Package 'piar'

September 17, 2025

Title Price Index Aggregation

Version 0.8.3

Description Most price indexes are made with a two-step procedure, where period-over-period elementary indexes are first calculated for a collection of elementary 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 collection of functions that revolve around this work flow, making it easy to build standard price indexes, and implement the methods described by Balk (2008, <doi:10.1017/CBO9780511720758>), von der Lippe (2007, <doi:10.3726/978-3-653-01120-3>), and the CPI manual (2020, <doi:10.5089/9781484354841.069>) for bilateral price indexes.

Depends R (>= 4.1)

Imports stats, utils, gpindex (>= 0.6.2), Matrix (>= 1.5-0)

Suggests data.tree, knitr, rmarkdown, sps, testthat (>= 3.0.0), treemap

License MIT + file LICENSE

Encoding UTF-8

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

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

LazyData true

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.2

NeedsCompilation no

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

Maintainer Steve Martin <marberts@protonmail.com>

Repository CRAN

Date/Publication 2025-09-17 04:10:02 UTC

Contents

`aggregate.piar_index` *Aggregate elementary price indexes*

Description

Aggregate elementary price indexes with a price index aggregation structure.

Usage

```
## S3 method for class 'chainable_piar_index'
aggregate(
  x,
  pias,
  ...,
  pias2 = NULL,
  na.rm = FALSE,
  contrib = TRUE,
  r = 1,
  include_ea = TRUE,
  duplicate_contrib = c("make.unique", "sum")
)

## S3 method for class 'direct_piar_index'
aggregate(
  x,
  pias,
  ...,
  pias2 = NULL,
  na.rm = FALSE,
  contrib = TRUE,
  r = 1,
  include_ea = TRUE,
  duplicate_contrib = c("make.unique", "sum")
)
```

Arguments

x	A price index, usually made by elementary_index().
pias	A price index aggregation structure or something that can be coerced into one. This can be made with aggregation_structure().
...	Not currently used.
pias2	An optional secondary aggregation structure, usually with current-period weights, to make a superlative index. See details.
na.rm	Should missing values be removed? By default, missing values are not removed. Setting na.rm = TRUE is equivalent to overall mean imputation.
contrib	Aggregate percent-change contributions in x (if any)?
r	Order of the generalized mean to aggregate index values. 0 for a geometric index (the default for making elementary indexes), 1 for an arithmetic index (the default for aggregating elementary 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. If pias2 is given then the index is aggregated with a quadratic mean of order 2*r.
include_ea	Should indexes for the elementary aggregates be included along with the aggregated indexes? By default, all index values are returned.

duplicate_contrib
: The method to deal with duplicate product contributions. Either 'make.unique' to treat duplicate products as distinct products and make their names unique with `make.unique()` or 'sum' to add contributions for each product.

Details

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

1. aggregates the elementary 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 elementary 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 elementary indexes. Otherwise, when `x` are fixed-base elementary indexes, the result is a collection of aggregated fixed-base (direct) indexes.

By default, missing elementary indexes will propagate when aggregating the index. Missing elementary indexes can be due to both missingness of these values in `x`, and the presence of elementary 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 and weights can't be price updated), 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), with analogous details in chapter 9 of the PPI manual (2004). Aggregating with a non-arithmetic mean follows the same steps, except that the elementary 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 elementary aggregate if the elementary 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` and `contrib = FALSE`).

If two aggregation structures are given then the steps above are done for each aggregation structure, with the aggregation for `pias` done with a generalized mean of order `r` the aggregation for `pias2` done with a generalized mean of order `-r`. The resulting indexes are combined with a geometric mean to make a superlative quadratic mean of order `2*r` index. Percent-change contributions are

combined using a generalized van IJzeren decomposition; see gpindex::nested_transmute() for details.

Value

An aggregate price index that inherits from 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 elementary indexes as a matrix operation when there are no missing values. See the examples for details.

References

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

ILO, IMF, UNECE, OECD, and World Bank. (2004). *Producer Price Index Manual: Theory and Practice*. International Monetary Fund.

IMF, ILO, OECD, Eurostat, UNECE, and World Bank. (2020). *Consumer Price Index Manual: Concepts and Methods*. International Monetary Fund.

von der Lippe, P. (2007). *Index Theory and Price Statistics*. Peter Lang.

See Also

Other index methods: [.piar_index(), as.data.frame.piar_index(), as.ts.piar_index(), chain(), contrib(), head.piar_index(), is.na.piar_index(), levels.piar_index(), mean.piar_index, merge.piar_index(), split.piar_index(), stack.piar_index(), time.piar_index(), window.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")),
  weights = 1:3
)

# Calculate Jevons elementary indexes

(elementary <- elementary_index(prices, rel ~ period + ea))

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

(index <- aggregate(elementary, 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 elementary 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 elementary aggregates, which should have no duplicates. All vectors should be the same length, without `NA`s, and there should be no duplicates across different levels of `x`. Names for `x` are used as level names; otherwise, levels are named 'level1', 'level2', ..., 'ea'.
`weights`	A numeric vector of aggregation weights for the elementary aggregates (i.e., the last vector in `x`), or something that can be coerced into one. The default is to give each elementary 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 named list of character vectors that give the levels of `x`.
`weights`	A vector giving the weight for each elementary aggregate.

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 `elementary_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)

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

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

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

with(
  aggregation_weights,
  aggregation_structure(expand_classification(ea), weight)
)
```

---

as.data.frame.piar_index

Coerce an index into a tabular form

Description

Turn an index into a data frame or a matrix.

Usage

```
## S3 method for class 'piar_index'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  ...,
  contrib = FALSE,
  stringsAsFactors = FALSE
)

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

Arguments

x	A price index, as made by, e.g., elementary_index().
row.names, stringsAsFactors	See as.data.frame().
optional	Not currently used.
...	Not currently used.
contrib	Include percent-change contributions (the default does not include them).

Value

as.data.frame() returns the index values in x as a data frame with three columns: period, level, and value. If contrib = TRUE then there is a fourth (list) column contrib containing percent-change contributions.

as.matrix() returns the index values in x as a matrix with a row for each level and a column for each time period in x.

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, as.ts.piar_index(), chain(), contrib(), head.piar_index(), is.na.piar_index(), levels.piar_index(), mean.piar_index, merge.piar_index(), split.piar_index(), stack.piar_index(), time.piar_index(), window.piar_index()

Examples

```
index <- as_index(matrix(1:6, 2))

as.data.frame(index)

as.matrix(index)
```

`as.matrix.piar_aggregation_structure` *Coerce an 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, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A price index aggregation structure, as made by `aggregation_structure()`.
`...`	Not currently used for the matrix method. Extra arguments to `as.data.frame.list()` for the data frame method.
`sparse`	Should the result be a sparse matrix from **Matrix**? This is faster for large aggregation structures. The default returns an ordinary dense matrix.
`row.names`	See `as.data.frame()`.
`optional`	Not currently used.

Value

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

`as.data.frame()` takes an aggregation structure and returns a data frame that could have generated it.

See Also

`as_aggregation_structure()` for coercing into an aggregation structure.

`treemap::treemap()` and `data.tree::as.Node()` for visualizing an aggregation structure.

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

Examples

```
# 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)

as.matrix(pias)

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

## Not run:
# Visualize as a treemap.
treemap::treemap(
  aggregation_weights,
  index = names(aggregation_weights)[-4],
  vSize = "weight",
  title = "aggregation structure"
)

# Or turn into a more genereal tree object and plot.
aggregation_weights$pathString <- do.call(
  \(...) paste(..., sep = "/"),
  aggregation_weights[-4]
)
plot(data.tree::as.Node(aggregation_weights))

## End(Not run)
```

---

as.ts.piar_index *Coerce an index into a time series*

Description

Turn an index into a regular time series, represented as a `ts` object.

Usage

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

Arguments

x	A price index, as made by, e.g., elementary_index().
...	Additional arguments passed to ts().

Value

A time series object.

See Also

Other index methods: [.piar_index(), aggregate.piar_index, as.data.frame.piar_index(), chain(), contrib(), head.piar_index(), is.na.piar_index(), levels.piar_index(), mean.piar_index, merge.piar_index(), split.piar_index(), stack.piar_index(), time.piar_index(), window.piar_index()

Examples

```
as.ts(as_index(matrix(1:9, 3)))
```

---

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, ...)
```

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 elementary aggregates. The default is to give each elementary aggregate the same weight.

Details

The default method attempts to coerce x into a list prior to calling aggregation_structure().

The data frame and matrix methods treat x as a table with a row for each elementary aggregate, a column of labels for each level in the aggregation structure, and a column of weights for the elementary aggregates.

Value

A price index aggregation structure that inherits from piar_aggregation_structure.

See Also

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

Examples

```
# 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 <- 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))
)
```

---

as_index *Coerce to a price index*

Description

Coerce pre-computed index values into an index object.

Usage

```
as_index(x, ...)

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

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

## S3 method for class 'data.frame'
as_index(x, ..., contrib = FALSE)

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

## S3 method for class 'direct_piar_index'
as_index(x, ..., chainable = FALSE)

## S3 method for class 'mts'
as_index(x, ...)
```

Arguments

x	An object to coerce into a price index.
...	Further arguments passed to or used by methods.
chainable	Are the index values in x period-over-period indexes, suitable for a chained calculation (the default)? This should be FALSE when x contains fixed-base (direct) index values.
contrib	Should the index values in x be used to construct percent-change contributions? The default does not make contributions.

Details

Numeric matrices are coerced into an index object by treating each column as a separate time period, and each row as a separate level of the index (e.g., an elementary aggregate). Column names are used to denote time periods, and row names are used to denote levels (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 and multiple time series methods 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[[1]]` as columns and the levels of `x[[2]]` as rows (coercing to a factor if necessary). It then populates this matrix with the corresponding values in `x[[3]]`, and uses the matrix method for `as_index()`. If `contrib = TRUE` and there is a fourth list column of product contributions then these are also included in the resulting 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

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

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

index <- elementary_index(prices, rel ~ period + ea)

all.equal(as_index(as.data.frame(index)), index)
all.equal(as_index(as.matrix(index)), index)
```

`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 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)), ...)

unchain(x, ...)

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

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

rebase(x, ...)

## Default S3 method:
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., elementary_index().
...	Further arguments passed to or used by methods.
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. If base is a length-one character vector giving a time period of x then the index values for this time period are used as the base-period values.

Details

The default methods attempt 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 does nothing, as does chaining a fixed-base index.

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.

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()`, `as.ts.piar_index()`, `contrib()`, `head.piar_index()`, `is.na.piar_index()`, `levels.piar_index()`, `mean.piar_index`, `merge.piar_index()`, `split.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `window.piar_index()`

Examples

```
index <- as_index(matrix(1:9, 3))

# Make period 0 the fixed base period

chain(index)

# Chaining and unchaining reverse each other

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

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

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

---

contrib *Extract percent-change contributions*

Description

Extract a matrix or data frame of percent-change contributions from a price index.

Usage

```
contrib(x, ...)

## S3 method for class 'piar_index'
contrib(x, level = levels(x)[1L], period = time(x), ..., pad = 0)

contrib2DF(x, ...)

## S3 method for class 'piar_index'
contrib2DF(x, level = levels(x)[1L], period = time(x), ...)

contrib(x, ...) <- value

## S3 replacement method for class 'piar_index'
contrib(x, level = levels(x)[1L], period = time(x), ...) <- value

set_contrib(x, ..., value)

set_contrib_from_index(x)
```

Arguments

`x`	A price index, as made by, e.g., `elementary_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). `contrib2DF()` can accept multiple levels.
`period`	The time periods for which percent-change contributions are desired, defaulting to all time periods.
`pad`	A numeric value to pad contributions so that they fit into a rectangular array when products differ over time. The default is 0.
`value`	A numeric matrix of replacement contributions with a row for each product and a column for each time period. Recycling occurs along time periods.

Value

`contrib()` returns 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 `pad` to fit into a rectangular array when products differ over time. The replacement methods returns a copy of `x` with contributions given by the matrix `value`. (`set_contrib()` is an alias that's easier to use with pipes.) `set_contrib_from_index()` is a helper to return a copy of `x` with all contributions set to the corresponding index value minus 1.

`contrib2DF()` returns a data frame of contributions with four columns: `period`, `level`, `product`, and `value`.

See Also

Other index methods: [.piar_index(), aggregate.piar_index, as.data.frame.piar_index(), as.ts.piar_index(), chain(), head.piar_index(), is.na.piar_index(), levels.piar_index(), mean.piar_index, merge.piar_index(), split.piar_index(), stack.piar_index(), time.piar_index(), window.piar_index()

Examples

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

index <- elementary_index(prices, rel ~ period + ea, contrib = TRUE)

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

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

# Percent-change contributions for the top-level index

contrib(index)

contrib2DF(index)

# Calculate EA contributions for the chained index

library(gpindex)

arithmetic_contributions(
  as.matrix(chain(index))[c("a", "b", "c"), 2],
  weights(pias)
)
```

---

cut.piar_aggregation_structure

Cut an aggregation structure

Description

Keep only the part of an aggregation structure above or below a certain level.

Usage

```
## S3 method for class 'piar_aggregation_structure'
cut(x, level, ..., na.rm = FALSE, upper = TRUE)
```

Arguments

x	A price index aggregation structure, as made by aggregation_structure().
level	A positive integer, or something that can be coerced into one, giving the level at which to cut x.
...	Not currently used.
na.rm	Should missing values be removed when aggregating the weights? By default, missing values are not removed.
upper	Keep only the part of x above level (the default); otherwise, return the part of x below level.

Value

A price index aggregation structure.

See Also

Other aggregation structure methods: as.matrix.piar_aggregation_structure(), levels.piar_aggregation_structu
update.piar_aggregation_structure(), weights.piar_aggregation_structure()

Examples

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

# Turn it into
#            1
#      |-----+-----|
#      11          12
```

```
#     (4)         (4)

cut(pias, 2)
```

`elementary_index` *Make elementary/elemental price indexes*

Description

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

Usage

```
elementary_index(x, ...)

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

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

## S3 method for class 'data.frame'
elementary_index(x, formula, ..., weights = NULL, product = NULL)

elemental_index(x, ...)
```

Arguments

`x`	Period-over-period or fixed-base price relatives. Currently there are methods for numeric vectors (which can be made with `price_relative()`) and data frames.
`...`	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 of the levels of `period`, to agree with `cut()`. The default makes an index for one time period.

ea	A factor, or something that can be coerced into one, giving the elementary aggregate associated with each price relative in x. The default makes an index for one elementary aggregate.
weights	A numeric vector of weights for the price relatives in x, or something that can be coerced into one. The default is equal weights. This is evaluated in x for the data frame method.
product	A character vector of product names, or something that can be coerced into one, for each price relative in x when making percent-change contributions. The default uses the names of x, if any; otherwise, elements of x are given sequential names within each elementary aggregate. This is evaluated in x for the data frame method.
chainable	Are the price relatives in x period-over-period relatives that are suitable 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.
contrib	Should percent-change contributions be calculated? The default does not calculate contributions.
r	Order of the generalized mean to aggregate price relatives. 0 for a geometric index (the default for making elementary indexes), 1 for an arithmetic index (the default for aggregating elementary 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.
formula	A two-sided formula with price relatives on the left-hand side, and time periods and elementary aggregates (in that order) on the right-hand side.

Details

When supplied with a numeric vector, elementary_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 elementary aggregate and time period, elementary_index() calculates an index based on a generalized mean of order r and, optionally, percent-change contributions. Product names should be unique within each time period when making contributions, and, if not, are passed to make.unique() with a warning. The default (r = 0 and no weights) makes Jevons elementary indexes. See chapter 8 (pp. 175–190) of the CPI manual (2020) for more detail about making elementary indexes, or chapter 9 of the PPI manual (2004), and chapter 5 of Balk (2008).

The default method simply coerces x to a numeric vector prior to calling the method above. The data frame method provides a formula interface to specify columns of price relatives, time periods, and elementary aggregates and call the method above.

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) elementary indexes; if these are fixed-base relatives, then the result is a collection of fixed-base (direct) elementary 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 `x` will propagate throughout the index calculation. Ignoring missing values with `na.rm = TRUE` is the same as overall mean (parental) imputation, and needs to be explicitly set in the call to `elementary_index()`. Explicit imputation of missing relatives, and especially imputation of missing prices, should be done prior to calling `elementary_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 `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.

`elemental_index()` is an alias for `elementary_index()`.

Value

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

References

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

ILO, IMF, UNECE, OECD, and World Bank. (2004). *Producer Price Index Manual: Theory and Practice*. International Monetary Fund.

IMF, ILO, OECD, Eurostat, UNECE, and World Bank. (2020). *Consumer Price Index Manual: Concepts and Methods*. International Monetary Fund.

von der Lippe, P. (2007). *Index Theory and Price Statistics*. Peter Lang.

See Also

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

`as_index()` to turn pre-computed (elementary) index values into an index object.

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

`aggregate()` to aggregate elementary indexes according to an aggregation structure.

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

Examples

```
library(gpindex)

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

# Calculate Jevons elementary indexes

elementary_index(prices, 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), 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

elementary_index(
  prices,
  rel ~ period + ea,
  weights = 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. Expanded classifications be interacted together to get all combinations of aggregation structures.

Usage

```
expand_classification(x, width = 1L, pad = NA)

interact_classifications(..., sep = ":")
```

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.
pad	A string used to pad the shorter labels for an unbalanced classification. The default pads with NA.
...	Lists 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 (as made by expand_classification()).
sep	A character used to combine codes/labels across elements of .... The default uses ':'.

Value

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

interact_classfications() returns a list of lists with the same structure as expand_classification().

See Also

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

split_classification() to expand a classification by splitting along a delimiter.

csh_from_digits() in the **accumulate** package for different handling of unbalanced classifications.

Examples

```
# A simple classification structure
#            1
#      |-----+-----|
#      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, ...)

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

Arguments

`x`	A price index, as made by, e.g., `elementary_index()`.
`n`	See `head()`/`tail()`. The default takes the first/last 6 levels of `x`.
`...`	Not currently used.

Value

A price index that inherits from the same class as `x`.

See Also

Other index methods: `[.piar_index()`, `aggregate.piar_index`, `as.data.frame.piar_index()`, `as.ts.piar_index()`, `chain()`, `contrib()`, `is.na.piar_index()`, `levels.piar_index()`, `mean.piar_index`, `merge.piar_index()`, `split.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `window.piar_index()`

Examples

```
index <- as_index(matrix(1:9, 3))

head(index, 1)

tail(index, 1)
```

impute_prices *Impute missing prices*

Description

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

Usage

```
shadow_price(x, ...)

## Default S3 method:
shadow_price(
  x,
  ...,
  period,
  product,
  ea,
  pias = NULL,
  weights = NULL,
  r1 = 0,
  r2 = 1
)

## S3 method for class 'data.frame'
shadow_price(x, formula, ..., weights = NULL)

carry_forward(x, ...)

## Default S3 method:
carry_forward(x, ..., period, product)

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

carry_backward(x, ...)

## Default S3 method:
carry_backward(x, ..., period, product)

## S3 method for class 'data.frame'
carry_backward(x, formula, ...)
```

Arguments

x	Either a numeric vector (or something that can be coerced into one) or data frame of prices.

...	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 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 elementary 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 elementary indexes only (i.e., not recursively).
weights	A numeric vector of weights for the prices in x (i.e., product weights), or something that can be coerced into one. The default is to give each price equal weight. This is evaluated in x for the data frame method.
r1	Order of the generalized-mean price index used to calculate the elementary 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 elementary 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.
formula	A two-sided formula with prices on the left-hand side. For carry_forward() and carry_backward(), the right-hand side should have time periods and products (in that order); for shadow_price(), the right-hand side should have time period, products, and elementary aggregates (in that order).

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 elementary index for the elementary 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 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 numeric vector of prices with missing values replaced (where possible).

References

IMF, ILO, OECD, Eurostat, UNECE, and World Bank. (2020). *Consumer Price Index Manual: Concepts and Methods*. International Monetary Fund.

See Also

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

Examples

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

carry_forward(prices, price ~ period + product)

shadow_price(prices, price ~ period + product + ea)
```

---

is.na.piar_index *Missing values in a price index*

Description

Identify missing values in a price index.

Usage

```
## S3 method for class 'piar_index'
is.na(x)

## S3 method for class 'piar_index'
anyNA(x, recursive = FALSE)
```

Arguments

x	A price index, as made by, e.g., `elementary_index()`.
recursive	Check if x also has missing percent-change contributions. By default only index values are checked for missingness.

Value

`is.na()` returns a logical matrix, with a row for each level of `x` and a columns for each time period, that indicates which index values are missing.

`anyNA()` returns `TRUE` if any index values are missing, or percent-change contributions (if `recursive = TRUE`).

See Also

Other index methods: `[.piar_index()`, `aggregate.piar_index`, `as.data.frame.piar_index()`, `as.ts.piar_index()`, `chain()`, `contrib()`, `head.piar_index()`, `levels.piar_index()`, `mean.piar_index`, `merge.piar_index()`, `split.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `window.piar_index()`

Examples

```
index <- as_index(matrix(c(1, 2, 3, NA, 5, NA), 2))

anyNA(index)
is.na(index)

# Carry forward imputation

index[is.na(index)] <- 1
index
```

---

is_aggregation_structure

Test if an object is an aggregation structure

Description

Test if an object is a price index aggregation structure.

Usage

```
is_aggregation_structure(x)
```

Arguments

`x`	An object to test.

Value

Returns `TRUE` if `x` inherits from `piar_aggregation_structure`.

is_index *Test if an object is a price index*

Description

Test if an object is a index object or a subclass of an index object.

Usage

```
is_index(x)

is_chainable_index(x)

is_direct_index(x)
```

Arguments

x	An object to test.

Value

`is_index()` returns `TRUE` if `x` inherits from `piar_index`.

`is_chainable_index()` returns `TRUE` if `x` inherits from `chainable_piar_index`.

`is_direct_index()` returns `TRUE` if `x` inherits from `direct_piar_index`.

levels.piar_aggregation_structure *Get the levels for an aggregation structure*

Description

Get the hierarchical list of levels for an aggregation structure. It is an error to try and replace these values.

Usage

```
## S3 method for class 'piar_aggregation_structure'
levels(x)
```

Arguments

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

Value

A list of character vectors giving the levels for each position in the aggregation structure.

See Also

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

`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

set_levels(x, value)
```

Arguments

x	A price index, as made by, e.g., elementary_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. (set_levels() is an alias that's easier to use with pipes.)

See Also

Other index methods: [.piar_index(), aggregate.piar_index, as.data.frame.piar_index(),
as.ts.piar_index(), chain(), contrib(), head.piar_index(), is.na.piar_index(), mean.piar_index,
merge.piar_index(), split.piar_index(), stack.piar_index(), time.piar_index(), window.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 'chainable_piar_index'
mean(
  x,
  ...,
  weights = NULL,
  window = ntime(x),
  na.rm = FALSE,
  contrib = TRUE,
  r = 1,
  duplicate_contrib = c("make.unique", "sum")
)

## S3 method for class 'direct_piar_index'
mean(
  x,
  ...,
  weights = NULL,
  window = ntime(x),
  na.rm = FALSE,
  contrib = TRUE,
  r = 1,
  duplicate_contrib = c("make.unique", "sum")
)
```

Arguments

`x`	A price index, as made by, e.g., `elementary_index()`.
`...`	Not currently used.
`weights`	A numeric vector of weights for the index values in `x`, or something that can be coerced into one. 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 averages over all periods in `x`. 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.

`contrib` Aggregate percent-change contributions in `x` (if any)?

`r` Order of the generalized mean to aggregate index values. 0 for a geometric index (the default for making elementary indexes), 1 for an arithmetic index (the default for aggregating elementary 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.

`duplicate_contrib` The method to deal with duplicate product contributions. Either 'make.unique' to make duplicate product names unique with `make.unique()` or 'sum' to add contributions for the same products across subperiods.

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 (with a warning), 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.

Percent-change contributions are aggregated if `contrib = TRUE` following the same approach as `aggregate()`.

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, averaged over subperiods, that inherits from the same class as `x`.

References

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

See Also

Other index methods: `[.piar_index()`, `aggregate.piar_index`, `as.data.frame.piar_index()`, `as.ts.piar_index()`, `chain()`, `contrib()`, `head.piar_index()`, `is.na.piar_index()`, `levels.piar_index()`, `merge.piar_index()`, `split.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `window.piar_index()`

Examples

```
index <- as_index(matrix(c(1:12, 12:1), 2, byrow = TRUE), chainable = FALSE)

# Turn a monthly index into a quarterly index
mean(index, window = 3)
```

`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 elementary aggregates come from different sources of data, or use different index-number formulas.

Usage

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

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

Arguments

`x`	A price index, as made by, e.g., `elementary_index()`.
`y`	A price index, or something that can coerced into one. If `x` is a period-over-period index then `y` is coerced into a chainable index; otherwise, `y` is coerced into a direct index.
`...`	Not currently used.

Value

A combined price index that inherits from the same class as `x`.

See Also

Other index methods: `[.piar_index()`, `aggregate.piar_index`, `as.data.frame.piar_index()`, `as.ts.piar_index()`, `chain()`, `contrib()`, `head.piar_index()`, `is.na.piar_index()`, `levels.piar_index()`, `mean.piar_index`, `split.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `window.piar_index()`

Examples

```
index1 <- as_index(matrix(1:6, 2))

index2 <- index1
levels(index2) <- 3:4

merge(index1, index2)
```

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`.

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 percent-change 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.

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

```
price_relative(x, ...)

## Default S3 method:
price_relative(x, ..., period, product)

## S3 method for class 'data.frame'
price_relative(x, formula, ...)
```

Arguments

`x`	Either a numeric vector (or something that can be coerced into one) or data frame of prices.
`...`	Further arguments passed to or used by methods.
`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`.
`formula`	A two-sided formula with prices on the left-hand side, and time periods and products (in that order) on the right-hand side.

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

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

`split.piar_index` *Split an index into groups*

Description

Split an index into groups of indexes according to a factor, along either the levels or time periods of the index.

Usage

```
## S3 method for class 'piar_index'
split(x, f, drop = FALSE, ..., margin = c("levels", "time"))

## S3 replacement method for class 'piar_index'
split(x, f, drop = FALSE, ..., margin = c("levels", "time")) <- value
```

Arguments

`x`	A price index, as made by, e.g., `elementary_index()`.
`f`	A factor or list of factors to group elements of `x`.
`drop`	Should levels that do not occur in `f` be dropped? By default all levels are kept.
`...`	Further arguments passed to `split.default()`.
`margin`	Either 'levels' to split over the levels of `x` (the default), or 'time' to split over the time periods of `x`.
`value`	A list of values compatible with the splitting of `x`, or something that can be coerced into one, recycled if necessary.

Value

`split()` returns a list of index objects for each level in `f`. The replacement method replaces these values with the corresponding element of `value`.

See Also

Other index methods: `[.piar_index()`, `aggregate.piar_index`, `as.data.frame.piar_index()`, `as.ts.piar_index()`, `chain()`, `contrib()`, `head.piar_index()`, `is.na.piar_index()`, `levels.piar_index()`, `mean.piar_index`, `merge.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `window.piar_index()`

Examples

```
index <- as_index(matrix(1:6, 2))

split(index, 1:2)

split(index, c(1, 1, 2), margin = "time")
```

`split_classification` *Split a hierarchical classification*

Description

Expand a character representation of a hierarchical classification to make a price index aggregation structure by splitting along a delimiter.

Usage

```
split_classification(x, split, ..., sep = ".", pad = NA)
```

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).
`split`	A regular expression to delineate and split the levels in `x`. See `strsplit()`.
`...`	Additional argument to pass to `strsplit()`.
`sep`	A character used to delineate levels in `x` in the result. The default separates levels by '.'.
`pad`	A string used to pad the shorter labels for an unbalanced classification. The default pads with NA.

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.

`expand_classification()` to expand a classification by the width of the levels.

Examples

```
#' # A simple classification structure
#            1
#      |-----+-----|
#      11          12
#  |---+---|       |
#  111     112     121

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

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

split_classification(c("01.1.1", "01.1.2", "01.2.1"), ".", fixed = TRUE)
```

`stack.piar_index` *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

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

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

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

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

Arguments

`x`	A price index, as made by, e.g., `elementary_index()`.
`y`	A price index, or something that can coerced into one. If `x` is a period-over-period index then `y` is coerced into a chainable index; otherwise, `y` is coerced into a direct index.
`...`	Not currently used.

Value

`stack()` returns a combined price index that inherits from the same class as `x`.

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

Note

It may be necessary to use `rebase()` prior to stacking fixed-based price indexes to ensure they have the same base period.

See Also

Other index methods: `[.piar_index()`, `aggregate.piar_index`, `as.data.frame.piar_index()`, `as.ts.piar_index()`, `chain()`, `contrib()`, `head.piar_index()`, `is.na.piar_index()`, `levels.piar_index()`, `mean.piar_index`, `merge.piar_index()`, `split.piar_index()`, `time.piar_index()`, `window.piar_index()`

Examples

```
index1 <- as_index(matrix(1:6, 2))

index2 <- index1
time(index2) <- 4:6

stack(index1, index2)

# Unstack does the reverse

all.equal(
  c(unstack(index1), unstack(index2)),
  unstack(stack(index1, index2))
)
```

---

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

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

time(x) <- value

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

```
set_time(x, value)

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

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

ntime(x)
```

Arguments

x	A price index, as made by, e.g., elementary_index().
...	Not currently used.
value	A character vector, or something that can be coerced into one, giving the replacement time periods for x.

Value

time() returns a character vector with the time periods for a price index. start() and end() return the first and last time period.

ntime() returns the number of time periods, analogous to nlevels().

The replacement method returns a copy of x with the time periods in value. (set_time() is an alias that's easier to use with pipes.)

See Also

Other index methods: [.piar_index(), aggregate.piar_index, as.data.frame.piar_index(), as.ts.piar_index(), chain(), contrib(), head.piar_index(), is.na.piar_index(), levels.piar_index(), mean.piar_index, merge.piar_index(), split.piar_index(), stack.piar_index(), window.piar_index()

---

update.piar_aggregation_structure

Update an aggregation structure

Description

Price update the weights in a price index aggregation structure.

Usage

```
## S3 method for class 'piar_aggregation_structure'
update(object, index, period = end(index), ..., r = 1)
```

Arguments

object	A price index aggregation structure, as made by aggregation_structure().
index	A fixed-base (direct) price index, or something that can be coerced into one. Usually 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 index.
...	Not currently used.
r	Order of the generalized mean to update the weights. The default is 1 for an arithmetic index.

Value

A copy of object with price-updated weights using the index values in index.

See Also

aggregate() to make an aggregated price index.

Other aggregation structure methods: as.matrix.piar_aggregation_structure(), cut.piar_aggregation_structure, levels.piar_aggregation_structure(), weights.piar_aggregation_structure()

Examples

```
# 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)

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

weights(pias, ea_only = FALSE)

weights(update(pias, index), ea_only = FALSE)
```

`weights.piar_aggregation_structure` *Get the weights for an aggregation structure*

Description

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

Usage

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

weights(object) <- value

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

set_weights(object, value)
```

Arguments

`object`	A price index aggregation structure, as made by `aggregation_structure()`.
`...`	Not currently used.
`ea_only`	Should weights be returned for only the elementary aggregates (the default)? Setting to `FALSE` 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.
`value`	A numeric vector of weights for the elementary aggregates of `object`.

Value

`weights()` returns a named vector of weights for the elementary aggregates. The replacement method replaces these values without changing the aggregation structure. (`set_weights()` is an alias that's easier to use with pipes.)

If `ea_only = FALSE` then the return value is a list with a named vector of weights for each level in the aggregation structure.

See Also

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

Examples

```
# 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)
```

`window.piar_index` *Index window*

Description

Extract and replace index values over a window of time periods.

Usage

```
## S3 method for class 'piar_index'
window(x, start = NULL, end = NULL, ...)

## S3 replacement method for class 'piar_index'
window(x, start = NULL, end = NULL, ...) <- value
```

Arguments

`x`	A price index, as made by, e.g., `elementary_index()`.
`start`	The time period to start the window. The default in the first period of `x`.
`end`	The time period to end the window. The default is the last period of `x`.

...	Not currently used.
value	A numeric vector or price index.

Value

window() extracts a price index over a window of time periods that inherits from the same class as x. The replacement method replaces these with value.

See Also

Other index methods: [.piar_index(), aggregate.piar_index, as.data.frame.piar_index(), as.ts.piar_index(), chain(), contrib(), head.piar_index(), is.na.piar_index(), levels.piar_index(), mean.piar_index, merge.piar_index(), split.piar_index(), stack.piar_index(), time.piar_index()

Examples

```
x <- as_index(matrix(1:9, 3))

window(x, "2")

window(x, "2") <- 1
x
```

[.piar_index *Extract and replace index values*

Description

Methods to extract and replace index values like a matrix.

Usage

```
## 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., elementary_index().
i, j	Indices for the levels and time periods of a price index. See details.
...	Not currently used.
value	A numeric vector or price index. See details.

Details

The extraction method treats `x` like a matrix of index values with (named) rows for each level and columns for each time 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 method similarly treat `x` like a matrix. If `value` is an index object with the same number of time periods as `x[i, j]` and it inherits from the same class as `x`, then the index values and percent-change contributions of `x[i, j]` are replaced with those for the corresponding levels of `value`. If `value` is not an index, then it is coerced to a numeric vector and behaves the same as replacing values in a matrix. Note that replacing the values of an index will remove the corresponding percent-change contributions (if any). Unlike extraction, it is possible to replace value in `x` using a logical matrix or a two-column matrix of indices.

Value

A price index that inherits from the same class as `x`.

See Also

Other index methods: `aggregate.piar_index`, `as.data.frame.piar_index()`, `as.ts.piar_index()`, `chain()`, `contrib()`, `head.piar_index()`, `is.na.piar_index()`, `levels.piar_index()`, `mean.piar_index`, `merge.piar_index()`, `split.piar_index()`, `stack.piar_index()`, `time.piar_index()`, `window.piar_index()`

Examples

```
index <- as_index(matrix(1:6, 2))

index["1", ]

index[, 2]

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

index
```

Index