# Package 'tatooheene'

May 8, 2026

**Title** Technology Appraisal Toolbox for Health Economic Evaluations in the Netherlands

**Version** 1.0.0

**Description** Functions to support economic modelling in R based on the methods of the Dutch guideline for economic evaluations in healthcare <https://www.zorginstituutnederland.nl/documenten/2024/01/16/richtlijn-voor-het-uitvoeren-van-economische-evaluaties-in-de-gezondheidszorg>, CBS data <https://www.cbs.nl/>, and OECD data <https://www.oecd.org/en.html>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Depends** R (>= 4.1)

**LazyData** true

**Imports** rlang, dplyr (>= 1.1.4), assertthat (>= 0.2.1), tidyr (>= 1.3.0)

**Suggests** cbsodataR, lubridate, tibble, lifecycle, here, testthat (>= 3.0.0), knitr, rmarkdown

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Stijn Peeters [aut, cre] (ORCID: <https://orcid.org/0009-0004-3684-3584>), Eline Krijkamp [aut] (ORCID: <https://orcid.org/0000-0003-3970-2205>), Frederick Thielen [aut] (ORCID: <https://orcid.org/0000-0002-0312-5891>)

**Maintainer** Stijn Peeters <s.b.peeters@eshpm.eur.nl>

**Repository** CRAN

**Date/Publication** 2025-12-15 10:10:02 UTC

# Contents



---

`apply_discounting` *Calculate the discounted (net present value) of costs or effects*

---

## Description

**[Experimental]** The apply_discounting function is designed to calculate the net present value of future costs or effects using a constant discount rate, following the Dutch guidelines for economic evaluations in health care. (section 2.6.1.2 version 2024). Here's a breakdown of how the function works:

## Usage

```
apply_discounting(
  values,
  discount_rate = c("costs", "effects"),
  times,
  aggregate = FALSE,
  digits = NULL
)
```

## Arguments

| | |
|---|---|
| `values` | A numeric (vector of) costs or effects over time (one value per period). |
| `discount_rate` | Specifies the discount rate to be used. Acceptable values are: "costs" (0.03), "effects" (0.015), or a custom numeric value such as 0.04. |
| `times` | A numeric (vector of) time points indicating the time used for the discounting. The length must match the length of the values vector. Since the default discounting is annual, the time points should be in years. The length of this vector should be the same as the length of the `values` vector. When the first year is not discounted, the time points should start at 0 (e.g., c(0, 1, 2) for three years with NO discounting in the first year). When the first year is discounted, the time |

| | |
|---|---|
| | points should start at 1 (e.g., c(1, 2, 3) for three years WITH discounting in the first year). In case costs or effects are accrued in time steps other they annual, the time points should be adjusted accordingly, see more details in the vignettes of this package about discounting. |
| aggregate | A logical: indicating whether to sum the discounted values. Default is FALSE. |
| digits | A numeric value to indicate the number of digits to round the value. Default is 3 digits |

## Details

This function ensures consistent application of discount rates in cost-effectiveness analyses, in line with Dutch guidelines. Custom rates can be specified when needed.

## Examples

```
# NO Discounting in First Year (t starts at 0)
# constant cost of 100 for 3 years
apply_discounting(values = rep(100, 3), discount_rate = "costs", times = c(0, 1, 2))

# WITH discounting in first year (t starts at 1)
# Constant cost of 100 for 3 years,
apply_discounting(values = rep(100, 3),
                  discount_rate = "costs",
                  times = c(1, 2, 3))

# Present value of 100 euro in 3 years
apply_discounting(values = 100, discount_rate = "costs", times = 3)

# Custom Discount Rate
# discount rate of 4%, no discounting in first year
apply_discounting(values = rep(100, 3),
                  discount_rate = 0.04,
                  times = c(0, 1, 2))
# This will give you a messages to inform you about the different discount rate

# Same applies to utility values
# Utility values with aggregation - NO discounting in first year
apply_discounting(values = c(0.98, 0.82, 0.79),
                  discount_rate = "effect",
                  times = c(0, 1, 2),
                  aggregate = TRUE, digits = 3)
```

---

# data_model_output_sick_sicker

*Example model output for the sick-sicker model*

---

## Description

This data structure with several lists and vectors collecting a subset of the model outputs from the Sick-Sicker model intended for demonstrating the discounting in this package. The full model is described in the DARTH "Sick-Sicker" example. Source code can be found here: https://github.com/DARTH-git/cohort-modeling-tutorial-intro

## Usage

```
data_model_output_sick_sicker
```

## Format

An object of class `list` of length 9.

## Details

For the full reference read: Alarid-Escudero F, Krijkamp EM, Enns EA, Yang A, Hunink MGM, Pechlivanoglou P, Jalal H. An Introductory Tutorial on Cohort State-Transition Models in R Using a Cost-Effectiveness Analysis Example. Medical Decision Making, 2023;43(1):3-20. https://doi.org/10.1177/0272989X22110

These data can be used for demonstrating how to apply some of the package functions like discounting for the results of a model-based cost-effectiveness model.

#' @format A list with the following components

**l_m_M_annual** Annual transition matrix for sick-sicker model.

**l_m_M_monthly** Monthly transition matrix for sick-sicker model.

**v_wcc_annual** Half-cycle correction weights for annual model.

**v_wcc_monthly** Half-cycle correction weights for monthly model.

**v_names_str** Names of the intervention strategies used in the model.

**l_u_annual** Annual utilities for each health state.

**l_u_monthly** Monthly utilities for each health state.

**l_c_annual** Annual costs for each health state.

**l_c_monthly** Monthly costs for each health state.

## Examples

```
# Load the dataset
data("data_model_output_sick_sicker.rda")

# Load list in global environment
list2env(data_model_output_sick_sicker, envir = .GlobalEnv)
# Explore the available objects
ls(pattern = "l_|v_")

# View names of health states
v_names_str

# Inspect the first few cycles of the annual Markov trace for the first strategy
```

```
head(l_m_M_annual[[1]])

# Compare dimensions of annual and monthly traces
dim(l_m_M_annual[[1]])
dim(l_m_M_monthly[[1]])

# Apply the half cycle correction
## Loop through each strategy and calculate total utilities and costs ----

v_tot_qaly <- v_tot_cost <- c()
for (i in 1:length(v_names_str)) {
 v_u_str <- l_u_monthly[[i]]  # select the vector of state utilities for the i-th strategy
 v_c_str <- l_c_monthly[[i]]   # select the vector of state costs for the i-th strategy

  ###* Expected QALYs and costs per cycle
  ##* Vector of QALYs and Costs
  #* Apply state rewards
 v_qaly_str <- l_m_M_monthly[[i]] %*% v_u_str # sum the utilities of all states for each cycle
  v_cost_str <- l_m_M_monthly[[i]] %*% v_c_str # sum the costs of all states for each cycle

 ###* Total expected QALYs and costs per strategy and apply half-cycle correction (if applicable)
  #* QALYs
  v_tot_qaly[i] <- t(v_qaly_str) %*% v_wcc_monthly
  #* Costs
  v_tot_cost[i] <- t(v_cost_str) %*% v_wcc_monthly
}
```

---

`depreciation_interest` *Annual depreciation + interest for medical equipment*

---

## Description

**[Experimental]** Compute the annuity factor and the annual depreciation-and-interest charge for medical equipment, following Section 7.3 of the Dutch costing manual; k = annual depreciation and interest expense jaarlijkse afschrijvings- en rentekosten

Let V be replacement value, R the salvage value, n the amortisation period (years), and i the interest rate (per year). The annuity factor is:

$$a_{n,i} = \frac{1}{i} * \left(1 - \frac{1}{(1+i)^n}\right)$$

#' The annual charge k is:

$$k = \frac{V - \frac{R}{(1+i)^n}}{a_{n,i}}$$

## Usage

```
depreciation_interest(
  v_replace_val,
```

```
  r_salvage_val,
  n_amortisation_period = 10,
  i_interest_rt = 0.025,
  output = c("dataframe", "annuity_factor", "annual_cost")
)
```

## Arguments

| | |
|---|---|
| v_replace_val | V: vervangingswaarde; replacement value (numeric scalar, > 0) |
| r_salvage_val | R: restwaarde; salvage (residual) value at end of period (numeric scalar, >= 0) |
| n_amortisation_period | n: afschrijvingstermijn; amortisation period in years (numeric scalar, > 0) |
| i_interest_rt | i: rentepercntage; annual interest rate as a decimal (numeric scalar, >= 0) |
| output | One of dataframe (default), annuity_factor, or annual_cost. |

## Value

- If output = "dataframe": a data.frame with two columns: Annuity factor and Yearly depreciation and interes
- If output = "annuity_factor": a single numeric (the annuity factor).
- If output = "annual_cost": a single numeric (the annual charge k).

## Examples

```
# Both values as a data frame (defaults: n=10, i=2.5%)
depreciation_interest(v_replace_val = 50000, r_salvage_val = 5000)

# Only the annuity factor
depreciation_interest(50000, 5000, output = "annuity_factor")

# Only the annual charge (k)
depreciation_interest(50000, 5000, output = "annual_cost")

# Zero interest (uses the i -> 0 limit): a = n, k = (V - R)/n
depreciation_interest(50000, 5000, n_amortisation_period = 8,
                      i_interest_rt = 0,
                      output = "dataframe")
```

---

# df_cpi_combined *Consumer Price Index (CPI) data from CBS*

---

## Description

A subset data frame of Consumentenprijzen; prijsindex 2015=100 from CBS. Identifier: 83131NED

## Usage

```
df_cpi_combined
```

## Format

`df_cpi_combined`**:**
A data frame with 11 rows and 8 columns:

**Year from** Year from in case of two consecutive years
**Year to** Year ending in case of two consecutive years
**Percentage** Percentage change in case of two consecutive years
**Factor** Factor for multiplication in case of two consecutive years
**Year from'** Year starting from, in the case of a range, the year up to the maximum year
**Year to'** Year ending in case of a range, the year up to the maximum year
**Percentage'** Percentage for multiplication in case of a range, the year up to the maximum year
**Factor'** Factor for multiplication in case of a range, the year up to the maximum year

## Source

https://www.cbs.nl/nl-nl/cijfers

---

# df_fp *Job vacancy data from CBS*

---

## Description

A subset data frame of job vacancies; SBI 2008; by economic activity and company size from CBS. Identifier: 80472NED

## Usage

```
df_fp
```

## Format

`df_fp`**:**
A data frame with 27 rows and 7 columns:

**Year** Year
**Filled vacancies** Vervulde vacatures
**Open vacancies** Openstaande vacatures
**Friction period in days** Calculation of the friction period in days
**Friction period in weeks** Calculation of the friction period in weeks
**Friction period days average over 5 years** Calculation of the 5 year average friction period in days
**Friction period weeks average over 5 years** Calculation of the 5 year average friction period in weeks

## Source

https://www.cbs.nl/nl-nl/cijfers

## df_ppp *Purchasing Power Parity (PPP) data from OECD*

### Description

A subset data frame of the OECD data set containing the Purchasing Power Parity (PPP) data.

### Usage

```
df_ppp
```

### Format

`df_ppp`**:**
A data frame with 64 rows and 2 columns:

**Year** Year of PPP
**PPP** Purchase Power Parity

### Source

https://sdmx.oecd.org/public

## df_ref_prices *Complete data of reference values from the costing manual*

### Description

Complete data of reference values from the costing manual

### Usage

```
df_ref_prices
```

### Format

`df_ref_prices`**:**
A data frame with 160 rows and 7 columns:

**Domain** Care domain, for example medical services
**Category** Service category such as nursing, intensive care or day treatment
**Unit** Description of the care unit or service
**short_var** Short variable name used in the package
**2022** Reference price for the year 2022
**2023** Reference price for the year 2023
**2024** Reference price for the year 2024

## Source

https://sdmx.oecd.org/public

---

`friction_period` *Friction period lookup (days/weeks; 1-year or 5-year average)*

---

## Description

**[Experimental]** Return friction periods from the internal CBS-based table.

## Usage

```
friction_period(
  year = NULL,
  units = "weeks",
  avg = "5yr",
  output = c("tibble", "value"),
  data = tatooheene::df_fp
)
```

## Arguments

| | |
|---|---|
| `year` | Integer vector of years to return. If `NULL`, returns all years. |
| `units` | One or more of `c("days","weeks")`. Default: `"weeks"`. |
| `avg` | One or more of `c("1yr","5yr")`. Default: `"5yr"`. |
| `output` | Either `"tibble"` (default) or `"value"`. If `"value"`, you must request exactly one year and one `(units, avg)` combo. |
| `data` | Data source (mainly for testing); default is `tatooheene::df_fp`. |

## Value

A tibble when `output = "tibble"`, or a single numeric when `output = "value"`.

## Examples

```
# All years, 5-year average in weeks (default)
friction_period()

# Specific year (2019), weeks 5-year average
friction_period(year = 2019)

# Days (1-year) for multiple years
friction_period(year = 2018:2020, units = "days", avg = "1yr")

# Single numeric value (requires one year + one combo)
friction_period(year = 2019, units = "weeks", avg = "5yr", output = "value")
```

## nl_ppp — *A function to obtain the Dutch PPP factor values in International Dollar (INT$)*

### Description

**[Experimental]** This function downloads the Purchasing Power Parity (PPP) factor values for the Netherlands from the OECD website per year in International Dollar (Int$).

### Usage

```
nl_ppp(year = "all")
```

### Arguments

| | |
|---|---|
| year | The year of which the PPP factor should be downloaded, multiple years are possible, default is the whole dataset. |

### Value

A dataframe or value with the PPP factor values for the specified years.

### Examples

```
# Example usage of the nl_ppp function
nl_ppp(year = 2019)
```

## nl_price_index — *A function to calculate the Consumer Price Index (CPI) for a given year range.*

### Description

**[Experimental]** This function provides the Consumer Price Index (CPI) for a given year range both in a factor or dataframe based on CBS data and further described in 2.6.1.1 of the Dutch EE guideline

### Usage

```
nl_price_index(
  start_year = 2013,
  end_year = 2023,
  output = c("table", "factor")
)
```

### Arguments

| | |
|---|---|
| start_year | start year for CPI output table or factor |
| end_year | End year for CPI output table or factor |
| output | Which output we would like to see. "factor": is the factor from start to end year, "table" is the table of all CPIs from start to end year |

### Value

Dataframe or factor with CPI data from start year to end year

### Examples

```
# Example usage of the nl_price_index function
# Get the CPI factor from 2013 to 2023
nl_price_index(start_year = 2013, end_year = 2023, output = "factor")

# Get the CPI table from 2013 to 2023
nl_price_index(start_year = 2013, end_year = 2023, output = "table")
```

---

## nl_ref_prices *A function to download the Reference prices of the Dutch Costing Manual for one or multiple years*

---

### Description

**[Experimental]** This function downloads the Reference prices of the Dutch Costing Manual for one or multiple years. The prices are available in Euro (EUR) or International Dollar (INT$).

### Usage

```
nl_ref_prices(
  year = "all",
  domain = "all",
  category = "all",
  unit = "all",
  short_unit = "all",
  currency = c("EUR", "INT$")
)
```

### Arguments

| | |
|---|---|
| year | The year of which the reference price should be downloaded, multiple years are possible, default is the whole dataset |
| domain | The domain of prices that should be included (one or more categories), default is including all categories |

| | |
|---|---|
| category | The category of prices that should be included (one or more categories), default is including all categories |
| unit | The reference price that should be included (one or multiple reference prices), default is including the whole dataframe |
| short_unit | The short variable name that should be included (one or more short variables), default is including all |
| currency | The currency of the output of the prices. A decision can be made between EUR and INT$, the default is EUR. |

## Value

A dataframe or value with the Medical Reference price(s) of the Dutch Costing Manual for the specified years

## Examples

```
# Example usage of the nl_med_prices function
# Calculate for year 2024 with the category Nursing
nl_ref_prices(year = "2024", category = "Nursing")

# Calculate for year 2022 and 2023 the category Nursing
nl_ref_prices(year = c(2022,2023), category = "Nursing")

# Calculate for year 2022 with the category Nursing in INT$
nl_ref_prices(year = "2022", category = "Nursing" , currency = "INT$")
```

---

# pretty_price *A function to write pretty prices in* bookdown *reports*

---

## Description

**[Experimental]** This function writes pretty prices in bookdown reports. The function uses the `formatC()` function to format the number and adds the currency to the end of the number.

## Usage

```
pretty_price(x, digits = 2, currency = "EUR", ...)
```

## Arguments

| | |
|---|---|
| x | A number to be printed |
| digits | Number of digits |
| currency | The name of the currency |
| ... | Extra arguments for `formatC()` |

## Value

A pretty price with the currency

## Examples

```
# Example usage of the pretty_price function
pretty_price(1000, currency = "EUR")
```

# Index