number of simulations used, number of missing values, mean, variance, standard deviation, standard error, minimum, maximum, selected quantiles, empirical VaR, and empirical TVaR.

Empirical VaR uses the lower quantile (the inverse empirical CDF; type 1), without interpolation between observed loss amounts. In contrast, the descriptive ‘q...‘ columns retain the default interpolated quantiles from [stats::quantile()].

TVaR uses the mean of the upper ‘1 - p‘ fraction of the empirical loss distribution, with the appropriate fraction of any probability mass at VaR. Equivalently, for ‘0 <= p < 1‘ and ‘v = VaR_p(X)‘, ‘TVaR_p(X) = v + E[(X - v)_+] / (1 - p)‘. For ‘p = 1‘, the finite empirical upper-endpoint limit (‘max(x)‘) is reported. At ‘p = 0‘, TVaR is the empirical mean. Negative simulated values are permitted because ‘summary_mc()‘ is a generic numeric summary. TVaR is evaluated from ordered empirical probability masses, avoiding subtraction of extreme losses. A fractional boundary atom is retained. Variance, standard deviation and error are evaluated independently in scaled form when squaring loses range: variance may be zero or infinite while the deviation or error remains representable. Undefined moments for nonfinite observations remain missing; extended upper tails can be infinite. A sample smaller than two has missing variance, deviation and error.

The column se_mean is the usual $s/\sqrt{n}$ estimate under independent, identically distributed observations. It is not a general Monte Carlo error estimate for antithetic, clustered, weighted or otherwise dependent samples. A method label alone does not identify independent sampling units after filtering, grouping or reordering, so pairs are not inferred automatically. For known independent antithetic pairs, summarise their pair means instead. An odd unpaired observation or unequal weights requires an error calculation appropriate to that sampling design.

Quantile labels retain the existing rounding to thousandths of probability; probabilities within a selector must yield distinct labels. Empty grouped inputs retain the statistical columns with zero output rows. An ungrouped empty effective sample retains one row of counts and missing statistics.

If `na_rm = TRUE`, missing values in `col_value` are removed before computing statistics. If `na_rm = FALSE` and missing values are present, most numerical statistics are returned as `NA_real_`, avoiding accidental silent deletion of missing reserve or loss scenarios.

Value

A tibble with summary statistics.

See Also

Other monte-carlo: `mc_annuity()`, `mc_insurance()`, `mc_loss()`, `mc_multilife_status()`, `mc_premium()`, `mc_reserve()`, `simulate_lifetime()`, `simulate_lifetimes()`

Examples

```
sim <- tibble::tibble(
  sim_id = 1:5,
  t = c(0, 0, 1, 1, 1),
  L_t = c(10, 12, 8, 15, 11)
)

summary_mc(sim, col_value = "L_t")
summary_mc(sim, col_value = "L_t", by = "t")

# Transitional compatibility with older argument names
summary_mc(sim, value_col = "L_t", group_cols = "t")
```

---

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 $x + t$:

$$ {}_tE_x = v^t \cdot {}_tp_x. $$

Usage

```
t_Ex(
  lt,
  x,
  t,
  i,
```

```
  i_type = "effective",
  m = 1L,
  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. Fractional durations are allowed and are handled through t_px.
i	Annual interest-rate input(s).
i_type	Character vector indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
m	Positive integer vector giving the conversion frequency for nominal rates. Ignored for "effective" and "force".
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, i_type, m, i_effective, frac, and nEx.
check	Logical. If TRUE, performs validity checks.
tol	Numeric tolerance for integer checks on x.

Details

This function follows the compact actuarial notation used throughout tidyactuarial: lt is the life table, x is the age, t is the duration, i is the interest-rate input, i_type is the interest-rate type, and m is the conversion frequency for nominal rates.

The pure endowment is a fundamental building block in life contingency mathematics. It serves as the actuarial discount factor, combining financial discounting with mortality:

$$ {}_tE_x = v^t \cdot {}_tp_x. $$

Discounted products use the original log accumulation rate, retaining representable values when the annual effective equivalent loses range. This is an actuarial present value, not a probability: it can exceed one with negative interest rates and is not capped. The equivalent rate in tidy output may round to -1 or overflow, without being used for valuation. In ordinary notation the discount factor is:

$$ v = (1 + i_e)^{-1}. $$

Key identities involving ${}_nE_x$:

- Actuarial accumulated value: $\ddot{s}_{x:\overline{n}|} = \ddot{a}_{x:\overline{n}|} / {}_nE_x$.

- Endowment insurance decomposition: $A_{x:\overline{n}|} = A^1_{x:\overline{n}|} + {}_nE_x$.
- Deferred annuity: ${}_{n|}\ddot{a}_x = {}_nE_x \cdot \ddot{a}_{x+n}$.

For a constant force of mortality μ and force of interest δ:

$$ {}_nE_x = e^{-n(\mu+\delta)}. $$

The variance of the pure endowment random variable is:

$$ \mathrm{Var}(Z) = v^{2n} \cdot {}_np_x \cdot {}_nq_x. $$

Value

Numeric vector of ${}_tE_x$, or a tibble if `tidy = TRUE`.

See Also

`t_px` for survival probabilities without discounting, `insurance_x` for insurance APVs, `annuity_x` for life annuity APVs.

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)

# Constant force of interest
lt_exp <- lifetable(x = 0:50, lx = 100000 * exp(-0.05 * (0:50)))
t_Ex(
  lt_exp,
  x = 30,
  t = 10,
  i = 0.10,
  i_type = "force"
)
exp(-1.5)

# Nominal annual interest rate convertible monthly
t_Ex(
  lt,
  x = 0,
  t = 3,
  i = 0.06,
  i_type = "nominal_interest",
  m = 12
)
```

```
# 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(x = 0:4, t = c(5, 4, 3, 2, 1)) |>
    dplyr::mutate(pure_endow = t_Ex(lt, x = x, t = t, 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): ${}_sp_y = \frac{p_y}{1-(1-s)\times q_y}$

Missing annual survival probabilities are derived directly from adjacent survivor counts, avoiding cancellation in $1-q_y$. CF and Balducci retain logarithmic survival when that count ratio underflows. Supplied `px` values remain in use; input columns should describe a consistent annual life table.

If $x + t > \omega$ (the terminal age), the function returns 0 by the established table-horizon convention, including open tables. This is truncation, not inference about survival beyond observed ages; extend the table before evaluating a longer horizon. Interval tables with nonannual widths from `km_lifetable` require annual conversion first.

Value

Numeric vector of ${}_tp_x$, or a tibble if `tidy = TRUE`.

`t_pxy` *Two-life survival probability for independent lives*

Description

Computes the survival probability for two independent lives with actuarial ages `x` and `y` at time 0 under a joint-life or last-survivor status.

Usage

```
t_pxy(lt, x, y, t, frac, status = c("joint", "last"))
```

Arguments

`lt`	Either: • a single life table data frame, used for both lives; or • a list of two life tables `list(lt_x, lt_y)`, one for each life. Each life table must contain columns `x` and `lx`.
`x`	Integer actuarial age of the first life at time 0.
`y`	Integer actuarial age of the second life at time 0.
`t`	Nonnegative time (may be fractional).
`frac`	Fractional-age assumption: `"UDD"`, `"CF"`, `"CML"` (alias of CF), or `"Balducci"`. If not specified and the supplied life table(s) carry a `frac` attribute, that value is used. If two tables are supplied and their `frac` attributes differ, `frac` must be supplied explicitly. Uses the same assumptions as `t_px`. Inherited `"CML"` is normalized to `"CF"` before comparing the two tables' assumptions.
`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 ${}_tp_y$ use observed `lx` ratios and fractional survival under UDD, constant force, or Balducci (Finan, Section 24). Logarithmic calculations retain small annual survival without subtracting a rounded death probability from one. Last-survivor survival is evaluated as $\max(p_x, p_y) + (1 - \max(p_x, p_y)) \min(p_x, p_y)$. Tables must have unique consecutive annual integer ages and finite, nonnegative, nonincreasing `lx`. Supplied px/qx must lie in [0,1] and agree with each other and observed annual `lx` ratios within `1e-10`; unknown probabilities may be `NA`. Ages must fit R's integer range and durations must be finite. The established `t_px` horizon and zero-duration conventions apply; survival beyond the last recorded age is a truncation, not extrapolation.

Value

A single numeric value.

See Also

`t_px` for single-life survival, `e_xy` for joint-life expectancy, `annuity_xy` for two-life annuity APVs, `insurance_xy` for two-life insurance APVs.

Examples

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

# Joint life, 2.5 years, UDD (Finan, Sec. 56)
t_pxy(lt, x = 60, y = 62, t = 2.5, frac = "UDD", status = "joint")

# Verify: joint = product of marginals
t_px(lt, x = 60, t = 2.5, frac = "UDD") *
  t_px(lt, x = 62, t = 2.5, frac = "UDD")

# Last survivor, 2.5 years, constant force (Finan, Sec. 57)
t_pxy(lt, x = 60, y = 62, t = 2.5, frac = "CF", status = "last")

# Same result if the same table is supplied explicitly for both lives
t_pxy(list(lt, lt), x = 60, y = 62, t = 2.5, frac = "UDD", status = "joint")

# Different life tables for the two lives
```

```
lt_m <- data.frame(
  x  = 60:66,
  lx = c(100000, 98500, 96800, 94800, 92400, 89500, 86000)
)
lt_f <- data.frame(
  x  = 60:66,
  lx = c(100000, 99000, 97800, 96400, 94700, 92700, 90300)
)
t_pxy(list(lt_m, lt_f), x = 60, y = 62, t = 2.5, frac = "UDD", status = "joint")

# Finan Example 56.2 style: integer survival
# 10_p_{50:60} = 10_p_50 * 10_p_60
lt_ilt <- data.frame(
  x  = 50:70,
  lx = c(8950901, 8879913, 8804189, 8723382, 8637048,
         8544731, 8445974, 8340310, 8227261, 8106334,
         7977338, 7839775, 7693040, 7536522, 7369603,
         7191658, 7002051, 6800139, 6585264, 6356752,
         6114913)
)
t_pxy(lt_ilt, x = 50, y = 60, t = 10, status = "joint")

# Finan Problem 56.1: t_q_xy = t_q_x + t_q_y - t_q_x * t_q_y
p_joint <- t_pxy(lt, x = 60, y = 62, t = 3, status = "joint")
q_joint <- 1 - p_joint
qx <- 1 - t_px(lt, x = 60, t = 3)
qy <- 1 - t_px(lt, x = 62, t = 3)
c(q_joint = q_joint, q_sum = qx + qy - qx * qy)  # should match
```

`t_qx` *t-year death probability from a life table*

Description

Computes the t-year death probability

$$ {}_tq_x = \Pr[T(x) \le t] = 1 - {}_tp_x $$

using an annual life table, allowing for fractional ages under standard actuarial assumptions.

Usage

```
t_qx(lt, x, t, frac, tidy = FALSE, check = TRUE, tol = 1e-10)
```

Arguments

`lt`	A lifetable object as produced by `lifetable`. Must contain columns `x` and `lx`. Columns `qx` or `px` are used if present.
`x`	Integer age(s) at which the interval starts.

t	Nonnegative numeric duration(s) in years (can be fractional).
frac	Fractional-age assumption: "UDD", "CF", "CML" (alias of CF), or "Balducci". If not specified and lt carries a frac attribute (set by lifetable), that value is used. Passed directly to t_px.
tidy	Logical. If TRUE, returns a tibble with columns x, t, frac, tqx.
check	Logical. If TRUE, performs validity checks.
tol	Numeric tolerance for integer checks on x.

Details

Uses t_px for input and horizon conventions; the identity is:

$$ {}_tq_x = 1 - {}_tp_x. $$

Small fractional losses are formed directly before subtracting from one, preserving probabilities whose survival rounds to one.

The identity ${}_tq_x = {}_td_x/\ell_x$ (Finan, Section 22) holds for integer t, where ${}_td_x = \ell_x - \ell_{x+t}$ is the expected number of deaths between ages x and $x + t$.

For fractional durations, the result depends on the chosen assumption (UDD, CF, or Balducci); see t_px for details and formulas (Finan, Section 24).

The deferred death probability ${}_{n|}q_x$ can be obtained as (Finan, Section 23.4):

$$ {}_{n|}q_x = {}_np_x \cdot q_{x+n} = {}_{n+1}q_x - {}_nq_x. $$

Value

Numeric vector of ${}_tq_x$, or a tibble if tidy = TRUE.

See Also

t_px for the complementary survival probability, t_Ex for the pure endowment (discounted survival), e_x for life expectancy, lifetable for building the life table input.

Examples

```
x  <- 0:5
lx <- c(100000, 99500, 99000, 98200, 97000, 95000)
lt <- lifetable(x = x, lx = lx, omega = 5, close = TRUE)

# Integer death probability (Finan, Section 22)
t_qx(lt, x = 0, t = 3)  # (l0 - l3) / l0

# t = 0 always returns 0
t_qx(lt, x = 0, t = 0)

# Fractional age under UDD (Finan, Section 24.1)
t_qx(lt, x = 0, t = 2.5, frac = "UDD")

# Finan Example 22.2a: number of deaths between ages 2 and 5
```

```
# 3_d_2 = l_2 - l_5 = 98995 - 97468 = 1527
lt_22 <- lifetable(
  x = 0:5,
  lx = c(100000, 99499, 98995, 98489, 97980, 97468)
)
t_qx(lt_22, x = 2, t = 3) * lt_22$lx[lt_22$x == 2]  # 1527

# Deferred death probability (Finan, Section 23.4):
# 2|1_q_0 = 2_p_0 * q_2 = 3_q_0 - 2_q_0
t_qx(lt, x = 0, t = 3) - t_qx(lt, x = 0, t = 2)

# Vectorized with tidy output
t_qx(lt, x = c(0, 1), t = c(1.5, 2.2), frac = "Balducci", tidy = TRUE)

# Use in a tidy pipeline
if (requireNamespace("dplyr", quietly = TRUE)) {
  tibble::tibble(age = c(0, 1, 2), duration = c(3, 2.5, 1.7)) |>
    dplyr::mutate(
      surv  = t_px(lt, x = age, t = duration),
      death = t_qx(lt, x = age, t = duration)
    )
}
```

`t_qxj` *t-year probability of decrement by cause j: t_qxj*

Description

Computes ${}_tq_x^{(j)}$, the probability that a life aged `x` decrements by a specific cause `j` within `t` years, using a multiple decrement table produced by `md_table`.

Usage

```
t_qxj(md, x, t, cause, frac = NULL, tidy = FALSE, check = TRUE, tol = 1e-10)
```

Arguments

`md`	A multiple decrement table produced by `md_table`. Must contain columns `x`, `p_total`, and the requested `cause`.
`x`	Numeric vector. Starting age(s) (integer-valued).
`t`	Numeric vector. Time horizon(s) in years (t >= 0). Can be non-integer if `frac` is supplied.
`cause`	Character. Name of the cause column in `md` (e.g., `"q_death"`).
`frac`	Character. Fractional-age assumption for non-integer `t`: one of `"UDD"`, `"CF"`, `"Balducci"`, or `NULL`. If `NULL` (default), `t` must be integer-valued.
`tidy`	Logical. If `TRUE`, returns a tibble with columns `x`, `t`, `cause`, `frac`, and `tqxj`.
`check`	Logical. If `TRUE`, performs input validation (default `TRUE`).
`tol`	Numeric tolerance for integer checks (default `1e-10`).

Details

Let $q_x^{(j)}$ be the annual decrement probability for cause j and $q_x^{(\tau)}$ be the total decrement probability. For integer t,

$$ {}_tq_x^{(j)} = \sum_{k=0}^{t-1} \left(\prod_{r=0}^{k-1} p_{x+r}^{(\tau)} \right) q_{x+k}^{(j)}. $$

For non-integer $t = n+s$ with $n = \lfloor t \rfloor$ and $s \in [0, 1)$, this function supports fractional-age assumptions specified by `frac` and uses the additional convention that the within-year cause proportions remain constant: ${}_sq_x^{(j)} = w_j \, {}_sq_x^{(\tau)}$ where $w_j = q_x^{(j)}/q_x^{(\tau)}$ (and 0 when $q_x^{(\tau)} = 0$).

Supported fractional-age assumptions for the total decrement:

- `"UDD"`: ${}_sq_x^{(\tau)} = sq_x^{(\tau)}$.
- `"CF"`: ${}_sp_x^{(\tau)} = (p_x^{(\tau)})^s$.
- `"Balducci"`: ${}_sq_x^{(\tau)} = \frac{sq_x^{(\tau)}}{1-(1-s)q_x^{(\tau)}}$.

Value

Numeric vector of ${}_tq_x^{(j)}$ (or tibble if `tidy=TRUE`).

Examples

```
qx_df <- tibble::tibble(
  x = 30:35,
  q_death = c(0.001, 0.0012, 0.0014, 0.0017, 0.0020, 1.0000),
  q_disability = c(0.002, 0.0021, 0.0022, 0.0023, 0.0024, 0.0000)
)
md <- md_table(qx_df, radix = 1e5, close = TRUE)
t_qxj(md, x = 30, t = 5, cause = "q_death")
t_qxj(md, x = 30, t = 2.5, cause = "q_death", frac = "CF", tidy = TRUE)
```

---

tail_value_at_risk *Tail Value at Risk (Expected Shortfall)*

Description

Calculates TVaR_p(X) = [E(X 1(X > v)) + v(F(v) - p)] / (1-p), where v = VaR_p(X). This correctly allocates only the necessary fraction of a probability atom at v to the upper tail.

Usage

```
tail_value_at_risk(x, level = 0.95, value = NULL)
```

Arguments

x	Finite nonnegative losses, a tibble of simulations, a supported severity specification, or an aggregate-distribution result.
level	Probabilities strictly between zero and one.
value	Optional loss column name for data frames.

Details

An aggregate distribution with unresolved probability mass cannot yield exact TVaR, even when its VaR is known: the missing loss expectation is not assumed to be zero. For continuous severity families with implemented stop-loss formulas, TVaR = VaR + E[(X-VaR)+]/(1-p).

Value

A tibble with ‘level‘, ‘tail_value_at_risk‘, and ‘method‘.

Examples

```
tail_value_at_risk(c(0, 100, 100, 300), level = c(0.5, 0.9))
tail_value_at_risk(severity_spec("exponential", rate = 0.01), level = 0.95)
```

---

to_payment_basis *Specify a change from per-loss to per-payment basis*

Description

Records the exact actuarial change of basis for a collective risk model whose severity is a per-loss payment specification from `apply_coverage()`. The conversion uses binomial thinning of the loss count and conditioning the payment amount on a strictly positive insurer payment.

Usage

```
to_payment_basis(x, label = NULL)
```

Arguments

x	A list-backed collective risk specification containing `frequency` (a `tidyact_frequency_spec`) and `severity` (a `tidyact_payment_spec` with `basis = "per_loss"`). The collective class `tidyact_collective_risk_spec`, as provided by `collective_risk_spec()`. The input must have `count_basis = "loss"`.
label	Optional nonempty character scalar identifying the transformed model. If `NULL`, the collective model label is inherited when present.

Details

Write N_L for the number of losses and Y_L for the insurer payment **per loss**, including zero payments. Put

$$q = P(Y_L > 0) = 1 - F_{Y_L}(0).$$

Assuming independent, identically distributed loss severities independent of N_L, the number of payments has the binomial-thinning law

$$N_P \mid N_L = n \sim Binomial(n, q).$$

For q > 0, the **per-payment** severity is

$$Y_P \stackrel{d}{=} (Y_L \mid Y_L > 0)$$

and its CDF is 0 for y <= 0 and

$$F_{Y_P}(y) = \frac{F_{Y_L}(y) - (1-q)}{q}, \quad y > 0.$$

Consequently, the probability generating functions satisfy

$$G_{N_P}(z) = G_{N_L}(1 - q + qz),$$

and the aggregate payments have the same distribution on either basis.

Poisson counts remain Poisson with mean q * lambda; binomial counts remain binomial with unchanged number of trials and probability multiplied by q. A negative binomial frequency with size = r and mu transforms to the same size and mean q * mu. For mixed Poisson models, the conditional intensity becomes q * Lambda **without** discarding the mixing variable. Other frequency families must be thinned via the general PGF rule rather than silently assumed to stay in the same family.

When q = 1, both basis representations coincide. When q = 0, payment frequency and aggregate payment are identically zero, while Y_L | Y_L > 0 is **undefined**. A fabricated point mass at zero must not be used as a per-payment severity.

The conversion is executable for registered Poisson, binomial, negative binomial, geometric and finite empirical frequencies. No unknown/custom frequency family is silently approximated. The probability of a positive payment is resolved from the exact source CDF or finite discrete masses. Continuous per-payment severities inherit only the registered exact operations of their source; unavailable moment orders remain unavailable. For negative binomial and geometric frequencies, thinning uses $p_P = p/(p + q(1-p))$. If a positive payment frequency would round to the degenerate parameter prob = 1, the conversion stops and the per-loss model should be retained; positive risk is never silently removed.

Value

For q > 0, a tidyact_payment_basis_spec that also inherits from tidyact_collective_risk_spec, with thinned frequency, conditional positive severity, count_basis = "payment", the unchanged source, and metadata. For q = 0, an explicit tidyact_zero_payment_basis_spec with no per-payment severity, since conditioning on zero-probability events is undefined. This degenerate result is not a collective risk specification.

References

Klugman, S. A., Panjer, H. H., and Willmot, G. E. (2012). *Loss Models: From Data to Decisions*, 4th ed., Appendix C.

Bowers, N. L., Gerber, H. U., Hickman, J. C., Jones, D. A., and Nesbitt, C. J. (1997). *Actuarial Mathematics*, 2nd ed., Chapter 12.

See Also

`coverage_spec()`, `apply_coverage()`

Examples

```
# Fully executable Poisson collective model with an ordinary deductible.
f <- frequency_spec("poisson", lambda = 4)
x <- severity_spec("discrete", values = c(50, 150, 250),
                   probs = c(0.2, 0.5, 0.3))
paid <- x |> apply_coverage(coverage_spec(deductible = 100))
model <- collective_risk_spec(f, paid)
by_payment <- model |> to_payment_basis()
by_payment$count_basis
by_payment$thinning$probability
aggregate_moments(by_payment, order = 1:2)
```

---

value_at_risk *Value at Risk for actuarial losses*

Description

Computes the lower quantile VaR at level p: the smallest x for which F_X(x) >= p. No interpolation across probability atoms; for discrete claims, the quantile is an observed support point, not an interpolated monetary amount.

Usage

```
value_at_risk(x, level = 0.95, value = NULL)
```

Arguments

`x`	A vector of observed losses, a tibble of simulations, a severity specification with an exact quantile, or an [aggregate_distribution()] result. A zero-payment specification is also accepted.
`level`	Probability levels strictly between zero and one.
`value`	Optional character name of a loss column in a data frame. By default, 'aggregate_loss', 'claim_amount', 'claim_count', 'loss', or 'payment' is selected (in that order).

Details

The tail probability retained by an aggregate-distribution object is never renormalized away. Quantiles in its uncomputed tail are rejected. For sampled losses the definition is empirical, not a model-based claim. FFT count truncation can omit probability at small losses as well as large losses. If the omitted count mass could move the selected quantile to a smaller amount, VaR is rejected; recompute with a smaller 'tol'.

Value

A tibble with 'level', 'value_at_risk', and 'method'.

Examples

```
value_at_risk(c(0, 100, 100, 300), level = c(0.5, 0.9))
value_at_risk(severity_spec("exponential", rate = 0.01), level = 0.95)
```

---

yield_curve *Validate a yield curve and compute discount factors*

Description

Builds a tibble-first representation of a discrete yield curve and computes the corresponding spot discount factors, using compact actuarial notation.

Usage

```
yield_curve(
  .data = NULL,
  t = NULL,
  i = NULL,
  col_t = "t",
  col_i = "i",
  i_type = "effective",
  m = 1L,
  plot = FALSE,
  .out = "v",
  .out_plot = "yield_curve_plot",
  .keep = c("all", "used", "none"),
  .na = c("propagate", "error", "drop")
)
```

Arguments

.data	A data frame or tibble. If NULL, t and i must be supplied as numeric vectors.
t	Numeric vector of maturities in years when .data = NULL.
i	Numeric vector of spot-rate values when .data = NULL.

col_t	Name of the list-column containing maturities.
col_i	Name of the list-column containing spot rates.
i_type	Character vector indicating the spot-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force". May have length 1 or the same length as each curve.
m	Positive integer vector giving the conversion frequency for nominal spot rates. May have length 1 or the same length as each curve.
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	Missing-value handling policy: "propagate", "error", or "drop".

Details

Each row is treated as one curve. For tibble input, col_t and col_i must identify list-columns of equal-length numeric vectors. When .data = NULL, t and i 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.

This function follows the compact actuarial notation used throughout tidyactuarial: t denotes maturity, i denotes the spot rate, i_type denotes the interest-rate type, and m denotes the conversion frequency for nominal spot rates.

Discount factors use the original log accumulation rate under the supplied convention. The reported annual effective equivalent may round to -1 or overflow to Inf while a finite discount factor remains recoverable. Final factors remain subject to double precision range. With .na = "propagate", an incomplete row keeps its position and has a NULL plot entry when plotting is requested.

Value

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

References

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

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

See Also

forward_rate, discount_factor_spot, standardize_interest

Other interest: accumulation_factor(), bootstrap_spot_curve(), discount_factor(), discount_factor_spot(), forward_curve(), forward_rate(), interest_equivalents(), par_rate(), standardize_interest()

Examples

```
# Simple example
res <- yield_curve(
  t = c(1, 2, 3, 4, 5),
  i = c(0.040, 0.045, 0.048, 0.050, 0.051),
  plot = TRUE
)

res$yield_curve_plot[[1]]

# Multiple curves in a tibble
curves <- tibble::tibble(
  curve_id = c("A", "B"),
  t = list(c(1, 2, 3), c(1, 3, 5)),
  i = list(c(0.04, 0.05, 0.06), c(0.03, 0.035, 0.04))
)

res2 <- yield_curve(
  curves,
  col_t = "t",
  col_i = "i",
  plot = TRUE,
  .out = "v",
  .out_plot = "curve_plot"
)

res2$curve_plot[[2]]

# Nominal annual spot rates convertible semiannually
yield_curve(
  t = c(1, 2, 3),
  i = c(0.05, 0.055, 0.06),
  i_type = "nominal_interest",
  m = 2
)
```

Index

∗ **amortization**
amort_schedule, 21
amort_schedule_general, 25
sinking_fund_schedule, 234
∗ **annuities**
a_angle, 4
annuity_arith, 28
annuity_geom, 31
s_angle, 218
∗ **bonds**
accrued_interest, 7
bond_book_value, 49
bond_callable_price, 52
bond_convexity, 56
bond_duration, 58
bond_price, 61
bond_ytm, 63
portfolio_convexity, 190
portfolio_duration, 193
∗ **datasets**
bonds_sample, 66
cash_flows_sample, 69
loans_sample, 144
mortality_colombia_tables, 170
mortality_world_sample_2015_2023, 175
mortality_world_sample_2023, 176
multiple_decrement_sample, 178
soa08lt, 237
∗ **immunization**
immunize_duration, 111
immunize_duration_convexity, 114
plot_immunization_gap, 187
∗ **interest**
accumulation_factor, 10
bootstrap_spot_curve, 67
discount_factor, 78
discount_factor_spot, 81
forward_curve, 101
forward_rate, 103
interest_equivalents, 130
par_rate, 179
standardize_interest, 241
yield_curve, 259
∗ **life-contingencies**
add_insurance, 13
add_premium_schedule, 14
annuity_multi, 34
annuity_x, 37
annuity_xy, 41
insurance_variable_k, 117
insurance_x, 120
insurance_xy, 126
life_contract, 139
premium_gross, 195
premium_x, 198
premium_xy, 202
reserve_x, 212
reserve_xy, 216
simulate_annuity_x, 225
simulate_insurance_x, 228
∗ **life-tables**
mortality_law_table, 171
∗ **monte-carlo**
mc_annuity, 147
mc_insurance, 152
mc_loss, 156
mc_multilife_status, 159
mc_premium, 161
mc_reserve, 164
simulate_lifetime, 229
simulate_lifetimes, 231
summary_mc, 244
∗ **multiple-decrements**
insurance_xj, 124
∗ **simulation**
simulate_annuity_x, 225
simulate_insurance_x, 228

∗ **time-value**
accumulation_factor, 10
discount_factor, 78
future_value, 107
fv_flow, 109
irr_flow, 132
irr_flow_multi, 134
plot_cash_flow, 181
present_value, 205
pv_flow, 210
solve_value, 238

a_angle, 4, *24*, *27*, *30*, *33*, *220*
a_angle(), *30*, *33*, *220*
A_xj *(*insurance_xj*)*, 124
accrued_interest, 7
accrued_interest(), *51*, *54*, *58*, *60*, *63*, *65*, *192*, *194*
accumulation_factor, 10, *80*
accumulation_factor(), *68*, *80*, *82*, *102*, *105*, *108*, *110*, *131*, *133*, *136*, *181*, *184*, *206*, *212*, *240*, *242*, *261*
add_insurance, 13, *201*, *204*, *215*
add_insurance(), *15*, *36*, *39*, *43*, *119*, *122*, *129*, *140*, *197*, *201*, *204*, *215*, *218*, *226*, *229*
add_premium_schedule, 14, *201*, *204*, *215*
add_premium_schedule(), *14*, *36*, *39*, *43*, *119*, *122*, *129*, *140*, *197*, *201*, *204*, *215*, *218*, *226*, *229*
aggregate_distribution, 15
aggregate_moments, 19
aggregate_moments(), *18*
amort_schedule, 21, *26, 27*, *236*
amort_schedule(), *27*, *236*
amort_schedule_general, *23*, 25
amort_schedule_general(), *24*, *236*
annuity_arith, 28, *33*
annuity_arith(), *6*, *33*, *220*
annuity_geom, 31
annuity_geom(), *6*, *30*, *220*
annuity_multi, 34
annuity_multi(), *14, 15*, *39*, *43*, *119*, *122*, *129*, *140*, *197*, *201*, *204*, *215*, *218*, *226*, *229*
annuity_x, *36*, 37, *43*, *119*, *122*, *143*, *197*, *200, 201*, *215*, *237*, *248*
annuity_x(), *14, 15*, *36*, *43*, *119*, *122*, *129*, *140*, *186*, *197*, *201*, *204*, *215*, *218*, *226*, *229*
annuity_xy, 41, *96*, *129*, *197*, *204*, *237*, *251*
annuity_xy(), *14, 15*, *36*, *39*, *119*, *122*, *129*, *140*, *197*, *201*, *204*, *215*, *218*, *226*, *229*
apply_coverage, 44
apply_coverage(), *256*, *258*
apv_life_flow, 46
apv_life_flow(), *183*, *185, 186*

bond_book_value, 49, *54*, *58*, *60*, *63*
bond_book_value(), *9*, *54*, *58*, *60*, *63*, *65*, *192*, *194*
bond_callable_price, 52, *63*, *65*
bond_callable_price(), *9*, *51*, *58*, *60*, *63*, *65*, *192*, *194*
bond_cash_flows, *51*, *54*, 55, *58*, *60*, *63*, *65*
bond_cash_flows(), *9*, *68*, *181*
bond_convexity, *51*, 56, *60*, *63*, *65*, *113*, *116*, *188*, *192*, *194*
bond_convexity(), *9*, *51*, *54*, *60*, *63*, *65*, *192*, *194*
bond_duration, *51*, *57*, 58, *58*, *63*, *65*, *113*, *116*, *188*, *192*, *194*
bond_duration(), *9*, *51*, *54*, *58*, *63*, *65*, *192*, *194*
bond_price, *51*, *54*, *58*, *60*, 61, *65*
bond_price(), *9*, *51*, *54*, *58*, *60*, *65*, *181*, *192*, *194*
bond_ytm, *54*, *58*, *60*, 63, *63*, *133*
bond_ytm(), *9*, *51*, *54*, *58*, *60*, *63*, *179*, *192*, *194*
bonds_sample, 66
bootstrap_spot_curve, 67
bootstrap_spot_curve(), *12*, *80*, *82*, *102*, *105*, *131*, *180, 181*, *242*, *261*

cash_flows_sample, 69
collective_risk_spec, 70
collective_risk_spec(), *18, 19*, *21*, *88*, *256*
commutation_table, 72
convexity_cash_flow, 74
coverage_spec, 76
coverage_spec(), *45, 46*, *258*

discount_factor, 78
discount_factor(), *12*, *68*, *82*, *102*, *105*, *108*, *110*, *131*, *133*, *136*, *181*, *184*, *206*, *212*, *240*, *242*, *261*

discount_factor_spot, *80*, 81, *105*, *131*, *242*, *261*
discount_factor_spot(), *12*, *68*, *80*, *102*, *105*, *131*, *180*, *181*, *242*, *261*
discretize_severity, 83
discretize_severity(), *16*, *18*
distribution_convolution, 86
distribution_convolution(), *85*
distribution_moments, 89
distribution_moments(), *20*, *21*, *72*, *98*, *100*
duration_cash_flow, 91

e_x, 93, *96*, *143*, *253*
e_xy, 95, *174*, *251*
empirical_frequency, 97
empirical_frequency(), *71*, *72*, *100*
empirical_severity, 99
empirical_severity(), *71*, *72*

forward_curve, 101
forward_curve(), *12*, *68*, *80*, *82*, *105*, *131*, *181*, *242*, *261*
forward_rate, 103, *261*
forward_rate(), *12*, *68*, *80*, *82*, *101*, *102*, *131*, *181*, *242*, *261*
frequency_spec, 106
frequency_spec(), *71*, *72*, *88*, *91*, *98*, *223*
future_value, *12*, 107, *110*, *206*, *220*
future_value(), *12*, *80*, *110*, *133*, *136*, *184*, *206*, *212*, *240*
fv_flow, *108*, 109, *212*
fv_flow(), *12*, *80*, *108*, *133*, *136*, *184*, *206*, *212*, *240*

geom_step, *190*
ggplot2::scale_x_date(), *185*

immunize_duration, 111, *116*, *188*
immunize_duration(), *116*, *188*
immunize_duration_convexity, *113*, 114, *188*
immunize_duration_convexity(), *113*, *188*
insurance_variable_k, 117, *122*, *201*
insurance_variable_k(), *14*, *15*, *36*, *39*, *43*, *122*, *129*, *140*, *197*, *201*, *204*, *215*, *218*, *226*, *229*
insurance_x, *39*, *119*, 120, *129*, *143*, *201*, *215*, *237*, *248*
insurance_x(), *14*, *15*, *36*, *39*, *43*, *119*, *129*, *140*, *186*, *197*, *201*, *204*, *215*, *218*, *226*, *229*
insurance_xj, 124
insurance_xy, *43*, *122*, 126, *204*, *237*, *251*
insurance_xy(), *14*, *15*, *36*, *39*, *43*, *119*, *122*, *140*, *197*, *201*, *204*, *215*, *218*, *226*, *229*
interest_equivalents, 130, *242*
interest_equivalents(), *12*, *68*, *80*, *82*, *102*, *105*, *181*, *242*, *261*
irr_flow, 132, *135*, *136*, *212*
irr_flow(), *12*, *80*, *108*, *110*, *136*, *184*, *206*, *212*, *240*
irr_flow_multi, *133*, 134
irr_flow_multi(), *12*, *80*, *108*, *110*, *133*, *184*, *206*, *212*, *240*

km_lifetable, 136, *143*, *189*, *190*

life_contract, *37*, *41*, *127*, 139, *199*, *201*, *202*, *204*, *213*, *215*
life_contract(), *14*, *15*, *36*, *39*, *43*, *119*, *122*, *129*, *186*, *197*, *201*, *204*, *215*, *218*, *226*, *229*
lifetable, *37*, *73*, *93*, *94*, *121*, *138*, 141, *146*, *174*, *247*, *249*, *252*, *253*
loans_sample, 144
lt_tau, *126*, 146

mc_annuity, 147, *154*, *157–159*, *161–164*, *168*, *231*, *233*
mc_annuity(), *154*, *159*, *161*, *164*, *168*, *231*, *233*, *246*
mc_insurance, *150*, 152, *159*, *161*, *162*, *164*, *168*, *231*, *233*
mc_insurance(), *150*, *159*, *161*, *164*, *168*, *231*, *233*, *246*
mc_loss, *150*, *154*, 156, *164*, *168*, *231*
mc_loss(), *150*, *154*, *161*, *164*, *168*, *231*, *233*, *246*
mc_multilife_status, *147*, *148*, *150*, *152–154*, 159, *233*
mc_multilife_status(), *150*, *154*, *159*, *164*, *168*, *231*, *233*, *246*
mc_premium, *150*, *154*, *158*, *159*, 161, *168*, *231*
mc_premium(), *150*, *154*, *159*, *161*, *168*, *231*, *233*, *246*

mc_reserve, *150*, *154*, *159*, *161*, 164, *164*, *231*, *233*
mc_reserve(), *150*, *154*, *159*, *161*, *164*, *231*, *233*, *246*
md_table, *124, 125*, *146*, 168, *254*
mortality_colombia_tables, 170
mortality_law_table, 171
mortality_world_sample_2015_2023, 175
mortality_world_sample_2023, 176
multiple_decrement_sample, 178

par_rate, 179
par_rate(), *12*, *68*, *80*, *82*, *102*, *105*, *131*, *242*, *261*
plot_cash_flow, 181
plot_cash_flow(), *12*, *80*, *108*, *110*, *133*, *136*, *185, 186*, *206*, *212*, *240*
plot_contingent_cash_flow, 184
plot_contingent_cash_flow(), *48*
plot_immunization_gap, 187
plot_immunization_gap(), *113*, *116*
plot_km, *138*, 189
portfolio_convexity, 190, *194*
portfolio_convexity(), *9*, *51*, *54*, *58*, *60*, *63*, *65*, *194*
portfolio_duration, *192*, 193
portfolio_duration(), *9*, *51*, *54*, *58*, *60*, *63*, *65*, *192*
premium_gross, 195, *201*
premium_gross(), *14, 15*, *36*, *39*, *43*, *119*, *122*, *129*, *140*, *201*, *204*, *215*, *218*, *226*, *229*
premium_x, *39*, *122*, *158, 159*, *164*, *195–197*, 198, *204*, *213*, *215*, *237*
premium_x(), *14, 15*, *36*, *39*, *43*, *119*, *122*, *129*, *140*, *197*, *204*, *215*, *218*, *226*, *229*
premium_xy, *43*, *129*, *158, 159*, *164*, *195–197*, *201*, 202, *237*
premium_xy(), *14, 15*, *36*, *39*, *43*, *119*, *122*, *129*, *140*, *197*, *201*, *215*, *218*, *226*, *229*
present_value, *6*, *12*, *24*, *80*, *82*, *108*, 205, *212*
present_value(), *12*, *80*, *108*, *110*, *133*, *136*, *184*, *212*, *240*
print.tidyact_aggregate_distribution, 207
print.tidyact_collective_risk_spec, 208
print.tidyact_convolution_spec, 208
print.tidyact_frequency_spec, 209
print.tidyact_payment_spec, 209
print.tidyact_severity_spec, 210
pv_flow, *24*, *80*, *82*, *110*, *133*, *136*, *206*, 210
pv_flow(), *12*, *80*, *108*, *110*, *133*, *136*, *184*, *206*, *240*

reserve_x, *39*, *122*, *158*, *168*, 212, *237*
reserve_x(), *14, 15*, *36*, *39*, *43*, *119*, *122*, *129*, *140*, *197*, *201*, *204*, *218*, *226*, *229*
reserve_xy, *158*, *168*, *204*, 216, *237*
reserve_xy(), *14, 15*, *36*, *39*, *43*, *119*, *122*, *129*, *140*, *197*, *201*, *204*, *215*, *226*, *229*

s_angle, *6*, *30*, *33*, 218, *236*
s_angle(), *6*, *30*, *33*
severity_spec, 221
severity_spec(), *71, 72*, *85*, *88*, *91*, *100*
simulate_aggregate, 224
simulate_annuity_x, 225
simulate_annuity_x(), *14, 15*, *36*, *39*, *43*, *119*, *122*, *129*, *140*, *197*, *201*, *204*, *215*, *218*, *229*
simulate_frequency, 227
simulate_insurance_x, 228
simulate_insurance_x(), *14, 15*, *36*, *39*, *43*, *119*, *122*, *129*, *140*, *197*, *201*, *204*, *215*, *218*, *226*
simulate_lifetime, *147, 148*, *150*, *152–154*, 229, *232, 233*
simulate_lifetime(), *150*, *154*, *159*, *161*, *164*, *168*, *233*, *246*
simulate_lifetimes, *148*, *150*, *153, 154*, *160, 161*, 231, *231*
simulate_lifetimes(), *150*, *154*, *159*, *161*, *164*, *168*, *231*, *246*
simulate_severity, 233
sinking_fund_schedule, 234
sinking_fund_schedule(), *24*, *27*
soa08lt, 237
solve_value, 238
solve_value(), *12*, *80*, *108*, *110*, *133*, *136*, *184*, *206*, *212*

standardize_interest, *6*, *12*, *24*, *27*, *29, 30*, *33*, *73*, *82*, *105*, *107, 108*, *110*, *130, 131*, *206*, *211, 212*, *220*, 241, *261*
standardize_interest(), *12*, *68*, *80*, *82*, *102*, *105*, *131*, *181*, *261*
stats::dexp(), *222*
stats::dgamma(), *222*
stats::dlnorm(), *222*
stats::dweibull(), *222*
stop_loss_premium, 243
summary_mc, *149, 150*, *154*, *226*, 244
summary_mc(), *150*, *154*, *159*, *161*, *164*, *168*, *231*, *233*

t_Ex, *39*, *122*, 246, *253*
t_px, *36*, *39*, *43*, *119*, *142, 143*, *174*, *247, 248*, 249, *250, 251*, *253*
t_pxy, *43*, *95, 96*, *129*, 250
t_qx, *143*, 252
t_qxj, *126*, 254
tail_value_at_risk, 255
tibble::tibble(), *20*, *90*, *98*, *100*
to_payment_basis, 256

uniroot, *64, 65*, *132, 133*, *135*

value_at_risk, 258

yield_curve, *105*, 259
yield_curve(), *12*, *68*, *80*, *82*, *102*, *105*, *131*, *180, 181*, *242*