Title  Price Index Aggregation
Version  0.6.0
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 (>= 4.0)
Imports  stats, utils, gpindex (>= 0.5.0), Matrix (>= 1.5-0)
Suggests  rmarkdown, knitr, sps, testthat (>= 3.0.0)
License  MIT + file LICENSE
Encoding  UTF-8


BugReports  https://github.com/marberts/piar/issues

LazyData  true
VignetteBuilder  knitr
Config/testthat/edition  3
RoxygenNote  7.2.3

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  2023-11-19 03:10:02 UTC
R topics documented:

aggregate.piar_index .......................................................... 2
aggregation_structure .......................................................... 5
as.data.frame.piar_index ......................................................... 6
as.matrix.piar_aggregation_structure ..................................... 8
as_aggregation_structure ....................................................... 9
as_index ........................................................................... 11
chain .............................................................................. 12
contrib .......................................................................... 15
elemental_index ................................................................. 16
expand_classification .......................................................... 19
head.piar_index ................................................................. 20
impute_prices ................................................................. 21
is_aggregation_structure ....................................................... 23
is_index ................................................................. 24
levels.piar_index .............................................................. 25
mean.piar_index .............................................................. 25
merge.piar_index ............................................................ 27
piar_index ................................................................. 28
price_data ................................................................. 29
price_relative .............................................................. 29
stack.piar_index ............................................................ 30
summary.piar_index .......................................................... 31
time.piar_index .............................................................. 32
update.piar_aggregation_structure ....................................... 33
vcov.aggregate_piar_index .................................................. 34
weights.piar_aggregation_structure ..................................... 36
[.piar_index .............................................................. 37

Index ................................. 40

aggregate.piar_index  Aggregate elemental price indexes

Description

Aggregate elemental price indexes with a price index aggregation structure.

Usage

```r
## S3 method for class 'chainable_piar_index'
aggregate(x, pias, na.rm = FALSE, r = 1, contrib = TRUE, ...)
```

```r
## S3 method for class 'direct_piar_index'
aggregate(x, pias, na.rm = FALSE, r = 1, contrib = TRUE, ...)
```
Arguments

- **x**: A price index, usually made by `elemental_index()`.  
- **pias**: A price index aggregation structure or something that can be coerced into one. This can be made with `aggregation_structure()`.  
- **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 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 `gpindex::generalized_mean()` for details.  
- **contrib**: Aggregate percent-change contributions in `x` (if any)?  
- **...**: Further arguments passed to or used by methods.

Details

The `aggregate()` method loops over each time period in `x` and

1. aggregates the elemental indexes with `gpindex::generalized_mean(r)()` for each level of `pias`;
2. aggregates percent-change contributions for each level of `pias` (if there are any and `contrib = TRUE`);
3. price updates the weights in `pias` with `gpindex::factor_weights(r)()` (only for period-over-period elemental indexes).

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. Note that, because aggregation is done with a generalized mean, the resulting index is consistent-in-aggregation at each point in time.

Aggregating percent-change 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 `gpindex::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).

Value

An aggregate price index that inherits from aggregate_piar_index and the class of x.

Note

For large indexes it can be much faster to turn the aggregation structure into an aggregation matrix with as.matrix(), then aggregate elemental indexes as a matrix operation when there are no missing values—see the examples for details.

References


See Also

Other index methods: [.piar_index(), as.data.frame.piar_index(), chain(), contrib(), head.piar_index(), levels.piar_index(), mean.piar_index(), merge.piar_index(), stack.piar_index(), time.piar_index(), vcov.aggregate_piar_index()]

Examples

```r
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)))

# 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]]))
```
aggregation_structure

Make a price index aggregation structure

Description

Create a price index aggregation structure from a hierarchical classification and aggregation weights that can be used to aggregate elemental indexes.

Usage

aggregation_structure(x, weights = NULL)

Arguments

x
A list of character vectors that give the codes/labels 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.

weights
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.

Value

A price index aggregation structure of class piar_aggregation_structure. This is a list-S3 class with 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.

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().
expand_classification() to make x from a character representation of a hierarchical aggregation structure.
as_aggregation_structure() to coerce tabular data into an aggregation structure.
as.data.frame() and as.matrix() to coerce an aggregation structure into a tabular form.
weights() to get the weights for an aggregation structure.
update() for updating a price index aggregation structure with an aggregated index.

Examples

# A simple aggregation structure
#
# |-----+-----|
# 1    |      |
# 11   12
# 111  112  121
# (1)  (3)  (4)

taggregation_weights <- data.frame(
  level1 = c("1", "1", "1"),
  level2 = c("11", "11", "12"),
  ea = c("111", "112", "121"),
  weight = c(1, 3, 4)
)
prias <- aggregation_structure(
  aggregation_weights[1:3],
  weights = aggregation_weights[[4]]
)

# The aggregation structure can also be made by expanding the # elemental aggregates

all.equal(
  with(
    aggregation_weights,
    aggregation_structure(expand_classification(ea), weight)
  ),
  prias
)

as.data.frame.piar_index

Coerce an index into a tabular form
as.data.frame.piar_index

Description

Turn an index into a data frame or a matrix.

Usage

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

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

Arguments

- `x` A price index, as made by, e.g., `elemental_index()`.
- `...` Further arguments passed to or used by methods.
- `stringsAsFactors` See `as.data.frame()`.

Value

`as.data.frame()` returns a data frame with three columns: period, level, and value.
`as.matrix()` returns a matrix with a row for each level and a column for each period.

See Also

`as_index()` to coerce a matrix/data frame of index values into an index object.

Other index methods: `.piar_index()`, `aggregate.piar_index()`, `chain()`, `contrib()`, `head.piar_index()`, `levels.piar_index()`, `mean.piar_index()`, `merge.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `vcov.aggregate_piar_index()`

Examples

```r
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))

as.data.frame(epr)
as.matrix(epr)
```
as.matrix.piar_aggregation_structure

Coerce a price index aggregation structure into a tabular form

Description

Coerce a price index aggregation structure into an aggregation matrix, or a data frame.

Usage

## S3 method for class 'piar_aggregation_structure'
as.matrix(x, sparse = FALSE, ...)

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

Arguments

x
A price index aggregation structure, as made by aggregation_structure().

sparse
Should the result be a sparse matrix from Matrix? This is faster for large aggregation structures. The default returns an ordinary dense matrix.

...
Further arguments passed to or used by methods.

stringsAsFactors
See as.data.frame().

Value

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.

See Also

as_aggregation_structure() for coercing into an aggregation structure.

Other aggregation structure methods: update.piar_aggregation_structure(), weights.piar_aggregation_structure()

Examples

# A simple aggregation structure
#     |-------
#     1     |
#     11   12
# |--------|   |
# 111   112  121
# (1)   (3) (4)
as_aggregation_structure

aggregation_weights <- data.frame(
  level1 = c("1", "1", "1"),
  level2 = c("11", "11", "12"),
  ea = c("111", "112", "121"),
  weight = c(1, 3, 4)
)

pias <- as_aggregation_structure(aggregation_weights)

as.matrix(pias)

all.equal(as.data.frame(pias), aggregation_weights)

as_aggregation_structure

Coerce to an aggregation structure

Description

Coerce an object into an aggregation structure object.

Usage

as_aggregation_structure(x, ...)

## Default S3 method:
as_aggregation_structure(x, weights = NULL, ...)

## S3 method for class 'data.frame'
as_aggregation_structure(x, ...)

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

## S3 method for class 'aggregate_piar_index'
as_aggregation_structure(x, weights = NULL, ...)

Arguments

x An object to coerce into an aggregation structure.

... Further arguments passed to or used by methods.

weights A numeric vector of aggregation weights for the elemental aggregates. The
default is to give each elemental aggregate the same weight.
Details

The default method attempts to coerce \( x \) into a list prior to calling \texttt{aggregation_structure()}. The data frame and matrix methods treat \( x \) as a table with a row for each elemental aggregate, a column of labels for each level in the aggregation structure, and a column of weights for the elemental aggregates.

The method for aggregate indexes reconstructs the aggregation structure used to generate the index (with optional weights).

Value

A price index aggregation structure that inherits from \texttt{piar_aggregation_structure}.

See Also

\texttt{as.matrix()} and \texttt{as.data.frame()} for coercing an aggregation structure into a tabular form.

Examples

```r
# A simple aggregation structure
# 1
# |-----+-----|
# 11 12
# |---+---| |
# 11 11 12 |
# (1) (3) (4)

aggregation_weights <- data.frame(
  level1 = c("1", "1", "1"),
  level2 = c("11", "1"", "12"),
  ea = c("111", "112", "121"),
  weight = c(1, 3, 4)
)

pias <- aggregation_structure(aggregation_weights[1:3],
  weights = aggregation_weights[4])

all.equal(
  pias,
  as_aggregation_structure(aggregation_weights)
)

all.equal(
  pias,
  as_aggregation_structure(as.matrix(aggregation_weights))
)
```
Coerce pre-computed index values into an index object.

Usage

\[
\text{as\_index}(x, \ldots)
\]

## Default S3 method:
\text{as\_index}(x, chainable = \text{TRUE, } \ldots)

## S3 method for class 'matrix'
\text{as\_index}(x, chainable = \text{TRUE, } \ldots)

## S3 method for class 'data.frame'
\text{as\_index}(x, cols = 1:3, chainable = \text{TRUE, } \ldots)

## S3 method for class 'chainable\_piar\_index'
\text{as\_index}(x, chainable = \text{TRUE, } \ldots)

## S3 method for class 'direct\_piar\_index'
\text{as\_index}(x, chainable = \text{FALSE, } \ldots)

Arguments

\(x\) An object to coerce into a price index.

\(\ldots\) Further arguments passed to or used by methods.

\(\text{chainable}\) Are the index values in \(x\) period-over-period indexes, suitable for a chained calculation (the default)? This should be \text{FALSE} when \(x\) is a fixed-base (direct) index.

\(\text{cols}\) 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.

Details

Numeric matrices are coerced into an 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 \text{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 levels of `x[[cols[1]]]` as columns and the levels of `x[[cols[2]]]` as rows (coercing to a factor if necessary). It then populates this matrix with the corresponding values in `x[[cols[3]]]`, and uses the matrix method for `as_index()`.

If `x` is a period-over-period index then it is returned unchanged when `chainable = TRUE` and chained otherwise. Similarly, if `x` is a fixed-base index then it is returned unchanged when `chainable == FALSE` and unchain otherwise.

**Value**

`as_index()` returns a price index that inherits from `piar_index`. If `chainable = TRUE` then this is a period-over-period price index that also inherits from `chainable_piar_index`; otherwise, it is a fixed-base index that inherits from `direct_piar_index`.

**See Also**

`as.matrix()` and `as.data.frame()` for coercing an index into a tabular form.

**Examples**

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

# Calculate period-over-period Jevons elemental indexes
epr <- with(prices, elemental_index(rel, period, ea))
all.equal(as_index(as.data.frame(epr)), epr)
all.equal(as_index(as.matrix(epr)), epr)
```

---

**Description**

Chain and rebase a price index

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 with the value of the index in the new base period.
Usage

chain(x, ...)

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

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

## S3 method for class 'direct_piar_index'
chain(x, ...)

unchain(x, ...)

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

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

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

rebase(x, ...)

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

## S3 method for class 'chainable_piar_index'
rebase(x, ...)

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

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.

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 \( t(\text{apply}(\text{as.matrix}(x), 1, \text{cumprod})) \times \text{link} \). Unchaining does the opposite, so these are inverse operations. Note that unchaining a period-over-period index (i.e., when \( \text{is\_chainable\_index}(x) == \text{TRUE} \)) does nothing, as does chaining a fixed-base index (i.e., when \( \text{is\_chainable\_index}(x) == \text{FALSE} \)).

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 \( \text{as.matrix}(x) / \text{base} \). Like unchaining, rebasing a period-over-period index does nothing.

Percent-change contributions are removed when chaining/unchaining/rebasing an index, as it’s not usually possible to update them correctly.

**Value**

`chain()` and `rebase()` return a fixed-base index that inherits from `direct_piar_index`. `unchain()` returns a period-over-period index that inherits from `chainable_piar_index`.

**See Also**

Other index methods: `.[.piar_index(), aggregate.piar_index(), as.data.frame.piar_index(), contrib(), head.piar_index(), levels.piar_index(), mean.piar_index(), merge.piar_index(), stack.piar_index(), time.piar_index(), vcov.aggregate_piar_index()`

**Examples**

```r
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])
```
Summary

Extract percent-change contributions from a price index.

Usage

contrib(x, ...)

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

Arguments

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

Value

A matrix of percent-change contributions with a column for each period and a row for each product (sorted) for which there are contributions in `level`. Contributions are padded with 0 to fit into a rectangular array when products differ over time.

See Also

Other index methods: `[, piar_index()`, `aggregate.piar_index()`, `as.data.frame.piar_index()`, `chain()`, `head.piar_index()`, `levels.piar_index()`, `mean.piar_index()`, `merge.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `vcov.aggregate_piar_index()`

Examples

```r
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
)```
elemental_index

Make elemental price indexes

Description

Compute period-over-period (chainable) or fixed-base (direct) elemental price indexes, with optional percent-change contributions.

Usage

elemental_index(x, ...)

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

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

)
Arguments

- **x**: Period-over-period or fixed-base price relatives. Currently there is only a method for numeric vectors; these can be made with `price_relative()`.
- **...**: Further arguments passed to or used by methods.
- **period**: A factor, or something that can be coerced into one, giving the time period associated with each price relative in x. The ordering of time periods follows 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 x. The default assumes that all price relatives belong to one elemental aggregate.
- **weights**: A numeric vector of weights for the price relatives in x. The default is equal weights.
- **contrib**: Should percent-change contributions be calculated? The default does not calculate contributions.
- **chainable**: Are the price relatives in x period-over-period relatives for a chained calculation (the default)? This should be `FALSE` when x contains 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. 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 `gpindex::generalized_mean()` for details.

Details

When supplied with a numeric vector, `elemental_index()` is a simple wrapper that applies `gpindex::generalized_mean(r)` and `gpindex::contributions(r)()` (if `contrib = TRUE`) to x and weights 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, percent-change contributions. 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 x to a numeric vector prior to calling the method above.

Names for x are used as product names when calculating percent-change contributions. Product names should be unique within each time period, and, if not, are passed to `make.unique()` with a warning. If x has no names then elements of x are given sequential names within each elemental aggregate.

The interpretation of the index depends on how the price relatives in x are made. If these are period-over-period relatives, then the result is a collection of period-over-period (chainable) 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.
elemental_index

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

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 \( \text{gpindex::nested_transmute()} \); see the example below. It is important to note that there are several ways to make these weights, and this affects how percent-change contributions are calculated.

Value

A price index that inherits from \( \text{piar_index} \). If \( \text{chainable = \text{TRUE}} \) then this is a period-over-period index that also inherits from \( \text{chainable_piar_index} \); otherwise, it is a fixed-based index that inherits from \( \text{direct_piar_index} \).

References


See Also

\( \text{price_relative()} \) for making price relatives for the same products over time, and \( \text{carry_forward()} \) and \( \text{shadow_price()} \) for imputation of missing prices.

\( \text{as_index() } \) to turn pre-computed (elemental) index values into an index object.

\( \text{chain()} \) for chaining period-over-period indexes, and \( \text{rebase()} \) for rebasing an index.

\( \text{aggregate()} \) to aggregate elemental indexes according to an aggregation structure.

\( \text{as.matrix()} \) and \( \text{as.data.frame()} \) for coercing an index into a tabular form.

Examples

```r
library(gpindex)

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

# Calculate Jevons elemental indexes
(epr <- with(prices, elemental_index(rel, period, ea)))

# Same as using \( \text{lm()} \) or \( \text{tapply()} \)
exp(coef(lm(log(rel) ~ ea:factor(period) - 1, prices)))

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

# 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))

# Calculate a CSWD index (same as the Jevons in this example)
# as an arithmetic index by using the appropriate weights
with(
  prices,
  elemental_index(
    rel, period, ea,
    fw(rel, group = interaction(period, ea)),
    r = 1
  )
)

expand_classification  Expand a hierarchical classification

Description

Expand a character representation of a hierarchical classification to make a price index aggregation structure.

Usage

expand_classification(x, width = 1L)

Arguments

x  
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  
An integer vector that gives the width of each digit in x. A single value is recycled to span the longest element in x. This cannot contain NAs. The default assumes each digit has a width of 1, as in the NAICS, NAPCS, and SIC classifications.

Value

A list with a entry for each level in x giving the “digits” that represent each level in the hierarchy.
See Also

aggregation_structure() to make a price-index aggregation structure.

Examples

# A simple classification structure
#  
# |-----+-----|
# 11  12
# |---+---| |
# 111 112 121

expand_classification(c("111", "112", "121"))

# 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 = 2)

head.piar_index

Return the first/last parts of an index

Description

Extract the first/last parts of an index as if it were a matrix.

Usage

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

tail(x, n = 6L, ...)
Arguments

\(x\)  
A price index, as made by, e.g., `elemental_index()`.

\(n\)  
See `head()/tail()`. The default takes the first/last 6 levels of \(x\).

\(...\)  
Further arguments passed to or used by methods.

Value

A price index that inherits from `chainable_piar_index` if \(x\) is a period-over-period index, or `direct_piar_index()` if \(x\) is a fixed-base index.

See Also

Other index methods: `.piar_index()`, `aggregate.piar_index()`, `as.data.frame.piar_index()`, `chain()`, `contrib()`, `levels.piar_index()`, `mean.piar_index()`, `merge.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `vcov.aggregate.piar_index()`

Examples

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

# Calculate Jevons elemental indexes
epr <- with(prices, elemental_index(rel, period, ea))

# Get the first/last time series
head(epr, 1)
tail(epr, 1)
```

---

### impute_prices

*Impute missing prices*

**Description**

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

```r
shadow_price(
  x,
  period,
  product,
  ea,
  pias = NULL,
  weights = NULL,
  r1 = 0,
  r2 = 1
)
```

carry_forward(x, period, product)

carry_backwards(x, period, product)

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 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, or something that can be coerced into one, as made with `aggregation_structure()`. The default imputes from elemental indexes only (i.e., not recursively).
- `weights`: A numeric vector of weights for the prices in `x` (i.e., product weights). The default is to give each price equal weight.
- `r1`: Order of the generalized-mean 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 `gpindex::generalized_mean()` for details.
- `r2`: Order of the generalized-mean 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 `gpindex::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 carry backwards method does the opposite, but this is rarely used in practice.
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 \( \text{pias} \), unless \( \text{pias} \) 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 percent-change 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**


**See Also**

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

**Examples**

```r
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))
```

---

**is_aggregation_structure**

*Test if an object is a price index aggregation structure*

**Description**

Test if an object is a price index aggregation structure.
Usage

\texttt{is\_aggregation\_structure(x)}

Arguments

\textit{x} \hspace{1cm} \text{An object to test.}

Value

Returns TRUE if \textit{x} inherits from \texttt{piar\_aggregation\_structure}

Description

Test if an object is a price index.

Usage

\texttt{is\_index(x)}

\texttt{is\_aggregate\_index(x)}

\texttt{is\_chainable\_index(x)}

\texttt{is\_direct\_index(x)}

Arguments

\textit{x} \hspace{1cm} \text{An object to test.}

Value

\texttt{is\_index()} returns TRUE if \textit{x} inherits from \texttt{piar\_index}.

\texttt{is\_chainable\_index()} returns TRUE if \textit{x} inherits from \texttt{chainable\_piar\_index}.

\texttt{is\_direct\_index()} returns TRUE if \textit{x} inherits from \texttt{direct\_piar\_index}.

\texttt{is\_aggregate\_index()} returns TRUE if \textit{x} inherits from \texttt{aggregate\_piar\_index}. 
levels.piar_index  

Get the levels for a price index

Description

Methods to get and set the levels for a price index.

Usage

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

## S3 replacement method for class 'piar_index'
levels(x) <- value

Arguments

x  A price index, as made by, e.g., elemental_index().
value  A character vector, or something that can be coerced into one, giving the replacement levels for x.

Value

levels() returns a character vector with the levels for a price index.
The replacement method returns a copy of x with the levels in value.

It’s not generally possible to change the levels of an aggregate price index, and in this case replacing the levels does not return an aggregate index.

See Also

Other index methods: [.piar_index(), aggregate.piar_index(), as.data.frame.piar_index(),
chain(), contrib(), head.piar_index(), mean.piar_index(), merge.piar_index(), stack.piar_index(),
time.piar_index(), vcov.aggregate_piar_index()}

mean.piar_index  

Aggregate a price index over subperiods

Description

Aggregate an index over subperiods by taking the (usually arithmetic) mean of index values over consecutive windows of subperiods.

Usage

## S3 method for class 'piar_index'
mean(x, weights = NULL, window = 3L, na.rm = FALSE, r = 1, ...)
mean.piar_index

Arguments

x
A price index, as made by, e.g., `elemental_index()`.

weights
A numeric vector of weights for the index values in x. The default is equal weights. 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.

window
A positive integer giving the size of the window used to average index values across subperiods. The default (3) turns a monthly index into a quarterly one. Non-integers are truncated towards 0.

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 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 `gpindex::generalized_mean()` for details.

... Further arguments passed to or used by methods.

Details

The `mean()` method constructs a set of non-overlapping windows of length `window`, starting in the first period of the index, and takes the 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 percent-change 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.

Value

A price index with the same class as x.

References


See Also

Other index methods: `.piar_index()`, `aggregate_piar_index()`, `as.data.frame.piar_index()`, `chain()`, `contrib()`, `head.piar_index()`, `levels.piar_index()`, `merge.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `vcov.aggregate_piar_index()`

Examples

```r
prices <- data.frame(
  rel = 1:8,
  period = rep(1:2, each = 4),
)```
merge.piar_index

```r
ea = rep(letters[1:2], 4)

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

mean(epr, window = 2)
```

merge.piar_index  Merge price indexes

Description

Combine two price indexes with common time periods, merging together the index values and percent-change contributions for each time period.

This is useful for building up an index when different elemental aggregates come from different sources of data, or use different index-number formulas.

Usage

```r
## S3 method for class 'piar_index'
merge(x, y, ...)
```

Arguments

- `x` A price index, as made by, e.g., `elemental_index()`.  
- `y` A price index, or something that can coerced into one. If `x` is a period-over-period index then `y` is coerce into a chainable index; otherwise, `y` is coerced into a direct index.  
- `...` Further arguments passed to or used by methods.

Value

A price index that inherits from `chainable_piar_index` if `x` is a period-over-period index, or `direct_piar_index` if `x` is a fixed-base index. It is not generally possible to merge aggregated indexes, as this would change the aggregation structure, so merging does not return an aggregated index.

See Also

Other index methods: `[, aggregate.piar_index(), as.data.frame.piar_index(), chain(), contrib(), head.piar_index(), levels.piar_index(), mean.piar_index(), stack.piar_index(), time.piar_index(), vcov.aggregate_piar_index()`
Examples

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

prices2 <- data.frame(
  rel = 1:8,
  period = rep(1:2, each = 4),
  ea = rep(letters[3:4], 4)
)

epr <- with(prices, elemental_index(rel, period, ea))
epr2 <- with(prices2, elemental_index(rel, period, ea))
merge(epr, epr2)
```

---

**piar_index**

**Price index objects**

**Description**

There are several classes to represent price indexes.

- All indexes inherit from the `piar_index` virtual class.
- Period-over-period indexes that can be chained over time inherit from `chainable_piar_index`.
- Fixed-base indexes inherit from `direct_piar_index`.
- Aggregate price indexes that are the result of aggregating elemental indexes with an aggregation structure further inherit from `aggregate_piar_index`.

**Details**

The `piar_index` object is a list-S3 class with the following components:

- **index**: A list with an entry for each period in `time` that gives a vector of index values for each level in `levels`.
- **contrib**: A list with an entry for each period in `time`, which itself contains a list with an entry for each level in `levels` with a named vector that gives the additive contribution for each price relative.
- **levels**: A character vector giving the levels of the index.
- **time**: A character vector giving the time periods for the index.
The `chainable_piar_index` and `direct_piar_index` subclasses have the same structure as the `piar_index` class, but differ in the methods used to manipulate the indexes. The `aggregate_piar_index` class further subclasses either `chainable_piar_index` or `direct_piar_index`, and adds the following components:

- `r` The order of the generalized mean used to aggregated the index (usually 1).
- `pias` A list containing the child, parent, eas, and height components of the aggregation structure used to aggregate the index.

---

### price_data

**Price data**

**Description**

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

---

### price_relative

**Calculate period-over-period price relatives**

**Description**

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

**Usage**

```r
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**

- `gpindex::back_period()` to get only the back price.
- `gpindex::base_period()` for making fixed-base price relatives.
- `carry_forward()` and `shadow_price()` to impute missing prices.
- `gpindex::outliers` for methods to identify outliers with price relatives.
Examples

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

---

**Stack price indexes**

**Description**

`stack()` combines two price indexes with common levels, stacking index values and percent-change contributions for one index after the other.

`unstack()` breaks up a price index into a list of indexes for each time period.

These methods can be used in a map-reduce to make an index with multiple aggregation structures (like a Paasche index).

**Usage**

```r
## S3 method for class 'piar_index'
stack(x, y, ...)
## S3 method for class 'piar_index'
unstack(x, ...)
```

**Arguments**

- `x`: A price index, as made by, e.g., `elemental_index()`.
- `y`: A price index, or something that can coerced into one. If `x` is a period-over-period index then `y` is coerce into a chainable index; otherwise, `y` is coerced into a direct index.
- `...`: Further arguments passed to or used by methods.

**Value**

A price index that inherits from `chainable_piar_index` if `x` is a period-over-period index, or `direct_piar_index` if `x` is a fixed-base index. If both `x` and `y` are aggregate indexes then the result will also inherit from `aggregate_piar_index`.

`unstack()` returns a list of price indexes with the same class as `x`.

**See Also**

Other index methods: `.piar_index()`, `aggregate.piar_index()`, `as.data.frame.piar_index()`, `chain()`, `contrib()`, `head.piar_index()`, `levels.piar_index()`, `mean.piar_index()`, `merge.piar_index()`, `time.piar_index()`, `vcov.aggregate_piar_index()`
Examples

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

prices2 <- data.frame(
  rel = 1:8,
  period = rep(3:4, each = 4),
  ea = rep(letters[1:2], 4)
)

epr <- with(prices, elemental_index(rel, period, ea))
epr2 <- with(prices2, elemental_index(rel, period, ea))

stack(epr, epr2)
# Unstack does the reverse

all.equal(
  c(unstack(epr), unstack(epr2)),
  unstack(stack(epr, epr2))
)
```

summary.piar_index  

Summarize a price index

Description

Summarize an index as a matrix of index values (i.e., the five-number summary for each period). If there are percent-change contributions, then these are also summarized as a matrix.

Usage

```r
## S3 method for class 'piar_index'
summary(object, ...)
```

Arguments

- `object`  
  A price index, as made by, e.g., `elemental_index()`.

- `...`  
  Further arguments passed to or used by methods.

Value

A list of five-number summaries.
Note

This function is still experimental and may change in the future.

Examples

```r
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))
summary(epr)
```

---

**time.piar_index**

Get the time periods for a price index

**Description**

Methods to get and set the time periods for a price index.

**Usage**

```r
## S3 method for class 'piar_index'
time(x, ...)
time(x) <- value

## S3 replacement method for class 'piar_index'
time(x) <- value

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

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

**Arguments**

- `x` A price index, as made by, e.g., `elemental_index()`.
- `...` Further arguments passed to or used by methods.
- `value` A character vector, or something that can be coerced into one, giving the replacement time periods for `x`. 


**Value**

time() return a character vector with the time periods for a price index. start() and end() return the first and last time period.
The replacement method returns a copy of x with the time periods in value.

**See Also**

Other index methods: [.piar_index(), aggregate.piar_index(), as.data.frame.piar_index(), chain(), contrib(), head.piar_index(), levels.piar_index(), mean.piar_index(), merge.piar_index(), stack.piar_index(), vcov.aggregate.piar_index()
Examples

```r
# A simple aggregation structure
# +--------+
# | 1      |
# +--------+
# | 1 2    |
# +--------+
# | 1 2 1  |
# +--------+

aggregation_weights <- data.frame(
  level1 = c("1", "1", "1"),
  level2 = c("11", "11", "12"),
  ea = c("111", "112", "121"),
  weight = c(1, 3, 4)
)

pias <- as_aggregation_structure(aggregation_weights)

index <- as_index(
  matrix(1:9, 3, dimnames = list(c("111", "112", "121"), NULL))
)

weights(pias)

weights(update(pias, index))
```

---

**vcov.aggregate_piar_index**

*Bootstrap variance for a price index with replicate weights*

Description

Estimate the sampling variance for an aggregate price index when using a sample of elemental aggregates.

Usage

```r
## S3 method for class 'aggregate_piar_index'
vcov(object, repweights, mse = TRUE, sparse = FALSE, ...)
```

Arguments

- `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 variance be centered off the value of the index in `object` (the default), or the mean of the replicates?
sparse  Use sparse matrices from Matrix when aggregating the index. Faster for indexes with large aggregation structures. The default uses regular dense matrices.

...  Further arguments passed to or used by methods.

Details
This function is a simple wrapper that reaggregates the elemental indexes in object using the bootstrap replicate weights in repweights to get a collection of aggregate indexes from which the variance is calculated.

This approach is usually applicable when elemental aggregates are sampled with a stratified design that follows the aggregation structure, so that there is no correlation between the index values for different levels of the index. It ignores any variation from the elemental indexes (which often use judgmental sampling), and ultimately depends on the method of generating replicate weights. (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.

Value
A matrix of variances with a row for each upper-level index and a column for each time period.

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

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

Other index methods: [.piar_index(), aggregate_piar_index(), as.data.frame.piar_index(), chain(), contrib(), head.piar_index(), levels.piar_index(), mean.piar_index(), merge.piar_index(), stack.piar_index(), time.piar_index()}

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)
)

repweights <- matrix(c(0, 2, 3, 1, 2, 4, 2, 3, 3), 3)

# Calculate Jevons elemental indexes

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

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

# Calculate variance
vcov(index, repweights)

---

weights.piar_aggregation_structure

*Get the weights for a price index aggregation structure*

**Description**

Get and set the weights for a price index aggregation structure.

**Usage**

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

weights(object) <- value

## S3 replacement method for class 'piar_aggregation_structure'
weights(object) <- value
```

**Arguments**

- **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 when aggregating the weights (i.e., when `ea_only = FALSE`)? By default, missing values are not removed.
- **...**: Further arguments passed to or used by methods.
- **value**: A numeric vector of weights for the elemental aggregates of `object`. 
Value

weights() returns a list with a named vector of weights for each level in the aggregation structure. If `ea_only = TRUE` then the return value is just a named vector of weights for the elemental aggregates. The replacement method replaces these values without changing the aggregation structure.

See Also

Other aggregation structure methods: `as.matrix.piar_aggregation_structure()`, `update.piar_aggregation_structure()`

Examples

```r
# A simple aggregation structure
# 1
# |-----+-----|
# 11 12
# |---+---| |
# 111 112 121
# (1) (3) (4)
aggregation_weights <- data.frame(
  level1 = c("1", "1", "1"),
  level2 = c("11", "11", "12"),
  ea = c("111", "112", "121"),
  weight = c(1, 3, 4)
)
pias <- as_aggregation_structure(aggregation_weights)

# Extract the weights
weights(pias)

# ... or update them
weights(pias) <- 1:3
weights(pias)
```

Description

Methods to extract and replace index values like a matrix.
Usage

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

Arguments

- `x`: A price index, as made by, e.g., `elemental_index()`.
- `i, j`: Indices for the levels and time periods of a price index. See details.
- `...`: Ignored.
- `value`: A numeric vector.

Details

The extraction methods treat `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 subscripting `x` always returns an index object. This means that subscripting with a matrix is not possible, and only a submatrix can be extracted. As `x` is not an atomic vector, subscripting with a single index like `x[1]` extracts all time periods for that level.

The replacement methods similarly treat `x` as a matrix, and behave 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 percent-change contributions (if any).

Subscripting an aggregate index cannot generally preserve the aggregation structure if any levels are removed or rearranged, and in this case the resulting index is not an aggregate index. Similarly, replacing the values for an aggregate index generally breaks consistency in aggregation, and therefore the result is not an aggregate index.

Value

A price index that inherits from `chainable_piar_index` if `x` is a period-over-period index, or `direct_piar_index` if `x` is a fixed-base index. If `x` inherits from `aggregate_piar_index` then `[` returns an aggregate index if the levels are unchanged.

See Also

Other index methods: `aggregate.piar_index()`, `as.data.frame.piar_index()`, `chain()`, `contrib()`, `head.piar_index()`, `levels.piar_index()`, `mean.piar_index()`, `merge.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `vcov.aggregate_piar_index()`

Examples

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

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

# Extract the indexes like a matrix

epr["a", ]

epr[, 2]

# can be useful for doing specific imputations

epr[1, ] <- 1

epr
Index

* aggregation structure methods
  * as.matrix.piar_aggregation_structure
  * update.piar_aggregation_structure
  * weights.piar_aggregation_structure

* index methods
  * [.piar_index
  * aggregate.piar_index
  * as.data.frame.piar_index
  * chain
  * contrib
  * head.piar_index
  * levels.piar_index
  * mean.piar_index
  * merge.piar_index
  * stack.piar_index
  * time.piar_index
  * vcov.aggregate_piar_index

* index method
  * chain
  * [.piar_index

as.data.frame.piar_aggregation_structure
as.data.frame.piar_index
as_aggregation_structure
as_aggregation_structure()
as_index
as_index()
gpindex::contributions(r)(), 17
gpindex::factor_weights(r)(), 3
gpindex::generalized_mean(), 3, 17, 22, 26
gpindex::generalized_mean(r)(), 3, 17
gpindex::nested_transmute(), 18
gpindex::outliers(), 29
head(), 21
head.piar_index, 4, 7, 14, 15, 20, 25–27, 30, 33, 35, 38
impute_prices, 21
is_aggregate_index(is_index), 24
is_aggregation_structure, 23
is_chainable_index(is_index), 24
is_direct_index(is_index), 24
is_index, 24
levels.piar_index, 4, 7, 14, 15, 21, 25, 26, 27, 30, 33, 35, 38
levels<-.piar_index
(make.unique()), 17
mean.piar_index, 4, 7, 14, 15, 21, 25, 27, 30, 33, 35, 38
merge(), 3
merge.piar_index, 4, 7, 14, 15, 21, 25, 26, 27, 30, 33, 35, 38
ms_prices(price_data), 29
ms_weights(price_data), 29
piar_aggregation_structure, 10, 24
piar_aggregation_structure
(aggregate_structure), 5
piar_index, 12, 18, 24, 28
price_data, 29
price_relative, 29
price_relative(), 17, 18, 23
rebase (chain), 12
rebase(), 18
shadow_price(impute_prices), 21
shadow_price(), 18, 29
stack.piar_index, 4, 7, 14, 15, 21, 25–27, 30, 33, 35, 38
start.piar_index(time.piar_index), 32
summary.piar_index, 31
tail(), 21
tail.piar_index(head.piar_index), 20
time.piar_index, 4, 7, 14, 15, 21, 25–27, 30, 32, 35, 38
time<-(time.piar_index), 32
unchain(chain), 12
unstack.piar_index(stack.piar_index), 30
update(), 6
update.piar_aggregation_structure, 8, 33, 37
vcov.aggregate_piar_index, 4, 7, 14, 15, 21, 25–27, 30, 33, 34, 38
weights(), 6
weights.piar_aggregation_structure, 8, 33, 36
weights<-.piar_aggregation_structure
(levels.piar_index), 25
(time.piar_index), 32
unchain(chain), 12