Package 'IndexNumR'

October 12, 2022

Type Package

Title Index Number Calculation

Version 0.5.0

Author Graham White

Maintainer Graham White <g.white@unswalumni.com>

Description Computes bilateral and multilateral index numbers.
It has support for many standard bilateral indexes as well as
multilateral index number methods such as GEKS, GEKS-Tornqvist
(or CCDI), Geary-Khamis and the weighted time product dummy
(for details on these methods see Diewert and Fox (2020)
<doi:10.1080/07350015.2020.1816176>).
It also supports updating of multilateral indexes using
several splicing methods.

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

URL https://github.com/grahamjwhite/IndexNumR

Depends R (>= 3.5.0)

Imports utils

Suggests testthat, knitr, rmarkdown, covr, tidyr

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2022-02-07 11:50:02 UTC

R topics documented:

`CESData` *Generate data assuming CES preferences*

Description

This function is useful for generating datasets that can be used for testing where the 'true' price index is known. The data are constructed using assumed prices and total expenditure in each period. Expenditure shares and quantities are then computed assuming CES preferences. For further details, see the references.

Usage

```
CESData(sigma)
```

Arguments

sigma	the elasticity of substitution parameter

Value

a dataframe containing time period, prices, quantities and product identifier.

References

W.E. Diewert and K.J. Fox (2017), "Substitution Bias in Multilateral Methods for CPI Construction Using Scanner Data", Discussion Paper 17-02, Vancouver School of Economics, The University of British Columbia.

Examples

```
## Not run:
# generate data assuming the elasticity of substitution is 2
CESData(2)

## End(Not run)
```

CES_sigma_2 *Dataset of prices and quantities on four products*

Description

A constructed dataset containing the prices and quantities of four products over a twelve month period, assuming CES preferences.

Usage

```
CES_sigma_2
```

Format

A data frame with 48 rows and 4 columns:

time time period

prices constructed prices

quantities constructed quantities

prodID product identifier

Source

Computed using procedure in W.E. Diewert and K.J. Fox (2017), "Substitution Bias in Multilateral Methods for CPI Construction Using Scanner Data", Discussion Paper 17-02, Vancouver School of Economics, The University of British Columbia.

`dominicksData` *Get data from the Dominicks dataset*

Description

The Dominicks Scanner data, provided by the University of Chicago Booth School of Business, contains around 5 years of product-level data from over 100 stores, collected from 1989-1994. The data consist of a UPC file that contains information on the products, and a movement file that contains the information on prices and sales. For a complete description of the data, see Dominicks data website and the Dominicks data user manual. This function downloads and merges the movement and UPC files, then merges the result with data detailing the dates of each of the weeks in the movement file.

Usage

```
dominicksData(x, movementcsv = NULL, UPCcsv = NULL)
```

Arguments

`x`	the name of the category to retrieve, see details for list.
`movementcsv`	the path to the movement csv file for one product category. The default is NULL, which downloads the file from the website.
`UPCcsv`	the path to the UPC csv file for one product category. The default is NULL, which downloads the file from the website.

Details

The following transformations are performed on the data:

- The quantity variable is set to MOVE, which is the number of individual units sold
- The price variable is set to PRICE/QTY, which is the unit price. This accounts for the fact that sometimes products are sold in bundles (e.g., two-for-one promotions).
- expenditure is given by PRICE*MOVE/QTY.
- All observations where the variable OK equals 0, or price is less than or equal to 0, are dropped.

If you have already downloaded the movement and UPC csv files for a category from the website, then you can pass the file paths of those files to the function and just have it combine them with the weeks dataset. The default is to download the files for you from the website.

The products available are:

- Analgesics
- Bath Soap
- Beer
- Bottled Juices

- Cereals
- Cheeses
- Cigarettes
- Cookies
- Crackers
- Canned Soup
- Dish Detergent
- Front-end-candies
- Frozen Dinners
- Frozen Entrees
- Frozen Juices
- Fabric Softeners
- Grooming Products
- Laundry Detergents
- Oatmeal
- Paper Towels
- Refrigerated Juices (not currently available)
- Soft Drinks
- Shampoos
- Snack Crackers
- Soaps
- Toothbrushes
- Canned Tuna
- Toothpastes
- Bathroom Tissues

References

James M. Kilts Center, University of Chicago Booth School of Business

Examples

```
## Not run:
analgesics <- dominicksData("Analgesics")

## End(Not run)
```

DominicksWeeks *Date information for the Dominicks data*

Description

Table from the Dominicks Data Manual, that gives the start and end date of each of the weeks in the movement files.

Usage

```
DominicksWeeks
```

Format

A data frame with 400 rows and 4 columns:

week the number of the week

start date the week started

end date the week ended

specialEvents special events, such as Halloween, that occurred during the week

Source

Dominicks Data Manual, Chicago Booth Kilts Center for Marketing, 2018, pages 21-28.

elasticity *Computes the elasticity of substitution*

Description

A function to estimate the elasticity of substitution

Usage

```
elasticity(
  x,
  pvar,
  qvar,
  pervar,
  prodID,
  compIndex = "ces",
  lower = -20,
  upper = 20
)
```

Arguments

x	A dataframe
pvar	A character string for the name of the price variable
qvar	A character string for the name of the quantity variable
pervar	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
prodID	A character string for the name of the product identifier
compIndex	The index number with which the CES index will be equated to calculate the elasticity. Acceptable options are lloydmoulton, fisher or satovartia. The lloyd-moulton option equates the 'base period' lloyd-moulton index with the 'current period' lloyd-moulton index.
lower	lower limit to search for sigma.
upper	upper limit to search for sigma.

Value

A list with three elements: sigma (the average elasticity over all time periods); allsigma (a T-1 by 1 matrix of the estimated elasticities for each time period, except period one); and diff (the value of the difference between the two indexes, check this is zero for all time periods).

Examples

```
elasticity(CES_sigma_2,pvar="prices",qvar="quantities",pervar="time",
prodID = "prodID")
```

---

evaluateMatched *Evaluate product overlap between periods*

Description

Evaluate the counts and expenditure for each period with and without matching items across periods.

Usage

```
evaluateMatched(x, pvar, qvar, pervar, prodID, output = "chained")
```

Arguments

x	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
pvar	A character string for the name of the price variable
qvar	A character string for the name of the quantity variable

pervar	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
prodID	A character string for the name of the product identifier
output	A character string specifying whether the matching should be done assuming a chained index or a fixed base index. No index is actually computed, but the matching needs to know which periods are being compared. Default is chained.

Value

A list of two matrices, one for expenditures and one for counts. The first four columns present the base period information base_index (the base time period), base (base period expenditure or count), base_matched (the expenditure or count of the base period after matching), base_share (share of total expenditure in the base period that remains after matching). Columns 5-8 are defined analogously for the current period. The matched numbers for the base period should be interpreted as the count or expenditure that remains after removal of products that exist in the base period, but not in the current period. That is, products that existed in the base period but no longer exist in the current period are removed by the matching. If new products exist in the current period that were not available in the base period, this does not affect the matched base period expenditure or count. The appearance of new products is captured in the current period matched expenditure and counts. Therefore, a base period share that is less than 1 indicates that products have disappeared, while a current period share less than 1 indicates that new products have appeared.

The count matrix has two additional columns, "new" and "leaving". The new column gives the number of products that exist in the current period but not the base period. The leaving column gives the count of products that exist in the base period but not the current period. Matching removes both of these types of products.

Examples

```
# create CES_sigma_2 dataset removing the observation in time period 4
# on product 1
df <- CES_sigma_2[!(CES_sigma_2$time==4 & CES_sigma_2$prodID==1),]
# evaluate the overlap between periods for this dataset assuming
# a chained index
evaluateMatched(df, pvar="prices", qvar="quantities", pervar="time",
prodID = "prodID", output="chained")
```

GEKSIndex *Compute a GEKS multilateral index*

Description

A function to calculate a GEKS multilateral price index

Usage

```
GEKSIndex(
  x,
  pvar,
  qvar,
  pervar,
  indexMethod = "tornqvist",
  prodID,
  sample = "matched",
  window = 13,
  splice = "mean",
  biasAdjust = FALSE,
  weights = "average",
  intGEKS = FALSE,
  imputePrices = NULL
)
```

Arguments

x	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
pvar	A character string for the name of the price variable
qvar	A character string for the name of the quantity variable
pervar	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
indexMethod	A character string to select the index number method. Valid index number methods are fisher, tornqvist, tpd, jevons or walsh. The default is tornqvist.
prodID	A character string for the name of the product identifier
sample	A character string specifying whether matching is to be performed. The default is to use matching. If sample=matched then any products that are not present in comparison periods are removed prior to estimating the index for those periods.
window	An integer specifying the length of the window.
splice	A character string specifying the splicing method. Valid methods are window, movement, half, mean, fbew or fbmw, wisp, hasp or mean_pub. The default is mean. See details for important considerations when using fbew and fbmw.
biasAdjust	whether to adjust for bias in the coefficients of the bilateral TPD index. The default is FALSE because making this adjustment will break transitivity of the GEKS index.
weights	the type of weighting for the bilateral TPD index. Options are "unweighted" to use ordinary least squares, "shares" to use weighted least squares with expenditure share weights, and "average" to use weighted least squares with the average of the expenditure shares over the two periods. See details for more information
intGEKS	whether to estimate the intersection GEKS method. This method performs additional product matching over the sample = "matched" option. See Lamboray and Krsinich 2015 for more information.

`imputePrices` the type of price imputation to use for missing prices. Currently only "carry" is supported to used carry-forward/carry-backward prices. Default is NULL to not impute missing prices.

Details

The splicing methods are used to update the price index when new data become available without changing prior index values. The window, movement, half and mean splices use the most recent index value as the base period, which is multiplied by a price movement computed using new data. The fbew (Fixed Base Expanding Window) and fbmw (Fixed Base Moving Window) use a fixed base onto which the price movement using new data is applied. The base period is updated periodically. IndexNumR calculates which periods are the base periods using `seq(from = 1, to = n, by = window - 1)`, so the data must be set up correctly and the right window length chosen. For example, if you have monthly data and want December of each year to be the base period, then the first period in the data must be December and the window must be set to 13.

References

Ivancic, L., W.E. Diewert and K.J. Fox (2011), "Scanner Data, Time Aggregation and the Construction of Price Indexes", Journal of Econometrics 161, 24-35.

Lamboray, C. and F. Krsinich (2015), "A Modification of the GEKS Index When Product Turnover is High", Paper presented at the fourteenth Ottawa Group meeting, 20-22 May 2015, Tokyo, Japan.

Examples

```
# compute a GEKS mutlilateral index with mean splicing
GEKSIndex(CES_sigma_2, pvar = "prices", qvar = "quantities", pervar = "time",
prodID = "prodID", indexMethod = "tornqvist", window=11, splice = "mean")

# compute a GEKS multilateral index with window splicing and the Fisher index method
GEKSIndex(CES_sigma_2, pvar = "prices", qvar = "quantities", pervar = "time",
prodID = "prodID", indexMethod = "fisher", window=11, splice = "mean")
```

GKIndex *Compute the Geary-Khamis index*

Description

Compute the Geary-Khamis index

Usage

```
GKIndex(
  x,
  pvar,
  qvar,
  pervar,
```

```
  prodID,
  sample = "",
  window,
  splice = "mean",
  imputePrices = NULL,
  solveMethod = "inverse",
  tolerance = 1/1e+12,
  maxIter = 100
)
```

Arguments

x	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
pvar	A character string for the name of the price variable
qvar	A character string for the name of the quantity variable
pervar	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
prodID	A character string for the name of the product identifier
sample	set to "matched" to only use products that occur across all periods in a given window. Default is not to match.
window	An integer specifying the length of the window.
splice	the splicing method to use to extend the index. Valid methods are window, movement, half, mean, fbew, fbmw, wisp, hasp or mean_pub. The default is mean. See details for important considerations when using fbew and fbmw.
imputePrices	the type of price imputation to use for missing prices. Currently only "carry" is supported to used carry-forward/carry-backward prices. Default is NULL to not impute missing prices.
solveMethod	the method to use to solve for the quality adjustment factors and the price levels. "inverse" uses a matrix inverse operation, is much more efficient, but may not work if there are many missing observations. "iterative" iterates between the equations for the quality adjustment factors and price levels and is much slower, but can be used even when there are a large number of missing observations.
tolerance	the tolerance for the iterative solving method. Smaller numbers will produce more accurate results, but take more iterations. Default is 1/1e12, which may be a little larger than machine precision, given by .Machine$double.eps.
maxIter	the maximum number of iterations for the iterative solving method.

Details

The splicing methods are used to update the price index when new data become available without changing prior index values. The window, movement, half and mean splices use the most recent index value as the base period, which is multiplied by a price movement computed using new data. The fbew (Fixed Base Expanding Window) and fbmw (Fixed Base Moving Window) use a fixed base onto which the price movement using new data is applied. The base period is updated

periodically. IndexNumR calculates which periods are the base periods using `seq(from = 1, to = n, by = window - 1)`, so the data must be set up correctly and the right window length chosen. For example, if you have monthly data and want December of each year to be the base period, then the first period in the data must be December and the window must be set to 13.

It is recommended to use the matrix inverse method of solving the GK equations (the default) because the performance difference can be significant. If the matrix inverse method does not work then switch to the iterative method. The tolerance and maximum number of iterations in the iterative method can be adjusted to balance performance and precision.

References

Ivancic, L., W.E. Diewert and K.J. Fox (2011), "Scanner Data, Time Aggregation and the Construction of Price Indexes", Journal of Econometrics 161, 24-35.

Geary, R. G. 1958. “A Note on Comparisons of Exchange Rates and Purchasing Power Between Countries.” Journal of the Royal Statistical Society Series A 121: 97–99.

Khamis, S. H. 1970. “Properties and Conditions for the Existence of a New Type of Index Number.” Sankhya: The Indian Journal of Statistics, Series B (1960-2002) 32: 81–98.

Examples

```
# compute a Geary-Khamis index with mean splicing
GKIndex(CES_sigma_2, pvar = "prices", qvar = "quantities", pervar = "time",
prodID = "prodID", window=11, splice = "mean")
```

`groupIndexes` *Calculate price indexes for product groups*

Description

Calculate price indexes for product groups

Usage

```
groupIndexes(group, indexFunction, indexArgs)
```

Arguments

`group`	the name of the variable containing the group ID. This must be a factor variable, or a variable coercible to a factor.
`indexFunction`	the name of the function to use to calculate the index as a string. Available options are ‘priceIndex‘, ‘GEKSIndex‘, ‘GKIndex‘, ‘WTPDIndex‘.
`indexArgs`	arguments for the price index function as a named list. All arguments must be named.

Value

a list of indexes, one for each group

Examples

```
df <- CES_sigma_2
df$groupID <- c(rep(1, 24), rep(2, 24))

argsList <- list(x = df, pvar = "prices", qvar = "quantities", pervar = "time",
prodID = "prodID", indexMethod = "fisher", output = "chained")

groupIndexes("groupID", "priceIndex", argsList)
```

---

imputeCarryPrices *Fill all missing prices with carry forward/backward prices*

Description

If a missing product has a previous price then that previous price is carried forward until the next real observation. If there is no previous price then the next real observation is found and carried backward. If a price observation is filled, and a quantity variable is specified, then the corresponding quantity is set to zero. Prices can be filled with no quantity variable by specifying qvar = "".

Usage

```
imputeCarryPrices(x, pvar, qvar, pervar, prodID)
```

Arguments

x	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
pvar	A character string for the name of the price variable
qvar	A character string for the name of the quantity variable. If there is no quantity variable you must specify qvar = "".
pervar	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
prodID	A character string for the name of the product identifier

Value

the input data frame with missing observations filled

Examples

```
# create a dataset with missing prices for products 1 and 2
df <- CES_sigma_2[-c(1,2,14,15),]
imputeCarryPrices(df, "prices", "quantities", "time", "prodID")
```

`imputeQuantities` *Impute quantities when only prices are available*

Description

This procedure calculates quantities in such a way that the expenditure shares on all products are equal in each period. It is used to compute quantities for the predicted share measure of relative price dissimilarity when there are none available.

Usage

```
imputeQuantities(x, pvar, pervar, prodID)
```

Arguments

`x`	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
`pvar`	A character string for the name of the price variable
`pervar`	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
`prodID`	A character string for the name of the product identifier

`IndexNumR` *IndexNumR: A package for computing index numbers*

Description

IndexNumR is a package for computing bilateral and multilateral index numbers. The package has been designed with performance in mind, to enable computing index numbers on large datasets within a reasonable timeframe. It also aims to make a large number of index number methods available, along with access to datasets to enable research and experimentation.

Author

Graham White

Notes

I'd like to thank all those that have commented on, or tested the code so that it could be improved. In particular, I'd like to thank Professor Kevin Fox at the University of New South Wales for his support and input.

Some function parameters can have a considerable impact on the outputs, so it is recommended that the user read the documentation for these functions carefully.

Vignettes

There is very detailed information about the functions in the package vignette, which can be accessed with,

`browseVignettes("IndexNumR")`.

Bilateral index functions

Compute bilateral indexes

- priceIndex
- quantityIndex

Similarity chain linking

Compute dissimilarity measures or chain links.

- relativeDissimilarity
- mixScaleDissimilarity
- maximumSimilarityLinks

Multilateral index functions

Compute multilateral indexes

- GEKSIndex
- GKIndex
- WTPDIndex

Other index number functions

- groupIndexes
- yearOverYearIndexes

Data preparation functions

Perform various operations on the data before using other functions, such as index number functions.

- unitValues
- imputeCarryPrices
- imputeQuantities

Data exploration functions

Learn more about the characteristics of your dataset.

- evaluateMatched
- values

Sample data

IndexNumR has one sample dataset,

- CES_sigma_2,

and a function for generating small datasets,

- CESData,

and a function for accessing the Dominicks Finer Foods scanner data,

- dominicksData.

Differences approach to index numbers

These functions are referred to as indicators, to distinguish them from the bilateral and multilateral index functions which use the ratio approach.

- priceIndicator
- quantityIndicator

Time index functions

Index functions in IndexNumR generally need a time period variable. These functions will compute the required time period variable, depending on the frequency required.

- weekIndex
- monthIndex
- quarterIndex
- yearIndex

---

maximumSimilarityLinks

Finds periods to link using minimum dissimilarity.

Description

Function to compute the maximum similarity chain links from a measure of dissimilarity. The procedure works as described in Diewert and Fox (2017). It first links period 2 to period 1. Then for each period t, from periods 3,...,T it searches among the periods 1,...,t-1 for the period that is most similar (least dissimilar) to period t.

Usage

```
maximumSimilarityLinks(x)
```

Arguments

x	a matrix containing a dissimilarity measure where the first two columns are the indices and the third column is the dissimilarity measure.

Examples

```
# find the linking periods in the CES_sigma_2 dataset that maximise
# the similarity between periods, using the absolute dissimilarity measure.
disMat <- mixScaleDissimilarity(CES_sigma_2, pvar = "prices", qvar = "quantities",
pervar = "time", prodID = "prodID", measure = "absolute",
combine = "geomean")
maximumSimilarityLinks(disMat)
```

---

mixScaleDissimilarity *Computes mix, scale and absolute dissimilarity measures*

Description

This is a function to compute the Fox, Hill and Diewert 2004 dissimilarity measures.

Usage

```
mixScaleDissimilarity(
  x,
  pvar,
  qvar,
  prodID,
  pervar,
  measure = "absolute",
  combine = "geomean"
)
```

Arguments

x	A dataframe
pvar	string identifying the price variable in x
qvar	string identifying the quantity variable in x
prodID	string identifying the product id variable in x
pervar	string identifying the time period variable in x
measure	choice of dissimilarity measure. Valid options are mix, scale or absolute.
combine	specifies how to combine the price and quantity vectors. "stack" stacks the price and quantity vectors, "geomean" computes separate dissimilarity measures for prices and quantities then takes the geometric mean of these.

Value

A matrix where the first two columns are the possible combinations of periods and the third column is the dissimilarity measure.

References

Fox, K.J., R.J. Hill and W.E. Diewert (2004), "Identifying outliers in multi-output models", Journal of Productivity Analysis, 22, 73-94, 2004.

Examples

```
# estimate the dissimilarity between periods in the CES_sigma_2 dataset
# using the absolute measure of dissimilarity and the geometric mean
# to combine price and quantity information.
mixScaleDissimilarity(CES_sigma_2, pvar = "prices", qvar = "quantities",
pervar = "time", prodID = "prodID", measure = "absolute",
combine = "geomean")
```

---

monthIndex *Generate an index of months*

Description

A function to create a month index variable

Usage

```
monthIndex(x, overlapWeeks = "naive")
```

Arguments

x	A vector or column of dates
overlapWeeks	Tells monthIndex how to deal with weeks that cross over two adjacent months. Options are "naive", "majority", "wholeOnly" or "fourWeek". "naive" simply takes the month number of the observation, ignoring where the week of that observation falls. "majority" will allocate the observation to the month that owns the majority of days in that week, assuming that Monday is day one of the week. "fourWeek" first calculates a week index, then calculates the month index assuming that there are four weeks in each month. "wholeOnly" will return NA for any dates falling inside a week that overlaps two adjacent months; that is, only weeks that are wholly within a month are given an index value. The default is "naive".

Examples

```
# given a vector of dates
df <- data.frame(date = as.Date(c("2017-01-01","2017-02-01","2017-03-01","2017-04-01"),
format = "%Y-%m-%d"))
# calculate the time period variable
df$period <- monthIndex(df$date, overlapWeeks = "naive")
df
```

`predictedShares` *Predicted shares for predicted share relative price dissimilarity*

Description

You should not need to call this function directly unless the shares themselves are of interest. Other functions will call this function internally.

Usage

```
predictedShares(x, pvar, qvar, pervar, prodID)
```

Arguments

`x`	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
`pvar`	A character string for the name of the price variable
`qvar`	A character string for the name of the quantity variable. For elementary indexes a quantity variable is not required for the calculations and you must specify qvar = "".
`pervar`	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
`prodID`	A character string for the name of the product identifier

Value

a list of matrices

`priceIndex` *Computes a bilateral price index*

Description

A function to compute a price index given data on products over time

Usage

```
priceIndex(
  x,
  pvar,
  qvar,
  pervar,
  indexMethod = "laspeyres",
  prodID,
  sample = "matched",
  output = "pop",
  chainMethod = "pop",
  sigma = 1.0001,
  basePeriod = 1,
  biasAdjust = TRUE,
  weights = "average",
  loweYoungBase = 1,
  imputePrices = NULL,
  ...
)
```

Arguments

`x`	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
`pvar`	A character string for the name of the price variable
`qvar`	A character string for the name of the quantity variable. For elementary indexes a quantity variable is not required for the calculations and you must specify qvar = "".
`pervar`	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
`indexMethod`	A character string to select the index number method. Valid index number methods are dutot, carli, jevons, laspeyres, paasche, fisher, cswd, harmonic, tornqvist, satovartia, walsh, CES, geomLaspeyres, geomPaasche, tpd, Geary-Khamis (gk), drobish, palgrave, stuvel, marshalledgeworth.
`prodID`	A character string for the name of the product identifier
`sample`	A character string specifying whether a matched sample should be used.

output	A character string specifying whether a chained (output="chained") , fixed base (output="fixedbase") or period-on-period (output="pop") price index numbers should be returned. Default is period-on-period.
chainMethod	A character string specifying the method of chain linking to use if the output option is set to "chained". Valid options are "pop" for period-on-period, and similarity chain linked options "plspread" for the Paasche-Laspeyres spread, "asymplinear" for weighted asymptotically linear, "logquadratic" for the weighted log-quadratic, and "mixScale" for the mix, scale or absolute dissimilarity measures, or "predictedshare" for the predicted share relative price dissimilarity. The default is period-on-period. Additional parameters can be passed to the mixScaleDissimilarity function using . . .
sigma	The elasticity of substitution for the CES index method.
basePeriod	The period to be used as the base when 'fixedbase' output is chosen. Default is 1 (the first period).
biasAdjust	whether to adjust for bias in the coefficients in the bilateral TPD index. The default is TRUE.
weights	the type of weighting for the bilateral TPD index. Options are "unweighted" to use ordinary least squares, "shares" to use weighted least squares with expenditure share weights, and "average" to use weighted least squares with the average of the expenditure shares over the two periods.
loweYoungBase	the period used as the base for the lowe or young type indexes. The default is period 1. This can be a vector of values to use multiple periods. For example, if the data are monthly and start in January, specifying 1:12 will use the first twelve months as the base.
imputePrices	the type of price imputation to use for missing prices. Currently only "carry" is supported to used carry-forward/carry-backward prices. Default is NULL to not impute missing prices.
...	this is used to pass additional parameters to the mixScaleDissimilarity function.

Examples

```
# period-on-period Laspeyres index for the CES_sigma_2 dataset
priceIndex(CES_sigma_2, pvar="prices", qvar="quantities", pervar="time",
prodID = "prodID", indexMethod = "laspeyres")

# chained Fisher index
priceIndex(CES_sigma_2, pvar="prices", qvar="quantities", pervar="time",
prodID = "prodID", indexMethod = "fisher", output="chained")

# chained Tornqvist index, with linking periods chosen by the
# weighted log-quadratic dissimilarity measure
priceIndex(CES_sigma_2, pvar="prices", qvar="quantities", pervar="time",
prodID = "prodID", indexMethod = "tornqvist", output="chained",
chainMethod = "logquadratic")
```

`priceIndicator` *Calculate a price indicator*

Description

This calculates a price indicator. This is calculated using the differences approach to index number theory, where the change in prices and quantities from one period to the next is additive. Therefore, the change in total value is the sum of the change in prices and the change in quantities. Such a value decomposition can be obtained using `valueDecomposition`.

See the vignette for more information on the calculations.

```
vignette(topic = "indexnumr", package = "IndexNumR")
```

Usage

```
priceIndicator(x, pvar, qvar, pervar, prodID, method, sample = "matched")
```

Arguments

`x`	data frame with input data
`pvar`	character string for the name of the price column
`qvar`	character string for the name of the quantity column
`pervar`	character string for the name of the time period variable
`prodID`	character string for the name of the product ID column
`method`	character string for the indicator method. Valid options are "laspeyres", "paasche", "bennet", or "montgomery".
`sample`	whether to use a matched sample (sample = "matched")

Value

an nx1 matrix containing the indicator

Examples

```
# compute a price indicator using the Montgomery method
priceIndicator(CES_sigma_2, pvar = "prices", qvar = "quantities",
prodID = "prodID", pervar = "time", method = "montgomery")
```

productChanges *Product ID's for appearing/disappearing products*

Description

This function will give the product ID's of products that appear or disappear in each period.

Usage

```
productChanges(x, pervar, prodID)
```

Arguments

x	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
pervar	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
prodID	A character string for the name of the product identifier

Value

a list containing one element for each time period, each element of which contains two vectors (one for appearing products, and one for disappearing products)

Examples

```
# create a dataset with some missing products
df <- CES_sigma_2[-c(3,4,15),]

# show the products that changed
productChanges(df, "time", "prodID")
```

quantityIndex *Computes a bilateral quantity index*

Description

A function to compute a quantity index given data on products over time

Usage

```
quantityIndex(
  x,
  pvar,
  qvar,
  pervar,
  indexMethod = "laspeyres",
  prodID,
  sample = "matched",
  output = "pop",
  chainMethod = "pop",
  sigma = 1.0001,
  basePeriod = 1,
  biasAdjust = TRUE,
  weights = "average",
  loweYoungBase = 1,
  imputePrices = NULL,
  ...
)
```

Arguments

x	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
pvar	A character string for the name of the price variable
qvar	A character string for the name of the quantity variable. For elementary indexes a quantity variable is not required for the calculations and you must specify qvar = "".
pervar	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
indexMethod	A character string to select the index number method. Valid index number methods are dutot, carli, jevons, laspeyres, paasche, fisher, cswd, harmonic, tornqvist, satovartia, walsh, CES, geomLaspeyres, geomPaasche, tpd, Geary-Khamis (gk), drobish, palgrave, stuvel, marshalledgeworth.
prodID	A character string for the name of the product identifier
sample	A character string specifying whether a matched sample should be used.
output	A character string specifying whether a chained (output="chained") , fixed base (output="fixedbase") or period-on-period (output="pop") price index numbers should be returned. Default is period-on-period.
chainMethod	A character string specifying the method of chain linking to use if the output option is set to "chained". Valid options are "pop" for period-on-period, and similarity chain linked options "plspread" for the Paasche-Laspeyres spread, "asymplinear" for weighted asymptotically linear, "logquadratic" for the weighted log-quadratic, and "mixScale" for the mix, scale or absolute dissimilarity measures,

	or "predictedshare" for the predicted share relative price dissimilarity. The default is period-on-period. Additional parameters can be passed to the mixScaleDissimilarity function using . . .
sigma	The elasticity of substitution for the CES index method.
basePeriod	The period to be used as the base when 'fixedbase' output is chosen. Default is 1 (the first period).
biasAdjust	whether to adjust for bias in the coefficients in the bilateral TPD index. The default is TRUE.
weights	the type of weighting for the bilateral TPD index. Options are "unweighted" to use ordinary least squares, "shares" to use weighted least squares with expenditure share weights, and "average" to use weighted least squares with the average of the expenditure shares over the two periods.
loweYoungBase	the period used as the base for the lowe or young type indexes. The default is period 1. This can be a vector of values to use multiple periods. For example, if the data are monthly and start in January, specifying 1:12 will use the first twelve months as the base.
imputePrices	the type of price imputation to use for missing prices. Currently only "carry" is supported to used carry-forward/carry-backward prices. Default is NULL to not impute missing prices.
...	this is used to pass additional parameters to the mixScaleDissimilarity function.

Examples

```
# chained Fisher quantity index for the CES_sigma_2 dataset
quantityIndex(CES_sigma_2, pvar="prices", qvar="quantities", pervar="time",
prodID = "prodID", indexMethod = "fisher", output="chained")
```

---

quantityIndicator *Compute a quantity indicator*

Description

This calculates a quantity indicator. This is calculated using the differences approach to index number theory, where the change in prices and quantities from one period to the next is additive. Therefore, the change in total value is the sum of the change in prices and the change in quantities. Such a value decomposition can be obtained using `valueDecomposition`.

See the vignette for more information on the calculations.

```
vignette(topic = "indexnumr", package = "IndexNumR")
```

Usage

```
quantityIndicator(x, pvar, qvar, pervar, prodID, method, sample = "matched")
```

Arguments

x	data frame with input data
pvar	character string for the name of the price column
qvar	character string for the name of the quantity column
pervar	character string for the name of the time period variable
prodID	character string for the name of the product ID column
method	character string for the quantity indicator method. Valid options are "laspeyres", "paasche", "bennet", or "montgomery".
sample	whether to use a matched sample (sample = "matched")

Value

an nx1 matrix containing the indicator

Examples

```
# compute a quantity indicator using the Bennet method
quantityIndicator(CES_sigma_2, pvar = "prices", qvar = "quantities",
prodID = "prodID", pervar = "time", method = "bennet")
```

---

quarterIndex *Generate an index of quarters*

Description

A function to create a quarter index variable

Usage

```
quarterIndex(x)
```

Arguments

x	A vector or column of dates

Examples

```
# given a vector of dates
df <- data.frame(date = as.Date(c("2017-01-01","2017-04-01","2017-07-01","2017-08-01"),
format = "%Y-%m-%d"))
# calculate the time period variable
df$period <- quarterIndex(df$date)
df
```

`relativeDissimilarity` *Computes measures of relative dissimilarity between all periods*

Description

A function to compute the relative price dissimilarity between two vectors of prices.

Usage

```
relativeDissimilarity(
  x,
  pvar,
  qvar,
  pervar,
  prodID,
  indexMethod = "fisher",
  similarityMethod = "logquadratic"
)
```

Arguments

`x`	A dataframe containing price, quantities, a time period index and a product identifier.
`pvar`	A string identifying the price variable.
`qvar`	A string identifying the quantity variable.
`pervar`	A string identifying the time index variable.
`prodID`	A string identifying the product ID.
`indexMethod`	A string identifying the index method to use in the calculation. Not relevant for similarityMethod = PLSpread. Supported methods are fisher and tornqvist. Default is Fisher.
`similarityMethod`	A string specifying the formula for calculating the relative dissimilarity. Valid options are logquadratic, asymplinear, PLSpread and predictedshare. Default is logquadratic.

Value

A matrix of dissimilarity measures. The first two columns are the possible combinations of bilateral comparisons and the third column is the dissimilarity measure.

References

Diewert, W.E. (2002). "Similarity and Dissimilarity Indexes: An Axiomatic Approach" Discussion Paper No. 0210, Department of Economics, University of British Columbia.

Examples

```
# estimate the dissimilarity between periods in the CES_sigma_2 dataset
# using the log quadratic measure of dissimilarity
relativeDissimilarity(CES_sigma_2, pvar = "prices", qvar="quantities",
pervar = "time", prodID = "prodID", indexMethod="fisher",
similarityMethod = "logquadratic")
```

---

shares *Compute expenditure shares for each product and time period*

Description

Compute expenditure shares for each product and time period

Usage

```
shares(x, pvar, qvar, pervar, prodID)
```

Arguments

x	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
pvar	A character string for the name of the price variable
qvar	A character string for the name of the quantity variable. For elementary indexes a quantity variable is not required for the calculations and you must specify qvar = "".
pervar	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
prodID	A character string for the name of the product identifier

Value

an n by p data frame of expenditure shares

unitValues *Aggregates prices to unit values and quantities to sums*

Description

A function to aggregate price and quantity data to unit values

Usage

```
unitValues(x, pvar, qvar, pervar, prodID)
```

Arguments

x	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
pvar	A character string for the name of the price variable
qvar	A character string for the name of the quantity variable
pervar	character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
prodID	A character string for the name of the product identifier

Value

A dataframe containing columns for product identifier, time period, quantities, and unit values.

Examples

```
# suppose the CES_sigma_2 dataset contains 12 monthly observations
# and suppose we want quarterly unit values.
df <- CES_sigma_2
# convert the monthly time variable into quarterly
df$time <- ceiling(CES_sigma_2$time/3)
# compute unit values using the quarterly time variable
unitValues(df,pvar="prices",qvar="quantities",pervar="time",prodID="prodID")
```

valueDecomposition *valueDecomposition*

Description

Perform a decomposition of value change using price and quantity indicators. This is an additive decomposition so that change due to price plus change due to quantity equals the total value change.

Usage

```
valueDecomposition(
  x,
  pvar,
  qvar,
  pervar,
  prodID,
  priceMethod,
  sample = "matched"
)
```

Arguments

x	data frame with input data
pvar	character string for the name of the price column
qvar	character string for the name of the quantity column
pervar	character string for the name of the time period variable
prodID	character string for the name of the product ID column
priceMethod	character string for the price indicator method. Valid options are "laspeyres", "paasche", "bennet", or "montgomery". This parameter also determines the method used for the quantity indicator. If a laspeyres price indicator is chosen, then a paasche quantity indicator is used. If a paasche price indicator is used then a laspeyres quantity indicator is used. For bennet and montgomery indicators, the same method is used for both the price and quantity indicators.
sample	whether to use a matched sample (sample = "matched")

Value

a dataframe containing the price indicator, quantity indicator the value change and the value level.

Examples

```
# decompose the value changes in the CES_sigma_2 dataset using the Bennet method
valueDecomposition(CES_sigma_2, pvar = "prices", qvar = "quantities",
prodID = "prodID", pervar = "time", priceMethod = "bennet")
```

---

values *Compute values (price x quantity)*

Description

Compute the total value (expenditure), for each time period in the sample.

Usage

```
values(
  x,
  pvar,
  qvar,
  pervar,
  prodID,
  sample = "matched",
  matchPeriod = "previous"
)
```

Arguments

x	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
pvar	A character string for the name of the price variable
qvar	A character string for the name of the quantity variable
pervar	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.
prodID	A character string for the name of the product identifier
sample	A character string specifying whether a matched sample should be used.
matchPeriod	A character string specifying which period is used to determine the set of products used for matching. Options are "following" or "previous". "following" calculates the expenditures in the current period, filtering out any products that do not appear in the following period. "previous" is calculated similarly, using the set of products in the previous period to filter the current period sample.

Examples

```
values(CES_sigma_2, pvar = "prices", qvar = "quantities", pervar = "time",
prodID = "prodID", matchPeriod = "previous")
```

---

weekIndex *Generate an index of weeks*

Description

Function to create a week index variable with weeks determined as defined in ISO 8601. If the week (starting on Monday) containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the 53rd week of the previous year, and the next week is week 1.

Usage

```
weekIndex(x)
```

Arguments

x	A vector of dates

Examples

```
# given a vector of dates
df <- data.frame(date = as.Date(c("2016-12-20","2016-12-27","2017-01-01","2017-01-07"),
format = "%Y-%m-%d"))
# calculate the time period variable
df$period <- weekIndex(df$date)
df
```

WTPDIndex *Compute a weighted time-product-dummy multilateral index*

Description

A function to calculate a weighted-time-product-dummy multilateral index.

Usage

```
WTPDIndex(
  x,
  pvar,
  qvar,
  pervar,
  prodID,
  sample = "",
  window = 13,
  splice = "mean",
  imputePrices = NULL
)
```

Arguments

x	A dataframe containing price, quantity, a time period identifier and a product identifier. It must have column names.
pvar	A character string for the name of the price variable
qvar	A character string for the name of the quantity variable
pervar	A character string for the name of the time variable. This variable must contain integers starting at period 1 and increasing in increments of 1 period. There may be observations on multiple products for each time period.

prodID	A character string for the name of the product identifier
sample	set to "matched" to only use products that occur across all periods in a given window. Default is not to match.
window	An integer specifying the length of the window.
splice	A character string specifying the splicing method. Valid methods are window, movement, half, mean, fbew, fbmw, wisp, hasp or mean_pub. The default is mean. See details for important considerations when using fbew and fbmw.
imputePrices	the type of price imputation to use for missing prices. Currently only "carry" is supported to used carry-forward/carry-backward prices. Default is NULL to not impute missing prices.

Details

When there are missing values in the dataset (e.g., from new or disappearing products), the default option is to treat the missing prices and quantities as zero. An alternative is to use a matched sample, where only products that appear throughout each window in the calculation are kept.

The splicing methods are used to update the price index when new data become available without changing prior index values. The window, movement, half and mean splices use the most recent index value as the base period, which is multiplied by a price movement computed using new data. The fbew (Fixed Base Expanding Window) and fbmw (Fixed Base Moving Window) use a fixed base onto which the price movement using new data is applied. The base period is updated periodically. IndexNumR calculates which periods are the base periods using `seq(from = 1, to = n, by = window - 1)`, so the data must be set up correctly and the right window length chosen. For example, if you have monthly data and want December of each year to be the base period, then the first period in the data must be December and the window must be set to 13.

References

Ivancic, L., W.E. Diewert and K.J. Fox (2011), "Scanner Data, Time Aggregation and the Construction of Price Indexes", Journal of Econometrics 161, 24-35.

Examples

```
# compute a wtpd index with mean splicing
WTPDIndex(CES_sigma_2, pvar = "prices", qvar = "quantities", pervar = "time",
prodID = "prodID", window=11, splice = "mean")
```

---

yearIndex *Generate an index of years*

Description

Function to create a year index variable

Usage

```
yearIndex(x)
```

Arguments

x	A vector or column of dates

Examples

```
# given a vector of dates
df <- data.frame(date = as.Date(c("2017-01-01","2018-04-01","2019-07-01","2019-08-01"),
format = "%Y-%m-%d"))
# calculate the time period variable
df$period <- yearIndex(df$date)
df
```

---

yearOverYearIndexes *Estimate year-over-year indexes*

Description

Year-over-year indexes are indexes where the months or quarters of the year are split in separate datasets and an index estimated on each. Therefore, year-over-year indexes estimated on a dataset with five full years of observations at a monthly frequency will have 12 separate indexes, each with 5 observations.

Usage

```
yearOverYearIndexes(freq, indexFunction, indexArgs)
```

Arguments

freq	the frequency of the data. Either "monthly" or "quarterly".
indexFunction	the name of the function to use to calculate the index as a string. Available options are ‘priceIndex‘, ‘GEKSIndex‘, ‘GKIndex‘, ‘WTPDIndex‘.
indexArgs	arguments for the price index function as a named list. All arguments must be named.

Value

a list of indexes with one element for each month or quarter

Examples

```
argsList <- list(x = CES_sigma_2, pvar = "prices", qvar = "quantities", pervar = "time",
prodID = "prodID", indexMethod = "fisher", output = "chained")

yearOverYearIndexes("quarterly", "priceIndex", argsList)
```

Index