Package 'piar'

April 30, 2022

Title Price Index Aggregation

Version 0.4.0

Description

Most price indexes are made with a two-step procedure, where period-over-period elemental indexes are first calculated for a collection of elemental aggregates at each point in time, and then aggregated according to a price index aggregation structure. These indexes can then be chained together to form a time series that gives the evolution of prices with respect to a fixed base period. This package contains a collections of functions that revolve around this work flow, making it easy to build standard price indexes, and implement the methods described by Balk (2008, ISBN:978-1-107-40496-0), von der Lippe (2001, ISBN:3-8246-0638-0), and the CPI manual (2020, ISBN:978-1-51354-298-0) for bilateral price indexes.

Depends R (>= 3.5)

Imports stats, utils, gpindex (>= 0.4.2)

Suggests rmarkdown, knitr, sps

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/marberts/piar

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Author Steve Martin [aut, cre, cph] (<https://orcid.org/0000-0003-2544-9480>)

Maintainer Steve Martin <stevemartin041@gmail.com>

Repository CRAN

Date/Publication 2022-04-30 18:30:02 UTC

R topics documented:

piar-package 2
aggregation_structure 3
chain 6

contrib 8
impute_prices 9
price_data 10
price_indexes 11
price_relative 19

Index **20**

`piar-package` *Price Index Aggregation*

Description

Most price indexes are made with a two-step procedure, where period-over-period *elemental indexes* are first calculated for a collection of *elemental aggregates* at each point in time, and then aggregated according to a *price index aggregation structure*. These indexes can then be chained together to form a time series that gives the evolution of prices with respect to a fixed base period. This package contains a collections of functions that revolve around this work flow, making it easy to build standard price indexes, and implement the methods described by Balk (2008), von der Lippe (2001), and the CPI manual (2020) for bilateral price indexes.

Usage

The vignette *Making price indexes* gives several extended examples of how to use the functions in this package to make different types of price indexes. Run `vignette("making_price_indexes", "piar")` to view it. But the basic work flow is fairly simple.

The starting point is to make period-over-period elemental price indexes with the `elemental_index()` function and an aggregation structure with the `aggregation_structure()` function. The `aggregate()` method can then be used to aggregate the elemental indexes according to the aggregation structure. There are a variety of methods to work with these index objects, such as chaining them over time.

The two-step workflow is described in chapter 8 of the CPI manual (2020) and chapter 5 of Balk (2008). A practical overview is given by Chiru et al. (2015) for the Canadian CPI, and a detailed discussion of chaining indexes is given by von der Lippe (2001).

Note

This package is designed to work with *bilateral* price indexes. The **IndexNumR** package on the CRAN has support for making *multilateral* price indexes.

Author(s)

Maintainer: Steve Martin <stevemartin041@gmail.com>

References

Balk, B. M. (2008). *Price and Quantity Index Numbers*. Cambridge University Press.

Chiru, R., Huang, N., Lequain, M. Smith, P., and Wright, A. (2015). *The Canadian Consumer Price Index Reference Paper*, Statistics Canada catalogue 62-553-X. Statistics Canada.

ILO, IMF, OECD, Eurostat, UN, and World Bank. (2020). *Consumer Price Index Manual: Theory and Practice*. International Monetary Fund.

von der Lippe, P. (2001). *Chain Indices: A Study in Price Index Theory*, Spectrum of Federal Statistics vol. 16. Federal Statistical Office, Wiesbaden.

See Also

https://github.com/marberts/piar

---

aggregation_structure *Aggregation structure*

Description

Create a price index aggregation structure from a hierarchical classification and aggregation weights.

Usage

```
aggregation_structure(x, w)

## S3 method for class 'pias'
weights(object, ea_only = FALSE, na.rm = FALSE, ...)

## S3 method for class 'pias'
update(object, index, period = end(index), ...)

## S3 method for class 'pias'
as.matrix(x, ...)

## S3 method for class 'pias'
as.data.frame(x, ..., stringsAsFactors = FALSE)

expand_classification(class, width)
```

Arguments

x	A list of character vectors (or vectors to be coerced to character) that give the codes/labels for the aggregation weights for each level of the classification, ordered so that moving down the list goes down the hierarchy. The last vector gives the elemental aggregates, which should have no duplicates. All vectors should be the same length, without NAs, and there should be no duplicates across different levels of x. This is usually a collection of columns in a data frame with a row

	for each aggregation weight; `expand_classification()` can be used when the classification is represented as a string.
`w`	A numeric vector of aggregation weights for the elemental aggregates (i.e., the last vector in `x`). The default is to give each elemental aggregate the same weight.
`object`	A price index aggregation structure, as made by `aggregation_structure()`.
`ea_only`	Should weights be returned for only the elemental aggregates? The default gives the weights for the entire aggregation structure.
`na.rm`	Should missing values be removed from `w` when aggregating the weights (i.e., when `ea_only = FALSE`)? By default, missing values are not removed.
`index`	An aggregate price index, as made by `aggregate()`.
`period`	The time period used to price update the weights. The default uses the last period in `object`.
`class`	A character vector, or something that can be coerced into one, of codes/labels for a specific level in a classification (e.g., 5-digit COICOP, 5-digit NAICS, 4-digit SIC).
`width`	A numeric vector that gives the width of each digit in `x`. This cannot contain NAs. The default assumes each digit has a width of 1, as in the NAICS, NAPCS, and SIC classifications.
`stringsAsFactors`	See `as.data.frame`.
`...`	Further arguments passed to or used by methods.

Value

`aggregation_structure()` returns a price index aggregation structure. This is an object of class `pias`, which has the following components.

`child`	A nested list that gives the positions of the immediate children for each node in each level of the aggregation structure above the terminal nodes.
`parent`	A list that gives the position of the immediate parent for each node of the aggregation structure below the initial nodes.
`levels`	A character vector that gives the levels of `x`.
`eas`	A character vector that gives the subset of `levels` that are elemental aggregates.
`weights`	A named vector giving the weight for each elemental aggregate.
`height`	The length of `x`.

`weights()` returns a list with a named vector of weights for each level in the aggregation structure, unless `ea_only = TRUE`; in this case the return value is just a named vector.

`update()` returns a copy of `object` with price-updated weights using the index values in `index`.

`as.matrix()` represents an aggregation structure as a matrix, such that multiplying with a (column) vector of elemental indexes gives the aggregated index.

`as.data.frame()` takes an aggregation structure and returns a data frame that could have generated it, with columns `level1`, `level2`, ..., `ea`, and `weight`.

`expand_classification()` returns a list with a entry for each level in `x` giving the "digits" that represent each level in the hierarchy.

Warning

The `aggregation_structure()` function does its best to check its arguments, but there should be no expectation that the result of `aggregation_structure()` will make any sense if `x` does not represent a nested hierarchy.

See Also

`aggregate` to aggregate price indexes made with `elemental_index()`.

Examples

```
# A simple example
#            1
#      |-----+-----|
#      11          12
#  |---+---|       |
#  111     112     121

x1 <- c("1",   "1",   "1")
x2 <- c("11",  "11",  "12")
x3 <- c("111", "112", "121")

aggregation_structure(list(x1, x2, x3))

# The aggregation structure can also be made by expanding 'x3'

expand_classification(x3)

all.equal(aggregation_structure(list(x1, x2, x3)),
          aggregation_structure(expand_classification(x3)))

# Unequal weights

aggregation_structure(list(x1, x2, x3), 1:3)

# Extract the weights

weights(aggregation_structure(list(x1, x2, x3)))

# Expanding more complex classifications
# ... if last 'digit' is either TA or TS

expand_classification(c("111TA", "112TA", "121TS"), width = c(1, 1, 1, 2))

# ... if first 'digit' is either 11 or 12

expand_classification(c("111", "112", "121"), width = c(2, 1))

# ...if there are delimiters in the classification (like COICOP)

expand_classification(c("01.1.1", "01.1.2", "01.2.1"), width = c(2, 2, 2))
```

chain *Chain and rebase a price index*

Description

Chain a period-over-period index by taking the cumulative product of its values to turn it into a fixed-base (direct) index. Unchain a fixed-base index by dividing its values for successive periods to get a period-over-period index. Rebase a fixed-base index by dividing its values by the value of the index in the new base period.

Usage

```
chain(x, ...)

## Default S3 method:
chain(x, ...)

## S3 method for class 'ind'
chain(x, link = rep(1, nlevels(x)), ...)

unchain(x, ...)

## Default S3 method:
unchain(x, ...)

## S3 method for class 'ind'
unchain(x, ...)

rebase(x, ...)

## Default S3 method:
rebase(x, ...)

## S3 method for class 'ind'
rebase(x, base = rep(1, nlevels(x)), ...)

is_chainable_index(x)

is_chain_index(x) # deprecated
```

Arguments

x	A price index, as made by, e.g., elemental_index().
link	A numeric vector, or something that can coerced into one, of link values for each level in x. The default is a vector of 1s so that no linking is done.

base	A numeric vector, or something that can coerced into one, of base-period index values for each level in x. The default is a vector of 1s so that the base period remains the same.
...	Further arguments passed to or used by methods.

Details

The default methods attempts to coerce x into an index with as_index() prior to chaining/unchaining/rebasing.

Chaining an index takes the cumulative product of the index values for each level; this is roughly the same as t(apply(as.matrix(x), 1, cumprod)) * link. Unchaining does the opposite, so these are inverse operations. Note that unchaining a period-over-period index (i.e., when is_chainable_index(x) == TRUE) does nothing, as does chaining a fixed-base index (i.e., when is_chainable_index(x) == FALSE).

Rebasing a fixed-base index divides the values for each level of this index by the corresponding values for each level in the new base period. It's roughly the same as as.matrix(x) / base. Like unchaining, rebasing a period-over-period index does nothing.

Quote contributions are removed when chaining/unchaining/rebasing an index, as it's not usually possible to update them correctly.

Value

Each of these functions returns an index of the same type as x, except is_chainable_index() which returns TRUE when x is a period-over-period index (so that chaining makes sense).

See Also

elemental_index for making elemental price indexes.

Examples

```
prices <- data.frame(rel = 1:8, period = rep(1:2, each = 4), ea = rep(letters[1:2], 4))

# A simple period-over-period elemental index

(epr <- with(prices, elemental_index(rel, period, ea)))

# Make period 0 the fixed base period

chain(epr)

# Chaining and unchaining reverse each other

all.equal(epr, unchain(chain(epr)))

# Change the base period to period 2 (note the loss of information for period 0)

epr <- chain(epr)
rebase(epr, epr[, 2])
```

contrib *Extract quote contributions*

Description

Extract quote contributions from an index object.

Usage

```
contrib(x, ...)

## S3 method for class 'ind'
contrib(x, level = levels(x), ...)
```

Arguments

x	A price index, as made by, e.g., elemental_index().
level	The level of an index for which quote contributions are desired, defaulting to the first level (usually the top-level for an aggregate index).
...	Further arguments passed to or used by methods.

Value

A matrix of quote contributions with a column for each period and a row for each product in level. Contributions are padded with 0 to fit into a rectangular array when products differ over time.

If x does not have quote contributions then the result is NULL.

See Also

elemental_index for making elemental price indexes.

Examples

```
prices <- data.frame(rel = 1:8, period = rep(1:2, each = 4), ea = rep(letters[1:2], 4))

epr <- with(prices, elemental_index(rel, period, ea, contrib = TRUE))

pias <- aggregation_structure(list(c("top", "top", "top"), c("a", "b", "c")), 1:3)

index <- aggregate(epr, pias, na.rm = TRUE)

# Quote contributions for the top-level index
contrib(index)

# Calculate EA contributions for the chained index
gpindex::arithmetic_contributions(as.matrix(chain(index))[c("a", "b", "c"), 2],
                                  weights(pias, ea_only = TRUE))
```

impute_prices *Impute prices*

Description

Impute missing prices using the carry forward or shadow price method.

Usage

```
carry_forward(x, period, product)

shadow_price(x, period, product, ea, pias, w, r1 = 0, r2 = 1)
```

Arguments

x	A numeric vector of prices.
period	A factor, or something that can be coerced into one, giving the time period associated with each price in x. The ordering of time periods follows of the levels of period, to agree with cut().
product	A factor, or something that can be coerced into one, giving the product associated with each price in x.
ea	A factor, or something that can be coerced into one, giving the elemental aggregate associated with each price in x.
pias	A price index aggregation structure, as made with aggregation_structure(). The default imputes from elemental indexes only (i.e., not recursively).
w	A numeric vector of weights for the prices in x (i.e., quote/product weights). The default is to give each price equal weight.
r1	Order of the price index used to calculate the elemental price indexes: 0 for a geometric index (the default), 1 for an arithmetic index, or -1 for a harmonic index. Other values are possible; see generalized_mean() for details.
r2	Order of the price index used to aggregate the elemental price indexes: 0 for a geometric index, 1 for an arithmetic index (the default), or -1 for a harmonic index. Other values are possible; see generalized_mean() for details.

Details

The carry forward method replaces a missing price for a product by the price for the same product in the previous period. It tends to push an index value towards 1, and is usually avoided; see paragraph 6.61 in the CPI manual (2020).

The shadow price method recursively imputes a missing price by the value of the price for the same product in the previous period multiplied by the value of the period-over-period elemental index for the elemental aggregate to which that product belongs. This requires computing and aggregating an index (according to pias, unless pias is not supplied) for each period, and so these imputations can take a while. The index values used to do the imputations are not returned because the index needs to be recalculated to get correct quote contributions.

Shadow price imputation is referred to as self-correcting overall mean imputation in chapter 6 of the CPI manual (2020). It is identical to simply excluding missing price relatives in the index calculation, except in the period that a missing product returns. For this reason care is needed when using this method. It is sensitive to the assumption that a product does not change over time, and in some cases it is safer to simply omit the missing price relatives instead of imputing the missing prices.

Value

A copy of `x` with missing values replaced (where possible).

References

ILO, IMF, OECD, Eurostat, UN, and World Bank. (2020). *Consumer Price Index Manual: Theory and Practice*. International Monetary Fund.

See Also

`price_relative` for making price relatives for the same products over time.

`elemental_index` for making elemental price indexes.

Examples

```
prices <- data.frame(price = c(1:7, NA), period = rep(1:2, each = 4),
                     product = 1:4, ea = rep(letters[1:2], 4))

with(prices, carry_forward(price, period, product))

with(prices, shadow_price(price, period, product, ea))
```

---

price_data *Price data*

Description

Sample price and weight data for both a match sample and fixed sample type index.

Usage

```
data("ms_prices")

data("ms_weights")

data("fs_prices")

data("fs_weights")
```

price_indexes *Price indexes*

Description

Compute period-over-period or fixed-base (direct) elemental price indexes (with optional quote contributions), or coerce pre-computed index values into an index object. There are methods to aggregate these indexes with a price index aggregation structure or average them over subperiods, combine indexes together, extract useful information, and coerce into a tabular form.

Usage

```
# Elemental indexes
elemental_index(rel, ...)

## Default S3 method:
elemental_index(rel, ...)

## S3 method for class 'numeric'
elemental_index(rel, period = gl(1, length(rel)),
                ea = gl(1, length(rel)), w, contrib = FALSE,
                chainable = TRUE, na.rm = FALSE, r = 0, ...)

# Aggregate
## S3 method for class 'ind'
aggregate(x, pias, na.rm = FALSE, r = 1, ...)

## S3 method for class 'agg_ind'
vcov(object, repweights, mse = TRUE, ...)

## S3 method for class 'ind'
mean(x, w, window = 3, na.rm = FALSE, r = 1, ...)

# Combine
## S3 method for class 'ind'
merge(x, y, ...)

## S3 method for class 'ind'
stack(x, y, ...)

## S3 method for class 'ind'
unstack(x, ...)

# Extract
## S3 method for class 'ind'
x[i, j]
```

```
## S3 replacement method for class 'ind'
x[i, j] <- value

## S3 method for class 'ind'
levels(x)

## S3 method for class 'ind'
time(x, ...)

## S3 method for class 'ind'
start(x, ...)

## S3 method for class 'ind'
end(x, ...)

## S3 method for class 'ind'
head(x, n = 6, ...)

## S3 method for class 'ind'
tail(x, n = 6, ...)

# Coerce
## S3 method for class 'ind'
as.matrix(x, ...)

## S3 method for class 'ind'
as.data.frame(x, ..., stringsAsFactors = FALSE)

as_index(x, ...)

## Default S3 method:
as_index(x, ...)

## S3 method for class 'matrix'
as_index(x, chainable = TRUE, ...)

## S3 method for class 'data.frame'
as_index(x, cols = 1:3, chainable = TRUE, ...)

is_index(x)

is_aggregate_index(x)
```

Arguments

`rel`	Period-over-period or fixed-base price relatives. Currently there is only a method for numeric vectors; these can be made with `price_relative()`.
`period`	A factor, or something that can be coerced into one, giving the time period

	associated with each price relative in `rel`. The ordering of time periods follows of the levels of `period`, to agree with `cut()`. The default assumes that all price relatives belong to one time period.
`ea`	A factor, or something that can be coerced into one, giving the elemental aggregate associated with each price relative in `rel`. The default assumes that all price relatives belong to one elemental aggregate.
`w`	A numeric vector of weights for the price relatives in `rel`/index values in `x`. The default is equal weights.
`contrib`	Should quote contributions be calculated? The default does not calculate contributions.
`chainable`	Are the price relatives in `rel` period-over-period relatives for a chained calculation (the default)? This should be `FALSE` when `rel` are fixed-base relatives.
`na.rm`	Should missing values be removed? By default, missing values are not removed. Setting `na.rm = TRUE` is equivalent to overall mean imputation.
`r`	Order of the generalized mean to aggregate price relatives/index values. 0 for a geometric index (the default for making elemental indexes), 1 for an arithmetic index (the default for aggregating elemental indexes and averaging indexes over subperiods), or -1 for a harmonic index (usually for a Paasche index). Other values are possible; see `generalized_mean()` for details.
`x, y`	A price index, usually made by `elemental_index()` or `aggregate()`.
`pias`	A price index aggregation structure. This can be made with `aggregation_structure()`.
`object`	An aggregate price index, as made by `aggregate()`.
`repweights`	A matrix, or something that can be coerced into one, of bootstrap replicate weights with a row for each elemental aggregate and a column for each replicate.
`mse`	Should covariances be centered off the value of the index in `object` (the default), or the mean of the replicates?
`window`	The size of the window used to average index values across subperiods. The default (3) turns a monthly index into into a quarterly one.
`i, j, value`	See Extract, with `value` being a numeric vector (or something that can coerced into one.)
`n`	See `head`.
`stringsAsFactors`	See `as.data.frame`.
`cols`	A vector giving the positions/names of the period, level, and value columns in `x`. The default assumes that the first column contains time periods, the second contains levels, and the third contains index values.
`...`	Further arguments passed to or used by methods.

Details

Elemental indexes: When supplied with a numeric vector, `elemental_index()` is a simple wrapper that applies `generalized_mean(r)` and `contributions(r)` (if `contrib = TRUE`) to `rel` and `w` grouped by `ea` and `period`. That is, for every combination of elemental aggregate and time period, `elemental_index()` calculates an index based on a generalized mean of order `r` and,

optionally, quote contributions (using names for `rel` as product names). The default (`r = 0` and no weights) makes Jevons elemental indexes. See chapter 8 (pp. 175–190) of the CPI manual (2020) for more detail about making elemental indexes, and chapter 5 of Balk (2008).

The default method simply coerces `rel` to a numeric prior to calling the method above.

The interpretation of the index depends on how the price relatives in `rel` are made. If these are period-over-period relatives, then the result is a collection of period-over-period elemental indexes; if these are fixed-base relatives, then the result is a collection of fixed-base (direct) elemental indexes. For the latter, `chainable` should be set to `FALSE` so that no subsequent methods assume that a chained calculation should be used.

By default, missing price relatives in `rel` will propagate throughout the index calculation. Ignoring missing values with `na.rm = TRUE` is the same as parental (or overall mean) imputation, and needs to be explicitly set in the call to `elemental_index()`. Explicit imputation of missing relatives, and especially imputation of missing prices, should be done prior to calling `elemental_index()`.

Indexes based on nested generalized means, like the Fisher index (and superlative quadratic mean indexes more generally), can be calculated by supplying the appropriate weights with `nested_transmute()`; see the example below. It is important to note that there are several ways to make these weights, and this affects how quote contributions are calculated.

Aggregate: The `aggregate()` method aggregates elemental indexes by looping over each time period in `x` and

1. aggregates the elemental indexes with `generalized_mean(r)` for each level of `pias`;
2. aggregates quote contributions for each level of `pias` (if there are any);
3. price updates the weights in `pias` with `factor_weights(r)` (only for period-over-period elemental indexes, i.e., `is_chainable_index(x) == TRUE`).

The result is a collection of aggregated period-over-period indexes that can be chained together to get a fixed-base index when `x` are period-over-period elemental indexes. Otherwise, when `x` are fixed-base elemental indexes, the result is a collection of aggregated fixed-base (direct) indexes.

By default, missing elemental indexes will propagate when aggregating the index. Missing elemental indexes can be due to both missingness of these values in `x`, and the presence of elemental aggregates in `pias` that are not part of `x`. Setting `na.rm = TRUE` ignores missing values, and is equivalent to parental (or overall mean) imputation. As an aggregated price index generally cannot have missing values (for otherwise it can't be chained over time), any missing values for a level of `pias` are removed and recursively replaced by the value of its immediate parent.

In most cases aggregation is done with an arithmetic mean (the default), and this is detailed in chapter 8 (pp. 190–198) of the CPI manual (2020). Aggregating with a non-arithmetic mean follows the same steps, except that the elemental indexes are aggregated with a mean of a different order (e.g., harmonic for a Paasche index), and the method for price updating the weights is slightly different.

Aggregating quote contributions uses the method in chapter 9 of the CPI manual (equations 9.26 and 9.28) when aggregating with an arithmetic mean. With a non-arithmetic mean, arithmetic weights are constructed using `transmute_weights(r, 1)` in order to apply this method.

There may not be contributions for all prices relatives in an elemental aggregate if the elemental indexes are built from several sources (as with `merge()`). In this case the contribution for a price relative in the aggregated index will be correct, but the sum of all contributions will not equal the change in the value of the index. This can also happen when aggregating an already aggregated index in which missing index values have been imputed (i.e., when `na.rm = TRUE`).

The vcov() method is a simple wrapper to calculate the variance matrix for an aggregated index when bootstrap replicate weights are available for the elemental aggregates. This approach is usually applicable when elemental aggregates are sampled, and provides an estimator of the sampling variance of the price index. It ignores any sampling variance from the elemental indexes (which often use judgmental sampling), and ultimately depends on the method of generating replicate weights. It returns a matrix of variances with a row for each upper-level index and a column for each time period. (Chapters 3 and 4 of Selvanathan and Rao (1994), especially section 4.7, provide analytic variance estimators for some common price indexes that are applicable with simple random sampling.) Note that any missing elemental indexes need to be explicitly imputed prior to using this method, otherwise they will propagate throughout the variance calculation.

Indexes can be aggregated over subperiods by taking the (usually arithmetic) mean of index values for each level over consecutive windows of subperiods. The mean() method constructs a set of windows of length window, starting in the first period of the index, and takes the unweighted mean of each index value in these windows for each level of the index. The last window is discarded if it is incomplete, so that index values are always averaged over window periods. The names for the first time period in each window form the new names for the aggregated time periods. Note that quote contributions are discarded when aggregating over subperiods.

An optional vector of weights can be specified when aggregating index values over subperiods, which is often useful when aggregating a Paasche index; see section 4.3 of Balk (2008) for details. It is usually easiest to specify these weights as a matrix with a row for each index value in x and a column for each time period.

Combine: The merge() method combines two index objects with common time periods, merging together the index values and quote contributions for each time period in x and y. This is useful for building up an index when different elemental aggregates come from different sources of data, or use different index-number formulas.

The stack() method combines two index objects with common levels, stacking index values and quote contributions for each level in y after those in x. The unstack() method breaks up x into a list of indexes, one for each period in x. These methods can be used in a map-reduce to make an index with multiple aggregation structures (like a Paasche index).

It is not generally possible to merge aggregated indexes, as this would change the aggregation structure, so merging always returns an index of class ind. If at least one of x or y is an aggregate index then the result of stacking these indexes is also an aggregate index; otherwise, it is the same class as x.

Extract: The extraction method treats x as a matrix of index values with (named) rows for each level and columns for each period in x. Unlike a matrix, dimensions are never dropped as indexing x always returns an index object. This means that indexing with a matrix is not possible, and only a submatrix can be extracted. As x is not an atomic vector, indexing with a single index like x[1] is taken to be the same as x[1,]. Note that indexing an aggregated index cannot generally preserve the aggregation structure if any levels are removed, and in this case the resulting index is *not* an aggregated index.

The replacement method similarly treats x as a matrix, and behaves the same as replacing values in a matrix (except that value is coerced to numeric). Note that replacing the values of an index will remove the corresponding quote contributions (if any).

The levels() method extracts the levels of an index, and the time() method extracts the time periods of the index. The start() and end() methods extract the first and last time period.

The head() and tail() methods act as if x is a matrix of index values, and by default extract the time series for the first/last six levels of x.

The summary() method summarizes x as a matrix of index values (i.e., the five-number summary for each period). If there are quote contributions, then these are also summarized as a matrix.

Coerce: The as.matrix() method turns an index into a matrix with a row for each level and a column for each period. The as.data.frame() method turns an index into a data frame with three columns: period, level, and value.

as_index() is useful to form an elemental index from pre-computed values. Numeric matrices are coerced into an elemental index object by treating each column as a separate time period, and each row as an elemental aggregate. Column names are used to denote time periods, and row names are used to denote elemental aggregates (so they must be unique). This essentially reverses calling as.matrix() on an index object. If a dimension is unnamed, then it is given a sequential label from 1 to the size of that dimension. The default method coerces x to a matrix prior to using the matrix method.

The data frame method for as_index() is best understood as reversing the effect of as.data.frame() on an index object. It constructs a matrix by taking the unique values of x[[cols[1]]] as columns and the unique values of x[[cols[2]]] as rows (in the order they appear). It then populates this matrix with the corresponding values in x[[cols[3]]], and uses the matrix method for as_index(). Note that the resulting index is therefore sensitive to the ordering of x.

Value

Most of these functions return index objects of class ind. These often behaves like a matrix with a row for each level of the index and a column for each time period, and have the following components.

index	A named list with an entry for each period that gives a named vector of index values for each level in ea.
contrib	A named list with an entry for each period, which itself contains a named list with an entry for each level in ea with a named vector that gives the additive contribution for each price relative. If contrib = FALSE, then each of these vectors is of length 0.
levels	The levels for ea.
time	The levels for period.
has_contrib	The value of contrib.
chainable	The value of chainable, usually TRUE.

aggregate() returns an aggregate index. This is an object of class agg_ind, inheriting from class ind, which has the following components.

index	A named list with an entry for each period in x that gives a named vector of index values for each level in pias.
contrib	A named list with an entry for each period, which itself contains a list with an entry for each level in pias with a named vector that gives the additive contribution for each price relative.
levels	The levels for pias.

time	The levels for period from x.
has_contrib	The value of has_contrib from x.
chainable	The value for chainable from x, usually TRUE.
r	The value for r, usually 1.
pias	A list containing the child, parent, eas, and height components of pias.

Source

The vcov() method was influenced by a SAS routine by Justin Francis that was first ported to R by Ambuj Dewan, and subsequently rewritten by Steve Martin.

References

Balk, B. M. (2008). *Price and Quantity Index Numbers*. Cambridge University Press.

ILO, IMF, OECD, Eurostat, UN, and World Bank. (2020). *Consumer Price Index Manual: Theory and Practice*. International Monetary Fund.

Selvanathan, E. A., and Rao, D. S. P. (1994). *Index Numbers: A Stochastic Approach*. MacMillan.

See Also

price_relative for making price relatives for the same products over time, and carry_forward and shadow_price for imputations for missing prices.

aggregation_structure for making a price index aggregation structure.

chain for chaining period-over-period indexes, and rebase for rebasing an index.

contrib for extracting quote contributions.

The sps_repweights() function in the **sps** package to generate replicates weights when elemental aggregates are sampled using sequential Poisson sampling.

Examples

```
prices <- data.frame(rel = 1:8, period = rep(1:2, each = 4), ea = rep(letters[1:2], 4))

# A two-level aggregation structure

pias <- aggregation_structure(list(c("top", "top", "top"), c("a", "b", "c")), 1:3)

# Calculate Jevons elemental indexes

(epr <- with(prices, elemental_index(rel, period, ea)))

# Same as using lm() or tapply()

exp(coef(lm(log(rel) ~ ea:factor(period) - 1, prices)))

with(prices, t(tapply(rel, list(period, ea), gpindex::geometric_mean, na.rm = TRUE)))

# Extract the indexes like a matrix
```

```
epr["a", ]

epr[, 2]

epr[1, ] <- 1 # can be useful for doing specific imputations

# Aggregate (note the imputation for elemental index 'c')

(index <- aggregate(epr, pias, na.rm = TRUE))

# Aggregation can equivalently be done as matrix multiplication

as.matrix(pias) %*% as.matrix(chain(index[letters[1:3]]))

# Merge two indexes prior to aggregation

prices2 <- data.frame(rel = 1:8, period = rep(1:2, each = 4), ea = rep(letters[3:4], 4))
epr2 <- with(prices2, elemental_index(rel, period, ea))
aggregate(merge(epr, epr2), pias)

# Stack two indexes prior to aggregation

prices3 <- data.frame(rel = 1:8, period = rep(3:4, each = 4), ea = rep(letters[1:2], 4))
epr3 <- with(prices3, elemental_index(rel, period, ea))
aggregate(stack(epr, epr3), pias)

# Unstack does the reverse

all.equal(c(unstack(epr), unstack(epr3)), unstack(stack(epr, epr3)))

# Extract useful features of the index

head(index, 1)
tail(index, 3)
levels(index)
time(index)
start(index)
end(index)

summary(index)

# Turn the index into a data frame/matrix

as.data.frame(index)
as.matrix(index)

all.equal(as_index(as.data.frame(epr)), epr)
all.equal(as_index(as.matrix(epr)), epr)

# Calculate a CSWD index (same as the Jevons in this example)
# as an arithmetic index by constructing appropriate weights

library(gpindex)
```

```
# A general function to calculate weights to turn the geometric
# mean of the arithmetic and harmonic mean (i.e., Fisher mean)
# into an arithmetic mean

fw <- grouped(nested_transmute(0, c(1, -1), 1))

with(
    prices,
    elemental_index(rel, period, ea, fw(rel, group = interaction(period, ea)), r = 1)
)
```

`price_relative` *Price relative*

Description

Construct period-over-period price relatives from information on prices over time.

Usage

```
price_relative(x, period, product)
```

Arguments

`x`	A numeric vector of prices.
`period`	A factor, or something that can be coerced into one, that gives the corresponding time period for each element in `x`. The ordering of time periods follows the levels of `period` to agree with `cut()`.
`product`	A factor, or something that can be coerced into one, that gives the corresponding product identifier for each element in `x`.

Value

A numeric vector of price relatives, with `product` as names.

See Also

`back_period` to get only the back price.

`base_period` for making fixed-base price relatives.

`carry_forward` and `shadow_price` for imputations for missing prices.

Examples

```
price_relative(1:6, rep(1:2, each = 3), rep(letters[1:3], 2))
```

Index

[.ind (price_indexes), 11
[<-.ind (price_indexes), 11

aggregate, *5*
aggregate(), *2*, *4*
aggregate.ind (price_indexes), 11
aggregation_structure, 3, *17*
aggregation_structure(), *2*, *9*, *13*
as.data.frame, *4*, *13*
as.data.frame.ind (price_indexes), 11
as.data.frame.pias
        (aggregation_structure), 3
as.matrix.ind (price_indexes), 11
as.matrix.pias (aggregation_structure),
        3
as_index (price_indexes), 11
as_index(), *7*

back_period, *19*
base_period, *19*

carry_forward, *17*, *19*
carry_forward (impute_prices), 9
chain, 6, *17*
contrib, 8, *17*
contributions(r), *13*
cut(), *9*, *13*, *19*

elemental_index, *7, 8*, *10*
elemental_index (price_indexes), 11
elemental_index(), *2*, *5, 6*, *8*
end.ind (price_indexes), 11
expand_classification
        (aggregation_structure), 3
Extract, *13*

factor_weights(r), *14*
fs_prices (price_data), 10
fs_weights (price_data), 10

generalized_mean(), *9*, *13*
generalized_mean(r), *13, 14*

head, *13*
head.ind (price_indexes), 11

impute_prices, 9
is_aggregate_index (price_indexes), 11
is_chain_index (chain), 6
is_chainable_index (chain), 6
is_index (price_indexes), 11

levels.ind (price_indexes), 11

mean.ind (price_indexes), 11
merge.ind (price_indexes), 11
ms_prices (price_data), 10
ms_weights (price_data), 10

nested_transmute(), *14*

piar (piar-package), 2
piar-package, 2
price_data, 10
price_indexes, 11
price_relative, *10*, *17*, 19
price_relative(), *12*

rebase, *17*
rebase (chain), 6

shadow_price, *17*, *19*
shadow_price (impute_prices), 9
stack.ind (price_indexes), 11
start.ind (price_indexes), 11
summary.ind (price_indexes), 11

tail.ind (price_indexes), 11
time.ind (price_indexes), 11

unchain (chain), 6
unstack.ind (price_indexes), 11

update.pias *(*aggregation_structure*)*, 3

vcov.agg_ind *(*price_indexes*)*, 11

weights.pias *(*aggregation_structure*)*, 3