

Package ‘tidyactuarial’

July 27, 2026

Type Package

Title Tidy Tools for Actuarial Mathematics and Life Contingencies

Version 0.1.6

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

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

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

Suggests purrr, testthat (>= 3.0.0)

LazyData true

LazyDataCompression xz

RoxygenNote 7.3.3

Config/testthat/edition 3

URL <https://github.com/JulianFajardo1908/tidyactuarial>

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

NeedsCompilation no

Author Julian Fajardo [aut, cre]

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

Repository CRAN

Date/Publication 2026-07-27 19:30:02 UTC

Contents

accumulation_factor	4
add_insurance	6
add_premium_schedule	8
amort_schedule	10
amort_schedule_general	13
annuity_arith	16
annuity_geom	18
annuity_multi	21
annuity_x	23
annuity_xy	27
apv_life_flow	30
a_angle	32
bonds_sample	35
bond_book_value	37
bond_callable_price	39
bond_cash_flows	42
bond_convexity	44
bond_duration	46
bond_price	49
bond_ytm	51
cash_flows_sample	53
commutation_table	54
convexity_cash_flow	56
discount_factor	58
discount_factor_spot	61
duration_cash_flow	62
e_x	64
e_xy	66
forward_rate	68
future_value	70
fv_flow	72
immunize_duration	75
immunize_duration_convexity	77
insurance_variable_k	80
insurance_x	83
insurance_xj	86
insurance_xy	88
interest_equivalents	91
irr_flow	93
irr_flow_multi	95
km_lifetable	97
lifetable	99
life_contract	103
loans_sample	105
lt_tau	106
mc_annuity	107

mc_insurance	112
mc_loss	117
mc_multilife_status	119
mc_premium	121
mc_reserve	124
md_table	127
mortality_colombia_tables	129
mortality_law_table	130
mortality_world_sample_2015_2023	134
mortality_world_sample_2023	135
multiple_decrement_sample	136
plot_cash_flow	138
plot_immunization_gap	140
plot_km	142
portfolio_convexity	144
portfolio_duration	146
premium_gross	148
premium_x	151
premium_xy	155
present_value	158
p_v_flow	160
reserve_x	162
reserve_xy	165
simulate_annuity_x	167
simulate_insurance_x	170
simulate_lifetime	171
simulate_lifetimes	173
sinking_fund_schedule	175
soa08lt	178
solve_value	179
standardize_interest	182
summary_mc	184
s_angle	185
t_Ex	188
t_px	190
t_pxy	192
t_qx	194
t_qxj	195
yield_curve	197

accumulation_factor	<i>Accumulation factor under conventional or time-varying interest</i>
---------------------	--

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

<code>s</code>	Numeric vector of starting times. Defaults to 0.
<code>t</code>	Numeric vector of ending times.
<code>i</code>	Optional numeric vector of conventional interest-rate values.
<code>i_type</code>	Character vector indicating the conventional interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force". Used only when <code>i</code> is supplied.
<code>m</code>	Positive integer vector giving the conversion frequency for nominal rates. Used only when <code>i</code> is supplied.
<code>a</code>	Optional accumulation function of one numeric time argument. The function must return one positive finite numeric value for each supplied time.
<code>delta</code>	Optional force-of-interest function of one numeric time argument. Numerical integration is performed separately over each interval $[s, t]$.
<code>subdivisions</code>	Positive integer giving the maximum number of subintervals used by <code>stats::integrate()</code> when <code>delta</code> is supplied.
<code>rel.tol</code>	Positive relative tolerance passed to <code>stats::integrate()</code> when <code>delta</code> 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 $0 \leq s \leq t$. Missing times propagate as missing accumulation factors.

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: [discount_factor\(\)](#), [discount_factor_spot\(\)](#), [forward_rate\(\)](#), [interest_equivalents\(\)](#), [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)

# 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 insurance contract to an existing `life_contract()` object. This function stores the specification only; it does not calculate an actuarial present value.

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")
)

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.

Value

The original contract with an insurance component.

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\(\)](#)

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\(\)](#)

Examples

```
lt <- data.frame(
  x = 40:90,
  lx = round(100000 * exp(-0.018 * (0:50)^1.35))
)

life_contract(
  lt = lt,
  lives = "single",
  x = 40,
  i = 0.05
) |>
add_insurance(
  type = "term",
  benefit = 100000,
  n = 20
)
```

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

Description

Adds the premium-payment side of a single-life insurance contract. The schedule defines when premiums are payable while the insured is alive. This function stores the specification only; [premium_x\(\)](#) later applies the equivalence principle.

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
  )

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".

Value

The original contract with a premium_schedule component.

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\(\)](#)

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\(\)](#)

Examples

```

lt <- data.frame(
  x = 40:90,
  lx = round(100000 * exp(-0.018 * (0:50)^1.35))
)
```

```

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"
)

```

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.

<code>i</code>	Numeric scalar. Annual interest-rate input.
<code>i_type</code>	Character string indicating the annual interest-rate type: "effective", "nominal_interest", "nominal_discount", or "force".
<code>m</code>	Positive integer. Conversion frequency for nominal annual rates.
<code>k</code>	Positive integer. Number of amortization periods per year.
<code>timing</code>	Character string. One of "immediate" or "due".
<code>payment</code>	Optional numeric scalar. Initial regular payment per period. If NULL, it is computed from the loan data.
<code>extra_principal</code>	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.
<code>adjust</code>	Character string. One of "none", "term", or "payment".
<code>tol</code>	Numeric tolerance 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.

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.

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 <code>period</code>, <code>ob_start</code>, and <code>i_effective_period</code>, returning one nonnegative payment.
<code>extra_principal</code>	Optional nonnegative extra-principal payments. It may be a scalar or a numeric vector of length n . Defaults to zero.
<code>output</code>	Character scalar. Either "schedule" for the complete amortization table or "summary" for a compact actuarial summary.
<code>tol</code>	Positive numeric scalar used for zero-balance detection.

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.

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 $0, 1, \dots, n - 1$.

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

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	<i>Arithmetic annuity factor</i>
---------------	----------------------------------

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. 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" 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 $1, 2, \dots, N$.
- "decreasing": payments $N, N - 1, \dots, 1$.
- "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.

The function first converts the supplied rate to the equivalent annual effective interest rate using [standardize_interest](#).

For each scenario, the total number of payments is

$$N = nk.$$

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.

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. Ignored for "effective" and "force".
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. Ignored for "effective" and "force".
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 $1 + i_p$.

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	<i>Actuarial present value of a multi-life annuity (up to 3 independent lives)</i>
---------------	--

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")
)

```

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 <code>list(lt_male, lt_female)</code> .
----	---

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".

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):** ${}_t p_{x_1 x_2 \dots x_n} = \prod_{j=1}^n {}_t p_{x_j}$
- **Last-survivor:** ${}_t p_{x_1 x_2 \dots x_n} = 1 - \prod_{j=1}^n (1 - {}_t p_{x_j})$

For annuity = "reversionary", the APV is a weighted combination of the two statuses:

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

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"
)
```

```
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. Optional when lt is a single-life tidyact_life_contract.
i	Numeric scalar. Annual interest-rate input. 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 a one-row tibble 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,

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

For annual annuities-immediate,

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

Deferment is handled through

$$v^h {}_h p_x,$$

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:

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

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 .

Value

If `tidy = FALSE`, a numeric scalar containing the actuarial present value.

If `tidy = TRUE`, a one-row tibble with 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")

```

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 <code>list(lt_x, lt_y)</code> , or a <code>tidyact_life_contract</code> object created by life_contract . Each life table must contain columns <code>x</code> and <code>lx</code> .
x	Integer actuarial age for the first life.
y	Integer actuarial age for the second life.

<code>i</code>	Numeric scalar. Annual interest-rate input.
<code>i_type</code>	Character string indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
<code>m</code>	Positive integer. Conversion frequency for nominal rates. Ignored for <code>i_type = "effective"</code> and <code>i_type = "force"</code> .
<code>status</code>	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.
<code>benefit</code>	Optional list with numeric scalar weights both, <code>x_only</code> , and <code>y_only</code> . These weights define the payment rate according to the survival state of the two lives.
<code>n</code>	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.
<code>h</code>	Nonnegative integer deferment period in years.
<code>k</code>	Positive integer. Number of payments per year. The annuity is normalized to an annual payment rate of 1, so each individual payment has amount $1 / k$.
<code>timing</code>	Payment timing. Use "immediate" or "due".
<code>woolhouse</code>	Woolhouse approximation for $k > 1$: "none", "first", or "second".
<code>frac</code>	Fractional-age assumption for exact k-thly computation: "UDD", "CF", "CML", or "Balducci".
<code>tidy</code>	Logical scalar. If FALSE, returns a numeric APV. If TRUE, returns a one-row tibble.
<code>...</code>	Transitional compatibility for older calls using <code>mortality_table</code> , <code>age_x</code> , <code>age_y</code> , <code>rate</code> , <code>rate_type</code> , <code>cohort</code> , <code>term_years</code> , <code>deferment_years</code> , <code>payments_per_year</code> , 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

$$b_{\text{both}} {}_t p_x {}_t p_y + b_{x \text{ only}} {}_t p_x (1 - {}_t p_y) + b_{y \text{ only}} {}_t p_y (1 - {}_t p_x).$$

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:

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

Woolhouse approximations are applied only to the standard joint-life and last-survivor statuses. For a deferred annuity, the Woolhouse endpoint corrections are multiplied by the issue-date value of the status at the beginning of the payment period. This preserves the deferment factor $v^h {}_h p_{xy}$ or its last-survivor analogue.

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"
)

```

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

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

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.

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").

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"
)
```

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:

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

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)
)

```

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)
```

bond_book_value	<i>Book value of a level coupon bond at a coupon date</i>
-----------------	---

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	Numeric scalar. Tolerance used in alignment checks.
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, which is converted first to annual effective yield and then to effective yield per coupon period.

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 $C_{s+1}, \dots, C_N$, 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.

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: [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
)
```

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	Numeric scalar. Tolerance used in alignment checks.
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 $\text{call_t} * k$ 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, which is converted first to annual effective yield and then to effective yield per coupon period.

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 $j = 1, \dots, J$, 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.

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.

See Also

[bond_price](#), [bond_cash_flows](#), [bond_book_value](#), [bond_ytm](#)

Other bonds: [bond_book_value\(\)](#), [bond_convexity\(\)](#), [bond_duration\(\)](#), [bond_price\(\)](#), [bond_ytm\(\)](#), [portfolio_convexity\(\)](#), [portfolio_duration\(\)](#)

Examples

```
# Callable bond with two possible call dates
bond_callable_price(
  face = 100,
  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
)
```

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	Numeric scalar. 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.

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	Numeric scalar. Tolerance used to check maturity alignment.
check	Logical scalar. If TRUE, performs input validation.

Details

Assumptions:

- Coupons are paid in arrears at regular intervals.
- $n * k$ 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, which is converted first to annual effective yield and then to effective yield per coupon period.

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 $r = 1, \dots, N$. 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 the second-order sensitivity of the bond price to changes in the yield per period. Together with `bond_duration`, it is used in the second-order Taylor approximation of price changes.

Value

A one-row tibble with:

price Dirty price at the given yield.

discrete_convexity_periods Discrete convexity in coupon periods.

discrete_convexity_years Discrete convexity in years.

yield_per_period Effective yield per coupon period.

yield_effective_annual Annual effective yield.

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: `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	<i>Macaulay and modified duration of a level coupon bond under a flat yield</i>
---------------	---

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	Numeric scalar. Tolerance used to check maturity alignment.
check	Logical scalar. If TRUE, performs input validation.

Details

Assumptions:

- Coupons are paid in arrears at regular intervals.
- $n * k$ 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, which is converted first to annual effective yield and then to effective yield per coupon period.

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 $r = 1, \dots, N$. With $v = 1/(1 + j)$ and $P = \sum_r C_r v^r$:

Macauley duration in coupon periods:

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

Macauley 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.

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: [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	<i>Price of a level coupon bond from its yield</i>
------------	--

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	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`.

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

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 $C = \text{face} \cdot c/k$ 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: [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.

<code>c</code>	Numeric scalar. Annual coupon rate as a proportion.
<code>n</code>	Numeric scalar. Time to maturity in years. Must be strictly positive.
<code>k</code>	Positive integer. Number of coupon payments per year.
<code>R</code>	Numeric scalar. Redemption value at maturity. If NULL, defaults to face.
<code>interval</code>	Optional numeric vector of length 2 giving a bracket for the effective yield per coupon period.
<code>tol</code>	Numeric scalar. Tolerance passed to uniroot .
<code>maxiter</code>	Positive integer. Maximum number of iterations passed to uniroot .
<code>check</code>	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

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

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

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

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

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: [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
)
```

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)
```

commutation_table	<i>Build an annual commutation table (discrete ages)</i>
-------------------	--

Description

The function builds annual commutation columns from *x*, *lx*, and an interest-rate specification. The interest rate is converted internally to an annual effective rate before constructing the discount factor.

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 annual effective rate is obtained 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

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

with reverse cumulative sums 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:

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

Equivalently,

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

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

$$\frac{\Delta P_{\tau}}{P_{\tau}} \approx -D_{\text{mod},\tau} \Delta i + \frac{1}{2} C_{\tau} (\Delta i)^2.$$

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.

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"
)
```

discount_factor	<i>Discount factor under conventional or time-varying interest</i>
-----------------	--

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 <code>stats::integrate()</code> when <code>delta</code> is supplied.
rel.tol	Positive relative tolerance passed to <code>stats::integrate()</code> when <code>delta</code> is supplied.
tidy	Logical scalar. If <code>FALSE</code> , returns a numeric vector. If <code>TRUE</code> , 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.

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\(\)](#), [discount_factor_spot\(\)](#), [forward_rate\(\)](#), [interest_equivalents\(\)](#), [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.
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.

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\(\)](#), [discount_factor\(\)](#), [forward_rate\(\)](#), [interest_equivalents\(\)](#), [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
)
```

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

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

The modified duration with respect to the effective yield is

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

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.

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 <code>lifetable</code> (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 <code>lifetable</code>), 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):

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

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

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

Complete life expectancy (Finan, Section 23.3):

$$\check{e}_x = \int_0^{\omega-x} {}_t p_x dt = \frac{T_x}{\ell_x}.$$

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

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

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

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

Value

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

e_xy

*Expected future lifetime for two independent lives***Description**

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

Usage

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

Arguments

lt	A life table data frame with columns x and lx .
x	Integer actuarial age for life 1.
y	Integer actuarial age for life 2.
t	Optional nonnegative numeric duration(s). If NULL, uses the maximum horizon allowed by the table.
type	Character: "curtate" or "complete".
frac	Fractional-age assumption for type = "complete", passed to <code>t_pxy</code> : "UDD", "CF", "CML", or "Balducci". If not specified and lt carries a frac attribute, that value is used.
cohort	Two-life cohort: "first" (joint-life) or "last" (last survivor).
tidy	Logical. If TRUE, returns a tibble.
check	Logical. If TRUE, performs basic input checks.
tol	Numeric tolerance for integer checks.

Details

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

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

Complete expectation (Finan, Section 56.4):

$$\overset{\circ}{e}_{xy} = \int_0^{\infty} {}_t p_{xy} dt.$$

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

Key identity (Finan, Example 57.4):

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

This can be used to cross-validate results.

Value

Numeric vector, or tibble if tidy = TRUE.

See Also

[e_x](#) for single-life expectancy, [t_pxy](#) for two-life survival probabilities, [annuity_xy](#) for two-life annuity APVs.

Examples

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

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

# Curtate last-survivor expectancy (Finan, Sec. 57)
e_xy(lt, x = 60, y = 62, type = "curtate", cohort = "last")

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

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

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

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

forward_rate	<i>Compute an implied forward rate from a discrete spot curve</i>
--------------	---

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.

<code>m</code>	Positive integer vector giving the conversion frequency for nominal spot rates. May have length 1 or the same length as each curve.
<code>method</code>	Spot extraction method: "exact" or "linear".
<code>plot</code>	Logical; if TRUE, adds a list-column of ggplot2 objects.
<code>.out</code>	Name of the output column containing the forward rate.
<code>.out_plot</code>	Name of the output list-column containing ggplot2 objects. Used only if <code>plot = TRUE</code> .
<code>.keep</code>	One of "all", "used", or "none".
<code>.na</code>	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.

Spot-rate inputs are converted to annual effective form using [standardize_interest](#) before interpolation and forward-rate calculation.

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\(\)](#), [discount_factor\(\)](#), [discount_factor_spot\(\)](#), [interest_equivalents\(\)](#), [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
)
```

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

<code>C</code>	Numeric vector of initial payment amounts or capitals.
<code>i</code>	Numeric vector of interest-rate values.
<code>i_type</code>	Character vector indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
<code>m</code>	Positive integer vector giving the conversion frequency for nominal rates. Ignored for "effective" and "force".
<code>t</code>	Numeric vector of times in years from valuation to accumulation.
<code>tidy</code>	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](#).

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	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".
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.

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	<i>Duration-based immunization with multiple assets</i>
-------------------	---

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:

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

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

$$(P_1, \dots, P_r)$$

and

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

and

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

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 the discrete convexity 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

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

$$(P_1, \dots, P_r),$$

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

and

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

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

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

is computed.

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),
  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),
  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),
  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 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. Ignored for "effective" and "force".

frac	Fractional-age assumption used in survival probabilities: "UDD", "CF", "CML", or "Balducci". If not specified and <code>lt</code> carries a <code>frac</code> 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 $1/k, 2/k, \dots, n$.

Fractional survival probabilities are computed via `t_px` under the selected fractional-age assumption.

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.
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".

<code>m</code>	Positive integer. Conversion frequency for nominal rates. Ignored for <code>i_type = "effective"</code> and <code>i_type = "force"</code> .
<code>n</code>	Integer insurance term in years. Required for <code>type = "term"</code> and <code>type = "endowment"</code> . Use <code>Inf</code> only for whole-life insurance.
<code>h</code>	Integer deferment period in years.
<code>type</code>	Character string. One of <code>"whole"</code> , <code>"term"</code> , or <code>"endowment"</code> .
<code>benefit</code>	Numeric scalar. Benefit amount.
<code>tidy</code>	Logical scalar. If <code>FALSE</code> , returns a numeric APV. If <code>TRUE</code> , returns a one-row tibble with intermediate quantities.
<code>...</code>	Transitional compatibility for older calls using <code>mortality_table</code> , <code>age</code> , <code>rate</code> , <code>rate_type</code> , <code>term_years</code> , <code>deferral_years</code> , <code>insurance_type</code> , and <code>output</code> .

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

$$v^h {}_h p_x.$$

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

$$v^n {}_n p_x.$$

Value

If `tidy = FALSE`, a numeric scalar containing the actuarial present value.

If `tidy = TRUE`, a one-row tibble with 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
)
```

insurance_xj	<i>Cause-specific term/whole-life insurance APV under multiple decrements</i>
--------------	---

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	Annual interest-rate input. Must produce an annual effective rate greater than -1.
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.

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 <code>list(lt_x, lt_y)</code> , or a <code>tidyact_life_contract</code> object created by <code>life_contract</code> . Each life table must contain columns <code>x</code> and <code>lx</code> .
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 <code>i_type = "effective"</code> and <code>i_type = "force"</code> .
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 <code>Inf</code> 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 <code>FALSE</code> , returns a numeric APV. If <code>TRUE</code> , returns a one-row tibble.
...	Transitional compatibility for older calls using <code>mortality_table</code> , <code>age_x</code> , <code>age_y</code> , <code>rate</code> , <code>rate_type</code> , <code>insurance_type</code> , <code>cohort</code> , <code>term_years</code> , <code>deferment_years</code> , and <code>output</code> .

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.

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

For a whole-life contract:

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

For an n -year term insurance:

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

For an n -year endowment insurance:

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

A deferred insurance is valued by multiplying the value at deferred ages $x + h$ and $y + h$ by the deferment factor

$$v^h {}_hp_{xy}$$

for the selected two-life status.

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

<code>i_type</code>	Character string indicating the input interest-rate type. Must be one of "effective", "nominal_interest", "nominal_discount", or "force".
<code>i</code>	Numeric scalar giving the interest-rate value.
<code>m</code>	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.

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

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

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

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

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

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

Value

A tibble with columns:

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

notation Actuarial notation for the equivalent rate.

m 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\(\)](#), [discount_factor\(\)](#), [discount_factor_spot\(\)](#), [forward_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	<i>Internal rate of return for a cash flow</i>
----------	--

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 <code>c(-0.99, 10)</code> .
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.

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

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

Details

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

The IRRs returned are interpreted as annual effective rates.

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

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 :

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

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

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

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).
- $e_x = T_x / \ell_x$: complete life expectancy (Finan, Sec. 23.3).
- $m_x = d_x / L_x$: central death rate (Finan, Sec. 23.9).
- $a_x = (\ell_x \Delta - L_x) / d_x$: average fraction of the interval lived by those who die.

Value

A list with two tibbles:

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

See Also

[lifetable](#) for building tables from known mortality inputs, [plot_km](#) for plotting the KM curve.

Examples

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

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

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

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

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

lifetable

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

Description

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

Usage

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

Arguments

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

Details

The table can be built from exactly one of:

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

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

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

By default, the table is actuarially closed at ω :

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

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

The basic identities are (Finan, Section 22):

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

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

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

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

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

Value

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

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

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

See Also

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

Examples

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

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

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

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

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

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

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

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

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

```
attr(lt1, "ax")      # 0.5
```

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

Description

Creates a lightweight 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 <code>list(lt_x, lt_y)</code> with one table for each life. Each life table must contain columns <code>x</code> and <code>lx</code> .
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 <code>lives</code> 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 <code>i_type = "effective"</code> and <code>i_type = "force"</code> .
...	Reserved for future extensions. Deprecated argument names such as <code>mortality_table</code> , <code>age</code> , <code>age_x</code> , <code>age_y</code> , <code>rate</code> , and <code>rate_type</code> 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.

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
)
```

)

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)
```

<code>lt_tau</code>	<i>Total-decrement lifetable from a multiple decrement table: <code>lt_tau</code></i>
---------------------	---

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

<code>.data</code>	A data frame or tibble containing simulated future lifetimes, typically returned by <code>simulate_lifetime</code> or by <code>mc_multilife_status</code> when working with multiple-life statuses.
<code>i</code>	Numeric scalar. Interest-rate input used for discounting.
<code>payment</code>	Numeric scalar. Amount paid at each annuity payment date. Default is 1. This is a per-payment amount, not an annualized payment rate.
<code>k</code>	Positive integer. Number of annuity payments per year. Default is 1, corresponding to annual payments.
<code>type</code>	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".
<code>n</code>	Numeric scalar. Term of the annuity in years. Required for "temporary", "deferred_temporary", and "certain" annuities.
<code>h</code>	Numeric scalar. Deferral period in years. Default is 0. Required to be positive for "deferred" and "deferred_temporary" annuities.
<code>n_guar</code>	Numeric scalar. Guaranteed payment period in years. Required for <code>type = "guaranteed"</code> .
<code>timing</code>	Character string specifying the annuity payment timing. Available options are "immediate" and "due".
<code>i_type</code>	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".
<code>m</code>	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.
<code>col_K</code>	Character string. Name of the column containing simulated curtate future lifetimes. Default is "Kx".
<code>col_T</code>	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.
<code>col_pv</code>	Character string. Name of the output column containing simulated present values of annuity payments. Default is "pv_annuity".
<code>...</code>	Transitional compatibility for older calls using <code>data</code> , <code>rate</code> , <code>payments_per_year</code> , <code>annuity</code> , <code>term</code> , <code>deferral_years</code> , <code>guarantee_years</code> , <code>interest_type</code> , <code>k_col</code> , <code>tx_col</code> , and <code>annuity_col</code> .

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.

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.

pvalue 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.
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.

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 .

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 <code>mc_annuity</code> .
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

$$P_{\text{per payment}} = qc, \quad P^{(k)} = kqc.$$

Value

A 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	<i>Compute multiple-life simulated status variables</i>
---------------------	---

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

<code>data</code>	A data frame or tibble, typically returned by simulate_lifetimes .
<code>status</code>	Multiple-life status. Canonical values are "joint" and "last_survivor". The aliases "joint_life", "last", and "last_survivor_life" are accepted.
<code>col_sim</code>	Character scalar naming the simulation identifier column.
<code>col_life</code>	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.
<code>col_K</code>	Character scalar naming the curtate future lifetime column.
<code>col_T</code>	Character scalar naming the complete future lifetime column.
<code>tol</code>	Nonnegative numeric tolerance used when checking that each complete future lifetime is compatible with its curtate lifetime.

Details

For simulated complete future lifetimes $T_1, \dots, T_r$, the multiple-life complete future lifetime is

$$T_{\text{joint}} = \min(T_1, \dots, T_r)$$

for the joint-life status and

$$T_{\text{last}} = \max(T_1, \dots, T_r)$$

for the last-survivor status.

The corresponding curtate future lifetime is obtained with the same minimum or maximum operation on $K_1, \dots, K_r$. Because the function validates $K_j \leq T_j < K_j + 1$, the returned values satisfy the same curtate-complete relationship.

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 <code>mc_insurance</code> and <code>mc_annuity</code> 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 $\text{payment} = 1 / k$, this coefficient is the annualized premium.
by	Optional character vector of grouping columns. If supplied, the premium is estimated separately within each group. If <code>by = NULL</code> and <code>.data</code> 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 <code>payments_per_year</code> or <code>k</code> in <code>.data</code> ; if neither exists, $k = 1$ is used.
annuity_payment	Optional positive numeric scalar giving the amount paid at each date when <code>col_Y</code> was constructed. It is normally inferred from <code>payment_per_payment</code> , <code>payment</code> , or <code>payment_annualized</code> in the output of <code>mc_annuity</code> .
tol	Nonnegative numeric tolerance for consistency checks.
...	Transitional compatibility for older calls using <code>data</code> , <code>benefit_col</code> , <code>annuity_col</code> , and <code>premium_col</code> .

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

$$\hat{q} = \frac{\bar{Z}}{\overline{Y_c}}.$$

The corresponding premium amount at each payment date is

$$\hat{P}_{\text{per payment}} = \hat{q}c,$$

and the annualized premium is

$$\hat{P}^{(k)} = k\hat{q}c.$$

The recommended premium-annuity normalization uses $\text{payment} = 1 / k$ in `mc_annuity`. In that case,

$$\hat{q} = \hat{P}^{(k)}, \quad \hat{P}_{\text{per payment}} = \frac{\hat{P}^{(k)}}{k}.$$

If instead $\text{payment} = 1$, then $\hat{q}$ is the amount paid at each premium date, while the annualized premium is $k\hat{q}$.

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.

Value

A tibble containing the original simulations and standardized premium columns:

premium_annualized Estimated annualized premium $\hat{P}^{(k)}$.

premium_per_payment Amount collected at each premium date.

payments_per_year Premium payment frequency k .

premium_annuity_scale Coefficient $\hat{q}$ multiplying `col_Y`.

mc_equivalence_residual Difference $\bar{Z} - \hat{q}\bar{Y}_c$ 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.

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

<code>.data</code>	A data frame or tibble containing simulated curtate future lifetimes and, when required, complete future lifetimes.
<code>t</code>	Nonnegative numeric vector of valuation durations.
<code>i</code>	Numeric scalar. Interest-rate input used for discounting.
<code>P</code>	Optional nonnegative numeric scalar supplied directly as the premium input.
<code>col_P</code>	Character scalar naming a premium column in <code>.data</code> . The default is "P". When it is not supplied explicitly, modern standardized premium columns are preferred.
<code>premium_unit</code>	Unit of <code>P</code> or the selected premium column: "annualized", "per_payment", or "annuity_scale". With "auto", <code>premium_annualized</code> , <code>premium_per_payment</code> , and <code>premium_annuity_scale</code> determine their own units, while legacy <code>P</code> and <code>premium</code> retain their historical annuity-scale interpretation.
<code>benefit</code>	Nonnegative numeric scalar. Insurance benefit amount.
<code>payment</code>	Positive numeric scalar. Amount used at each premium-annuity payment date. For the normalized k -thly convention, use <code>payment = 1 / k</code> .
<code>k</code>	Positive integer number of premium payments per year.
<code>type</code>	Insurance type. Supported values are "whole", "term", "deferred", "deferred_term", "pure_endowment", and "endowment".
<code>annuity_type</code>	Premium-annuity type. Supported values are "whole", "temporary", "deferred", "deferred_temporary", "certain", and "guaranteed".
<code>n</code>	Optional contract or premium term in years.
<code>h</code>	Nonnegative deferral period in years.
<code>n_guar</code>	Optional guaranteed premium-payment period in years.
<code>timing</code>	Death-benefit timing: "end_of_year" or "moment_of_death".
<code>premium_timing</code>	Premium-annuity timing: "due" or "immediate".
<code>reserve_timing</code>	Whether cash flows exactly at the valuation duration are included: "before_payment" or "after_payment".
<code>i_type</code>	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:

$$\hat{q} = \frac{\bar{Z}_0}{\bar{Y}_{0,c}}.$$

This makes the sample mean issue loss equal to zero, up to numerical error, when reserve_timing = "before_payment".

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.

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

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

Details

Let the cause columns be $q_x^{(1)}, \dots, q_x^{(J)}$. 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 $\ell_{x_0} = \text{radix}$.

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	<i>Generate a tidy life table from a theoretical mortality law</i>
---------------------	--

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 assumption used to convert μ_x to q_x . 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.
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 $px = 1 - qx$.

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:** $\mu_x = (\text{shape}/\text{scale})(x/\text{scale})^{\text{shape}-1}$.
- **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 $X/\omega \sim \text{Beta}(\alpha, \beta)$.
- **HeligmanPollard:** odds model returning q_x directly.

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, and G. 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 q_x is obtained using frac:

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

Direct qx laws

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

Closure

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

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 = 0.03,
  D = 10,
  E = 20,
  F_hp = 0.00005,
  G = 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 = 0.03,
```

```

D = 10,
E = 20,
F = 0.00005,
G = 1.08
)

```

mortality_world_sample_2015_2023

World mortality sample panel, 2015–2023

Description

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

Usage

```
mortality_world_sample_2015_2023
```

Format

A tibble with 35,451 rows and 14 variables:

country Country name.

country_code Numeric ISO country code.

continent Continent.

region Geographic region.

year Calendar year, from 2015 to 2023.

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

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

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 $l_0 = 100000$.

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

source Data source.

Details

The dataset is derived from central death rates m_x . Death probabilities q_x 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 $q_x = 1$. Survivors l_x and expected deaths dx are then reconstructed recursively from $l_0 = 100000$.

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	<i>World mortality sample, 2023</i>
-----------------------------	-------------------------------------

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 $l_0 = 100000$.
dx Expected number of deaths between ages x and $x + 1$.
source Data source.

Details

The dataset is derived from central death rates m_x . Death probabilities q_x 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 $q_x = 1$. Survivors l_x and expected deaths dx are then reconstructed recursively from $l_0 = 100000$.

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)
```

plot_cash_flow	<i>Plot a cash-flow diagram</i>
----------------	---------------------------------

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.
-------	---

<code>C</code>	Numeric vector of cash flows, or a column name when <code>.data</code> is supplied.
<code>t</code>	Optional numeric vector of times in years, or a column name when <code>.data</code> is supplied.
<code>date</code>	Optional date vector, or a column name when <code>.data</code> is supplied.
<code>i</code>	Optional annual interest-rate input used to compute present value if <code>PV</code> is <code>NULL</code> .
<code>i_type</code>	Character string indicating the interest-rate type.
<code>m</code>	Positive integer. Conversion frequency for nominal rates.
<code>PV</code>	Optional numeric present value to display.
<code>payment</code>	Deprecated. Use <code>C</code> .
<code>time</code>	Deprecated. Use <code>t</code> . Kept explicit to avoid partial matching with <code>timeline_size</code> .
<code>rate</code>	Deprecated. Use <code>i</code> .
<code>pv</code>	Deprecated. Use <code>PV</code> .
<code>title</code>	Optional plot title.
<code>subtitle</code>	Optional plot subtitle.
<code>x_label</code>	Optional x-axis label.
<code>amount_label</code>	Optional y-axis label when <code>normalize = FALSE</code> .
<code>financial</code>	Logical. If <code>TRUE</code> , positive cash flows point upward and negative cash flows point downward.
<code>normalize</code>	Logical. If <code>TRUE</code> , arrow heights are normalized.
<code>aggregate</code>	Logical. If <code>TRUE</code> , cash flows occurring at the same time or date are summed before plotting.
<code>show_labels</code>	Logical. If <code>TRUE</code> , labels are shown next to cash-flow arrows.
<code>label_size</code>	Numeric text size for labels.
<code>arrow_size</code>	Numeric line width for arrows.
<code>timeline_size</code>	Numeric line width for the time axis.
<code>label_digits</code>	Integer number of decimal digits for cash-flow labels.
<code>label_format</code>	Character string controlling label notation. Use <code>"auto"</code> to abbreviate values from one million onward, <code>"full"</code> for unabridged values, or <code>"compact"</code> for abbreviated values such as 1.25M.
<code>currency</code>	Optional currency or unit prefix, such as \$.
<code>col_inflow</code>	Character color for inflow arrows.
<code>col_outflow</code>	Character color for outflow arrows.
<code>date_labels</code>	Character date-label format passed to <code>ggplot2::scale_x_date()</code> .
<code>day_count</code>	Day-count convention used when date is supplied.
<code>...</code>	Reserved for future extensions.

Details

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_immunization_gap` *Plot immunization performance under interest-rate shifts*

Description

Computes and plots the difference between the present value of liabilities and the present value of an immunized asset portfolio under small interest rate changes. This allows visual evaluation of duration or duration-convexity immunization quality.

Usage

```
plot_immunization_gap(
  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.

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.

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.

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: [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	<i>Compute portfolio duration as a market-value-weighted average</i>
--------------------	--

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

<code>.data</code>	A data.frame or tibble. If NULL, P and D must be supplied as vectors.
<code>portfolio_id</code>	Optional vector of portfolio identifiers when <code>.data = NULL</code> . If omitted, all positions are treated as belonging to a single portfolio.
<code>P</code>	Numeric vector of present values, prices, or market values when <code>.data = NULL</code> .
<code>D</code>	Numeric vector of individual durations when <code>.data = NULL</code> .
<code>col_portfolio</code>	Name of the portfolio identifier column. If NULL, all rows are treated as one portfolio.
<code>col_P</code>	Name of the numeric column containing present values, prices, or market values.
<code>col_D</code>	Name of the numeric column containing individual durations.
<code>.out</code>	Name of the output column containing portfolio duration.
<code>.out_value</code>	Name of the output column containing total portfolio value.
<code>.out_n</code>	Name of the output column containing the number of positions used in the calculation.
<code>.na</code>	NA handling policy: "propagate", "error", or "drop".
<code>...</code>	Transitional compatibility for older calls using <code>market_value</code> , <code>duration</code> , <code>col_market_value</code> , and <code>col_duration</code> . These names are mapped to <code>P</code> , <code>D</code> , <code>col_P</code> , and <code>col_D</code> .

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.

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: [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 <code>premium_x(..., output = "summary")</code> or <code>premium_xy(..., output = "summary")</code> . 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 <code>a_premiums</code> is normalized with payments of $1/k$.
k	Optional positive integer payment frequency. By default it is inferred from <code>payments_per_year</code> or <code>k</code> in <code>prem</code> ; if neither is present, <code>k = 1</code> 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 <code>output = "summary"</code> and FALSE to <code>output = "value"</code> .
check	Logical scalar. If TRUE, validates the inputs.
tol	Nonnegative numeric tolerance used for consistency checks.
...	Transitional compatibility. The deprecated argument <code>payments_per_year</code> is mapped to <code>k</code> .

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,

$$G^{(k)} = \frac{P^{(k)} + k\gamma}{(1 - \beta) - \alpha/(ka^{(k)})}.$$

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 γ 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 γ .

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.

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:

$$\text{APV}(\text{premiums}) = \text{APV}(\text{benefits}).$$

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.
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, $E[L_0] = 0$, gives

$$P^{(k)} = \frac{\text{APV}(\text{benefits})}{\text{APV}(\text{premium annuity})}.$$

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.

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 <code>list(lt_x, lt_y)</code> , or a two-life contract created with life_contract .
x	Integer actuarial age for the first life. Optional for a <code>life_contract</code> .

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

$$P^{(k)} = \frac{\text{APV}(\text{benefits})}{\text{APV}(\text{premium annuity})}.$$

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 obtained as the difference between the corresponding endowment and term insurance values.

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	<i>Present value of a single payment</i>
---------------	--

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. Ignored for "effective" and "force".
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](#).

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)
)

```

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](#).

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".

<code>t</code>	Optional integer vector of policy durations. If NULL, all supported integer durations are returned.
<code>method</code>	Reserve method: "prospective" or "recursive". The recursive method currently supports annual premiums only ($k = 1$).
<code>output</code>	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".
<code>tidy</code>	Deprecated compatibility argument. TRUE maps to <code>output = "summary"</code> and FALSE to <code>output = "value"</code> .
<code>check</code>	Logical scalar. If TRUE, validates inputs.
<code>...</code>	Transitional compatibility for older calls using <code>mortality_table</code> , <code>age</code> , <code>rate</code> , <code>rate_type</code> , <code>insurance_type</code> , <code>term_years</code> , <code>premium</code> , <code>premium_term_years</code> , and <code>durations</code> .

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

$${}_tV_x = \text{APV}_t(\text{future benefits}) - P^{(k)} \text{APV}_t(\text{future premium annuity}).$$

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

$${}_{t+1}V_x = \frac{({}_tV_x + P_t)(1 + i) - b_{t+1}q_{x+t}}{p_{x+t}}.$$

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

<code>lt</code>	One life table, <code>list(lt_x, lt_y)</code> , or a two-life <code>life_contract()</code> object.
<code>x, y</code>	Integer ages at issue. Optional for a life contract.
<code>i</code>	Annual interest-rate input. Optional for a life contract.
<code>i_type</code>	Interest-rate type.
<code>m</code>	Conversion frequency for nominal rates.
<code>type</code>	One of "whole", "term", "endowment", or "pure_endowment".
<code>status</code>	"joint" or "last".
<code>n</code>	Insurance term in years after deferment.
<code>h</code>	Nonnegative integer deferment in years.
<code>benefit</code>	Positive benefit amount.
<code>P</code>	Optional annualized premium. If NULL, <code>premium_xy()</code> is used.
<code>n_prem</code>	Premium-paying term. Fractional values require <code>n_prem * k</code> to be an integer.
<code>k</code>	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

$${}_tV = \text{APV}_t(\text{future benefits}) - P^{(k)} \text{APV}_t(\text{future premium annuity}).$$

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 . Pure endowments are valued as endowment APV minus term-insurance APV.

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()`

simulate_annuity_x	<i>Monte Carlo simulation of a life annuity</i>
--------------------	---

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

<code>lt</code>	A life table or a <code>tidyact_life_contract</code> . A life table must contain columns <code>x</code> and <code>lx</code> .
<code>x</code>	Integer actuarial age. Optional for a life contract.
<code>i</code>	Numeric scalar interest-rate input.
<code>i_type</code>	Interest-rate convention.
<code>m</code>	Positive integer nominal conversion frequency.
<code>n</code>	Positive integer term in years, or <code>Inf</code> for whole life.
<code>k</code>	Positive integer number of annuity payments per year.
<code>timing</code>	Payment timing: "due" or "immediate".
<code>payment</code>	Nonnegative amount paid at each payment date. To simulate an annualized payment rate of 1 with <code>k</code> payments per year, use <code>payment = 1 / k</code> .
<code>frac</code>	Fractional-age assumption used to simulate the complete lifetime within the year of death: "udd", "cml", or "balducci".
<code>method</code>	Simulation method. Currently only "inverse".
<code>n_sim</code>	Positive integer number of simulations.
<code>seed</code>	Optional nonnegative integer seed.
<code>output</code>	"simulations" or "summary".
<code>...</code>	Transitional compatibility for <code>mortality_table</code> , <code>age</code> , <code>rate</code> , <code>rate_type</code> , <code>term_years</code> , and <code>payments_per_year</code> .

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.

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_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.

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	<i>Simulate future lifetimes from a life table</i>
-------------------	--

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

$$P(K_x = k) = {}_k p_x q_{x+k}.$$

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 ${}_k p_x$.

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.

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	<i>Simulate future lifetimes for multiple lives</i>
--------------------	---

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

<code>data</code>	A data frame or tibble containing a life table.
<code>x</code>	Nonempty numeric vector of nonnegative integer initial ages. Each position represents one life; repeated ages are allowed.
<code>n_sim</code>	Positive integer number of simulations per life.
<code>frac</code>	Fractional-age assumption. Canonical values are "udd", "cml", and "balducci". The historical values "constant", "cfm", and "constant_force" are accepted as aliases for "cml".
<code>method</code>	Simulation method passed to <code>simulate_lifetime</code> : "inverse", "multinomial", or "antithetic".
<code>seed</code>	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.
<code>truncation</code>	Treatment of a mortality table that does not exhaust the lifetime distribution: "conditional" retains the historical conditional simulation with one warning, while "error" stops.
<code>tol</code>	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.

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
)
```

`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

<code>principal</code>	Numeric scalar. Initial loan amount.
<code>n</code>	Positive integer. Number of schedule periods.
<code>i</code>	Numeric scalar. Annual interest-rate input for the loan.
<code>j</code>	Numeric scalar. Annual accumulation-rate input for the sinking fund.
<code>k</code>	Positive integer. Number of schedule periods per year.
<code>i_type</code>	Character string indicating the loan interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
<code>j_type</code>	Character string indicating the sinking-fund accumulation-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
<code>m</code>	Positive integer. Conversion frequency for nominal loan rates. Ignored for <code>i_type = "effective"</code> and <code>i_type = "force"</code> .
<code>j_m</code>	Positive integer. Conversion frequency for nominal sinking-fund rates. Ignored for <code>j_type = "effective"</code> and <code>j_type = "force"</code> .
<code>deposit</code>	Optional numeric scalar. Level sinking-fund deposit per period. If NULL, it is computed so that the fund accumulates to <code>principal</code> at time <code>n</code> .
<code>tol</code>	Numeric tolerance used for zero checks and final-balance checks.

Details

The borrower pays:

- interest on the loan each period, and
- a level deposit into the sinking fund.

At maturity, the sinking fund is used to redeem the principal.

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:

$$i_p = (1 + i_e)^{1/k} - 1, \quad j_p = (1 + j_e)^{1/k} - 1.$$

If `deposit` is NULL and the sinking-fund periodic rate `j_p` is approximately zero, the deposit is computed as `principal / n`.

Otherwise, the standard sinking-fund formula is used:

$$deposit = \frac{principal}{s_{\bar{n}|j_p}}$$

where

$$s_{\bar{n}|j_p} = \frac{(1 + j_p)^n - 1}{j_p}.$$

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")
)

```

Arguments

<code>fn</code>	A function, or the name of a function, whose scalar result is to be matched to target.
<code>solve_for</code>	Character scalar. Name of the scalar argument to solve for.
<code>target</code>	Finite numeric scalar. Desired value of the selected result.
<code>args</code>	Named list containing all known arguments passed to <code>fn</code> . Do not supply a non-NULL value for the argument named in <code>solve_for</code> .
<code>result</code>	Optional result extractor. Use NULL when <code>fn</code> 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 <code>fn</code> .
<code>interval</code>	Optional finite numeric vector of length 2. It defines the admissible search interval. When omitted, a conservative interval is inferred from <code>solve_for</code> , <code>target</code> , and the numeric values in <code>args</code> .
<code>start</code>	Optional finite numeric scalar used as the initial value for Newton–Raphson. It is also used as a fallback starting point when <code>method = "auto"</code> cannot bracket a root.
<code>derivative</code>	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.
<code>method</code>	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 <code>start</code> is available.
<code>multiple</code>	Character scalar controlling multiple roots found inside the interval: "error", "all", or "first".
<code>tol</code>	Positive numeric scalar. Convergence and residual tolerance.
<code>maxiter</code>	Positive integer. Maximum number of iterations.
<code>scan_points</code>	Positive integer. Number of points used to detect sign changes and possible multiple roots.
<code>max_expand</code>	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.
<code>output</code>	Character scalar. One of "value", "summary", or "audit".

Details

The equation solved is

$$g(x) = f(\dots, x, \dots) - \text{target} = 0,$$

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.

`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()`, `pvalue_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"
)

```

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

<code>i_type</code>	Character vector indicating the interest-rate type. Must be one of "effective", "nominal_interest", "nominal_discount", or "force".
<code>i</code>	Numeric vector of interest-rate values.
<code>m</code>	Positive integer vector giving the conversion frequency for nominal rates. Ignored for "effective" and "force".
<code>...</code>	Transitional compatibility for older internal calls using <code>type =</code> and <code>rate =</code> . These names are accepted and mapped to <code>i_type</code> and <code>i</code> .

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.

Value

Numeric vector of annual effective rates. Missing values are propagated.

See Also

[interest_equivalents](#), [discount_factor_spot](#)

Other interest: [accumulation_factor\(\)](#), [discount_factor\(\)](#), [discount_factor_spot\(\)](#), [forward_rate\(\)](#), [interest_equivalents\(\)](#), [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
    )
  )
}
```

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

<code>.data</code>	A data.frame or tibble containing simulation output.
<code>col_value</code>	Character string. Name of the numeric column to summarise. Defaults to "present_value".
<code>by</code>	Optional character vector of grouping columns. If supplied, the summary is computed separately within each group. If <code>by = NULL</code> and <code>.data</code> is already grouped with <code>dplyr::group_by()</code> , the current grouping structure is used.
<code>probs</code>	Numeric vector of probabilities for quantiles.
<code>var_probs</code>	Numeric vector of probabilities for VaR and TVaR.
<code>na_rm</code>	Logical scalar. If TRUE, missing values are removed before computing statistics.
<code>...</code>	Transitional compatibility for older calls using <code>data</code> , <code>value_col</code> , and <code>group_cols</code> .

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.

TVaR is computed empirically as the mean of simulated values greater than or equal to the corresponding empirical VaR.

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")
```

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 time of the last payment. 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.

The future value of a perpetuity diverges, so perpetuities are not supported in `s_angle()`.

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:

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

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")
)
```

<i>t</i> _Ex	<i>Pure endowment (discounted survival): ${}_tE_x$</i>
--------------	---

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).

<code>i_type</code>	Character vector indicating the interest-rate type. Allowed values are "effective", "nominal_interest", "nominal_discount", and "force".
<code>m</code>	Positive integer vector giving the conversion frequency for nominal rates. Ignored for "effective" and "force".
<code>frac</code>	Fractional-age assumption passed to <code>t_px</code> : "UDD", "CF", "CML" (alias of CF), or "Balducci". If not specified and <code>lt</code> carries a <code>frac</code> attribute, that value is used.
<code>tidy</code>	Logical. If TRUE, returns a tibble with columns <code>x</code> , <code>t</code> , <code>i</code> , <code>i_type</code> , <code>m</code> , <code>i_effective</code> , <code>frac</code> , and <code>nEx</code> .
<code>check</code>	Logical. If TRUE, performs validity checks.
<code>tol</code>	Numeric tolerance for integer checks on <code>x</code> .

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:

$${}_tE_x = v^t \cdot {}_tp_x.$$

The interest-rate input is converted to an annual effective rate before applying the discount factor:

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

Key identities involving ${}_nE_x$:

- Actuarial accumulated value: $\ddot{s}_{x:\overline{n}|} = \ddot{a}_{x:\overline{n}|} / {}_nE_x$.
- Endowment insurance decomposition: $A_{x:\overline{n}|} = A_{x:\overline{n}|}^1 + {}_nE_x$.
- 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:

$$\text{Var}(Z) = v^{2n} \cdot {}_np_x \cdot {}_nq_x.$$

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:  ${}_3E_0 = v^3 \cdot {}_3p_0$ 
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):

$${}_n p_x = \frac{\ell_{x+n}}{\ell_x}$$

For non-integer durations, let $t = n + s$ with $n = \lfloor t \rfloor$ and $s \in [0, 1)$. Then (Finan, Section 24):

$${}_t p_x = {}_n p_x \times {}_s p_{x+n}$$

The fractional-year factor ${}_s p_y$ depends on the assumption:

- UDD (Finan, Sec. 24.1): ${}_s p_y = 1 - s \times q_y$
- CF (Finan, Sec. 24.2): ${}_s p_y = (p_y)^s$
- Balducci (Finan, Sec. 24.3): ${}_s p_y = \frac{p_y}{1 - (1-s) \times q_y}$

If $x + t > \omega$ (the terminal age), the function returns 0 since no survival is possible beyond the table's limiting age.

Value

Numeric vector of ${}_t p_x$, 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 <code>list(lt_x, lt_y)</code>, one for each life. Each life table must contain columns x and lx .
x	Integer actuarial age of the first life at time 0.
y	Integer actuarial age of the second life at time 0.
t	Nonnegative time (may be fractional).
$frac$	Fractional-age assumption: "UDD", "CF", "CML" (alias of CF), or "Balducci". If not specified and the supplied life table(s) carry a $frac$ attribute, that value is used. If two tables are supplied and their $frac$ attributes differ, $frac$ must be supplied explicitly. Passed to t_{px} .
$status$	Two-life status: "joint" or "last".

Details

Independence is assumed throughout (Finan, Sections 56–57).

Joint-life status (Finan, Section 56): the status survives as long as *both* lives are alive.

$${}_tp_{xy} = {}_tp_x \cdot {}_tp_y.$$

Last-survivor status (Finan, Section 57): the status survives as long as *at least one* life is alive.

$${}_tp_{\overline{xy}} = {}_tp_x + {}_tp_y - {}_tp_x \cdot {}_tp_y.$$

Individual survival probabilities ${}_tp_x$ and ${}_tp_y$ are computed via [\$t_{px}\$](#) , which supports fractional ages under UDD, constant force, and Balducci assumptions (Finan, Section 24).

Value

A single numeric value.

See Also

[t_px](#) for single-life survival, [e_xy](#) for joint-life expectancy, [annuity_xy](#) for two-life annuity APVs, [insurance_xy](#) for two-life insurance APVs.

Examples

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

# Joint life, 2.5 years, UDD (Finan, Sec. 56)
t_pxy(lt, x = 60, y = 62, t = 2.5, frac = "UDD", status = "joint")

# Verify: joint = product of marginals
t_px(lt, x = 60, t = 2.5, frac = "UDD") *
  t_px(lt, x = 62, t = 2.5, frac = "UDD")

# Last survivor, 2.5 years, constant force (Finan, Sec. 57)
t_pxy(lt, x = 60, y = 62, t = 2.5, frac = "CF", status = "last")

# Same result if the same table is supplied explicitly for both lives
t_pxy(list(lt, lt), x = 60, y = 62, t = 2.5, frac = "UDD", status = "joint")

# Different life tables for the two lives
lt_m <- data.frame(
  x = 60:66,
  lx = c(100000, 98500, 96800, 94800, 92400, 89500, 86000)
)
lt_f <- data.frame(
  x = 60:66,
  lx = c(100000, 99000, 97800, 96400, 94700, 92700, 90300)
)
t_pxy(list(lt_m, lt_f), x = 60, y = 62, t = 2.5, frac = "UDD", status = "joint")

# Finan Example 56.2 style: integer survival
# 10_p_{50:60} = 10_p_50 * 10_p_60
lt_ilt <- data.frame(
  x = 50:70,
  lx = c(8950901, 8879913, 8804189, 8723382, 8637048,
        8544731, 8445974, 8340310, 8227261, 8106334,
        7977338, 7839775, 7693040, 7536522, 7369603,
        7191658, 7002051, 6800139, 6585264, 6356752,
        6114913)
)
t_pxy(lt_ilt, x = 50, y = 60, t = 10, status = "joint")

# Finan Problem 56.1: t_q_xy = t_q_x + t_q_y - t_q_x * t_q_y
p_joint <- t_pxy(lt, x = 60, y = 62, t = 3, status = "joint")
q_joint <- 1 - p_joint
qx <- 1 - t_px(lt, x = 60, t = 3)
qy <- 1 - t_px(lt, x = 62, t = 3)
```

```
c(q_joint = q_joint, q_sum = qx + qy - qx * qy) # should match
```

${}_tq_x$	<i>t</i> -year death probability from a life table
-----------	--

Description

Computes the t -year death probability

$${}_tq_x = \Pr[T(x) \leq t] = 1 - {}_tp_x$$

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

<code>lt</code>	A lifetable object as produced by lifetable . Must contain columns <code>x</code> and <code>lx</code> . Columns <code>qx</code> or <code>px</code> are used if present.
<code>x</code>	Integer age(s) at which the interval starts.
<code>t</code>	Nonnegative numeric duration(s) in years (can be fractional).
<code>frac</code>	Fractional-age assumption: "UDD", "CF", "CML" (alias of CF), or "Balducci". If not specified and <code>lt</code> carries a <code>frac</code> attribute (set by lifetable), that value is used. Passed directly to t_px .
<code>tidy</code>	Logical. If TRUE, returns a tibble with columns <code>x</code> , <code>t</code> , <code>frac</code> , <code>txq</code> .
<code>check</code>	Logical. If TRUE, performs validity checks.
<code>tol</code>	Numeric tolerance for integer checks on <code>x</code> .

Details

This is a thin wrapper around [t_px](#):

$${}_tq_x = 1 - {}_tp_x.$$

The identity ${}_tq_x = {}_td_x/\ell_x$ (Finan, Section 22) holds for integer t , where ${}_td_x = \ell_x - \ell_{x+t}$ is the expected number of deaths between ages x and $x + t$.

For fractional durations, the result depends on the chosen assumption (UDD, CF, or Balducci); see [t_px](#) for details and formulas (Finan, Section 24).

The deferred death probability ${}_n|q_x$ can be obtained as (Finan, Section 23.4):

$${}_n|q_x = {}_np_x \cdot q_{x+n} = {}_{n+1}q_x - {}_nq_x.$$

Value

Numeric vector of ${}_tq_x$, or a tibble if `tidy = TRUE`.

See Also

[t_px](#) for the complementary survival probability, [t_Ex](#) for the pure endowment (discounted survival), [e_x](#) for life expectancy, [lifetable](#) for building the life table input.

Examples

```
x <- 0:5
lx <- c(100000, 99500, 99000, 98200, 97000, 95000)
lt <- lifetable(x = x, lx = lx, omega = 5, close = TRUE)

# Integer death probability (Finan, Section 22)
t_qx(lt, x = 0, t = 3) # (10 - 13) / 10

# 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 <code>md_table</code> . Must contain columns <code>x</code> , <code>p_total</code> , and the requested cause.
x	Numeric vector. Starting age(s) (integer-valued).
t	Numeric vector. Time horizon(s) in years ($t \geq 0$). Can be non-integer if <code>frac</code> is supplied.
cause	Character. Name of the cause column in <code>md</code> (e.g., "q_death").
frac	Character. Fractional-age assumption for non-integer <code>t</code> : one of "UDD", "CF", "Balducci", or NULL. If NULL (default), <code>t</code> must be integer-valued.
tidy	Logical. If TRUE, returns a tibble with columns <code>x</code> , <code>t</code> , <code>cause</code> , <code>frac</code> , and <code>tqxj</code> .
check	Logical. If TRUE, performs input validation (default TRUE).
tol	Numeric tolerance for integer checks (default $1e-10$).

Details

Let $q_x^{(j)}$ be the annual decrement probability for cause j and $q_x^{(\tau)}$ be the total decrement probability. For integer t ,

$${}_tq_x^{(j)} = \sum_{k=0}^{t-1} \left(\prod_{r=0}^{k-1} p_{x+r}^{(\tau)} \right) q_{x+k}^{(j)}.$$

For non-integer $t = n + s$ with $n = \lfloor t \rfloor$ and $s \in [0, 1)$, this function supports fractional-age assumptions specified by `frac` and uses the additional convention that the within-year cause proportions remain constant: ${}_sq_x^{(j)} = w_j {}_sq_x^{(\tau)}$ where $w_j = q_x^{(j)} / q_x^{(\tau)}$ (and 0 when $q_x^{(\tau)} = 0$).

Supported fractional-age assumptions for the total decrement:

- "UDD": ${}_sq_x^{(\tau)} = sq_x^{(\tau)}$.
- "CF": ${}_sp_x^{(\tau)} = (p_x^{(\tau)})^s$.
- "Balducci": ${}_sq_x^{(\tau)} = \frac{sq_x^{(\tau)}}{1 - (1-s)q_x^{(\tau)}}$.

Value

Numeric vector of ${}_tq_x^{(j)}$ (or tibble if `tidy=TRUE`).

Examples

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

```
t_qxj(md, x = 30, t = 5, cause = "q_death")
t_qxj(md, x = 30, t = 2.5, cause = "q_death", frac = "CF", tidy = TRUE)
```

yield_curve

Validate a yield curve 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.

The spot-rate input is converted to annual effective form through `standardize_interest` before discount factors are computed.

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\(\)](#), [discount_factor\(\)](#), [discount_factor_spot\(\)](#), [forward_rate\(\)](#), [interest_equivalents\(\)](#), [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, 10
 - amort_schedule_general, 13
 - sinking_fund_schedule, 175
- * **annuities**
 - a_angle, 32
 - annuity_arith, 16
 - annuity_geom, 18
 - s_angle, 185
- * **bonds**
 - bond_book_value, 37
 - bond_callable_price, 39
 - bond_convexity, 44
 - bond_duration, 46
 - bond_price, 49
 - bond_ytm, 51
 - portfolio_convexity, 144
 - portfolio_duration, 146
- * **datasets**
 - bonds_sample, 35
 - cash_flows_sample, 53
 - loans_sample, 105
 - mortality_colombia_tables, 129
 - mortality_world_sample_2015_2023, 134
 - mortality_world_sample_2023, 135
 - multiple_decrement_sample, 136
 - soa08lt, 178
- * **immunization**
 - immunize_duration, 75
 - immunize_duration_convexity, 77
 - plot_immunization_gap, 140
- * **interest**
 - accumulation_factor, 4
 - discount_factor, 58
 - discount_factor_spot, 61
 - forward_rate, 68
 - interest_equivalents, 91
 - standardize_interest, 182
 - yield_curve, 197
- * **life-contingencies**
 - add_insurance, 6
 - add_premium_schedule, 8
 - annuity_multi, 21
 - annuity_x, 23
 - annuity_xy, 27
 - insurance_variable_k, 80
 - insurance_x, 83
 - insurance_xy, 88
 - life_contract, 103
 - premium_gross, 148
 - premium_x, 151
 - premium_xy, 155
 - reserve_x, 162
 - reserve_xy, 165
 - simulate_annuity_x, 167
 - simulate_insurance_x, 170
- * **life-tables**
 - mortality_law_table, 130
- * **monte-carlo**
 - mc_annuity, 107
 - mc_insurance, 112
 - mc_loss, 117
 - mc_multilife_status, 119
 - mc_premium, 121
 - mc_reserve, 124
 - simulate_lifetime, 171
 - simulate_lifetimes, 173
 - summary_mc, 184
- * **multiple-decrements**
 - insurance_xj, 86
- * **simulation**
 - simulate_annuity_x, 167
 - simulate_insurance_x, 170
- * **time-value**
 - accumulation_factor, 4
 - discount_factor, 58
 - future_value, 70

- fv_flow, 72
- irr_flow, 93
- irr_flow_multi, 95
- plot_cash_flow, 138
- present_value, 158
- pv_flow, 160
- solve_value, 179
- a_angle, 12, 14, 17, 20, 32, 187
- a_angle(), 181
- A_xj (insurance_xj), 86
- accumulation_factor, 4, 59, 62, 70, 71, 74, 92, 94, 96, 140, 159, 161, 181, 183, 198
- add_insurance, 6, 9, 23, 26, 29, 81, 84, 90, 104, 150, 154, 157, 164, 167, 169, 171
- add_premium_schedule, 8, 8, 23, 26, 29, 81, 84, 90, 104, 150, 154, 157, 164, 167, 169, 171
- amort_schedule, 10, 14, 177
- amort_schedule_general, 12, 13, 177
- annuity_arith, 16, 20, 34, 187
- annuity_geom, 17, 18, 34, 187
- annuity_multi, 8, 9, 21, 26, 29, 81, 84, 90, 104, 150, 154, 157, 164, 167, 169, 171
- annuity_x, 8, 9, 23, 23, 29, 81, 84, 90, 102, 104, 150, 153, 154, 157, 164, 167, 169, 171, 178, 189
- annuity_x(), 104
- annuity_xy, 8, 9, 23, 26, 27, 67, 81, 84, 90, 104, 150, 154, 157, 164, 167, 169, 171, 178, 193
- annuity_xy(), 104, 167
- apv_life_flow, 30
- bond_book_value, 37, 42, 45, 48, 50, 53, 145, 148
- bond_callable_price, 38, 39, 45, 48, 50, 53, 145, 148
- bond_cash_flows, 38, 42, 42, 45, 48, 50, 53
- bond_convexity, 38, 42, 44, 47, 48, 50, 53, 76, 79, 142, 145, 148
- bond_duration, 38, 42, 45, 46, 50, 53, 76, 79, 142, 145, 148
- bond_price, 38, 42, 45, 48, 49, 53, 145, 148
- bond_ytm, 38, 42, 45, 48, 50, 51, 94, 145, 148
- bond_ytm(), 181
- bonds_sample, 35
- cash_flows_sample, 53
- commutation_table, 54
- convexity_cash_flow, 56
- discount_factor, 5, 58, 62, 70, 71, 74, 92, 94, 96, 140, 159, 161, 181, 183, 198
- discount_factor_spot, 5, 59, 61, 70, 92, 183, 198
- duration_cash_flow, 62
- duration_cash_flow(), 57
- e_x, 64, 67, 102, 195
- e_xy, 66, 133, 193
- forward_rate, 5, 59, 62, 68, 92, 183, 198
- future_value, 5, 59, 70, 74, 94, 96, 140, 159, 161, 181, 187
- fv_flow, 5, 59, 71, 72, 94, 96, 140, 159, 161, 181
- geom_step, 143
- immunize_duration, 75, 79, 142
- immunize_duration_convexity, 76, 77, 142
- insurance_variable_k, 8, 9, 23, 26, 29, 80, 84, 90, 104, 150, 154, 157, 164, 167, 169, 171
- insurance_x, 8, 9, 23, 26, 29, 81, 83, 90, 102, 104, 150, 154, 157, 164, 167, 169, 171, 178, 189
- insurance_x(), 104
- insurance_xj, 86
- insurance_xy, 8, 9, 23, 26, 29, 81, 84, 88, 104, 150, 154, 157, 164, 167, 169, 171, 178, 193
- insurance_xy(), 104, 167
- interest_equivalents, 5, 59, 62, 70, 91, 183, 198
- irr_flow, 5, 59, 71, 74, 93, 96, 140, 159, 161, 181
- irr_flow(), 181
- irr_flow_multi, 5, 59, 71, 74, 94, 95, 140, 159, 161, 181
- irr_flow_multi(), 181
- km_lifetable, 97, 102, 142, 143

- life_contract, 8, 9, 23, 24, 26, 27, 29, 81, 84, 89, 90, 103, 150, 152, 154, 155, 157, 163, 164, 167, 169, 171
- life_contract(), 6, 166
- lifetable, 24, 55, 65, 83, 99, 99, 106, 107, 133, 188, 191, 194, 195
- loans_sample, 105
- lt_tau, 87, 106
- mc_annuity, 107, 115, 118, 119, 121–123, 127, 173, 175, 185
- mc_insurance, 110, 112, 119, 121–123, 127, 173, 175, 185
- mc_loss, 110, 115, 117, 121, 123, 127, 173, 175, 185
- mc_multilife_status, 108, 110, 112, 113, 115, 119, 119, 123, 127, 173, 175, 185
- mc_premium, 110, 115, 119, 121, 121, 127, 173, 175, 185
- mc_reserve, 110, 115, 119, 121, 123, 124, 173, 175, 185
- md_table, 86, 87, 106, 127, 195, 196
- mortality_colombia_tables, 129
- mortality_law_table, 130
- mortality_world_sample_2015_2023, 134
- mortality_world_sample_2023, 135
- multiple_decrement_sample, 136
- plot_cash_flow, 5, 59, 71, 74, 94, 96, 138, 159, 161, 181
- plot_immunization_gap, 76, 79, 140
- plot_km, 99, 142
- portfolio_convexity, 38, 42, 45, 48, 50, 53, 144, 148
- portfolio_duration, 38, 42, 45, 48, 50, 53, 145, 146
- premium_gross, 8, 9, 23, 26, 29, 81, 84, 90, 104, 148, 154, 157, 164, 167, 169, 171
- premium_x, 8, 9, 23, 26, 29, 81, 84, 90, 104, 118, 119, 123, 148–150, 151, 157, 163, 164, 167, 169, 171, 178
- premium_x(), 8, 104
- premium_xy, 8, 9, 23, 26, 29, 81, 84, 90, 104, 118, 119, 123, 148–150, 154, 155, 164, 167, 169, 171, 178
- premium_xy(), 104, 166, 167
- present_value, 5, 12, 34, 59, 62, 71, 74, 94, 96, 140, 158, 161, 181
- present_value(), 181
- pv_flow, 5, 12, 59, 62, 71, 74, 94, 96, 140, 159, 160, 181
- reserve_x, 8, 9, 23, 26, 29, 81, 84, 90, 104, 118, 127, 150, 154, 157, 162, 167, 169, 171, 178
- reserve_x(), 104, 167
- reserve_xy, 8, 9, 23, 26, 29, 81, 84, 90, 104, 118, 127, 150, 154, 157, 164, 165, 169, 171, 178
- reserve_xy(), 104
- s_angle, 17, 20, 34, 177, 185
- simulate_annuity_x, 8, 9, 23, 26, 29, 81, 84, 90, 104, 150, 154, 157, 164, 167, 167, 171
- simulate_insurance_x, 8, 9, 23, 26, 29, 81, 84, 90, 104, 150, 154, 157, 164, 167, 169, 170
- simulate_lifetime, 108, 110, 112, 113, 115, 119, 121, 123, 127, 171, 173–175, 185
- simulate_lifetimes, 108, 110, 113, 115, 119–121, 123, 127, 173, 173, 185
- sinking_fund_schedule, 12, 14, 175
- soa08lt, 178
- solve_value, 5, 59, 71, 74, 94, 96, 140, 159, 161, 179
- standardize_interest, 5, 12, 14, 17, 20, 34, 55, 59, 61, 62, 69–71, 73, 74, 92, 159, 161, 182, 187, 198
- stats::uniroot(), 181
- summary_mc, 109, 110, 114, 115, 119, 121, 123, 127, 169, 173, 175, 184
- t_Ex, 26, 84, 188, 195
- t_px, 23, 26, 29, 81, 100, 102, 133, 188, 189, 190, 192–195
- t_pxy, 29, 66, 67, 90, 192
- t_qx, 102, 194
- t_qxj, 87, 195
- uniroot, 52, 93–96
- yield_curve, 5, 59, 62, 70, 92, 183, 197