Package 'gpindex'

July 25, 2020

Title Generalized Price and Quantity Indexes

Version 0.1.2

Description A small package for calculating lots of different price indexes, and by extension quantity indexes. Provides tools to build and work with any type of generalized bilateral index (of which most price indexes are), along with a few important indexes that don't belong to the generalized family. Implements and extends many of the methods in Balk (2008, ISBN:978-1-107-40496-0) and ILO, IMF, OECD, Eurostat, UN, and World Bank (2004, ISBN:92-2-113699-X) for bilateral price indexes.

Depends R (>= 4.0)

License MIT + file LICENSE

Encoding UTF-8

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

LazyData true

NeedsCompilation no

Author Steve Martin [aut, cre, cph]

Maintainer Steve Martin <stevemartin041@gmail.com>

Repository CRAN

Date/Publication 2020-07-25 04:50:02 UTC

R topics documented:

gpindex-package *Generalized Price and Quantity Indexes*

Description

A small package for calculating lots of different price indexes, and by extension quantity indexes. Provides tools to build and work with any type of generalized bilateral index (of which most price indexes are), along with a few important indexes that don't belong to the generalized family. Implements and extends many of the methods in Balk (2008, ISBN:978-1-107-40496-0) and ILO, IMF, OECD, Eurostat, UN, and World Bank (2004, ISBN:92-2-113699-X) for bilateral price indexes.

Details

To avoid duplication, everything is framed as a price index; it is trivial to turn a price index into its analogous quantity index by simply switching prices and quantities.

Generalized indexes are a large family of price indexes that are consistent in aggregation (Balk, 2008, section 3.7.3). Almost all bilateral price indexes used in practice are either generalized indexes (like the Laspeyres and Paasche index) or are nested generalized indexes (like the Fisher index).

All generalized indexes are based on the generalized mean, which is provided by the `mean_generalized()` function. Given a set of price relatives and weights, any generalized price index is easily calculated as a generalized mean.

Two important functions for decomposing generalized means are given by `weights_change()` and `weights_factor()`. These functions can be used to calculate quote contributions and price-update weights for generalized indexes.

Together these functions, along with `logmean_generalized()`, provide the key mathematical apparatus to work with any type of generalized index, and those that are nested generalized indexes.

On top of these basic mathematical tools are functions for making standard price indexes when both prices and quantities are known. Weights for a large variety of indexes can be calculated with `index_weights()`, which can be plugged into the relevant generalized mean to calculate most common price indexes, and many uncommon ones. The `index` functions provide a simple wrapper.

Note

There are a number of R packages on the CRAN for working with price/quantity indexes (e.g., 'IndexNumber', 'productivity', 'IndexNumR', 'micEconIndex'). Compared to existing libraries, this package provides greater flexibility for building index numbers in the class of generalized price and quantity indexes.

While there is support for a large number of index-number formulas out-of-the box, the focus is on the tools to easily make and work with any type of generalized price index. No assumptions are made about how data are stored or arranged; rather, the functions in the package are designed to work with atomic vectors, and can be used with R's standard data-manipulation functions for more complex data structures. Compared to existing libraries, this library is suitable for building custom price/quantity indexes, and learning about different types of index-number formulas.

Author(s)

Maintainer: Steve Martin <stevemartin041@gmail.com>

References

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

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

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

See Also

https://github.com/marberts/gpindex

---

change weights — *Change the weights in a generalized mean*

Description

Calculate the weights to turn an r-generalized mean into a k-generalized mean.

Usage

```
weights_change(x, w = rep(1, length(x)), r, k, na.rm = FALSE, scale = TRUE, M)

weights_g2a(x, w = rep(1, length(x)), na.rm = FALSE, scale = TRUE, M)

weights_h2a(x, w = rep(1, length(x)), na.rm = FALSE, scale = TRUE, M)

weights_a2g(x, w = rep(1, length(x)), na.rm = FALSE, scale = TRUE, M)

weights_h2g(x, w = rep(1, length(x)), na.rm = FALSE, scale = TRUE, M)
```

Arguments

`x`	A numeric or logical vector.
`w`	A vector of numeric or logical weights, the same length as `x`. The default is to equally weight each element of `x`.
`r`	A number giving the exponent of the generalized mean associated with `w`. See details.
`k`	A number giving the exponent of the generalized mean for which weights are desired. See details.
`na.rm`	Should missing values be removed when calling `mean_generalized()` and `weights_scale()`?
`scale`	Should the weights be scaled to sum to 1?
`M`	The value of the r-generalized mean, if known.

Details

The function `weights_change()` returns a vector of weights `v` such that

`mean_generalized(x,w,r) = mean_generalized(x,v,k)`.

These weights are calculated as

`M <-mean_generalized(x,w,r)`

`w * logmean_generalized(x,M,r)^(r -1) / logmean_generalized(x,M,k)^(k -1)`.

This generalizes the result in section 4.2 of Balk (2008) when `r` and `k` are 0 or 1, although these are usually the most important cases.

The functions `weights_g2a()` and `weights_h2a()` calculate the weights to turn a geometric and harmonic average into an arithmetic average (i.e., setting `r = 0` and `r = -1` when `k = 1` in `weights_change()`). The functions `weights_a2g()` and `weights_h2g()` are similarly for turning an arithmetic and harmonic average into a geometric average.

As a matter of definition, both `x` and `w` should be strictly positive. This is not enforced here, but the results may not make sense in cases where the generalized mean and generalized logarithmic mean are not defined.

The weights depend on the value of `mean_generalized(x,w,r)`. In many cases this value is known prior to calling the function, and can be supplied to save some computations. Otherwise, it will be calculated from the values of `x` and `w`.

As the return value is the same length as `w`, any NAs in `x` or `w` will return NA. Setting `na.rm = TRUE` simply sets `na.rm = TRUE` in the call to `mean_generalized()`, and `weights_scale()` if `scale = TRUE`.

Value

A numeric vector, the same length as `x`.

References

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

See Also

`mean_generalized` for the generalized mean.

`logmean_generalized` for the generalized logarithmic mean.

`weights_scale` to scale the weights to sum to 1.

Examples

```
# Calculate the geometric mean as an arithmetic mean and harmonic mean by changing the weights

x <- 1:10
mean_geometric(x)
mean_arithmetic(x, weights_g2a(x))
mean_harmonic(x, weights_change(x, r = 0, k = -1))

# Works for nested means, too
```

```
w1 <- runif(10)
w2 <- runif(10)

mean_geometric(c(mean_arithmetic(x, w1), mean_harmonic(x, w2)))

v0 <- weights_g2a(c(mean_arithmetic(x, w1), mean_harmonic(x, w2)))
v1 <- weights_scale(w1)
v2 <- weights_h2a(x, w2)
mean_arithmetic(x, v0[1] * v1 + v0[2] * v2)
```

`factor weights` *Factor the weights in a generalized mean*

Description

Calculate the weights to turn the generalized mean of a product into the product of generalized means.

Usage

```
weights_factor(x, w = rep(1, length(x)), r, na.rm = FALSE, scale = TRUE)

weights_update(x, w = rep(1, length(x)), na.rm = FALSE, scale = TRUE)
```

Arguments

`x`	A numeric or logical vector.
`w`	A vector of numeric or logical weights, the same length as `x`. The default is to equally weight each element of `x`.
`r`	A number giving the exponent of the generalized mean.
`na.rm`	Should missing values be removed when calling `weights_scale()`?
`scale`	Should the weights be scaled to sum to 1?

Details

The function `weights_factor()` returns a vector of weights `v` such that

`mean_generalized(x * y,w,r) == mean_generalized(x,w,r) * mean_generalized(y,v,r)`.

This generalizes the result in section C.5 of Chapter 9 of the PPI Manual for the Young index. Factoring weights with `r = 1` sometimes gets called price-updating weights; `weights_update()` simply calls `weights_factor()` with `r = 1`.

As a matter of definition, both `x` and `w` should be strictly positive. This is not enforced here, but the results may not make sense in cases where the generalized mean is not well defined.

Value

A numeric vector, the same length as `x`.

References

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

See Also

`mean_generalized` for the generalized mean.

`weights_scale` to scale the weights to sum to 1.

Examples

```
# Make some data

x <- 1:10
y <- 11:20
w <- runif(10)

# Calculate the harmonic mean

mean_harmonic(x * y, w)

# The same as

mean_harmonic(x, w) * mean_harmonic(y, weights_factor(x, w, -1))

# The common case of an arithmetic mean

mean_arithmetic(x * y, w)

mean_arithmetic(x, w) * mean_arithmetic(y, weights_update(x, w))

# In cases where x and y have the same order, Chebyshev's inequality
# implies that the chained calculation is too small

mean_arithmetic(x * y, w) > mean_arithmetic(x, w) * mean_arithmetic(y, w)
```

`generalized logarithmic mean`

Generalized logarithmic mean

Description

Calculate a generalized logarithmic mean.

Usage

```
logmean_generalized(a, b, r, tol = .Machine$double.eps^0.5)

logmean(a, b, tol = .Machine$double.eps^0.5)
```

Arguments

`a, b`	A numeric vector.
`r`	A number giving the order of the generalized logarithmic mean.
`tol`	The tolerance used to determine if a == b.

Details

The function `logmean_generalized()` returns the value of the generalized logarithmic mean of `a` and `b` of order `r`. See Bullen (2003, pp. 385, 393) for a precise statement, or `https://en.wikipedia.org/wiki/Stolarsky_mean`.

The function `logmean()` returns the ordinary logarithmic mean, the most useful generalized logarithmic mean, and corresponds to setting `r = 0` in `logmean_generalized()`.

As a matter of definition, both `a` and `b` should be strictly positive. This is not enforced here, but the results may not make sense when the generalized logarithmic mean is not defined.

By definition, the generalized logarithmic mean of `a` and `b` is `a` when `a == b`. The `tol` argument is used to test equality by checking if `abs(a -b) <= tol`. The default value is the same as in `all.equal()`. Setting `tol = 0` will test for exact equality, but can give misleading results in certain applications when `a` and `b` are computed values.

Value

A numeric vector, the same length as `max(length(a),length(b))`.

References

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

Bullen, P. S. (2003). *Handbook of Means and Their Inequalities*. Springer.

See Also

`mean_generalized` for the generalized mean.

`weights_change` uses the generalized logarithmic mean to turn an r-generalized mean into a k-generalized mean.

Examples

```
# The arithmetic and geometric means are special cases of the generalized logarithmic mean

x <- 8:5
y <- 1:4
all.equal(logmean_generalized(x, y, 2), (x + y) / 2)
all.equal(logmean_generalized(x, y, -1), sqrt(x * y))

# A useful identity

all.equal(logmean(x, y) * log(x / y), x - y)

# Works for other orders, too
```

```
r <- 2

all.equal(logmean_generalized(x, y, r) * (r * (x - y))^(1 / (r - 1)),
          (x^r - y^r)^(1 / (r - 1)))

# Some other identities from Bullen (example iii on page 385)

all.equal(logmean_generalized(1, 2, -2),
          (mean_harmonic(1:2) * mean_geometric(1:2)^2)^(1/3))

all.equal(logmean_generalized(1, 2, 0.5),
          (mean_arithmetic(1:2) + mean_geometric(1:2)) / 2)

all.equal(logmean(1, 2),
          mean_geometric(1:2)^2 * logmean(1, 1/2))
```

`generalized mean` *Generalized mean*

Description

Calculate a generalized mean.

Usage

```
mean_generalized(x, w = rep(1, length(x)), r, na.rm = FALSE, scale = TRUE)

mean_arithmetic(x, w = rep(1, length(x)), na.rm = FALSE, scale = TRUE)

mean_geometric(x, w = rep(1, length(x)), na.rm = FALSE, scale = TRUE)

mean_harmonic(x, w = rep(1, length(x)), na.rm = FALSE, scale = TRUE)
```

Arguments

`x`	A numeric or logical vector.
`w`	A vector of numeric or logical weights, the same length as `x`. The default is to equally weight each element of `x`.
`r`	A number giving the exponent of the generalized mean.
`na.rm`	Should missing values in `x` and `w` be removed?
`scale`	Should the weights be scaled to sum to 1?

Details

The function `mean_generalized()` returns the value of the generalized mean of `x` with weights `w` and exponent `r` (i.e., the weighted mean of `x` to the power of `r`, all raised to the power of 1 / `r`). This is also called the power mean or Holder mean. See Bullen (2003, p. 175) for a precise definition, or https://en.wikipedia.org/wiki/Power_mean.

The functions `mean_arithmetic()`, `mean_geometric()`, and `mean_harmonic()` are the most useful generalized means, and correspond to setting `r = 1`, `r = 0`, and `r = -1` in `mean_generalized()`.

As a matter of definition, both `x` and `w` should be strictly positive, especially for the purpose of making a price index. This is not enforced here, but the results may not make sense if the generalized mean in not defined. There are two exceptions to this.

1. The convention in Hardy et al. (1934, p. 13) is used in cases where `x` has zeros: the generalized mean is 0 whenever `w` is strictly positive and `r` < 0.
2. Some authors let `w` be non-negative and sum to 1 (e.g., Sydsaeter et al., 2005, p. 47). If `w` has zeros, then the corresponding element of `x` has no impact on the mean, unless it is missing (unlike `weighted.mean()`).

The weights should almost always be scaled to sum to 1 to satisfy the definition of a generalized mean, although there are certain types of price indexes where the weight should not be scaled (e.g., the Vartia-I index).

The underlying calculation for `mean_generalized()` is mostly identical to `weighted.mean()`, with a few exceptions.

1. Missing values in the weights are not treated differently than missing values in `x`. Setting `na.rm = TRUE` drops missing values in both `x` and `w`, not just `x`. This ensures that certain useful identities are satisfied with missing values in `x`.
2. To speed up execution when there are NAs in `x` or `w`, the return value is always NA whenever `na.rm = FALSE` and `anyNA(x) == TRUE` or `anyNA(w) == TRUE`. This means that NaNs can be handled slightly differently than `weighted.mean()`.

In most cases `mean_arithmetic()` is a drop-in replacement for `weighted.mean()`.

Value

A numeric value.

Note

There are a number of existing functions for calculating *unweighted* geometric and harmonic means, namely the `geometric.mean()` and `harmonic.mean()` functions in the `'psych'` package, the `geomean()` function in the `'FSA'` package, the `GMean()` and `HMean()` functions in the `'DescTools'` package, and the `geoMean()` function in the `'EnvStats'` package.

References

Bullen, P. S. (2003). *Handbook of Means and Their Inequalities*. Springer.

Hardy, G., Littlewood, J. E., and Polya, G. (1934). *Inequalities*. Cambridge University Press.

Lord, N. (2002). Does Smaller Spread Always Mean Larger Product? *The Mathematical Gazette*, 86(506): 273-274.

Sydsaeter, K., Strom, A., and Berck, P. (2005). *Economists' Mathematical Manual* (4th edition). Springer.

See Also

`logmean_generalized` for the generalized logarithmic mean.

`weights_change` can be used to turn an r-generalized mean into a k-generalized mean.

`weights_factor` can be used to factor the weights to a turn a mean of products into a product of means.

Examples

```
# Arithmetic mean

x <- 1:3
w <- c(0.25, 0.25, 0.5)

mean_arithmetic(x, w)
stats::weighted.mean(x, w) # same as stats::weighted.mean

# Geometric mean

mean_geometric(x, w)
prod(x^w) # same as manual calculation

# Using prod to manually calculate the geometric mean can give misleading results

z <- 1:1000
prod(z)^(1 / length(z)) # overflow
mean_geometric(z)

z <- seq(0.0001, by = 0.0005, length.out = 1000)
prod(z)^(1 / length(z)) # underflow
mean_geometric(z)

# Harmonic mean

mean_harmonic(x, w)
1 / stats::weighted.mean(1 / x, w) # same as manual calculation

# Quadratic mean / root mean sqaure

mean_generalized(x, w, r = 2)

# Cubic mean

mean_generalized(x, w, r = 3)

# Example of how missing values are handled
```

```
mean_arithmetic(x, c(0.25, NA, 0.5))
mean_arithmetic(x, c(0.25, NA, 0.5), na.rm = TRUE) # stats::weighted.mean returns NA

# Example from Lord (2002) where the dispersion between arithmetic and geometric means
# decreases as the variance increases

x <- c((5 + sqrt(5)) / 4, (5 - sqrt(5)) / 4, 1 / 2)
y <- c((16 + 7 * sqrt(2)) / 16, (16 - 7 * sqrt(2)) / 16, 1)

stats::sd(x) > stats::sd(y)
mean_arithmetic(x) - mean_geometric(x) < mean_arithmetic(y) - mean_geometric(y)
```

`index weights` *Calculate the weights for a variety of price indexes*

Description

Calculate the weights for a variety of price indexes using information on prices and quantities.

Usage

```
index_weights(p1, p0, q1, q0, type, na.rm = FALSE,
              scale = !is.element(type, c("Vartia1", "MontgomeryVartia")))
```

Arguments

`p1`	Current-period prices.
`p0`	Base-period prices.
`q1`	Current-period quantities.
`q0`	Base-period quantities.
`type`	The name of the index. See details.
`na.rm`	Should missing values be removed when calling `weights_scale()`?
`scale`	Should the weights be scaled to sum to 1?

Details

Weights for the following types of indexes can be calculated.

- Carli / Jevons / Coggeshall
- Dutot
- Laspeyres / Lloyd-Moulton
- Hybrid Laspeyres (for use in a harmonic mean)
- Paasche / Palgrave
- Hybrid Paasche (for use in an arithmetic mean)

- Tornqvist / Unnamed
- Drobish
- Walsh-I (for an arithmetic Walsh index)
- Walsh-II (for a geometric Walsh index)
- Marshall-Edgeworth
- Geary-Khamis
- Montgomery-Vartia / Vartia-I
- Sato-Vartia / Vartia-II
- Lowe
- Young

Naming for the weights follows the CPI/PPI manual first, then Balk (2008) for indexes not listed (or not named) in the CPI/PPI manual. In several cases two or more names correspond to the same weights (e.g., Paasche and Palgrave, or Sato-Vartia and Vartia-II). The explicit calculations are given in the examples.

The weights need not sum to 1, as this normalization isn't always appropriate (i.e., for the Vartia-I weights).

Value

A numeric vector, the same length as either `p0` or `p1`.

Note

Dealing with missing values is cumbersome when making weights, and best avoided. As there are different approaches for dealing with missing values in a price index, missing values should be dealt with prior to calculating weights. Setting `na.rm = TRUE` simply remove missing values when scaling the weights if `scale = TRUE`.

References

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

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

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

See Also

`mean_generalized` for the generalized mean.

`index` provides a wrapper for common indexes.

`weights_scale` to scale the weights to sum to 1.

Examples

```
# Make some data

p0 <- price6[[2]]
p1 <- price6[[3]]
q0 <- quantity6[[2]]
q1 <- quantity6[[3]]
pb <- price6[[1]]
qb <- quantity6[[1]]

# Can be used with the mean_generalized function to make most types of price indexes
# Arithmetic Laspeyres index

mean_arithmetic(p1 / p0, index_weights(p1, p0, q1, q0, type = "Laspeyres"))

# Trivial to turn this into a basket-style index

qs <- index_weights(p1, p0, q1, q0, type = "Laspeyres") / p0
sum(p1 * qs) / sum(p0 * qs)

# Harmonic calculation
mean_harmonic(p1 / p0, index_weights(p1, p0, q1, q0, type = "HybridLaspeyres"))

# Explicit calculation for each of the different weights
# Carli/Jevons/Coggeshall

index_weights(p1, p0, q1, q0, type = "Carli")
rep(1 / length(p0), length(p0))

# Dutot

index_weights(p1, p0, q1, q0, type = "Dutot")
p0 / sum(p0)

# Laspeyres / Lloyd-Moulton

index_weights(p1, p0, q1, q0, type = "Laspeyres")
p0 * q0 / sum(p0 * q0)

# Hybrid Laspeyres

index_weights(p1, p0, q1, q0, type = "HybridLaspeyres")
p1 * q0 / sum(p1 * q0)

# Paasche / Palgrave

index_weights(p1, p0, q1, q0, type = "Paasche")
p1 * q1 / sum(p1 * q1)

# Hybrid Paasche

index_weights(p1, p0, q1, q0, type = "HybridPaasche")
```

```
p0 * q1 / sum(p0 * q1)

# Tornqvist / Unnamed

index_weights(p1, p0, q1, q0, type = "Tornqvist")
0.5 * p0 * q0 / sum(p0 * q0) + 0.5 * p1 * q1 / sum(p1 * q1)

# Drobish

index_weights(p1, p0, q1, q0, type = "Drobish")
0.5 * p0 * q0 / sum(p0 * q0) + 0.5 * p0 * q1 / sum(p0 * q1)

# Walsh-I

index_weights(p1, p0, q1, q0, type = "Walsh1")
p0 * sqrt(q0 * q1) / sum(p0 * sqrt(q0 * q1))

# Marshall-Edgeworth

index_weights(p1, p0, q1, q0, type = "MarshallEdgeworth")
p0 * (q0 + q1) / sum(p0 * (q0 + q1))

# Geary-Khamis

index_weights(p1, p0, q1, q0, type = "GearyKhamis")
p0 / (1 / q0 + 1 / q1) / sum(p0 / (1 / q0 + 1 / q1))

# Montgomery-Vartia / Vartia-I

index_weights(p1, p0, q1, q0, type = "MontgomeryVartia")
logmean(p0 * q0, p1 * q1) / logmean(sum(p0 * q0), sum(p1 * q1))

# Sato-Vartia / Vartia-II

index_weights(p1, p0, q1, q0, type = "SatoVartia")
logmean(p0 * q0 / sum(p0 * q0), p1 * q1 / sum(p1 * q1)) /
sum(logmean(p0 * q0 / sum(p0 * q0), p1 * q1 / sum(p1 * q1)))

# Walsh-II

index_weights(p1, p0, q1, q0, type = "Walsh2")
sqrt(p0 * q0 * p1 * q1) / sum(sqrt(p0 * q0 * p1 * q1))

# Lowe

index_weights(p0 = p0, q0 = qb, type = "Lowe")
p0 * qb / sum(p0 * qb)

# Young

index_weights(p0 = pb, q0 = qb, type = "Young")
pb * qb / sum(pb * qb)
```

`price indexes` *Calculate a variety of price indexes*

Description

Calculate a variety of price indexes using information on prices and quantities.

Usage

```
index_arithmetic(p1, p0, q1, q0, type, na.rm = FALSE)

index_lowe(p1, p0, qb, na.rm = FALSE)

index_young(p1, p0, pb, qb, na.rm = FALSE)

index_geometric(p1, p0, q1, q0, type, na.rm = FALSE)

index_harmonic(p1, p0, q1, q0, type, na.rm = FALSE)

index_fisher(p1, p0, q1, q0, na.rm = FALSE)

index_hlp(p1, p0, q1, q0, na.rm = FALSE)

index_lm(p1, p0, q0, elasticity, na.rm = FALSE)

index_cswd(p1, p0, na.rm = FALSE)

index_cswdb(p1, p0, q1, q0, na.rm = FALSE)

index_bw(p1, p0, na.rm = FALSE)

index_stuval(p1, p0, q1, q0, a, b, na.rm = FALSE)
```

Arguments

`p1`	Current-period prices.
`p0`	Base-period prices.
`q1`	Current-period quantities.
`q0`	Base-period quantities.
`pb`	Period-b prices for the Lowe/Young index.
`qb`	Period-b quantities for the Lowe/Young index.
`type`	The name of the index. See details.
`na.rm`	Should missing values be removed?
`elasticity`	The elasticity of substitution for the Lloyd-Moulton index.
`a, b`	Parameters for the generalized Stuval index.

Details

The functions for the arithmetic, geometric, harmonic, and Lloyd-Moulton indexes are just convenient wrappers for `mean_generalized(p1 / p0,index_weights(p1,p0,q1,q0))`. Together, the arithmetic, geometric, and harmonic index functions can calculate the following indexes.

- **Arithmetic indexes**
- Carli
- Dutot
- Laspeyres
- Palgrave
- Unnamed index (arithmetic analog of the Fisher)
- Drobish
- Walsh-I (arithmetic Walsh)
- Marshall-Edgeworth
- Geary-Khamis
- Lowe
- Young
- **Geometric indexes**
- Jevons
- Geometric Laspeyres
- Geometric Paasche
- Tornqvist
- Montgomery-Vartia / Vartia-I
- Sato-Vartia / Vartia-II
- Walsh-II (geometric Walsh)
- **Harmonic indexes**
- Coggeshall (equally weighted harmonic index)
- Paasche
- Harmonic Laspeyres

In addition to these generalized indexes, there are also functions for calculating a variety of non-generalized indexes. The Fisher index is the geometric mean of the arithmetic Laspeyres and Paasche indexes; the Harmonic Laspeyres Paasche index is the harmonic analog of the Fisher index. The Carruthers-Sellwood-Ward-Dalen and Carruthers-Sellwood-Ward-Dalen-Balk indexes are sample analogs of the Fisher index; the Balk-Walsh index is the sample analog of the Walsh index.

Naming for the indexes follows the CPI/PPI manual first, then Balk (2008) for indexes not listed (or not named) in the CPI/PPI manual.

Value

A numeric value.

Note

Dealing with missing values is cumbersome when making a price index, and best avoided. As there are different approaches for dealing with missing values in a price index, missing values should be dealt with prior to calculating the index.

The approach taken here when `na.rm = TRUE` is to remove price relatives with missing information, either because of a missing price or a missing weight. Certain properties of an index-number formula may not work as expected with missing values, however, if there is ambiguity about how to remove missing values from the weights (as in, e.g., a Tornqvist or Sato-Vartia index).

References

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

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

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

See Also

`mean_generalized` for the generalized mean.

`index_weights` calculates weights for the different types of indexes.

Examples

```
# Most of these indexes can be calculated by combining the
# appropriate weights with the correct type of mean

p0 <- price6[[1]]
p1 <- price6[[2]]
q0 <- quantity6[[1]]
q1 <- quantity6[[2]]

index_geometric(p1, p0, q0 = q0, type = "Laspeyres")
mean_geometric(p1 / p0, index_weights(p1, p0, q0 = q0, type = "Laspeyres"))

# If only weights are available (no quantity information), then use one of the generalized means

w <- c(0.1, 0.2, 0.4, 0.15, 0.10, 0.05)
mean_geometric(p1 / p0, w)

# Chain an index by price updating the weights

p2 <- price6[[3]]
index_arithmetic(p2, p0, q0 = q0, type = "Laspeyres")

I1 <- index_arithmetic(p1, p0, q0 = q0, type = "Laspeyres")
w_pu <- weights_update(p1 / p0, index_weights(p1, p0, q0 = q0, type = "Laspeyres"))
I2 <- mean_arithmetic(p2 / p1, w_pu)
I1 * I2
```

```
# Works for other types of indexes, too

index_harmonic(p2, p0, q0 = q0, type = "Laspeyres")

I1 <- index_harmonic(p1, p0, q0 = q0, type = "Laspeyres")
w_pu <- weights_factor(p1 / p0, index_weights(p1, p0, q0 = q0, type = "Laspeyres"), r = -1)
I2 <- mean_harmonic(p2 / p1, w_pu)
I1 * I2

# Quote contribution for the Tornqvist index

(con <- weights_g2a(p1 / p0, index_weights(p1, p0, q1, q0, "Tornqvist")) * (p1 / p0 - 1))

sum(con)
index_geometric(p1, p0, q1, q0, "Tornqvist") - 1

# Quote contribution for the Fisher index

wl <- index_weights(p1, p0, q1, q0, "Laspeyres")
wp <- index_weights(p1, p0, q1, q0, "HybridPaasche")
wf <- weights_g2a(c(mean_arithmetic(p1 / p0, wl), mean_arithmetic(p1 / p0, wp)))
(con <- (wf[1] * wl + wf[2] * wp) * (p1 / p0 - 1))

sum(con)
index_fisher(p1, p0, q1, q0) - 1

# NAs get special treatment

p0[6] <- NA

# Drops the last price relative

index_arithmetic(p1, p0, q0 = q0, type = "Laspeyres", na.rm = TRUE)

# Only drops the last period-0 price

sum(p1 * q0, na.rm = TRUE) / sum(p0 * q0, na.rm = TRUE)
```

---

price/quantity data — *Sample price/quantity data*

Description

Prices and quantities for six products over five periods.

Usage

```
price6
quantity6
```

Format

Each data frame has 6 rows and 5 columns, with each row corresponding to a product and each column corresponding to a time period.

Note

Adapted from tables 3.1 and 3.2 in Balk (2008), which were adapted from tables 19.1 and 19.2 in the PPI manual.

Source

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

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

Examples

```
# Recreate table 3.6 from Balk (2008)

index_formulas <- function(p1, p0, q1, q0) {
  c(fisher = index_fisher(p1, p0, q1, q0),
    tornqvist = index_geometric(p1, p0, q1, q0, type = "Tornqvist"),
    marshall_edgeworth = index_arithmetic(p1, p0, q1, q0, type = "MarshallEdgeworth"),
    walsh1 = index_arithmetic(p1, p0, q1, q0, type = "Walsh1")
  )
}

round(t(mapply(index_formulas, price6, price6[1], quantity6, quantity6[1])), 4)
```

`scale weights` *Scale weights*

Description

Scale a vector of weights so they sum to 1.

Usage

```
weights_scale(w, na.rm = FALSE)
```

Arguments

`w`	A numeric or logical vector.
`na.rm`	Should missing values be removed?

Details

This function is a simple way to call `w / sum(w)`.

To speed up execution when there are NAs in `w`, the return value is always a vector of NAs whenever `na.rm = FALSE` and `anyNA(w) == TRUE`. This means that NaNs can be handled slightly differently than `w / sum(w)`.

Value

A numeric vector, the same length as `w`.

See Also

`weights_change`, `weights_factor`, and `index_weights` for an application.

Examples

```
w <- 1:4
weights_scale(w)
```

Index