Package: hicp (via r-universe)

October 25, 2024

Type Package

Title Harmonised Index of Consumer Prices

Version 0.6.1

Description The Harmonised Index of Consumer Prices (HICP) is the key economic figure to measure inflation in the euro area. The methodology underlying the HICP is documented in the HICP Methodological Manual (<https://ec.europa.eu/eurostat/web/products-manuals-and-guidelines/w/ks-gq-24-003>). Based on the manual, this package provides functions to access and work with HICP data from Eurostat's public database (<https://ec.europa.eu/eurostat/data/database>).

License EUPL

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports restatapi (>= 0.21.0), data.table (>= 1.14.0)

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

URL https://github.com/eurostat/hicp

BugReports https://github.com/eurostat/hicp/issues

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

RemoteUrl https://github.com/eurostat/hicp

RemoteRef HEAD

RemoteSha 6ae8b9f956f49652f716cc29fe10ef13115e7f17

Contents

`chaining` *Chain-linking, rebasing and frequency conversion*

Description

Function `unchain()` decouples a chained index series with monthly frequency. These unchained index series can be aggregated into higher-level indices using `aggregate()`. To obtain a longterm index series, the higher-level indices must be chained using function `chain()`. Finally, `rebase()` sets the index reference period. Monthly indices can be converted into annual or quarterly indices using function `convert()`.

Usage

```
unchain(x, t, by=12)

chain(x, t, by=12)

rebase(x, t, t.ref, verbose=FALSE)

convert(x, t, freq="annual")
```

Arguments

`x`	numeric vector of index values
`t`	date vector
`by`	for annual overlap `NULL`; for one-month overlap a single integer between 1 and 12 specifying the price reference month
`t.ref`	character specifying the index reference period. Could be a whole year (`YYYY`) or a single year-month (`YYYY-MM`).
`verbose`	logical indicating if messages regarding the index reference period should be printed to the console or not.
`freq`	frequency of converted index. Either `annual` or `quarterly`.

Details

Function `unchain()` sets the value of the first price reference period to `NA` although the value could be set to 100 (if `by` is not `NULL`) or 100 divided by the average of the year (if `by=NULL`). This is wanted to avoid aggregation of these values. Function `chain()` finally sets the values back to 100.

Value

Functions `unchain()`, `chain()` and `rebase()` return numeric values of the same length as `x`.

Function `convert()` returns a named vector of the length of quarter or years available in `t`, where the names correspond to the years or quarters.

Author(s)

Sebastian Weinand

References

European Commission, Eurostat, *Harmonised Index of Consumer Prices (HICP) - Methodological Manual - 2024 edition*, Publications Office of the European Union, 2024, https://data.europa.eu/doi/10.2785/055028.

See Also

aggregate

Examples

```
### EXAMPLE 1

t <- seq.Date(from=as.Date("2021-12-01"), to=as.Date("2024-12-01"), by="1 month")
p <- rnorm(n=length(t), mean=100, sd=5)

100*p/p[1]
chain(unchain(p, t, by=12), t, by=12)

convert(x=p, t=t, freq="q") # quarterly index

t <- seq.Date(from=as.Date("2021-01-01"), to=as.Date("2024-12-01"), by="1 month")
p <- rnorm(n=length(t), mean=100, sd=5)

100*p/mean(p[1:12])
(res <- chain(unchain(p, t, by=NULL), t, by=NULL))
# note that for backwards compability, each month in the first
# year receives an index value of 100. this allows the same
# computation again:
chain(unchain(res, t, by=NULL), t, by=NULL)

### EXAMPLE 2

# set cores for testing on CRAN:
library(restatapi)
```

```
options(restatapi_cores=1)
library(data.table)

# get hicp index values for euro area with base 2015:
dt <- hicp.dataimport(id="prc_hicp_midx", filter=list(unit="I15", geo="EA"))
dt[, "time":=as.Date(paste0(time, "-01"))]
setkeyv(x=dt, cols=c("unit","coicop","time"))

# check chain-linked indices against published data:
dt[, "dec_ratio" := unchain(x=values, t=time), by="coicop"]
dt[, "chained_index" := chain(x=dec_ratio, t=time), by="coicop"]
dt[, "index_own" := rebase(x=chained_index, t=time, t.ref="2015"), by="coicop"]
dt[abs(values-index_own)>0.01,] # should be empty

# check converted indices against published data:
dta <- dt[, as.data.table(convert(x=values, t=time), keep.rownames=TRUE), by="coicop"]
setnames(x=dta, c("coicop","time","index"))
aind <- hicp.dataimport(id="prc_hicp_aind", filter=list(unit="INX_A_AVG", geo="EA"))
aind[, c("geo","unit") := NULL]
dtcomp <- merge(x=aind, y=dta, by=c("coicop","time"), all=TRUE)
dtcomp[abs(values-index)>0.01,] # should be empty
```

`coicop` *Working with COICOP codes*

Description

Function `is.coicop()` checks if the input is a valid COICOP code while `level()` returns the COICOP level (e.g. division or subclass). Function `parent()` derives the higher-level parent of a COICOP code if available in the data supplied. The function `child()` does the same for lower-level children.

Usage

```
is.coicop(id, settings=list())

level(id, label=FALSE, settings=list())

child(id, flag=TRUE, direct=FALSE, settings=list())

parent(id, flag=TRUE, direct=FALSE, settings=list())
```

Arguments

`id`	character vector of COICOP ids.
`label`	logical indicating if the number of digits or the labels (e.g., division, subclass) should be returned.
`flag`	for `flag=TRUE`, parent or child codes are available in the data are flagged by a logical. Otherwise, the parent or child codes are returned.

`direct` logical indicating if only direct relatives should be flagged as `TRUE` (e.g. `03->031`) or also indirect relatives (e.g. `03->0311`) if direct relatives in between are missing.

`settings` a list of control settings to be used. The following settings are supported:

- `coicop.version` : character specifying the COICOP version to be used when checking for valid COICOP codes. See details for the allowed values. The default is `getOption("hicp.coicop.version")`.
- `unbundle` : logical indicating if COICOP bundles (e.g. 08X, 0531_2) as defined in `coicop.bundles` should be taken into account or not. The default is `getOption("hicp.unbundle")`.
- `all.items.code` : character specifying the code internally used for the all-items index. The default is taken from `getOption("hicp.all.items.code")`. Not used by `is.coicop()`.

Details

The following COICOP versions are supported:

- Classification of Individual Consumption According to Purpose (COICOP-1999): `coicop1999`
- European COICOP (ECOICOP): `ecoicop`
- ECOICOP adopted to the needs of the HICP (ECOICOP-HICP): `ecoicop-hicp`
- COICOP-2018: `coicop2018`

None of the COICOP versions include a code for the all-items index. The internal package code for the all-items index is globally defined by `options(hicp.all.items.code="00")` but can be changed by the user. The `level()` is always 1.

If `settings$unbundle=TRUE`, COICOP bundle codes are resolved into their component ids and processed in that way. By contrast, if `settings$unbundle=FALSE`, COICOP bundle codes are internally set to `NA`. Consequently, they can't be a parent or a child of some other COICOP code.

Value

Function `is.coicop()` returns a logical vector and function `level()` a numeric vector. If argument `flag=TRUE`, functions `parent()` and `child()` both return a logical vector. If `flag=FALSE`, `parent()` gives a character vector, while `child()` returns a list. In any case, all function outputs have the same length as `id`.

Author(s)

Sebastian Weinand

See Also

`unbundle`, `tree`

Examples

```
### EXAMPLE 1

# validity of coicop id:
is.coicop(id=c("00","CP00","13","08X"), settings=list(unbundle=TRUE))
is.coicop(id=c("00","CP00","13","08X"), settings=list(unbundle=FALSE))

# coicop level:
level(id=c("00","05","053","0531_2"))
level(id=c("00","05","053","0531_2"), label=TRUE)

# check for children in data:
child(id=c("0111"), flag=FALSE) # false, no child found
child(id=c("0111", "01"), flag=FALSE, direct=TRUE) # still false
child(id=c("0111", "01"), flag=FALSE, direct=FALSE) # now TRUE

# check for parent in data, including coicop bundles:
ids <- c("053","0531_2","05311","05321")
parent(id=ids, flag=FALSE, direct=TRUE, settings=list(unbundle=FALSE))
parent(id=ids, flag=FALSE, direct=TRUE, settings=list(unbundle=TRUE))

### EXAMPLE 2

# set cores for testing on CRAN:
library(restatapi)
options(restatapi_cores=1)
library(data.table)

# load hicp item weights:
coicops <- hicp.dataimport(id="prc_hicp_inw", filter=list(geo="EA"))
coicops <- coicops[grepl("^CP", coicop),]
coicops[, "coicop":=gsub("^CP", "", coicop)]

# get frequency of coicop levels:
coicops[, .N, by=list(time, "lvl"=level(coicop))]

# get coicop parent from the data:
coicops[, "parent":=parent(id=coicop, flag=FALSE), by="time"]

# flag if coicop has child available in the data:
coicops[, "has_child":=child(id=coicop, flag=TRUE), by="time"]
coicops[has_child==FALSE, sum(values, na.rm=TRUE), by="time"]
# coicop bundles and their component ids are both taken into
# account. this double counting explains some differences
```

`coicop.bundles` *COICOP bundle codes*

Description

HICP data follow the COICOP classification system. However, sometimes COICOP ids are merged into bundles, deviating from the usual structure of ids (e.g. 08X, 0531_2). Function `is.bundle()` flags if a COICOP id is a bundle or not, while `unbundle()` splits the bundles into their original ids. Both functions make use of the bundle dictionary `coicop.bundles`.

Usage

```
is.bundle(id)

unbundle(id)

# list of coicop bundles:
coicop.bundles
```

Arguments

`id`	character vector of COICOP ids.

Value

For `is.bundle()`, a logical vector of the same length as `id`. For `unbundle()` a vector of ids with length greater or equal to the length of `id`.

Author(s)

Sebastian Weinand

Examples

```
ids <- c("011",NA,"08X","112","0531_2")
is.bundle(ids)
unbundle(ids)
```

---

coicop.tree *Derive and fix COICOP tree*

Description

Function `tree()` derives the COICOP tree at the lowest possible level. In HICP data, this can be done separately for each reporting month and country. Consequently, the COICOP tree can differ across space and time. If needed, specifying argument by in `tree()` allows to merge the COICOP trees at the lowest possible level, e.g. to obtain a unique composition of COICOP codes over time.

Usage

```
tree(id, by=NULL, w=NULL, max.lvl=NULL, settings=list())
```

Arguments

id	character vector of COICOP ids
by	vector specifying the variable to be used for merging the tree, e.g. vector of dates for merging over time or a vector of countries for merging across space. Can be NULL if no merging is required.
w	numeric weight of id. If supplied, it is checked that the weight of children add up to the corresponding weight of the parent (allowing for tolerance w.tol). If w=NULL (the default), no checking of weight aggregation is performed.
max.lvl	integer specifying the maximum depth or deepest COICOP level allowed. If NULL (the default), the deepest level found in id is used.
settings	a list of control settings to be used. The following settings are supported:

- coicop.version : the COICOP version to be used when checking for valid COICOP codes. See coicop for the allowed values. The default is getOption("hicp.coicop.version").
- unbundle : logical indicating if COICOP bundles (e.g. 08X, 0531_2) as defined in coicop.bundles should be taken into account or not. The default is getOption("hicp.unbundle").
- all.items.code : character specifying the code internally used for the all-items index. The default is taken from getOption("hicp.all.items.code").
- w.tol : numeric tolerance for checking of weights. Only relevant in case w is not NULL. The default is 1/100.

Value

A logical vector of the same length as id.

Author(s)

Sebastian Weinand

See Also

unbundle, child

Examples

```
### EXAMPLE 1

# flag lowest possible level to be used as COICOP tree:
tree(id=c("01","011","012"), w=NULL) # true
tree(id=c("01","011","012"), w=c(0.2,0.08,0.12)) # true, weights add up
tree(id=c("01","011","012"), w=c(0.2,0.08,0.10)) # false, weights do not add up

# set maximum (or deepest) coicop level to 3:
tree(id=c("01","011","012","0111","0112","01121"),
     w=c(0.2,0.08,0.12,0.02,0.06,0.06),
     max.lvl=3)
```

```
# maximum level=3, but weights do not add up:
tree(id=c("01","011","012","0111","0112","01121"),
     w=c(0.2,0.08,0.07,0.02,0.06,0.06),
     max.lvl=3)

# coicop bundles:
tree(id=c("08","081","082_083"), w=c(0.25,0.05,0.2))
tree(id=c("08","081","082_083"), w=c(0.25,0.05,0.2), settings=list(unbundle=FALSE))

# merge (or fix) coicop tree over time:
tree(id=c("08","081","082","08"), by=c(1,1,1,2))

### EXAMPLE 2

# set cores for testing on CRAN:
library(restatapi)
options(restatapi_cores=1)
library(data.table)

# load hicp item weights:
coicops <- hicp.dataimport(
              id="prc_hicp_inw",
              filter=list(geo=c("EA","DE","FR")),
              date.range=c("2005", NA))
coicops <- coicops[grepl("^CP", coicop),]
coicops[, "coicop":=gsub("^CP", "", coicop)]

# derive seperate trees for each time period and country:
coicops[, "t1" := tree(id=coicop, w=values, settings=list(w.tol=0.1)), by=c("geo","time")]
coicops[t1==TRUE,
        list("n"=uniqueN(coicop),           # varying coicops over time and space
             "w"=sum(values, na.rm=TRUE)),  # weight sums should equal 1000
        by=c("geo","time")]

# derive merged trees over time, but not across countries:
coicops[, "t2" := tree(id=coicop, by=time, w=values, settings=list(w.tol=0.1)), by="geo"]
coicops[t2==TRUE,
        list("n"=uniqueN(coicop),           # same selection over time in a country
             "w"=sum(values, na.rm=TRUE)),  # weight sums should equal 1000
        by=c("geo","time")]

# derive merged trees over countries and time:
coicops[, "t3" := tree(id=coicop, by=paste(geo,time), w=values, settings=list(w.tol=0.1))]
coicops[t3==TRUE,
       list("n"=uniqueN(coicop),          # same selection over time and across countries
             "w"=sum(values, na.rm=TRUE)),  # weight sums should equal 1000
        by=c("geo","time")]
```

---

countries *Country metadata*

Description

This dataset contains metadata for the euro area, EU, EFTA, and candidate countries that submit(ted) HICP data on a regular basis.

Usage

```
# country metadata:
countries
```

Format

A data.table with metadata on the individual euro area (EA), EU, EFTA, and candidate countries producing the HICP.

- `code`: the country code
- `name_[en|fr|de]`: the country name in English, French, and German
- `protocol_order`: the official protocol order of countries
- `is_eu`, `is_ea`, `is_efta`, `is_candidate`: a logical indicating if a country belongs to the EU, the euro area, or if it's an EFTA or candidate country, respectively
- `eu_since`, `eu_until`: date of joining and leaving the European Union
- `ea_since`: the date of introduction of the euro as the official currency
- `index_decimals`: the number of index decimals used for dissemination

Author(s)

Sebastian Weinand

Examples

```
# subset to euro area countries:
countries[is_ea==TRUE, ]
```

---

hicp.data *Download HICP data*

Description

These functions are simple wrappers of functions in the `restatapi` package. Function `hicp.datasets()` lists all available HICP datasets in Eurostat's public database, while `hicp.datafilters()` gives the allowed values that can be used for filtering a dataset. `hicp.dataimport()` downloads a specific dataset with filtering on key parameters and time, if supplied.

Usage

```
hicp.datasets()

hicp.datafilters(id)

hicp.dataimport(id, filters=list(), date.range=NULL, flags=FALSE)
```

Arguments

id	A dataset identifier, which can be obtained from `hicp.datasets()`.
filters	A named list of filters to be applied to the data request. Allowed values for filtering can be retrieved from `hicp.datafilters()`. For HICP data, typical filter variables are the index reference period (`unit: I96, I05, I15`), the country (`geo: EA, DE, FR, ...`), or the COICOP code (`coicop: CP00, CP01, SERV, ...`).
date.range	A vector of start and end date used for filtering on time dimension. These must follow the pattern `YYYY(-MM)?`. An open interval can be defined by setting one date to `NA`.
flags	A logical indicating if data flags should be returned or not.

Value

A data.table.

Author(s)

Sebastian Weinand

Source

See Eurostat's public database at https://ec.europa.eu/eurostat/web/main/data/database.

See Also

get_eurostat_toc, get_eurostat_dsd, get_eurostat_data

Examples

```
# set cores for testing on CRAN:
library(restatapi)
options(restatapi_cores=1)

# view available datasets:
hicp.datasets()

# get allowed filters for item weights:
hicp.datafilters(id="prc_hicp_inw")

# download item weights for euro area from 2015 on:
hicp.dataimport(id="prc_hicp_inw", filters=list("geo"="EA"), date.range=c("2015", NA))
```

`index.aggregation` *Index number functions and aggregation*

Description

Lower-level price relatives or price indices can be aggregated into higher-level indices in a single step using one of the bilateral index number methods listed below. Function `aggregate()` uses these bilateral indices (or others defined by the user) for step-wise aggregation of lower-level subindices into the overall index following the COICOP hierarchy.

Usage

```
# bilateral price indices:
jevons(x, w0=NULL, wt=NULL)
carli(x, w0=NULL, wt=NULL)
harmonic(x, w0=NULL, wt=NULL)
laspeyres(x, w0, wt=NULL)
paasche(x, w0=NULL, wt)
fisher(x, w0, wt)
toernqvist(x, w0, wt)
walsh(x, w0, wt)

# step-wise index aggregation:
aggregate(x, w0, wt, grp, index=laspeyres, add=list(), settings=list())
```

Arguments

`x`	numeric vector of price relatives obtained by unchaining some HICP index series.
`w0`, `wt`	numeric vector of weights in the base period `w0` (e.g., for the Laspeyres index) or current period `wt` (e.g., for the Paasche index), respectively.
`grp`	grouping variable to be used. These must be valid COICOP codes according to `is.coicop()`.
`index`	a function or named list of functions specifying the index formula used for aggregation. Each function must have arguments `x`, `w0` and `wt`, even if `w0` and/or `wt` are not used (this can be indicated by setting this argument to `NULL`). Each function must return a scalar. The default is `index=laspeyres` since the HICP is calculated as a Laspeyres-type index.
`add`	a named list of user-defined aggregates to be calculated. Each list element is a vector of ids that can be found in `grp`. See `settings$add.exact` for further specification of this argument.
`settings`	A list of control settings to be used. The following settings are supported: • `keep.lowest` : logical indicating if the lowest-level indices that form the base of all aggregation steps should be kept in the function output. The default is `TRUE`.

- `add.exact` : logical indicating if the ids in `add` must **all** be present in `grp` for aggregation or not. If `FALSE`, aggregation is carried out using the available ids in `add`. If `TRUE` and some ids are missing in `add`, `NA` is returned. The default is `TRUE`.
- `coicop.version` : the COICOP version to be used when checking for valid COICOP codes. See `coicop` for the allowed values. The default is `getOption("hicp.coicop.version")`.
- `unbundle` : logical indicating if COICOP bundles (e.g. 08X, 0531_2) as defined in `coicop.bundles` should be taken into account or not. The default is `getOption("hicp.unbundle")`.
- `all.items.code` : character specifying the code internally used for the all-items index. The default is taken from `getOption("hicp.all.items.code")`.

Details

The price indices currently available use price relatives `x`. The Dutot index is therefore not implemented.

The functions `jevons()`, `carli()`, and `harmonic()` do not make use of any weights in the calculations. However, they are implemented in a way such that the weights `w0` are considered, that is, elements in `x` where the weight `w0` is `NA` are excluded from the calculations. This mimics the behavior of the weighted index functions like `laspeyres()` and can be useful in situations where indices are present but the weight is missing. If, for example, subindices are newly introduced, the index in December is usually set to 100 while the weight of this subindex is not available. The subindex's value in December can thus be excluded by using the weights `w0` also in the unweighted price indices.

Value

Functions `jevons()`, `carli()`, `harmonic()`, `laspeyres()`, `paasche()`, `fisher()`, `toernqvist()`, and `walsh()` return a single (aggregated) value.

Function `aggregate()` returns a `data.table` of aggregated values at each `grp`-level with the following variables:

`grp`	*character*	the grouping variable
`is_aggregated`	*logical*	is the value an aggregate (`TRUE`) or not (`FALSE`); column available if `settings$keep.lowes`
`w0, wt`	*numeric*	sum of weights `w0` and `wt`; columns available if weights were provided
`index`	*numeric*	aggregates for each `index` function

Author(s)

Sebastian Weinand

References

European Commission, Eurostat, *Harmonised Index of Consumer Prices (HICP) - Methodological Manual - 2024 edition*, Publications Office of the European Union, 2024, https://data.europa.eu/doi/10.2785/055028.

See Also

unchain, chain, rebase

Examples

```
library(data.table)

### EXAMPLE 1

# data for two times periods:
dt <- data.table(
  "time"=rep(1:2, each=5),
  "coicop"=rep(c("01111","01112","0112","0113","021"), times=2),
  "price"=c(105,103,102,99,120, 105,104,110,98,125),
  "weight"=rep(c(0.05,0.15,0.3,0.2,0.3), times=2),
  "weight_lag"=rep(c(0.03,0.12,0.33,0.2,0.32), times=2))

# aggregate directly to overall index:
dt[, laspeyres(x=price, w0=weight), by="time"]

# gives identical results at top level as with stepwise
# aggregation through all coicop levels:
dt[, aggregate(x=price, w0=weight, grp=coicop, index=laspeyres), by="time"]

# this is no longer the case for the superlative indices as shown
# here for the walsh index:
dt[, walsh(x=price, w0=weight, wt=weight_lag), by="time"]
dt[, aggregate(x=price, w0=weight, wt=weight_lag, grp=coicop, index=walsh), by="time"]

# see also for example Auer and Wengenroth (2017, p. 2)

# apply user-defined function:
dt[, aggregate(x=price, w0=weight, grp=coicop,
               index=list("carli"=function(x,w0=NULL,wt=NULL) mean(x))),
     by="time"]

# add additional, user-defined aggregates (e.g. special aggregates):
dt[, aggregate(x=price, w0=weight, grp=coicop,
               add=list("FOOD"=c("01111","021"), "MISS"=c("021","09"))),
     by="time"]

# aggregate 'MISS' is computed if settings$add.exact=FALSE:
dt[, aggregate(x=price, w0=weight, grp=coicop,
               add=list("FOOD"=c("01111","021"), "MISS"=c("021","09")),
               settings=list("add.exact"=FALSE)),
     by="time"]

### EXAMPLE 2: Index aggregation using published HICP data

# set cores for testing on CRAN:
library(restatapi)
options(restatapi_cores=1)
```

```
# import monthly price indices:
prc <- hicp.dataimport(id="prc_hicp_midx", filter=list(unit="I15", geo="EA"))
prc[, "time":=as.Date(paste0(time, "-01"))]
prc[, "year":=as.integer(format(time, "%Y"))]
setnames(x=prc, old="values", new="index")

# unchaining indices:
prc[, "dec_ratio" := unchain(x=index, t=time), by="coicop"]

# import item weights:
inw <- hicp.dataimport(id="prc_hicp_inw", filter=list(geo="EA"))
inw[, "time":=as.integer(time)]
setnames(x=inw, old=c("time","values"), new=c("year","weight"))

# derive coicop tree:
inw[grepl("^CP",coicop),
    "tree":=tree(id=gsub("^CP","",coicop), w=weight, settings=list(w.tol=0.1)),
    by=c("geo","year")]

# except for rounding, we receive total weight of 1000 in each period:
inw[tree==TRUE, sum(weight), by="year"]

# merge price indices and item weights:
hicp.data <- merge(x=prc, y=inw, by=c("geo","coicop","year"), all.x=TRUE)
hicp.data <- hicp.data[year <= year(Sys.Date())-1 & grepl("^CP\\d+", coicop),]
hicp.data[, "coicop" := gsub(pattern="^CP", replacement="", x=coicop)]

# compute all-items HICP:
hicp.own <- hicp.data[tree==TRUE,
                      list("laspey"=laspeyres(x=dec_ratio, w0=weight)),
                      by="time"]
setorderv(x=hicp.own, cols="time")
hicp.own[, "chain_laspey" := chain(x=laspey, t=time, by=12)]
hicp.own[, "chain_laspey_15" := rebase(x=chain_laspey, t=time, t.ref="2015")]

# add published all-items HICP for comparison:
hicp.own <- merge(
            x=hicp.own,
            y=hicp.data[coicop=="00", list(time, index)],
            by="time",
            all.x=TRUE)
plot(index-chain_laspey_15~time, data=hicp.own, type="l")
head(hicp.own[abs(index-chain_laspey_15)>0.1,])

# compute all-items HICP stepwise through all higher-levels:
hicp.own.all <- hicp.data[, aggregate(x=dec_ratio, w0=weight, grp=coicop, index=laspeyres),
                          by="time"]
setorderv(x=hicp.own.all, cols="time")
hicp.own.all[, "chain_laspey" := chain(x=laspeyres, t=time, by=12), by="grp"]
hicp.own.all[, "chain_laspey_15" := rebase(x=chain_laspey, t=time, t.ref="2015"), by="grp"]

# add published indices for compariosn:
```

```
hicp.own.all <- merge(
                x=hicp.own.all,
                y=hicp.data[, list(time,"grp"=coicop,index,weight)],
                by=c("time","grp"),
                all.x=TRUE)
hicp.own.all[, "diff" := index-chain_laspey_15]
head(hicp.own.all[abs(diff)>0.1,])
head(hicp.own.all[abs(w0-weight)>0.1,])

# compare all-items HICP from direct and step-wise aggregation:
agg.comp <- merge(
        x=hicp.own.all[grp=="00", list(time, "index_stpwse"=chain_laspey_15)],
        y=hicp.own[, list(time, "index_direct"=chain_laspey_15)],
        by="time")

# no differences -> consistent in aggregation:
head(agg.comp[abs(index_stpwse-index_direct)>1e-4,])
```

`linking` *Linking-in new index series*

Description

Function `link()` links a new index series to an existing one by an overlap period supplied. In the resulting linked index series, the new index series starts after the existing one. Function `lsf()` computes the level-shift factors for linking via the overlap periods in `t.overlap`. The level-shift factors can be applied to an index series that has already been linked by the standard HICP one-month overlap method using December of year `t-1`.

Usage

```
link(x, x.new, t, t.overlap=NULL)

lsf(x, x.new, t, t.overlap=NULL)
```

Arguments

`x`, `x.new`	numeric vector of index values. `NA`-values in the vectors indicate when the index series discontinues (for `x`) or starts (for `x.new`).
`t`	date vector
`t.overlap`	character specifying the overlap period to be used. Could be a whole year (`YYYY`) or a single year-month (`YYYY-MM`). Multiple periods can be provided. If `NULL`, all available overlap periods are considered.

Value

Function `link()` returns a numeric vector or a matrix of the same length as `t`, while `lsf()` provides a named numeric vector of the same length as `t.overlap`.

Author(s)

Sebastian Weinand

See Also

chain

Examples

```
# input data:
set.seed(1)
t <- seq.Date(from=as.Date("2015-01-01"), to=as.Date("2024-05-01"), by="1 month")
x.new <- rnorm(n=length(t), mean=100, sd=5)
x.new <- rebase(x=x.new, t=t, t.ref="2019-12")
x.old <- x.new + rnorm(n=length(x.new), sd=5)
x.old <- rebase(x=x.old, t=t, t.ref="2015")
x.old[t>as.Date("2021-12-01")] <- NA # current index discontinues in 2021
x.new[t<as.Date("2020-01-01")] <- NA # new index starts in 2019-12

# linking in new index in different periods:
plot(x=t, y=link(x=x.old, x.new=x.new, t=t, t.overlap="2021-12"),
     col="red", type="l", xlab=NA, ylab="Index", ylim=c(80,120))
lines(x=t, y=link(x=x.old, x.new=x.new, t=t, t.overlap="2020"), col="blue")
lines(x=t, y=link(x=x.old, x.new=x.new, t=t, t.overlap="2021"), col="green")
lines(x=t, y=x.old, col="black")
abline(v=as.Date("2021-12-01"), lty="dashed")
legend(x="topleft",
       legend=c("One-month overlap using December 2021",
                "Annual overlap using 2021",
                "Annual overlap using 2020"),
       fill=c("red","green","blue"), bty = "n")

# compute level-shift factors:
lsf(x=x.old, x.new=x.new, t=t, t.overlap=c("2020","2021"))

# level-shift factors can be applied to already chain-linked index series
# to obtain linked series using another overlap period:
x.new.chained <- link(x=x.old, x.new=x.new, t=t, t.overlap="2021-12")

# level-shift adjustment:
x.new.adj <- ifelse(test=t>as.Date("2021-12-01"),
                yes=x.new.chained*lsf(x=x.old, x.new=x.new, t=t, t.overlap="2020"),
                no=x.new.chained)

# compare:
all.equal(x.new.adj, link(x=x.old, x.new=x.new, t=t, t.overlap="2020"))
```

`rates` *Change rates and contributions*

Description

Function `rates()` computes monthly, annual and annual average rates of change for an index series. Function `contrib()` computes the contributions of a subcomponent to the annual change rate of the overall index.

Usage

```
rates(x, t=NULL, type="monthly")

contrib(x, w, t, x.all, w.all, method="ribe")
```

Arguments

`x`, `x.all`	numeric vector of index values.
`w`, `w.all`	numeric vector of weights of the subcomponent (`w`) and the overall index (`w.all`).
`t`	date vector.
`type`	character specifying the type of change rate. Allowed values are `monthly` for monthly change rates, `annual` for annual change rates, and `annual-average` for annual average change rates.
`method`	character specifying the method used for the calculations. Allowed values are `ribe` and `kirchner`.

Value

For `rates()`, a numeric vector of the same length as `x` if `type='monthly'` or `type='annual'`. If `type='annual-average'`, same length as years available.

For `contrib()`, a numeric vector of the same length as `x`.

Author(s)

Sebastian Weinand

References

European Commission, Eurostat, *Harmonised Index of Consumer Prices (HICP) - Methodological Manual - 2024 edition*, Publications Office of the European Union, 2024, https://data.europa.eu/doi/10.2785/055028.

Examples

```
### EXAMPLE 1

P <- rnorm(n=25,mean=100,sd=5)
t <- seq.Date(from=as.Date("2021-01-01"), by="1 month", length.out=length(P))

rates(x=P, type="monthly")
rates(x=P, type="annual")
rates(x=P, type="annual-average")
rates(x=P, t=t, type="annual-average")

### EXAMPLE 2: Contributions using published HICP data

# set cores for testing on CRAN:
library(restatapi)
options(restatapi_cores=1)
library(data.table)

# import monthly price indices:
prc <- hicp.dataimport(id="prc_hicp_midx", filter=list(unit="I15", geo="EA"))
prc[, "time":=as.Date(paste0(time, "-01"))]
prc[, "year":=as.integer(format(time, "%Y"))]
setnames(x=prc, old="values", new="index")

# import item weights:
inw <- hicp.dataimport(id="prc_hicp_inw", filter=list(geo="EA"))
inw[, "time":=as.integer(time)]
setnames(x=inw, old=c("time","values"), new=c("year","weight"))

# merge price indices and item weights:
hicp.data <- merge(x=prc, y=inw, by=c("geo","coicop","year"), all.x=TRUE)

# add all-items hicp:
hicp.data <- merge(x=hicp.data,
                   y=hicp.data[coicop=="CP00", list(geo,time,index,weight)],
                   by=c("geo","time"), all.x=TRUE, suffixes=c("","_all"))

# ribe decomposition:
hicp.data[, "ribe" := contrib(x=index, w=weight, t=time,
                              x.all=index_all, w.all=weight_all), by="coicop"]

# annual change rates over time:
plot(rates(x=index, t=time, type="annual")~time,
     data=hicp.data[coicop=="CP00",],
     type="l", ylim=c(-2,12))

# add contribution of energy:
lines(ribe~time, data=hicp.data[coicop=="NRG"], col="red")

# compare to published contributions:
hicp.ctrb <- hicp.dataimport(id="prc_hicp_ctrb")
hicp.ctrb[, "time":=as.Date(paste0(time, "-01"))]
```

```
dt.comp <- merge(x=hicp.ctrb,
                 y=hicp.data[, list(coicop, time, ribe)],
                 by=c("coicop","time"),
                 all=TRUE)
head(dt.comp[!is.na(values) & abs(values-ribe)>0.1, ]) # should be empty
```

`spec.aggs` *Special aggregates*

Description

This dataset contains the special aggregates and their composition of COICOP codes valid since 2017.

Usage

```
# special aggregates:
spec.aggs
```

Format

A data.table with the following variables.

- `code`: the special aggregate code
- `name_[en|fr|de]`: the special aggregate description in English, French, and German
- `composition`: a list of the COICOP product codes forming the special aggregate

Author(s)

Sebastian Weinand

Examples

```
# subset to services:
spec.aggs[code=="SERV", composition[[1]]]
```

Index