Package 'R4GoodPersonalFinances'

May 7, 2026

Title Make Optimal Financial Decisions

Version 1.2.0

Description Make optimal decisions for your personal or household finances. Use tools and methods that are selected carefully to align with academic consensus, bridging the gap between theoretical knowledge and practical application. They help you find your own personalized optimal discretionary spending or optimal asset allocation, and prepare you for retirement or financial independence.
The optimal solution to this problems is extremely complex, and we only have a single lifetime to get it right.
Fortunately, we now have the user-friendly tools implemented, that integrate life-cycle models with single-period net-worth mean-variance optimization models.
Those tools can be used by anyone who wants to see what highly-personalized optimal decisions can look like.
For more details see:
Idzorek T., Kaplan P. (2024, ISBN:9781952927379),
Haghani V., White J. (2023, ISBN:9781119747918).

License MIT + file LICENSE

URL https://www.r4good.academy/,
https://r4goodacademy.github.io/R4GoodPersonalFinances/,
https://github.com/R4GoodAcademy/R4GoodPersonalFinances

BugReports https://github.com/R4GoodAcademy/R4GoodPersonalFinances/issues

Depends R (>= 4.1.0)

Imports bsicons, bslib, dplyr, ggplot2, ggrepel, ggtext, gt, glue, PrettyCols, scales, shiny, tidyr, readr, fs, purrr, stringr, nloptr, cli, furrr, future, progressr, lubridate, memoise, cachem, rlang

Suggests spelling, tibble, withr, microbenchmark, testthat (>= 3.0.0), vdiffr

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first plot_*, simulate_*

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

LazyData true

NeedsCompilation no

Author Kamil Wais [aut, cre, cph, fnd] (ORCID:
<https://orcid.org/0000-0002-4062-055X>),
Olesia Wais [aut] (ORCID: <https://orcid.org/0000-0002-8741-8674>)

Maintainer Kamil Wais <kamil.wais@gmail.com>

Repository CRAN

Date/Publication 2025-11-23 11:10:02 UTC

Contents

`R4GoodPersonalFinances-package`

R4GoodPersonalFinances: Make Optimal Financial Decisions

Description

Make optimal decisions for your personal or household finances. Use tools and methods that are selected carefully to align with academic consensus, bridging the gap between theoretical knowledge and practical application. They help you find your own personalized optimal discretionary spending or optimal asset allocation, and prepare you for retirement or financial independence. The optimal solution to this problems is extremely complex, and we only have a single lifetime to get it right. Fortunately, we now have the user-friendly tools implemented, that integrate life-cycle models with single-period net-worth mean-variance optimization models. Those tools can be used by anyone who wants to see what highly-personalized optimal decisions can look like. For more details see: Idzorek T., Kaplan P. (2024, ISBN:9781952927379), Haghani V., White J. (2023, ISBN:9781119747918).

Author(s)

Maintainer: Kamil Wais <kamil.wais@gmail.com> (ORCID) [copyright holder, funder]

Authors:

- Olesia Wais <olesia.wais@gmail.com> (ORCID)

See Also

Useful links:

- https://www.r4good.academy/
- https://r4goodacademy.github.io/R4GoodPersonalFinances/
- https://github.com/R4GoodAcademy/R4GoodPersonalFinances
- Report bugs at https://github.com/R4GoodAcademy/R4GoodPersonalFinances/issues

`calc_effective_tax_rate` *Calculate Effective Tax Rate*

Description

Calculate Effective Tax Rate

Usage

```
calc_effective_tax_rate(portfolio, tax_rate_ltcg, tax_rate_ordinary_income)
```

Arguments

`portfolio`	A nested `tibble` of class `Portfolio`.
`tax_rate_ltcg`	A numeric. Tax rate for long-term capital gains.
`tax_rate_ordinary_income`	A numeric. Tax rate for ordinary income.

Value

A `portfolio` object augmented with nested columns with effective tax rates calculations.

Examples

```
portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio <-
  calc_effective_tax_rate(
    portfolio,
    tax_rate_ltcg = 0.20,
    tax_rate_ordinary_income = 0.40
  )
portfolio$aftertax$effective_tax_rate
```

`calc_gompertz_joint_parameters` *Calculating the Gompertz model parameters for joint survival*

Description

Calculating the Gompertz model parameters for joint survival

Usage

```
calc_gompertz_joint_parameters(
  p1 = list(age = NULL, mode = NULL, dispersion = NULL),
  p2 = list(age = NULL, mode = NULL, dispersion = NULL),
  max_age = 120
)
```

Arguments

p1	A list with age, mode and dispersion parameters for the first person (p1).
p2	A list with age, mode and dispersion parameters for the second person (p2).
max_age	A numeric. The maximum age for the Gompertz model.

Value

A list containing:

data	A data frame with survival rates for 'p1', 'p2', 'joint' survival, and the fitted Gompertz model
mode	The mode of the joint Gompertz distribution
dispersion	The dispersion parameter of the joint Gompertz distribution

Examples

```
calc_gompertz_joint_parameters(
  p1 = list(
    age        = 65,
    mode       = 88,
    dispersion = 10.65
  ),
  p2 = list(
    age        = 60,
    mode       = 91,
    dispersion = 8.88
  ),
  max_age = 110
)
```

---

calc_gompertz_mode *Calculate Gompertz mode for a given life expectancy*

Description

Calculate Gompertz mode for a given life expectancy

Usage

```
calc_gompertz_mode(life_expectancy, current_age, dispersion, max_age = 120)
```

Arguments

life_expectancy	A numeric. Desired life expectancy.
current_age	A numeric. Current age.
dispersion	A numeric. Dispersion of the Gompertz distribution.
max_age	A numeric. Maximum age. Defaults to 120.

Value

A numeric. Mode of the Gompertz distribution.

Examples

```
calc_gompertz_mode(
 life_expectancy = 86,
  current_age    = 25,
  dispersion     = 8.88,
  max_age        = 115
)
```

---

calc_gompertz_parameters

Calculating Gompertz model parameters

Description

Calculating Gompertz model parameters

Usage

```
calc_gompertz_parameters(
  mortality_rates,
  current_age,
  estimate_max_age = FALSE
)
```

Arguments

mortality_rates	A data frame with columns mortality_rate and age. Usually the output of read_hmd_life_tables() function or filtered data from life_tables object.
current_age	A numeric. Current age.
estimate_max_age	A logical. Should the maximum age be estimated?

Value

A list containing:

`data`	The input mortality rates data frame with additional columns like 'survival_rate' and 'probability_of_death'
`mode`	The mode of the Gompertz distribution
`dispersion`	The dispersion parameter of the Gompertz distribution
`current_age`	The current age parameter
`max_age`	The maximum age parameter

References

Blanchet, David M., and Paul D. Kaplan. 2013. "Alpha, Beta, and Now... Gamma." Journal of Retirement 1 (2): 29-45. doi:10.3905/jor.2013.1.2.029.

Examples

```
mortality_rates <-
  dplyr::filter(
    life_tables,
    country == "USA" &
    sex     == "male" &
    year    == 2022
  )

calc_gompertz_parameters(
  mortality_rates = mortality_rates,
  current_age     = 65
)
```

`calc_gompertz_survival_probability`
Calculating Gompertz survival probability

Description

Calculating Gompertz survival probability

Usage

```
calc_gompertz_survival_probability(
  current_age,
  target_age,
  mode,
  dispersion,
  max_age = NULL
)
```

Arguments

current_age	Current age
target_age	Target age
mode	Mode of the Gompertz distribution
dispersion	Dispersion of the Gompertz distribution
max_age	Maximum age. Defaults to NULL.

Value

A numeric. The probability of survival from 'current_age' to 'target_age' based on the Gompertz distribution with the given parameters.

Examples

```
calc_gompertz_survival_probability(
  current_age = 65,
  target_age  = 85,
  mode        = 80,
  dispersion  = 10
)
```

---

calc_life_expectancy *Calculate Life Expectancy*

Description

Calculate Life Expectancy

Usage

```
calc_life_expectancy(current_age, mode, dispersion, max_age = 120)
```

Arguments

current_age	A numeric. Current age.
mode	A numeric. Mode of the Gompertz distribution.
dispersion	A numeric. Dispersion of the Gompertz distribution.
max_age	A numeric. Maximum age. Defaults to 120.

Value

A numeric. Total life expectancy in years.

Examples

```
calc_life_expectancy(
  current_age = 65,
  mode        = 80,
  dispersion  = 10
)
```

`calc_optimal_asset_allocation` *Calculate optimal asset allocation*

Description

Calculate optimal asset allocation

Usage

```
calc_optimal_asset_allocation(
  household,
  portfolio,
  current_date = get_current_date()
)
```

Arguments

`household`	An R6 object of class `Household`.
`portfolio`	A nested `tibble` of class `Portfolio`.
`current_date`	A character. Current date in the format `YYYY-MM-DD`. By default, it is the output of `get_current_date()`.

Value

The `portfolio` with additional nested columns:

- `allocations$optimal` - optimal joint net-worth portfolio allocations
- `allocations$current` - current allocations

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "1980-02-15",
  mode       = 80,
  dispersion = 10
)
household <- Household$new()
household$add_member(older_member)
```

```
household$expected_income <- list(
  "income" = c(
    "members$older$age <= 65 ~ 7000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "TRUE ~ 5000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)

portfolio <-
  portfolio |>
  calc_effective_tax_rate(
    tax_rate_ltcg = 0.20,
    tax_rate_ordinary_income = 0.40
  )

portfolio <-
  calc_optimal_asset_allocation(
   household = household,
   portfolio = portfolio,
   current_date = "2020-07-15"
  )

portfolio$allocations
```

`calc_optimal_risky_asset_allocation` *Calculate optimal risky asset allocation*

Description

Calculates the optimal allocation to the risky asset using the Merton Share formula.

Usage

```
calc_optimal_risky_asset_allocation(
  risky_asset_return_mean,
  risky_asset_return_sd,
  safe_asset_return,
  risk_aversion
)
```

Arguments

`risky_asset_return_mean`	A numeric. The expected (average) yearly return of the risky asset.
`risky_asset_return_sd`	A numeric. The standard deviation of the yearly returns of the risky asset.
`safe_asset_return`	A numeric. The expected yearly return of the safe asset.
`risk_aversion`	A numeric. The risk aversion coefficient.

Details

Can be used to calculate the optimal allocation to the risky asset for vectors of inputs.

Value

A numeric. The optimal allocation to the risky asset. In case of `NaN()` (because of division by zero) the optimal allocation to the risky asset is set to 0.

See Also

- How to Determine Our Optimal Asset Allocation?
- Haghani V., White J. (2023) "The Missing Billionaires: A Guide to Better Financial Decisions." ISBN:978-1-119-74791-8.

Examples

```
calc_optimal_risky_asset_allocation(
  risky_asset_return_mean = 0.05,
  risky_asset_return_sd   = 0.15,
  safe_asset_return       = 0.02,
  risk_aversion           = 2
)

calc_optimal_risky_asset_allocation(
  risky_asset_return_mean = c(0.05, 0.06),
  risky_asset_return_sd   = c(0.15, 0.16),
  safe_asset_return       = 0.02,
  risk_aversion           = 2
)
```

`calc_portfolio_parameters` *Calculate Portfolio Parameters*

Description

Calculate Portfolio Parameters

Usage

```
calc_portfolio_parameters(portfolio)
```

Arguments

portfolio	A tibble of class Portfolio. Usually created using create_portfolio_template and customised.

Value

A list with the following elements:

- value: The value of the portfolio.
- weights: The weights of assets in the portfolio.
- expected_return: The expected return of the portfolio.
- standard_deviation: The standard deviation of the portfolio.

Examples

```
portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
calc_portfolio_parameters(portfolio)
```

---

calc_purchasing_power *Calculate purchasing power*

Description

Calculates changes in purchasing power over time, taking into account the real interest rate.

Usage

```
calc_purchasing_power(x, years, real_interest_rate)
```

Arguments

x	A numeric. The initial amount of money.
years	A numeric. The number of years.
real_interest_rate	A numeric. The yearly real interest rate.

Details

The real interest rate is the interest rate after inflation. If negative (e.g. equal to the average yearly inflation rate) it can show diminishing purchasing power over time. If positive, it can show increasing purchasing power over time, and effect of compounding interest on the purchasing power.

Value

A numeric. The purchasing power.

See Also

- How to Determine Our Optimal Asset Allocation?

Examples

```
calc_purchasing_power(x = 10, years = 30, real_interest_rate = -0.02)
calc_purchasing_power(x = 10, years = 30, real_interest_rate = 0.02)
```

---

calc_retirement_ruin *Calculating retirement ruin probability*

Description

Calculating retirement ruin probability

Usage

```
calc_retirement_ruin(
  portfolio_return_mean,
  portfolio_return_sd,
  age,
  gompertz_mode,
  gompertz_dispersion,
  portfolio_value,
  monthly_spendings,
  yearly_spendings = 12 * monthly_spendings,
  spending_rate = yearly_spendings/portfolio_value
)
```

Arguments

portfolio_return_mean	A numeric. Mean of portfolio returns.
portfolio_return_sd	A numeric. Standard deviation of portfolio returns.
age	A numeric. Current age.
gompertz_mode	A numeric. Gompertz mode.
gompertz_dispersion	A numeric. Gompertz dispersion.
portfolio_value	A numeric. Initial portfolio value.

`monthly_spendings`
: A numeric. Monthly spendings.

`yearly_spendings`
: A numeric. Yearly spendings.

`spending_rate`
: A numeric. Spending rate (initial withdrawal rate).

Value

A numeric. The probability of retirement ruin (between 0 and 1), representing the likelihood of running out of money during retirement.

References

Milevsky, M.A. (2020). Retirement Income Recipes in R: From Ruin Probabilities to Intelligent Drawdowns. Use R! Series. doi:10.1007/9783030514341.

Examples

```
calc_retirement_ruin(
  age                  = 65,
  gompertz_mode        = 88,
  gompertz_dispersion  = 10,
  portfolio_value      = 1000000,
  monthly_spendings    = 3000,
  portfolio_return_mean = 0.02,
  portfolio_return_sd  = 0.15
)
```

---

calc_risk_adjusted_return

Calculate risk adjusted return

Description

Calculates the risk adjusted return for portfolio of given allocation to the risky asset.

Usage

```
calc_risk_adjusted_return(
  safe_asset_return,
  risky_asset_return_mean,
  risky_asset_allocation,
  risky_asset_return_sd = NULL,
  risk_aversion = NULL
)
```

Arguments

`safe_asset_return`
: A numeric. The expected yearly return of the safe asset.

`risky_asset_return_mean`
: A numeric. The expected (average) yearly return of the risky asset.

`risky_asset_allocation`
: A numeric. The allocation to the risky asset. Could be a vector. If it is the optimal allocation then parameters `risky_asset_return_sd` and `risk_aversion` can be omitted.

`risky_asset_return_sd`
: A numeric. The standard deviation of the yearly returns of the risky asset.

`risk_aversion`
: A numeric. The risk aversion coefficient.

Value

A numeric. The risk adjusted return.

See Also

- How to Determine Our Optimal Asset Allocation?
- Haghani V., White J. (2023) "The Missing Billionaires: A Guide to Better Financial Decisions." ISBN:978-1-119-74791-8.

Examples

```
calc_risk_adjusted_return(
  safe_asset_return = 0.02,
  risky_asset_return_mean = 0.04,
  risky_asset_return_sd = 0.15,
  risky_asset_allocation = 0.5,
  risk_aversion = 2
)

calc_risk_adjusted_return(
  safe_asset_return = 0.02,
  risky_asset_return_mean = 0.04,
  risky_asset_allocation = c(0.25, 0.5, 0.75),
  risky_asset_return_sd = 0.15,
  risk_aversion = 2
)
```

`create_portfolio_template` *Create Portfolio Template*

Description

Creates a template for default portfolio with two asset classes:

- `GlobalStocksIndexFund`
- `InflationProtectedBonds`

Usage

```
create_portfolio_template()
```

Details

The template is used as a starting point for creating a portfolio. The asset classes have some reasonable default values of expected returns and standard deviations of returns. The template assumes no correlations between asset classes in the `correlations` matrix. Please check and update the template assumptions if necessary.

The nested `pretax` columns contain default values for parameters needed for calculating effective tax rates. The template assumes only capital gains tax is paid. Please customise this template to your individual situation.

The `accounts` nested columns have zero values for all assets by default in both taxable and tax-advantaged accounts. The template assumes that there is currently no financial wealth allocated to those accounts. Please customise this template to your individual situation.

The `weights` nested columns define weights of assets in portfolios representative of the household human capital and liabilities. The template assumes equal weights for all assets for both portfolios. Please customise this template to your individual situation.

Value

A nested `tibble` of class 'Portfolio' with columns:

- `name`
- `expected_return`
- `standard_deviation`
- `accounts`
  - `taxable`
  - `taxadvantaged`
- `weights`
  - `human_capital`
  - `liabilities`
- `correlations`
- `pretax`
  - `turnover`
  - `income_qualified`
  - `capital_gains_long_term`
  - `income`
  - `capital_gains`
  - `cost_basis`

See Also

Possible sources of market assumptions:

- https://elmwealth.com/capital-market-assumptions/
- https://www.obligacjeskarbowe.pl/oferta-obligacji/obligacje-10-letnie-edo/
- https://www.msci.com/indexes/index/664204
- (PDF) https://research.ftserussell.com/Analytics/FactSheets/Home/DownloadSingleIssue?issueName=AWORLDS&isl

Examples

```
portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio
```

---

format_currency *Printing currency values or percentages*

Description

Wrapper functions for printing nicely formatted values.

Usage

```
format_currency(
  x,
  prefix = "",
  suffix = "",
  big.mark = ",",
  accuracy = NULL,
  min_length = NULL,
  ...
)

format_percent(x, accuracy = 0.1, ...)
```

Arguments

x	A numeric vector
prefix, suffix	Symbols to display before and after value.
big.mark	Character used between every 3 digits to separate thousands. The default (NULL) retrieves the setting from the number options.
accuracy	A number to round to. Use (e.g.) 0.01 to show 2 decimal places of precision. If NULL, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.
min_length	A numeric. Minimum number of characters of the string with the formatted value.
...	Other arguments passed on to base::format().

Value

A character. Formatted value.

A character. Formatted value.

See Also

scales::dollar()

scales::percent()

Examples

```
format_currency(2345678, suffix = " PLN")
format_percent(0.52366)
```

`get_cache_info` *Working with cache*

Description

Get information about the cache

Reset the cache

Set the cache directory

Usage

```
get_cache_info()

reset_cache()

set_cache(path = file.path(getwd(), ".cache"))
```

Arguments

`path`	The path to the cache directory. Defaults to the '.cache' folder in the current working directory.

Value

Invisibly returns the path to the cache directory or a list containing:

`path`	The path to the cache directory.
`files`	The number of files in the cache.

Examples

```
get_cache_info()


reset_cache()


set_cache()
```

`get_current_date` *Get current date*

Description

If `R4GPF.current_date` option is not set, the current system date is used.

Usage

```
get_current_date()
```

Value

A date.

Examples

```
get_current_date()
# Setting custom date using `R4GPF.current_date` option
options(R4GPF.current_date = as.Date("2023-01-01"))
get_current_date()
options(R4GPF.current_date = NULL) # Reset default date#' Working with cache

get_current_date()
```

`get_default_gompertz_parameters`
Get default Gompertz parameters

Description

Calculates default Gompertz parameters for a given age, country and sex, based on the package build-in HMD life tables.

Usage

```
get_default_gompertz_parameters(
  age,
  country = unique(life_tables$country),
  sex = c("both", "male", "female")
)
```

Arguments

age	A numeric. The age of the individual.
country	A character. The name of the country.
sex	A character. The sex of the individual.

Value

A list containing:

mode	The mode of the Gompertz distribution
dispersion	The dispersion parameter of the Gompertz distribution
current_age	The current age parameter
max_age	The maximum age parameter

See Also

calc_gompertz_parameters()

Examples

```
get_default_gompertz_parameters(
  age     = 65,
  country = "USA",
  sex     = "male"
)
```

Household *Household class*

Description

The Household class aggregates information about a household and its members.

Value

An object of class Household.

Active bindings

`expected_income` Set of rules that are used to generate streams of expected income

`expected_spending` Set of rules that are used to generate streams of expected spending

`risk_tolerance` Risk tolerance of the household

`consumption_impatience_preference` Consumption impatience preference of the household - subjective discount rate (rho). Higher values indicate a stronger preference for consumption today versus in the future.

`smooth_consumption_preference` Smooth consumption preference of the household - Elasticity of Intertemporal Substitution (EOIS) (eta). Higher values indicate more flexibility and a lower preference for smooth consumption.

Methods

Public methods:

- `Household$print()`
- `Household$get_members()`
- `Household$add_member()`
- `Household$set_member()`
- `Household$set_lifespan()`
- `Household$get_lifespan()`
- `Household$calc_survival()`
- `Household$get_min_age()`
- `Household$clone()`

Method `print()`**:** Printing the household object

Usage:

```
Household$print(current_date = get_current_date())
```

Arguments:

`current_date` A date in the format "YYYY-MM-DD".

Method `get_members()`**:** Getting members of the household

Usage:

```
Household$get_members()
```

Method `add_member()`**:** Adding a member to the household It will fail if a member with the same name already exists.

Usage:

```
Household$add_member(household_member)
```

Arguments:

`household_member` A `HouseholdMember` object.

Method `set_member()`**:** Setting a member of the household If a member already exists, it will be overwritten.

Usage:

```
Household$set_member(member)
```

Arguments:

`member` A `HouseholdMember` object.

Method `set_lifespan()`**:** Setting an arbitrary lifespan of the household

Usage:

```
Household$set_lifespan(value)
```

Arguments:

`value` A number of years.

Method `get_lifespan()`**:** Getting a lifespan of the household If not set, it will be calculated based on the members' lifespans.

Usage:

```
Household$get_lifespan(current_date = get_current_date())
```

Arguments:

`current_date` A date in the format "YYYY-MM-DD".

Method `calc_survival()`**:** Calculating a survival rate of the household based on its members' parameters of the Gompertz model.

Usage:

```
Household$calc_survival(current_date = get_current_date())
```

Arguments:

`current_date` A date in the format "YYYY-MM-DD".

Method `get_min_age()`**:** Calculating a minimum age of the household members.

Usage:

```
Household$get_min_age(current_date = get_current_date())
```

Arguments:

`current_date` A date in the format "YYYY-MM-DD".

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Household$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
household <- Household$new()
household$risk_tolerance
household$consumption_impatience_preference
household$smooth_consumption_preference
```

HouseholdMember *HouseholdMember class*

Description

The HouseholdMember class aggregates information about a single member of a household.

Value

An object of class HouseholdMember.

Active bindings

max_age The maximum age of the household member

mode The Gompertz mode parameter

dispersion The Gompertz dispersion parameter

Methods

Public methods:

- HouseholdMember$new()
- HouseholdMember$print()
- HouseholdMember$get_name()
- HouseholdMember$get_birth_date()
- HouseholdMember$calc_age()
- HouseholdMember$get_lifespan()
- HouseholdMember$calc_life_expectancy()
- HouseholdMember$calc_survival_probability()
- HouseholdMember$get_events()
- HouseholdMember$set_event()
- HouseholdMember$clone()

Method new(): Creating a new object of class HouseholdMember

Usage:

```
HouseholdMember$new(name, birth_date, mode = NULL, dispersion = NULL)
```

Arguments:

name The name of the member.

birth_date The birth date of the household member in the format YYYY-MM-DD.

mode The Gompertz mode parameter.

dispersion The Gompertz dispersion parameter.

Method print(): Printing the household member object

Usage:

```
HouseholdMember$print(current_date = get_current_date())
```

Arguments:

`current_date` A date in the format "YYYY-MM-DD".

Method `get_name()`**:** Getting the name of the household member

Usage:

```
HouseholdMember$get_name()
```

Method `get_birth_date()`**:** Getting the birth date of the household member

Usage:

```
HouseholdMember$get_birth_date()
```

Method `calc_age()`**:** Calculating the age of the household member

Usage:

```
HouseholdMember$calc_age(current_date = get_current_date())
```

Arguments:

`current_date` A date in the format "YYYY-MM-DD".

Method `get_lifespan()`**:** Calculating a lifespan of the household member

Usage:

```
HouseholdMember$get_lifespan(current_date = get_current_date())
```

Arguments:

`current_date` A date in the format "YYYY-MM-DD".

Method `calc_life_expectancy()`**:** Calculating a life expectancy of the household member

Usage:

```
HouseholdMember$calc_life_expectancy(current_date = get_current_date())
```

Arguments:

`current_date` A date in the format "YYYY-MM-DD".

Method `calc_survival_probability()`**:** Calculating a survival probability of the household member

Usage:

```
HouseholdMember$calc_survival_probability(
  target_age,
  current_date = get_current_date()
)
```

Arguments:

`target_age` Target age (numeric, in years).

`current_date` A date in the format "YYYY-MM-DD".

Method `get_events()`**:** Getting the events related to the household member

Usage:

```
HouseholdMember$get_events()
```

Method `set_event()`**:** Setting an event related to the household member

Usage:

```
HouseholdMember$set_event(event, start_age, end_age = Inf, years = Inf)
```

Arguments:

`event` The name of the event.

`start_age` The age of the household member when the event starts.

`end_age` The age of the household member when the event ends.

`years` The number of years the event lasts.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
HouseholdMember$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
member <- HouseholdMember$new(
  name       = "Isabela",
  birth_date = "1980-07-15",
  mode       = 91,
  dispersion = 8.88
)
member$calc_age()
member$calc_life_expectancy()
```

`life_tables` *HMD life tables*

Description

A data frame based on: HMD. Human Mortality Database. Max Planck Institute for Demographic Research (Germany), University of California, Berkeley (USA), and French Institute for Demographic Studies (France). Available at www.mortality.org.

Usage

```
life_tables
```

Format

`life_tables`**:**

A data frame with 6 columns:

country Country name

sex Sex: "male", "female", "both"

year Year
age Age
mortality_rate Mortality rate
life_expectancy Life expectancy

Source

https://www.mortality.org

`plot_expected_allocation` *Plot expected allocation over household life cycle*

Description

If multiple Monte Carlo samples are provided in the `scenario` argument, the normalized median of expected allocation is plotted.

Usage

```
plot_expected_allocation(
  scenario,
  accounts = c("all", "taxable", "taxadvantaged")
)
```

Arguments

`scenario`	A `tibble` with nested columns - the result of `simulate_scenario()`. Data for a single scenario.
`accounts`	A character. Plot allocation for specified types of accounts.

Value

A `ggplot2::ggplot()` object.

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "2000-02-15",
  mode       = 80,
  dispersion = 10
)
household <- Household$new()
household$add_member(older_member)

household$expected_income <- list(
  "income" = c(
```

```
    "members$older$age <= 65 ~ 10000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "TRUE ~ 5000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio$weights$human_capital <- c(0.2, 0.8)
portfolio$weights$liabilities <- c(0.1, 0.9)
portfolio <-
  portfolio |>
  calc_effective_tax_rate(
    tax_rate_ltcg            = 0.20,
    tax_rate_ordinary_income = 0.40
  )

scenario <-
  simulate_scenario(
   household    = household,
   portfolio    = portfolio,
   current_date = "2020-07-15"
  )

plot_expected_allocation(scenario)
```

`plot_expected_capital` *Plot expected capital over household life cycle*

Description

Plots financial capital, human capital, total capital, and liabilities.

Usage

```
plot_expected_capital(scenario)
```

Arguments

`scenario`	A `tibble` with nested columns - the result of `simulate_scenario()`. Data for a single scenario.

Value

A `ggplot2::ggplot()` object.

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "2000-02-15",
  mode       = 80,
  dispersion = 10
)
household <- Household$new()
household$add_member(older_member)

household$expected_income <- list(
  "income" = c(
    "members$older$age <= 65 ~ 10000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "TRUE ~ 5000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio$weights$human_capital <- c(0.2, 0.8)
portfolio$weights$liabilities <- c(0.1, 0.9)
portfolio <-
  portfolio |>
  calc_effective_tax_rate(
    tax_rate_ltcg            = 0.20,
    tax_rate_ordinary_income = 0.40
  )

scenario <-
  simulate_scenario(
   household    = household,
   portfolio    = portfolio,
   current_date = "2020-07-15"
  )

plot_expected_capital(scenario)
```

`plot_future_income` *Plot future income structure over household life cycle*

Description

Plot future income structure over household life cycle

Usage

```
plot_future_income(
  scenario,
  period = c("yearly", "monthly"),
  y_limits = c(NA, NA)
)
```

Arguments

scenario	A tibble with nested columns - the result of simulate_scenario(). Data for a single scenario.
period	A character. The amounts can be shown as yearly values (default) or averaged per month values.
y_limits	A numeric vector of two values. Y-axis limits.

Value

A ggplot2::ggplot() object.

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "1980-02-15",
  mode       = 80,
  dispersion = 10
)
household <- Household$new()
household$add_member(older_member)

household$expected_income <- list(
  "income" = c(
    "members$older$age <= 65 ~ 7000 * 12",
    "members$older$age > 65 ~ 3000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "TRUE ~ 5000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio <-
  portfolio |>
  calc_effective_tax_rate(
    tax_rate_ltcg = 0.20,
    tax_rate_ordinary_income = 0.40
  )
```

```
scenario <-
  simulate_scenario(
   household = household,
   portfolio = portfolio,
   current_date = "2020-07-15"
  )

plot_future_income(scenario, "monthly")
```

`plot_future_saving_rates` *Plotting future saving rates*

Description

This function plots the future saving rates from a scenario object.

Usage

```
plot_future_saving_rates(
  scenario,
  aggregation_function = stats::median,
  y_limits = c(NA, NA)
)
```

Arguments

`scenario`	A `tibble` with nested columns - the result of `simulate_scenario()`. Data for a single scenario.
`aggregation_function`	A function used to aggregate the saving rates for multiple Monte Carlo samples. Default is `median`. If `NULL`, no aggregation is performed.
`y_limits`	A numeric vector of two values. Y-axis limits.

Value

A `ggplot2::ggplot()` object.

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "2000-02-15",
  mode       = 80,
  dispersion = 10
)
```

```
household <- Household$new()
household$add_member(older_member)

household$expected_income <- list(
  "income" = c(
    "members$older$age <= 65 ~ 10000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "TRUE ~ 5000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio$weights$human_capital <- c(0.2, 0.8)
portfolio$weights$liabilities <- c(0.1, 0.9)
portfolio <-
  portfolio |>
  calc_effective_tax_rate(
    tax_rate_ltcg            = 0.20,
    tax_rate_ordinary_income = 0.40
  )

scenario <-
  simulate_scenario(
   household    = household,
   portfolio    = portfolio,
   current_date = "2020-07-15"
  )

plot_future_saving_rates(scenario)
```

`plot_future_spending` *Plot future spending structure over household life cycle*

Description

Plot future spending structure over household life cycle, including discretionary and non-discretionary spending. You can also plot discretionary and non-discretionary spending separately, to see structure of non-discretionary spending and possible levels of discretionary spending over time based on Monte Carlo simulations.

Usage

```
plot_future_spending(
  scenario,
  period = c("yearly", "monthly"),
  type = c("both", "discretionary", "non-discretionary"),
```

```
  discretionary_spending_position = c("bottom", "top"),
  y_limits = c(NA, NA)
)
```

Arguments

scenario	A tibble with nested columns - the result of simulate_scenario(). Data for a single scenario.
period	A character. The amounts can be shown as yearly values (default) or averaged per month values.
type	A character. Type of spending to plot: discretionary, non-discretionary, or both (default).
discretionary_spending_position	A character. Position of discretionary spending in plot. Bottom is the default.
y_limits	A numeric vector of two values. Y-axis limits.

Value

A ggplot2::ggplot() object

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "1980-02-15",
  mode       = 80,
  dispersion = 10
)
household <- Household$new()
household$add_member(older_member)

household$expected_income <- list(
  "income" = c(
    "members$older$age <= 65 ~ 9000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "members$older$age <= 65 ~ 5000 * 12",
    "TRUE ~ 4000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio <-
  portfolio |>
  calc_effective_tax_rate(
    tax_rate_ltcg = 0.20,
    tax_rate_ordinary_income = 0.40
```

```
  )

scenario <-
  simulate_scenario(
   household = household,
   portfolio = portfolio,
   # monte_carlo_samples = 100,
   current_date = "2020-07-15"
  )

plot_future_spending(scenario, "monthly")
plot_future_spending(
  scenario,
  "monthly",
  discretionary_spending_position = "top"
)
plot_future_spending(scenario, "monthly", "non-discretionary")
# If Monte Carlo samples are present:
# plot_future_spending(scenario, "monthly", "discretionary")
```

`plot_gompertz_calibration` *Plotting the results of Gompertz model calibration*

Description

Plotting the results of Gompertz model calibration

Usage

```
plot_gompertz_calibration(params, mode, dispersion, max_age)
```

Arguments

`params`	A list returned by `calc_gompertz_parameters()` function.
`mode`	A numeric. The mode of the Gompertz model.
`dispersion`	A numeric. The dispersion of the Gompertz model.
`max_age`	A numeric. The maximum age of the Gompertz model.

Value

A `ggplot2::ggplot()` object showing the comparison between actual survival rates from life tables and the fitted Gompertz model.

Examples

```
mortality_rates <-
  dplyr::filter(
    life_tables,
    country == "USA" &
    sex     == "female" &
    year    == 2022
  )

params <- calc_gompertz_parameters(
  mortality_rates = mortality_rates,
  current_age     = 65
)

plot_gompertz_calibration(params = params)
```

`plot_joint_survival` *Plotting the results of Gompertz model calibration for joint survival*

Description

Plotting the results of Gompertz model calibration for joint survival

Usage

```
plot_joint_survival(params, include_gompertz = FALSE)
```

Arguments

`params`	A list returned by `calc_gompertz_joint_parameters()` function.
`include_gompertz`	A logical. Should the Gompertz survival curve be included in the plot?

Value

A `ggplot2::ggplot()` object showing the survival probabilities for two individuals and their joint survival probability.

Examples

```
params <- calc_gompertz_joint_parameters(
  p1 = list(
    age        = 65,
    mode       = 88,
    dispersion = 10.65
  ),
  p2 = list(
    age        = 60,
    mode       = 91,
```

```
    dispersion = 8.88
  ),
  max_age = 110
)

plot_joint_survival(params = params, include_gompertz = TRUE)
```

`plot_life_expectancy` *Plot life expectancy of household members*

Description

Probability of dying at a given age is plotted for each member of a household. Also for each member the life expectancy is shown as dashed vertical line.

Usage

```
plot_life_expectancy(household)
```

Arguments

`household`	An R6 object of class `Household`.

Value

A `ggplot` object.

Examples

```
hm1 <-
 HouseholdMember$new(
   name       = "member1",
   birth_date = "1955-01-01",
   mode       = 88,
   dispersion = 10.65
 )
hm2 <-
 HouseholdMember$new(
   name       = "member2",
   birth_date = "1965-01-01",
   mode       = 91,
   dispersion = 8.88
 )
household <- Household$new()
household$add_member(hm1)
household$add_member(hm2)

plot_life_expectancy(household = household)
```

plot_optimal_portfolio *Plot optimal portfolio allocations*

Description

The function plots current versus optimal portfolio allocations for each asset class and for taxable and tax-advantaged accounts.

Usage

```
plot_optimal_portfolio(portfolio)
```

Arguments

portfolio	A nested tibble of class Portfolio.

Value

A ggplot2::ggplot() object.

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "1980-02-15",
  mode       = 80,
  dispersion = 10
)
household <- Household$new()
household$add_member(older_member)

household$expected_income <- list(
  "income" = c(
    "members$older$age <= 65 ~ 7000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "TRUE ~ 5000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio$accounts$taxadvantaged <- c(0, 20000)
portfolio <-
  portfolio |>
  calc_effective_tax_rate(
```

```
    tax_rate_ltcg = 0.20,
    tax_rate_ordinary_income = 0.40
  )

portfolio <-
  calc_optimal_asset_allocation(
   household = household,
   portfolio = portfolio,
   current_date = "2020-07-15"
  )

plot_optimal_portfolio(portfolio)
```

`plot_purchasing_power` *Plotting changes to the purchasing power over time*

Description

Plots the effect of real interest rates (positive or negative) on the purchasing power of savings over the span of 50 years (default).

Usage

```
plot_purchasing_power(
  x,
  real_interest_rate,
  years = 50,
  legend_title = "Real interest rate",
  seed = NA
)
```

Arguments

`x`	A numeric. The initial amount of money.
`real_interest_rate`	A numeric. The yearly real interest rate.
`years`	A numeric. The number of years.
`legend_title`	A character.
`seed`	A numeric. Seed passed to `geom_label_repel()`.

Value

A `ggplot2::ggplot()` object.

See Also

- How to Determine Our Optimal Asset Allocation?

Examples

```
plot_purchasing_power(
  x = 10,
  real_interest_rate = seq(-0.02, 0.04, by = 0.02)
)
```

`plot_retirement_ruin` *Plotting retirement ruin*

Description

Plotting retirement ruin

Usage

```
plot_retirement_ruin(
  portfolio_return_mean,
  portfolio_return_sd,
  age,
  gompertz_mode,
  gompertz_dispersion,
  portfolio_value,
  monthly_spendings = NULL
)
```

Arguments

`portfolio_return_mean`	A numeric. Mean of portfolio returns.
`portfolio_return_sd`	A numeric. Standard deviation of portfolio returns.
`age`	A numeric. Current age.
`gompertz_mode`	A numeric. Gompertz mode.
`gompertz_dispersion`	A numeric. Gompertz dispersion.
`portfolio_value`	A numeric. Initial portfolio value.
`monthly_spendings`	A numeric. Monthly spendings.

Value

A `ggplot2::ggplot()` object showing the probability of retirement ruin for different monthly spending levels. If a specific 'monthly_spendings' value is provided, it will be highlighted on the plot with annotations.

Examples

```
plot_retirement_ruin(
  portfolio_return_mean = 0.034,
  portfolio_return_sd   = 0.15,
  age                   = 65,
  gompertz_mode         = 88,
  gompertz_dispersion   = 10,
  portfolio_value       = 1000000,
  monthly_spendings     = 3000
)
```

---

plot_risk_adjusted_returns

Plotting risk adjusted returns

Description

Plots the risk adjusted returns for portfolios of various allocations to the risky asset.

Usage

```
plot_risk_adjusted_returns(
  safe_asset_return,
  risky_asset_return_mean,
  risky_asset_return_sd,
  risk_aversion = 2,
  current_risky_asset_allocation = NULL
)
```

Arguments

`safe_asset_return`
: A numeric. The expected yearly return of the safe asset.

`risky_asset_return_mean`
: A numeric. The expected (average) yearly return of the risky asset.

`risky_asset_return_sd`
: A numeric. The standard deviation of the yearly returns of the risky asset.

`risk_aversion`
: A numeric. The risk aversion coefficient.

`current_risky_asset_allocation`
: A numeric. The current allocation to the risky asset. For comparison with the optimal allocation.

Value

A `ggplot2::ggplot()` object.

See Also

- How to Determine Our Optimal Asset Allocation?
- Haghani V., White J. (2023) "The Missing Billionaires: A Guide to Better Financial Decisions." ISBN:978-1-119-74791-8.

Examples

```
plot_risk_adjusted_returns(
  safe_asset_return              = 0.02,
  risky_asset_return_mean        = 0.04,
  risky_asset_return_sd          = 0.15,
  risk_aversion                  = 2,
  current_risky_asset_allocation = 0.8
)
```

---

plot_scenarios *Plot scenarios metrics*

Description

The plot allows to compare metrics for multiple scenarios.

If scenarios are simulated without Monte Carlo samples, so they are based only on expected returns of portfolio, two metrics are available for each scenario:

- constant discretionary spending - certainty equivalent constant level of consumption that would result in the same lifetime utility as a given series of future consumption in a given scenario (the higher, the better).
- utility of discretionary spending - normalized to minimum and maximum values of constant discretionary spending (the higher, the better).

If scenarios are simulated with additional Monte Carlo samples, there are four more metrics available per scenario:

- constant discretionary spending (for Monte Carlo samples),
- normalized median utility of discretionary spending (for Monte Carlo samples),
- median of missing funds that need additional income or additional savings at the expense of non-discretionary spending, (of yearly averages of Monte Carlo samples),
- median of discretionary spending (of yearly averages of Monte Carlo samples).

Usage

```
plot_scenarios(scenarios, period = c("yearly", "monthly"))
```

Arguments

scenarios	A tibble with nested columns - the result of simulate_scenarios().
period	A character. The amounts can be shown as yearly values (default) or averaged per month values.

Value

A ggplot2::ggplot() object.

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "1980-02-15",
  mode       = 80,
  dispersion = 10
)
household <- Household$new()
household$add_member(older_member)

household$expected_income <- list(
  "income" = c(
    "is_not_on('older', 'retirement') ~ 7000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "TRUE ~ 4000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(100000, 300000)
portfolio <-
  portfolio |>
  calc_effective_tax_rate(
    tax_rate_ltcg = 0.20,
    tax_rate_ordinary_income = 0.40
  )

start_ages <- c(60, 65, 75)
scenarios_parameters <-
  tibble::tibble(
    member    = "older",
    event      = "retirement",
    start_age = start_ages,
    years     = Inf,
    end_age   = Inf
   ) |>
  dplyr::mutate(scenario_id = start_age) |>
  tidyr::nest(events = -scenario_id)

scenarios <-
  simulate_scenarios(
    scenarios_parameters = scenarios_parameters,
    household            = household,
    portfolio            = portfolio,
    maxeval              = 100,
```

```
    current_date         = "2020-07-15"
  )

plot_scenarios(scenarios, "monthly")
```

`plot_survival` *Plot survival of household members*

Description

Plot survival probabilities for each household members and for the entire household when at least one member is alive. The household joint survival probability is also approximated by a Gompertz model.

Usage

```
plot_survival(household, current_date = get_current_date())
```

Arguments

`household`	An R6 object of class `Household`.
`current_date`	A character. Current date in the format `YYYY-MM-DD`. By default, it is the output of `get_current_date()`.

Value

A `ggplot` object.

Examples

```
hm1 <-
 HouseholdMember$new(
   name       = "member1",
   birth_date = "1955-01-01",
   mode       = 88,
   dispersion = 10.65
 )
hm2 <-
 HouseholdMember$new(
   name       = "member2",
   birth_date = "1965-01-01",
   mode       = 91,
   dispersion = 8.88
 )
hm3 <-
 HouseholdMember$new(
   name       = "member3",
   birth_date = "1975-01-01",
```

```
   mode       = 88,
   dispersion = 7.77
 )
household <- Household$new()
household$add_member(hm1)
household$add_member(hm2)
household$add_member(hm3)

plot_survival(
 household    = household,
 current_date = "2020-01-01"
)
```

`read_hmd_life_tables` *Reading HMD life tables*

Description

Reading HMD life tables

Usage

```
read_hmd_life_tables(
  path = getwd(),
  files = c("mltper_1x1.txt", "fltper_1x1.txt", "bltper_1x1.txt")
)
```

Arguments

`path`	A character. Path to the folder with life tables.
`files`	A character. Names of files with life tables.

Value

A data frame containing mortality data with columns:

`sex`	Character - sex ('male', 'female', or 'both')
`year`	Integer - the year of the data
`age`	Integer - age
`mortality_rate`	Numeric - mortality rate
`life_expectancy`	Numeric - life expectancy

References

HMD. Human Mortality Database. Max Planck Institute for Demographic Research (Germany), University of California, Berkeley (USA), and French Institute for Demographic Studies (France). Available at www.mortality.org

Examples

```
## Not run:
# Download 'txt' files
# ("mltper_1x1.txt", "fltper_1x1.txt", "bltper_1x1.txt")
# for a given country to the working directory
# from https://www.mortality.org after registration.

read_hmd_life_tables(path = getwd())

## End(Not run)
```

`render_scenario_snapshot` *Rendering a scenario snapshot*

Description

Rendering a scenario snapshot

Usage

```
render_scenario_snapshot(scenario, index = 0, currency = "", big_mark = " ")
```

Arguments

`scenario`	A `tibble` with nested columns - the result of `simulate_scenario()`. Data for a single scenario.
`index`	The index of the scenario year to render. By default, it is 0, which corresponds to the current year.
`currency`	The currency symbol to use as a suffix.
`big_mark`	The character to use as a big mark. It separates thousands.

Value

A `gt::gt()` object.

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "1980-02-15",
  mode       = 80,
  dispersion = 10
)
household <- Household$new()
household$add_member(older_member)

household$expected_income <- list(
```

```
  "income" = c(
    "members$older$age <= 65 ~ 9000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "members$older$age <= 65 ~ 5000 * 12",
    "TRUE ~ 4000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio <-
  portfolio |>
  calc_effective_tax_rate(
    tax_rate_ltcg = 0.20,
    tax_rate_ordinary_income = 0.40
  )

scenario <-
  simulate_scenario(
   household = household,
   portfolio = portfolio,
   current_date = "2020-07-15"
  )
render_scenario_snapshot(scenario)
```

`run_app` *Run a package app*

Description

Run a package app

Usage

```
run_app(
  which = c("risk-adjusted-returns", "purchasing-power", "retirement-ruin"),
  res = 120,
  shinylive = FALSE
)
```

Arguments

`which`	A character. The name of the app to run. Currently available: • `risk-adjusted-returns` - Plotting risk-adjusted returns for various allocations to the risky asset allows you to find the optimal allocation.

- `purchasing-power` - Plotting the effect of real interest rates (positive or negative) on the purchasing power of savings over time.
- `retirement-ruin` - Plotting the probability of retirement ruin.

`res` A numeric. The initial resolution of the plots.

`shinylive` A logical. Whether to use `shinylive` for the app.

Value

A `shiny::shinyApp()` object if `shinylive` is `TRUE`. Runs the app if `shinylive` is `FALSE` with `shiny::runApp()`.

Examples

```
run_app("risk-adjusted-returns")
run_app("purchasing-power")
run_app("retirement-ruin")
```

---

simulate_scenario *Simulate a scenario of household lifetime finances*

Description

The function simulates a scenario of household lifetime finances and returns a `tibble` with nested columns. By default no Monte Carlo samples are generated, and only single sample based on portfolio expected returns are returned with column `sample=0`. If the additional Monte Carlo samples are generated, they have consecutive IDs starting from 1 in the `sample` column.

Usage

```
simulate_scenario(
  household,
  portfolio,
  scenario_id = "default",
  current_date = get_current_date(),
  monte_carlo_samples = NULL,
  seeds = NULL,
  use_cache = FALSE,
  auto_parallel = FALSE,
  debug = FALSE,
  ...
)
```

Arguments

household	An R6 object of class Household.
portfolio	A nested tibble of class Portfolio.
scenario_id	A character. ID of the scenario.
current_date	A character. Current date in the format YYYY-MM-DD. By default, it is the output of get_current_date().
monte_carlo_samples	An integer. Number of Monte Carlo samples. If NULL (default), no Monte Carlo samples are generated.
seeds	An integer or integer vector. If integer vector, it is a vector of random seeds for the random number generator used to generate random portfolio returns for each Monte Carlo sample. If NULL (default), random seed is generated automatically. If a single integer is provided, it is used to generate a vector of random seeds for each Monte Carlo sample.
use_cache	A logical. If TRUE, the function uses memoised functions to speed up the simulation. The results are cached in the folder set by set_cache().
auto_parallel	A logical. If TRUE, the function automatically detects the number of cores and uses parallel processing to speed up the Monte Carlo simulations. The results are cached in the folder set by set_cache().
debug	A logical. If TRUE, additional information is printed during the simulation.
...	Additional arguments passed to simulation and optimization functions. You can pass a list named opts as parameter to the optimization function to select the optimization algorithm and its parameters. See nloptr::nloptr() and nloptr::nloptr.print.options() for more information.

Value

A tibble with nested columns including:

- scenario_id - (character) ID of the scenario
- sample - (integer) ID of the Monte Carlo sample
- index - (integer) year index starting from 0
- years_left - (integer) years left to the end of the household lifespan
- date - (date) date after index years from the current date
- year - (integer) calendar year
- survival_prob - (double) survival probability of the household
- members - (nested tibble) data of each member in the household
- income - (nested tibble) income streams
- total_income - (double) total income of the household from all income streams
- spending - (nested tibble) non-discretionary spending streams
- nondiscretionary_spending - (double) total non-discretionary spending of the household from all non-discretionary spending streams

- human_capital - (double) human capital of the household
- liabilities - (double) liabilities of the household
- portfolio - (nested tibble) state of investment portfolio
- financial_wealth - (double) financial wealth of the household at the beginning of the year
- net_worth - (double) net worth of the household
- discretionary_spending - (double) optimal discretionary spending of the household
- total_spending - (double) total spending of the household (discretionary + non-discretionary)
- financial_wealth_end - (double) financial wealth of the household at the end of the year
- risk_tolerance - (double) risk tolerance of the household
- smooth_consumption_preference - (double) smooth consumption preference of the household
- consumption_impatience_preference - (double) consumption impatience preference of the household
- time_value_discount - (double) time value discount based on consumption impatience of the household
- discretionary_spending_utility - (double) discretionary spending utility of the household based on the smooth consumption preference
- discretionary_spending_utility_weighted - (double) discretionary spending utility of the household weighted by survival probability and time value discount.

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "1980-02-15",
  mode       = 80,
  dispersion = 10
)
household <- Household$new()
household$add_member(older_member)

household$expected_income <- list(
  "income" = c(
    "members$older$age <= 65 ~ 7000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "TRUE ~ 5000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio <-
  portfolio |>
  calc_effective_tax_rate(
```

```
    tax_rate_ltcg = 0.20,
    tax_rate_ordinary_income = 0.40
  )

scenario <-
  simulate_scenario(
   household = household,
   portfolio = portfolio,
   current_date = "2020-07-15"
  )
names(scenario)
```

`simulate_scenarios` *Simulate multiple scenarios of household lifetime finances*

Description

Simulate multiple scenarios of household lifetime finances

Usage

```
simulate_scenarios(
  scenarios_parameters,
  household,
  portfolio,
  current_date = get_current_date(),
  monte_carlo_samples = NULL,
  seeds = NULL,
  auto_parallel = FALSE,
  use_cache = FALSE,
  debug = FALSE,
  ...
)
```

Arguments

`scenarios_parameters`	A `tibble` with column `scenario_id` and nested column `events`. Each scenario has defined one or more events in the tibbles that are stored in as a list in the `events` column.
`household`	An R6 object of class `Household`.
`portfolio`	A nested `tibble` of class `Portfolio`.
`current_date`	A character. Current date in the format `YYYY-MM-DD`. By default, it is the output of `get_current_date()`.
`monte_carlo_samples`	An integer. Number of Monte Carlo samples. If `NULL` (default), no Monte Carlo samples are generated.

seeds	An integer or integer vector. If integer vector, it is a vector of random seeds for the random number generator used to generate random portfolio returns for each Monte Carlo sample. If NULL (default), random seed is generated automatically. If a single integer is provided, it is used to generate a vector of random seeds for each Monte Carlo sample.
auto_parallel	A logical. If TRUE, the function automatically detects the number of cores and uses parallel processing to speed up the Monte Carlo simulations. The results are cached in the folder set by set_cache().
use_cache	A logical. If TRUE, the function uses memoised functions to speed up the simulation. The results are cached in the folder set by set_cache().
debug	A logical. If TRUE, additional information is printed during the simulation.
...	Additional arguments passed to simulation and optimization functions. You can pass a list named opts as parameter to the optimization function to select the optimization algorithm and its parameters. See nloptr::nloptr() and nloptr::nloptr.print.options() for more information.

Value

A tibble with nested columns.

Examples

```
older_member <- HouseholdMember$new(
  name       = "older",
  birth_date = "1980-02-15",
  mode       = 80,
  dispersion = 10
)
household <- Household$new()
household$add_member(older_member)

household$expected_income <- list(
  "income" = c(
    "members$older$age <= 65 ~ 7000 * 12"
  )
)
household$expected_spending <- list(
  "spending" = c(
    "TRUE ~ 5000 * 12"
  )
)

portfolio <- create_portfolio_template()
portfolio$accounts$taxable <- c(10000, 30000)
portfolio <-
  portfolio |>
  calc_effective_tax_rate(
    tax_rate_ltcg = 0.20,
    tax_rate_ordinary_income = 0.40
  )
```

```
start_ages <- c(60, 65, 70)
scenarios_parameters <-
  tibble::tibble(
    member    = "older",
    event      = "retirement",
    start_age = start_ages,
    years     = Inf,
    end_age   = Inf
   ) |>
  dplyr::mutate(scenario_id = start_age) |>
  tidyr::nest(events = -scenario_id)

scenarios_parameters

scenarios <-
  simulate_scenarios(
    scenarios_parameters = scenarios_parameters,
    household            = household,
    portfolio            = portfolio,
    current_date         = "2020-07-15"
  )
scenarios$scenario_id |> unique()
```

Index