Package 'gpindex'

September 11, 2020

Title Generalized Price and Quantity Indexes

Version 0.2.0

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)

Suggests stats

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 2020-09-11 06:00:06 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_transmute()` 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 packages, 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 packages, this package 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

`generalized mean` *Generalized mean*

Description

Calculate a generalized mean.

Usage

```
mean_generalized(r)

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

`r`	A finite number giving the order of the generalized mean.
`x`	A numeric vector.
`w`	A numeric vector of weights, the same length as `x`. The default is to equally weight each element of `x`.
`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 a function to compute 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 definition, or https://en.wikipedia.org/wiki/Power_mean.

The functions `mean_arithmetic()`, `mean_geometric()`, and `mean_harmonic()` compute the Pythagorean means, the most useful means for making price indexes, and correspond to setting `r = 1`, `r = 0`, and `r = -1` in `mean_generalized()`.

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. (1952, 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 returned by `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

`mean_generalized()` returns a function:

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

`mean_arithmetic()`, `mean_geometric()`, and `mean_harmonic()` each return a numeric value.

Warning

Passing very small values for `r` can give misleading results, and warning is given whenever `abs(r)` is sufficiently small. In general, `r` should not be a computed value.

Note

`mean_generalized()` can be defined on the extended real line, so that `r = -Inf/Inf` returns `min()`/`max()`, to agree with the definition in, e.g., Bullen (2003). This is not implemented, and `r` must be finite.

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 Science+Business Media.

Hardy, G., Littlewood, J. E., and Polya, G. (1952). *Inequalities* (2nd edition). 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_transmute` transforms the weights to turn an r-generalized mean into an s-generalized mean.

`weights_factor` factors the weights to a turn a mean of products into a product of means.

Examples

```
# Make some data

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

# Arithmetic mean

mean_arithmetic(x, w)
stats::weighted.mean(x, w)

# Geometric mean

mean_geometric(x, w)
prod(x^w)

# 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)

# Quadratic mean / root mean square

mean_generalized(2)(x, w)

# Cubic mean

mean_generalized(3)(x, w)

# This is larger than the other means so far because the generalized mean is
# increasing in r

#--------------------

# 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)

#--------------------

# When r < 1, the generalized mean is larger than the corresponding
# counter-harmonic (Lehmer) mean

r <- -1
sum(w * x^r) / sum(w * x^(r - 1)) < mean_generalized(r)(x, w)

# The reverse is true when r > 1

r <- 2
sum(w * x^r) / sum(w * x^(r - 1)) > mean_generalized(r)(x, w)

#--------------------

# Example of how missing values are handled

w <- replace(x, 2, NA)
```

```
mean_arithmetic(x, w)
mean_arithmetic(x, w, na.rm = TRUE)
stats::weighted.mean(x, w, na.rm = TRUE)
```

`logarithmic means` *Generalized logarithmic mean / extended mean*

Description

Calculate a generalized logarithmic mean / extended mean.

Usage

```
mean_extended(r, s)

logmean_generalized(r)

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

Arguments

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

Details

The function `mean_extended()` returns a function to compute the extended mean of `a` and `b` of order `r` and `s`. See Bullen (2003, p. 393) for a definition. This is also called the difference mean, Stolarsky mean, or extended mean-value mean.

The function `logmean_generalized()` returns a function to compute the generalized logarithmic mean of `a` and `b` of order `r`. See Bullen (2003, p. 385) for a definition, or https://en.wikipedia.org/wiki/Stolarsky_mean. The generalized logarithmic mean is a special case of the extended mean, `mean_extended(r,1)`, but is more commonly used for price indexes.

The function `logmean()` returns the ordinary logarithmic mean, and corresponds to `logmean_generalized(1)`.

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 / extended mean is not defined.

By definition, the generalized logarithmic mean / extended 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

`logmean_generalized()` and `mean_extended()` return a function:

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

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

Warning

Passing very small values for `r` or `s` can give misleading results, and warning is given whenever `abs(r)` or `abs(s)` is sufficiently small. Similarly, values for `r` and `s` that are very close in value, but not equal, can give misleading results. In general, `r` and `s` should not be computed values.

Note

`logmean_generalized()` can be defined on the extended real line, so that `r = -Inf/Inf` returns `pmin()`/`pmax()`, to agree with the definition in, e.g., Bullen (2003). This is not implemented, and `r` must be finite.

References

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

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

Stolarsky, K. B. (1975). Generalizations of the Logarithmic Mean. *Mathematics Magazine*, 48(2): 87-92.

See Also

`mean_generalized` for the generalized mean.

`weights_transmute` uses the extended mean to turn an r-generalized mean into an s-generalized mean.

Examples

```
# Make some data

x <- 8:5
y <- 1:4

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

all.equal(logmean_generalized(2)(x, y), (x + y) / 2)
all.equal(logmean_generalized(-1)(x, y), sqrt(x * y))

# The logarithmic mean lies between the arithmetic and geometric means
# because the generalized logarithmic mean is increasing in r

all(logmean(x, y) < (x + y) / 2) & all(logmean(x, y) > sqrt(x * y))
```

```
# It can be approximated as a convex combination of the arithmetic and
# geometric means that gives more weight to the geometric mean

# always a positive approximation error
1 / 3 * (x + y) / 2 + 2 / 3 * sqrt(x * y)
# can have a smaller approximation error
((x + y) / 2)^(1 / 3) * (sqrt(x * y))^(2 / 3)
logmean(x, y)

# A better approximation

correction <- (log(x / y) / pi)^4 / 32
(1 / 3 * (x + y) / 2 + 2 / 3 * sqrt(x * y)) / (1 + correction)

# The harmonic mean cannot be expressed as a logarithmic mean, but can be
# expressed as an extended mean

all.equal(mean_extended(-2, -1)(x, y), 2 / (1 / x + 1 / y))

# The quadratic mean is also a type of extended mean

all.equal(mean_extended(2, 4)(x, y), sqrt(x^2 / 2 + y^2 / 2))

#--------------------

# A useful identity for turning an additive change into a proportionate
# change

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

# Works for other orders, too

r <- 2

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

# Some other identities

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

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

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

#--------------------

# Logarithmic means can be represented as integrals

logmean(2, 3)
```

```
stats::integrate(function(t) 2^(1 - t) * 3^t, 0, 1)$value
1 / stats::integrate(function(t) 1 / (2 * (1 - t) + 3 * t), 0, 1)$value
```

---

price indexes *Calculate a variety of price indexes*

Description

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

Usage

```
index_arithmetic(type)

index_geometric(type)

index_harmonic(type)

index_laspeyres(p1, p0, q0, na.rm = FALSE)

index_paasche(p1, p0, q1, na.rm = FALSE)

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

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

index_young(p1, p0, pb, qb, 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(a, b)

index_weights(type)
```

Arguments

type	The name of the index. See details.

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.
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 arithmetic_mean(), geometric_mean(), and harmonic_mean() functions return a function to calculate a given type of arithmetic, geometric, and harmonic index. Together, these functions produce functions to 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
- Geometric Young
- Tornqvist
- Montgomery-Vartia / Vartia-I
- Sato-Vartia / Vartia-II
- Walsh-II (geometric Walsh)
- **Harmonic indexes**
- Coggeshall (equally weighted harmonic index)

- Paasche
- Harmonic Laspeyres

Along with the `index_lm()` function to calculate the Lloyd-Moulton index, these are just convenient wrappers for `mean_generalized()`.

The Laspeyres, Paasche, Jevons, Lowe, and Young indexes are among the most common price indexes. The `index_laspeyres()`, `index_lowe()`, and `index_young()` functions correspond to setting the appropriate `type` in `index_arithmetic()`; `index_paasche()` and `index_jevons()` instead come from the `index_harmonic()` and `index_geometric()` functions.

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. The `index_stuval()` function returns a function to calculate a Stuval index of the given parameters.

The `index_weights()` function returns a function to calculate weights for a variety of price indexes. 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

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

Naming for the indexes and weights generally 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 calculations are given in the examples.

Value

`index_arithmetic()`, `index_geometric()`, `index_harmonic()`, `index_stuval()`, and `index_weights()` each return a function; the others return 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.

`contributions` for calculating quote contributions.

Examples

```
# Make some data

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

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

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

# Arithmetic Laspeyres index

index_laspeyres(p1, p0, q0)
mean_arithmetic(p1 / p0, index_weights("Laspeyres")(p0, q0))
```

```
# Trivial to turn weights for an arithmetic index
# into a basket-style index

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

# Harmonic calculation for the arithmetic Laspeyres

mean_harmonic(p1 / p0, index_weights("HybridLaspeyres")(p1, q0))

# Unlike its arithmetic counterpart, the geometric Laspeyres can increase
# when base-period prices increase if some of these prices are small

p0_small <- replace(p0, 1, p0[1] / 5)
p0_dx <- replace(p0_small, 1, p0_small[1] + 0.01)
index_geometric("Laspeyres")(p1, p0_small, q0) <
    index_geometric("Laspeyres")(p1, p0_dx, q0)

#--------------------

# Chain an index by price updating the weights

p2 <- price6[[4]]
index_laspeyres(p2, p0, q0)

I1 <- index_laspeyres(p1, p0, q0)
w_pu <- weights_update(p1 / p0, index_weights("Laspeyres")(p0, q0))
I2 <- mean_arithmetic(p2 / p1, w_pu)
I1 * I2

# Works for other types of indexes, too

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

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

#--------------------

# Quote contributions for the Tornqvist index

w <- index_weights("Tornqvist")(p1, p0, q1, q0)
(con <- contributions_geometric(p1 / p0, w))

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

# Quote contributions for the Fisher index

wl <- weights_scale(index_weights("Laspeyres")(p0, q0))
wp <- weights_scale(index_weights("HybridPaasche")(p0, q1))
```

```
wf <- weights_transmute(0, 1)(c(mean_arithmetic(p1 / p0, wl),
                                mean_arithmetic(p1 / p0, wp)))
wf <- weights_scale(wf)
(con <- (wf[1] * wl + wf[2] * wp) * (p1 / p0 - 1))

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

# The same as the decomposition in section 4.2.2 of Balk (2008)

Qf <- index_fisher(q1, q0, p1, p0)
Ql <- index_laspeyres(q1, q0, p0)
con2 <- (Qf / (Qf + Ql) * wl + Ql / (Qf + Ql) * wp) * (p1 / p0 - 1)
all.equal(con, con2)

#--------------------

# NAs get special treatment

p_na <- replace(p0, 6, NA)

# Drops the last price relative

index_laspeyres(p1, p_na, q0, na.rm = TRUE)

# Only drops the last period-0 price

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

#--------------------

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

index_weights("Carli")(p1)
rep(1, length(p0))

# Dutot

index_weights("Dutot")(p0)
p0

# Laspeyres / Lloyd-Moulton

index_weights("Laspeyres")(p0, q0)
p0 * q0

# Hybrid Laspeyres

index_weights("HybridLaspeyres")(p1, q0)
p1 * q0

# Paasche / Palgrave
```

```
index_weights("Paasche")(p1, q1)
p1 * q1

# Hybrid Paasche

index_weights("HybridPaasche")(p0, q1)
p0 * q1

# Tornqvist / Unnamed

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

# Drobish

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

# Walsh-I

index_weights("Walsh1")(p0, q1, q0)
p0 * sqrt(q0 * q1)

# Marshall-Edgeworth

index_weights("MarshallEdgeworth")(p0, q1, q0)
p0 * (q0 + q1)

# Geary-Khamis

index_weights("GearyKhamis")(p0, q1, q0)
p0 / (1 / q0 + 1 / q1)

# Montgomery-Vartia / Vartia-I

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

# Sato-Vartia / Vartia-II

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

# Walsh-II

index_weights("Walsh2")(p1, p0, q1, q0)
sqrt(p0 * q0 * p1 * q1)

# Lowe

index_weights("Lowe")(p0, qb)
p0 * qb
```

```
# Young

index_weights("Young")(pb, qb)
pb * qb
```

`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("Tornqvist")(p1, p0, q1, q0),
    marshall_edgeworth = index_arithmetic("MarshallEdgeworth")(p1, p0, q1, q0),
    walsh1 = index_arithmetic("Walsh1")(p1, p0, q1, q0)
  )
}

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

`transform weights` *Transformations for weights in a generalized mean*

Description

Useful transformations for the weights in a generalized mean.

- Transmute weights to turn an r-generalized mean into an s-generalized mean.
- Factor weights to turn the generalized mean of a product into the product of generalized means.
- Scale weights so they sum to 1.

Usage

```
weights_transmute(r, s)

contributions(r)

contributions_geometric(x, w = rep(1, length(x)))

contributions_harmonic(x, w = rep(1, length(x)))

weights_factor(r)

weights_update(x, w = rep(1, length(x)))

weights_scale(x)
```

Arguments

`r, s`	A number giving the order of the generalized mean. See details.
`x`	A numeric vector.
`w`	A numeric vector of weights, the same length as `x`. The default is to equally weight each element of `x`.

Details

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.

Transmute weights The function `weights_transmute()` returns a function to compute a vector of weights `v(x,w)` such that

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

These weights are calculated as

`v(x,w) = w * mean_extended(r,s)(x,mean_generalized(r)(x,w))^(r -s)`.

This generalizes the result for turning a geometric mean into an arithmetic mean (and vice versa) in section 4.2 of Balk (2008), although this is usually the most important case.

Contributions The function `contributions()` is a simple wrapper for `weights_transmute(r,1)` to calculate (additive) quote contributions for a price index. It returns a function to compute a vector `k(x,w)` such that

`mean_generalized(r)(x,w) -1 == sum(k(x,w))`.

That is, `k(x,w)` gives the additive contribution for each element of `x` in an r-generalized mean. The `contributions_geometric()` and `contributions_harmonic()` functions cover the most important cases. This generalizes the approach for calculating quote contributions in section 4.2 of Balk (2008).

Factor weights The function `weights_factor()` returns a function to compute weights `u(x,w)` such that

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

These weights are calculated as `u(x,w) = w * x^r`.

This generalizes the result in section C.5 of Chapter 9 of the PPI Manual for chaining the Young index, and gives a way to chain generalized price indexes over time. Factoring weights with `r = 1` sometimes gets called price-updating weights; `weights_update()` simply calls `weights_factor(1)`.

Scale weights The function `weights_scale()` scales a vector of weights so they sum to 1 by calling `x / sum(x,na.rm = TRUE)`.

Value

`weights_transmute()`, `contributions()`, and `weights_factor()` return a function:

`function(x,w = rep(1,length(x))`.

`contributions_geometric()`, `contributions_harmonic()`, `weights_update()`, and `weights_scale()` return a numeric vector the same length as `x`.

Note

Transmuting, factoring, and scaling weights will return a value that is the same length as `x`, so any NAs in `x` or `w` will return NA. Unless all values are NA, however, the result for transmuting or factoring will still satisfy the above identities when `na.rm = TRUE` in `mean_generalized()`. Similarly, the result of scaling will sum to 1 when NAs are removed.

References

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.

See Also

`mean_generalized` for the generalized mean.

`mean_extended` for the extended mean.

Examples

```
# Make some data

x <- 2:3
y <- 4:5
w <- runif(2)

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

mean_geometric(x)
mean_arithmetic(x, weights_transmute(0, 1)(x))
mean_harmonic(x, weights_transmute(0, -1)(x))

# Works for nested means, too

w1 <- runif(2)
w2 <- runif(2)

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

v0 <- weights_transmute(0, 1)(c(mean_arithmetic(x, w1),
                                mean_harmonic(x, w2)))
v0 <- weights_scale(v0)
v1 <- weights_scale(w1)
v2 <- weights_scale(weights_transmute(-1, 1)(x, w2))
mean_arithmetic(x, v0[1] * v1 + v0[2] * v2)

#--------------------

# Transmuted weights can be used to calculate quote contributions for,
# e.g., a geometric price index

weights_scale(weights_transmute(0, 1)(x)) * (x - 1)
contributions_geometric(x) # the more convenient way

# Not the only way to calculate contributions

transmute2 <- function(x) {
  m <- mean_geometric(x)
  (m - 1) / log(m) * log(x) / (x - 1) / length(x)
}

transmute2(x) * (x - 1) # this isn't proportional to the method above
all.equal(sum(transmute2(x) * (x - 1)), mean_geometric(x) - 1)

# But these "transmuted" weights don't recover the geometric mean!
# Not a particularly good way to calculate contributions

isTRUE(all.equal(mean_arithmetic(x, transmute2(x)), mean_geometric(x)))

# There are infinitely many ways to calculate contributions, but the weights
```

```
# from weights_transmute(0, 1) are the *unique* weights that recover the
# geometric mean

perturb <- function(w, e) {
  w + c(e, -e) / (x - 1)
}

perturb(transmute2(x), 0.1) * (x - 1)
all.equal(sum(perturb(transmute2(x), 0.1) * (x - 1)),
          mean_geometric(x) - 1)
isTRUE(all.equal(mean_arithmetic(x, perturb(transmute2(x), 0.1)),
                 mean_geometric(x)))

#--------------------

# Factor the harmonic mean by chaining the calculation

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

# 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)
```

Index