Package 'tsutils'

November 6, 2026

Type Package

Title Time Series Exploration, Modelling and Forecasting

Version 0.9.5

Description Includes: (i) tests and visualisations that can help the modeller explore time series components and perform decomposition; (ii) modelling shortcuts, such as functions to construct lagmatrices and seasonal dummy variables of various forms; (iii) an implementation of the Theta method; (iv) tools to facilitate the design of the forecasting process, such as ABC-XYZ analyses; and (v) ``quality of life" functions, such as treating time series for trailing and leading values.

Imports RColorBrewer, forecast, MAPA, plotrix

Suggests thief

License GPL-3

Encoding UTF-8

LazyData true

URL https://github.com/trnnick/tsutils/

Config/roxygen2/version 8.1.1

RoxygenNote 7.3.3

NeedsCompilation no

Author Nikolaos Kourentzes [aut, cre],
Ivan Svetunkov [ctb],
Oliver Schaer [ctb]

Maintainer Nikolaos Kourentzes <nikolaos@kourentzes.com>

Repository CRAN

Date/Publication 2026-11-06 18:00:02 UTC

Contents

abc 2
abcxyz 4
cmav 5

colour.blend . . . 6
coxstuart . . . 7
decomp . . . 8
gdpchain . . . 10
geomean . . . 11
getOptK . . . 12
lagmatrix . . . 13
lambdaseq . . . 14
leadtrail . . . 15
nemenyi . . . 16
nemenyi.sens . . . 19
plotS . . . 22
plotSblock . . . 23
plotSthief . . . 26
plotStree . . . 27
plotThief . . . 30
referrals . . . 33
residout . . . 33
Saugment . . . 34
seasdummy . . . 37
seasplot . . . 38
seastest . . . 40
Sgroup . . . 42
Smean . . . 44
Sthief . . . 46
theta . . . 47
trendtest . . . 50
tsutils . . . 51
wins . . . 53
xyz . . . 54

Index **56**

---

abc *ABC analysis*

Description

Perform ABC analysis on a set of time series.

Usage

```
abc(x, prc = c(0.2, 0.3, 0.5))

## S3 method for class 'abc'
plot(x, cex.prc = 0.8, ...)
```

Arguments

x	this can either be an array, where each column is a series, or a vector of values. If x is an array of time series, then the importance of each series is calculated as the mean of the observations (sales volume). Otherwise, value can be whatever quantity is provided.
prc	a vector of percentages indicating how many items are included in each class. By default this is c(0.2,0.3,0.5), but any set of percentage values can be used as long as 0<=prc[i]<=1 and sum(prc)==1.
cex.prc	font size of percentages reported in plot.
...	additional arguments passed to the plot.

Value

Return object of class abc and contains:

- value: a vector containing the importance value of each series.
- class: a vector containing the class membership of each series.
- rank: a vector containing the rank of each series, with 1 being the highest ranking series.
- conc: the importance concentration of each class, as percentage of total value.

Methods (by generic)

- plot(abc): plot ABC or XYZ analyses.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

Ord K., Fildes R., Kourentzes N. (2017) Principles of Business Forecasting, 2e. *Wessex Press Publishing Co.*, p.515-518.

See Also

xyz, abcxyz.

Examples

```
x <- abs(matrix(cumsum(rnorm(5400,0,1)),36,150))
z <- abc(x)
print(z)
plot(z)
```

abcxyz *ABC-XYZ visualisation*

Description

Jointly visualise ABC and XYZ analyses.

Usage

```
abcxyz(imp, frc, outplot = c(TRUE, FALSE), error = NULL, ...)
```

Arguments

`imp`	an obkect of class `abc` that is the output of function `abc`.
`frc`	an obkect of class `abc` that is the output of function `xyz`.
`outplot`	if `TRUE`, then provide a visualisation of the analyses.
`error`	vector of forecast errors for each series that will be distributed in each class, presented as an average.
`...`	additional arguments passed to the plot.

Value

A list containing:

- `class`: a matrix containing the number of time series in each class.
- `error`: a matrix containing the averaged error for each class, if the argument `error` was used.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

Ord K., Fildes R., Kourentzes N. (2017) Principles of Business Forecasting, 2e. *Wessex Press Publishing Co.*, p.515-518.

See Also

`abc`, `xyz`.

Examples

```
x <- abs(matrix(cumsum(rnorm(5400,0,1)),36,150))
abcxyz(abc(x),xyz(x,type="cv"))
```

cmav *Centred moving average*

Description

Calculate the Centred Moving Average (CMA) for time series.

Usage

```
cmav(
  y,
  ma = NULL,
  fill = c(TRUE, FALSE),
  outplot = c(FALSE, TRUE),
  fast = c(TRUE, FALSE)
)
```

Arguments

y	input time series. Can be ts or msts object.
ma	length of centred moving average. If y is a ts object then the default is its frequency. If it is a msts object the default is the maximum frequency.
fill	if TRUE, then fill first and last ma/2 observations using exponential smoothing.
outplot	if TRUE, then output a plot of the time series and the moving average.
fast	if TRUE, then only a limited set of models are evaluated for CMA extrapolation.

Value

Centred moving average. If y is a ts object, then cma has the same properties.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

Ord K., Fildes R., Kourentzes N. (2017) Principles of Business Forecasting, 2e. *Wessex Press Publishing Co.*, p.109.

Examples

```
cmav(referrals,outplot=TRUE)
```

`colour.blend` *Opaque colour blending*

Description

Produce the opaque colour that looks like a semi-transparent colour drawn over a background, by mixing the two colours. This is useful for graphics devices that do not support semi-transparency, such as `postscript` (EPS), where semi-transparent elements are not drawn.

Usage

```
colour.blend(col, alpha, bg = NULL)
```

Arguments

`col`	colours to blend. Any form accepted by `col2rgb`. Their own alpha channel is ignored.
`alpha`	opacity of `col`, between 0 (only the background) and 1 (only `col`). `col` and `alpha` are recycled to a common length.
`bg`	background colour. If `NULL` (default), the background colour of the current device, `par("bg")`, is used if a device is open, otherwise white. A transparent background is treated as white.

Details

Each colour channel is mixed as `alpha * col + (1 - alpha) * bg`, which is how a semi-transparent colour is drawn over an opaque background. The result is identical to using `adjustcolor(col, alpha.f=alpha)` on that background, but overlapping elements do not add up: the colour drawn last covers the ones below it. If the plot is placed on a different background afterwards, the blended colours keep the tint of `bg`.

Value

A character vector of opaque colours in hexadecimal format.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

See Also

`adjustcolor`

Examples

```
colour.blend("red", 0.25)
colour.blend("steelblue", c(0.2, 0.5, 0.8))
colour.blend(c("red", "blue"), 0.5, bg="black")
plot(NA, xlim=c(0,1), ylim=c(0,1))
rect(0.1, 0.1, 0.9, 0.9, col=colour.blend("steelblue", 0.3))
```

---

coxstuart *Cox-Stuart test*

Description

Perform Cox-Stuart test for location or dispersion.

Usage

```
coxstuart(y, type = c("trend", "deviation", "dispersion"), alpha = 0.05)
```

Arguments

y	input data.
type	type of test. Can be: • "trend": test for changes in trend. • "deviation": test for changes in deviation. • "dispersion": test for changes in dispersion (range).
alpha	significance level.

Value

A list containing:

- H: hypothesis outcome.
- p.value: corresponding p-value.
- Htxt: textual description of the hypothesis outcome.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

Examples

```
coxstuart(referrals)
```

decomp *Classical time series decomposition*

Description

Perform classical time series decomposition. Optionally, the seasonal component is only estimated when there is evidence of seasonality.

Usage

```
decomp(
  y,
  m = NULL,
  s = NULL,
  trend = NULL,
  outplot = c(FALSE, TRUE),
  decomposition = c("multiplicative", "additive", "auto"),
  h = 0,
  type = c("mean", "median", "pure.seasonal"),
  w = NULL,
  seastest = c(FALSE, TRUE),
  alpha = 0.05,
  ...
)
```

Arguments

y	input time series. Can be ts object.
m	seasonal period. If y is a ts object then the default is its frequency.
s	starting period in the season. If y is a ts object then this is picked up from y.
trend	vector of the level/trend of y. Use NULL to estimate internally.
outplot	if TRUE, then provide a plot of the decomposed components.
decomposition	type of decomposition. This can be "multiplicative", "additive" or "auto". If y contains non-positive values then this is forced to "additive".
h	forecast horizon for seasonal component.
type	calculation for seasonal component: • "mean": the mean of each seasonal period. • "median": the median of each seasonal period. • "pure.seasonal": estimate using a pure seasonal model.
w	percentage or number of observations to winsorise in the calculation of mean seasonal indices. If w>1 then it is the number of observations, otherwise it is a percentage. If type != "mean" then this is ignored.

`seastest` if `TRUE`, then the detrended series is tested for seasonality with `seastest` and the seasonal component is only estimated if there is evidence of seasonality. Otherwise, the seasonal component is constant (1 for multiplicative and 0 for additive decomposition). If `FALSE` the seasonal component is always estimated.

`alpha` significance level for the seasonality test. Used only if `seastest=TRUE`.

`...` additional arguments to control the plot:

- `main`: title of the plot.
- `xlab`: label of the time axis.
- `ylab`: labels of the four panels.
- `col`: colours of the data and components, the reconstructed series, and the seasonal forecast, recycled as needed.
- `lwd`: line width.
- `las`, `cex.axis`, `cex.lab`: axis label orientation, size of axis annotation and of panel labels.

Value

A list containing:

- `trend`: trend component.
- `season`: season component.
- `irregular`: irregular component.
- `f.season`: forecasted seasonal component if `h>0`.
- g: pure seasonal model parameters.
- `decomposition`: type of decomposition used.
- `season.exist`: `TRUE`/`FALSE` result of the seasonality test, or `NULL` if not tested.
- `season.pval`: p-value of the seasonality test, or `NULL` if not tested.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

Ord K., Fildes R., Kourentzes N. (2017) Principles of Business Forecasting, 2e. *Wessex Press Publishing Co.*, p.106-111.

See Also

`seastest`, `seasplot`, `cmav`.

Examples

```
decomp(referrals)
decomp(referrals,outplot=TRUE,h=12,seastest=TRUE)
```

gdpchain *Hierarchy of US real GDP in chained dollars*

Description

Quarterly US real gross domestic product and its expenditure components in chained 2017 dollars, forming a hierarchy that is not coherent: chain-linked volumes are not additive, so each aggregate differs from the sum of its components.

Format

A list with:

- `y`: quarterly `ts` matrix of 26 series from 2007Q1 to 2026Q2 (78 quarters), in billions of chained 2017 dollars, seasonally adjusted annual rate. Columns are ordered as the rows of `S`.
- `S`: summing matrix (26×15) of the expenditure-side hierarchy, following NIPA Table 1.1.6: 11 aggregates and 15 bottom level series. Imports enter net exports, and hence GDP, with `-1`.
- `levels`: level of each row of `S` in the hierarchy, for use with `plotS`.
- `codes`: FRED series code of each row of `S`.
- `titles`: FRED series title of each row of `S`.

Details

The hierarchy is: GDP = PCE + GPDI + NetEx + Gov; personal consumption expenditures, PCE = Goods + Services, with Goods = Durable + Nondurable; gross private domestic investment, GPDI = FPI + CBI, with fixed investment FPI = PNFI + Residential and nonresidential PNFI = Structures + Equipment + IPP; net exports, NetEx = Exports - Imports, each split into goods and services; government, Gov = Federal + StateLocal, with Federal = Defense + Nondefense.

Because of chain-linking, the observed series do not satisfy `y = B %*% t(S)`, with `B` the bottom level series. The discrepancy of GDP against the bottom level equals, up to rounding, the residual published by the Bureau of Economic Analysis (FRED code A960RX1Q020SBEA). Net exports are computed as exports minus imports, so their discrepancy is implied by those of exports and imports. Use `Saugment` to obtain a coherent augmented hierarchy. The data are rounded to three decimals.

The data are the vintage of 30 September 2026. National accounts are revised, and comprehensive revisions change the reference year, so later vintages differ, including the discrepancies.

Source

Bureau of Economic Analysis, retrieved from FRED, Federal Reserve Bank of St. Louis, https://fred.stlouisfed.org.

References

Kourentzes, N., et al. (in preparation). Forecast reconciliation under observational incoherence. Working paper.

See Also

Saugment, plotS, Smean.

Examples

```
S <- gdpchain$S
y <- gdpchain$y
cols <- RColorBrewer::brewer.pal(3,"Pastel1")

# Hierarchy, highlighting the series that enter with -1 (imports)
neg <- rownames(S) %in% colnames(S)[apply(S < 0,2,any)]
plotS(S,levels=gdpchain$levels,fill=ifelse(neg,cols[1],"grey88"))

# Produce plot similar to reference paper
plotS(S,levels=gdpchain$levels,fill=ifelse(neg,cols[1],"grey88"),
      view="block",skip="expand",vgap=0.1)

# Augment with discrepancy series. The data are rounded to three decimals.
aug <- Saugment(S,y,tol=0.01)
aug$status
aug$error

# Augmented hierarchy, highlighting the discrepancy series. Discrepancies are
# measured against the bottom level, so they are not nested, and the block
# view is clearer than the tree.
k <- ncol(aug$d)
lev <- c(gdpchain$levels,rep(max(gdpchain$levels),k))
neg <- rownames(aug$S) %in% colnames(aug$S)[apply(aug$S < 0,2,any)]
fill <- ifelse(neg,cols[1],"grey88")
fill[nrow(S)+(1:k)] <- cols[2]
plotS(aug$S,levels=lev,fill=fill,view="block",skip="expand",vgap=0.1)

# Discrepancy series: close to zero around the 2017 reference year
plot(aug$d,main="")
```

---

geomean *Geometric mean*

Description

Calculate the geometric mean.

Usage

```
geomean(x, na.rm = c(FALSE, TRUE), ...)
```

Arguments

x	input data (will be considered as a vector).
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
...	further arguments passed to or from other methods.

Value

The geometirc mean of the values in x.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

Examples

```
geomean(c(0.5,1,1.5))
```

---

getOptK *Optimal temporal aggregation level for AR(1), MA(1), ARMA(1,1)*

Description

Calculate the theoretically optimal temporal aggregation level for AR(1), MA(1) and ARMA(1,1) time series.

Usage

```
getOptK(y, m = 12, type = c("ar", "ma", "arma"))
```

Arguments

y	a time series that must be of either ts or msts class.
m	maximum aggregation level.
type	type of data generating process. Can be: • "ar": For AR(1) series. • "ma": For MA(1) series. • "arma": For ARMA(1,1) series.

Value

Identified optimal temporal aggregation level.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

- Kourentzes, N., Rostami-Tabar, B., & Barrow, D. K. (2017). Demand forecasting by temporal aggregation: using optimal or multiple aggregation levels?. Journal of Business Research, 78, 1-9.
- Rostami-Tabar, B., Babai, M. Z., Syntetos, A., & Ducq, Y. (2013). Demand forecasting by temporal aggregation. Naval Research Logistics (NRL), 60(6), 479-498.
- Rostami-Tabar, B., Babai, M. Z., Syntetos, A., & Ducq, Y. (2014). A note on the forecast performance of temporal aggregation. Naval Research Logistics (NRL), 61(7), 489-500.

Examples

```
getOptK(referrals)
```

`lagmatrix` *Create lead/lags of a variable*

Description

Create an array with lead/lags of an input variable.

Usage

```
lagmatrix(x, lag)
```

Arguments

`x`	input variable.
`lag`	vector of leads and lags. Positive numbers are lags, negative are leads. O is the original `x`.

Value

An array with the resulting leads and lags (columns).

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

Examples

```
x <- rnorm(10)
lagmatrix(x,c(0,1,-1))
```

`lambdaseq` *Generate sequence of lambda for LASSO regression*

Description

Calculates the `lambdaMax` value, which is the penalty term (lambda) beyond which coefficients are guaranteed to be all zero and provides a sequence of `nLambda` values to `lambdaMin` in logarithmic descent.

Usage

```
lambdaseq(
  x,
  y,
  weight = NA,
  alpha = 1,
  standardise = TRUE,
  lambdaRatio = 1e-04,
  nLambda = 100,
  addZeroLambda = FALSE
)
```

Arguments

`x`	matrix of regressors. See `glmnet`.
`y`	response variable. See `glmnet`.
`weight`	vector of `length(nrow(y))` for weighted LASSO estimation. See `glmnet`.
`alpha`	elastic net mixing value. See `glmnet`.
`standardise`	if `TRUE`, then variables are standardised.
`lambdaRatio`	ratio between `lambdaMax` and `lambdaMin`. That is, `lambdaMin <- lambdaMax * lambdaRatio`.
`nLambda`	length of the lambda sequence.
`addZeroLambda`	if `TRUE`, then set the last value in the lambda sequence to 0, which is the OLS solution.

Value

A list that contains:

- `lambda`: sequence of lambda values, from `lambdaMax` to `lambdaMin`.
- `lambdaMin`: minimal lambda value.
- `lambdaMax`: maximal lambda value.
- `nullMSE`: MSE of the fit using just a constant term.

Author(s)

Oliver Schaer, <info@oliverschaer.ch>,

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

Hastie, T., Tibshirani, R., & Wainwright, M. (2015). Statistical learning with sparsity: the lasso and generalizations. CRC press.

Examples

```
y <- mtcars[,1]
x <- as.matrix(mtcars[,2:11])
lambda <- lambdaseq(x, y)$lambda

## Not run:
  library(glmnet)
  fit.lasso <- cv.glmnet(x, y, lambda = lambda)
  coef.lasso <- coef(fit.lasso, s = "lambda.1se")

## End(Not run)
```

---

leadtrail *Remove leading/training zeros/NAs*

Description

Remove leading or trailing zeros or NAs from a vector.

Usage

```
leadtrail(
  x,
  rm = c("zeros", "na"),
  lead = c(TRUE, FALSE),
  trail = c(TRUE, FALSE)
)
```

Arguments

x	vector of values to check.
rm	what to remove, can be "zeros" or "na".
lead	If TRUE, then leading values are removed.
trail	If TRUE, then trailing values are removed.

Value

Resulting vector.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

Examples

```
x <- c(rep(0,5),rnorm(100),rep(0,5))
leadtrail(x)
```

`nemenyi` *Nonparametric multiple comparisons (Nemenyi test)*

Description

Perform nonparametric multiple comparisons, across columns, using the Friedman and the post-hoc Nemenyi tests.

Usage

```
nemenyi(
  data,
  conf.level = 0.95,
  sort = c(TRUE, FALSE),
 plottype = c("vline", "none", "mcb", "vmcb", "line", "matrix", "lmatrix", "ranges"),
  select = NULL,
  labels = NULL,
  group = NULL,
  line = c(main = NA, xlab = NA, ylab = NA, xaxis = NA, yaxis = NA, group = NA),
  ...
)
```

Arguments

`data`	an array that includes values to be compared for several treatments (in columns) for several observations (rows), of size n x k. For example, if these are forecast errors, different methods should be in columns and errors for different time series or forecast origins in rows.
`conf.level`	the confidence level used for the comparison. Default is 0.95.
`sort`	if `TRUE`, then function sorts the outputted values of mean ranks. If plots are request, this is forced to `TRUE`.
`plottype`	type of plot to produce: • `"none"`: no plot.

- "mcb": *Multiple Comparison with the Best* style plot.
- "vmcb": vertical *MCB* plot.
- "line": summarised *line* plot. Each line connects a group of treatments that are all not significantly different.
- "vline": vertical *line* plot.
- "matrix": *matrix* visualisation. Both axes are in mean rank order and each row/column is a tested treatment (dark cell on the diagonal). Each off-diagonal cell gives the comparison of a pair of treatments; shaded cells indicate no evidence of significant difference. As the comparison is symmetric, the upper triangle mirrors the lower, so each row can be read on its own as the outcome for the tested treatment.
- "lmatrix": lower triangular *matrix* visualisation. Both axes are in mean rank order, each row/column is a tested treatment (dark cell on the diagonal), and only the lower triangle of the matrix is shown, giving the comparison for each pair once. Shaded cells indicate no evidence of significant difference.
- "ranges": *anchored* plot. One row per tested treatment on a common mean rank axis. The shaded band is the tested treatment's mean rank +/- the critical distance; treatments inside the band (filled points) are not significantly different from it, treatments outside (open points) are.

select — highlight selected treatment (column). Number 1 to k. Use NULL for no highlighting.

labels — optional labels for models. If NULL column names of data will be used.

group — if TRUE, show the group membership with a compact letter display: treatments that share a letter are not significantly different. For "ranges", "vline" and "matrix" the letters are shown on the right axis, for "line" on the top axis, for "lmatrix" in the empty upper triangle, next to each diagonal cell, and for "mcb" and "vmcb" they are appended to the labels. If NULL (default), they are shown for all plots except "mcb" and "vmcb".

line — margin lines (as in the line argument of title and mtext) where the plot annotation is placed. A named vector with elements:

- main: plot title.
- xlab: x-axis title.
- ylab: y-axis title.
- xaxis: x-axis labels (treatment labels or numbers).
- yaxis: y-axis labels (treatment labels or numbers).
- group: compact letter display, see group. For "lmatrix" this is the distance from the right edge of the diagonal cell.

Elements that are NA are placed automatically: axis labels and letters at par("mgp")[2] (letters at 0.3 for "lmatrix"), axis titles outside the axis labels and the title above the letters when these are on the top axis (otherwise in its default position). Only the elements to change need to be given, e.g. line=c(main=0.5, ylab=10); an unnamed vector is assigned in the order above. Margins are enlarged to fit the requested lines.

`...` additional arguments passed to the `plot` function. Axis annotation arguments (`cex.axis`, `font.axis`, `col.axis`, `cex.lab`, `font.lab`, `col.lab`, `cex.main`, `font.main`, `col.main`, `mgp`) are also used for the axes and titles the function draws.

Details

Plots respect the graphical parameters currently set with `par()` (for example `mfrow`, `cex`, `mgp`, `fg`, `bg`, `family` or `las` for numerical axes). The only parameter changed is `mar`: margins that are too small for the treatment labels, the group letters or the lines requested in `line` are enlarged, using the actual width of the labels, while all other margins are kept as set by the user. The user's `mar` is restored when the function exits. As the plot region depends on the margins, low-level additions to the plot after the call (e.g. `abline`) may not align; set `par(mar=...)` large enough for the labels beforehand to avoid this.

Value

Return object of class `nemenyi` and contains:

- `means`: mean rank of each treatment.
- `intervals`: intervals within there is no evidence of significance difference according to the Nemenyi test at requested confidence level.
- `fpval`: Friedman test p-value.
- `fH`: Friedman test hypothesis outcome.
- `cd`: Nemenyi critical distance. Output `intervals` is calculate as `means` +/- `cd`.
- `conf.level`: confidence level used for testing.
- `k`: number of treatments (columns).
- `n`: number of observations (rows).
- `group`: group membership as a compact letter display; treatments sharing a letter are not significantly different.
- `diff`: k x k matrix of absolute differences in mean ranks. Pairs with a difference smaller than `cd` are not significantly different.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>,

Ivan Svetunkov, <ivan@svetunkov.ru>.

References

- The tests are detailed by Hollander, M., Wolfe, D.A. and Chicken, E. (2014) Nonparametric Statistical Methods. 3rd Edition, John Wiley & Sons, Inc., New York.
- The *line* plot is introduced here and a first example of its use, along with a short description is provided by Kourentzes, N. (2013). Intermittent demand forecasts with neural networks. International Journal of Production Economics, 143(1), 198-206.

- The *matrix* plot is introduced by Kourentzes, N., & Athanasopoulos, G. (2018). Cross-temporal coherent forecasts for Australian tourism (No. 24/18). Monash University, Department of Econometrics and Business Statistics.
- The *MCB* plot is described by Koning, A. J., Franses, P. H., Hibon, M., & Stekler, H. O. (2005). The M3 competition: Statistical tests of the results. International Journal of Forecasting, 21(3), 397-409.

Examples

```
x <- matrix( rnorm(50*4,mean=0,sd=1), 50, 4)
x[,2] <- x[,2]+1
x[,3] <- x[,3]+0.7
x[,4] <- x[,4]+0.5
colnames(x) <- c("Method A","Method B","Method C - long name","Method D")
nemenyi(x,conf.level=0.95,plottype="vline")
nemenyi(x,plottype="matrix")
nemenyi(x,plottype="lmatrix")
nemenyi(x,plottype="ranges")
nemenyi(x,plottype="line",ylab="Groups",line=c(main=0.5,group=0.2))
```

`nemenyi.sens` *Sensitivity of the Nemenyi test to the confidence level*

Description

Shows how the grouping of treatments by the Nemenyi test changes with the confidence level. As the test uses a single critical distance for all pairs, each pair of treatments changes from significantly different to not significantly different at exactly one confidence level, given by its pairwise p-value. The groupings therefore change only at these levels and are constant in between, so all distinct groupings can be found exactly.

Usage

```
nemenyi.sens(
  data,
  conf.level = NULL,
  conf.range = c(0.8, 0.999),
  ref = NULL,
  plottype = c("groups", "letters", "pvalue", "none"),
  labels = NULL,
  line = c(main = NA, xlab = NA, ylab = NA, xaxis = NA, yaxis = NA, group = NA),
  ...
)
```

Arguments

`data`	an array that includes values to be compared for several treatments (in columns) for several observations (rows), of size n x k. See `nemenyi`.
`conf.level`	confidence levels to show. If `NULL` (default), one level is shown for each distinct grouping within `conf.range`, labelled with the range of confidence levels that give that grouping.
`conf.range`	range of confidence levels searched for distinct groupings, when `conf.level` is `NULL`.
`ref`	optional reference confidence level, e.g. `0.95`. If given, the column of this level is outlined in the `"groups"` and `"letters"` plots, and the matching significance level is marked on the colour key of the `"pvalue"` plot. If `NULL` (default), nothing is highlighted.
`plottype`	type of plot to produce: • `"none"`: no plot. • `"groups"`: *group stability* plot. Columns are confidence levels and rows are treatments in mean rank order, best at the top. In each column, every vertical bar connects a group of treatments that are all not significantly different, as in the *line* plot of `nemenyi`. • `"letters"`: as `"groups"`, but showing the compact letter display in each column: treatments that share a letter are not significantly different. • `"pvalue"`: lower triangular matrix of the pairwise Nemenyi p-values. The shading increases with the p-value, on a log scale; shaded cells indicate little evidence of significant difference. A pair is not significantly different at a confidence level above 1 - p-value. Changes from the previous column are highlighted. In `"groups"` these are the treatments whose set of not significantly different treatments changed, i.e. the members of the pairs that changed outcome, which mark the edges of the groups that changed. In `"letters"` these are all the letters that changed, which also includes treatments whose letters were relabelled because other groups changed. In both plots, columns where the Friedman test is not significant are shaded grey, and, if `ref` is given, the column of that confidence level is outlined.
`labels`	optional labels for models. If NULL column names of `data` will be used.
`line`	margin lines where the plot annotation is placed, see `nemenyi`. For `"pvalue"`, `group` is the distance of the colour key from the plot.
`...`	additional arguments passed to the `plot` function, see `nemenyi`.

Details

The pairwise p-value of treatments i and j is `1 - ptukey(|R_i - R_j| / sqrt(k(k+1)/(12n)), k, Inf)`, where `R` are the mean ranks. The pair is not significantly different at confidence levels above `1 - p`, which is the same outcome as `nemenyi` at that level. Graphical parameters and colours are handled as in `nemenyi`.

Value

Return object of class `nemenyi.sens` and contains:

- `means`: mean rank of each treatment.
- `pval`: k x k matrix of pairwise Nemenyi p-values.
- `pairs`: table of all pairs, ordered by p-value, with the absolute difference in mean ranks, the p-value and the confidence level above which the pair is not significantly different.
- `fpval`: Friedman test p-value. The Friedman test is significant for confidence levels below `1 - fpval`.
- `levels`: confidence levels shown.
- `range`: range of confidence levels that each shown level represents. For user specified levels this is the level itself.
- `cd`: Nemenyi critical distance at each shown level.
- `group`: k x levels matrix of compact letter displays.
- `changed`: list of two k x levels logical matrices, `TRUE` where a treatment changed from the previous level: `groups`, if its set of not significantly different treatments changed (highlighted in the `"groups"` plot), and `letters`, if its letters changed (highlighted in the `"letters"` plot).
- `k`: number of treatments (columns).
- `n`: number of observations (rows).

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

See Also

`nemenyi`

Examples

```
x <- matrix( rnorm(50*4,mean=0,sd=1), 50, 4)
x[,2] <- x[,2]+1
x[,3] <- x[,3]+0.7
x[,4] <- x[,4]+0.5
colnames(x) <- c("Method A","Method B","Method C - long name","Method D")
nemenyi.sens(x)
nemenyi.sens(x,plottype="letters")
nemenyi.sens(x,plottype="pvalue",ref=0.95)
nemenyi.sens(x,conf.level=c(0.8,0.9,0.95,0.99),ref=0.95)
```

plotS *Plot hierarchy*

Description

Plots a hierarchy from its summing matrix S, either as a tree or as blocks. This is a convenience function that calls plotStree or plotSblock. These can also be used directly, and their help files describe all the options available.

Usage

```
plotS(
  S,
  view = c("tree", "block"),
  labels = TRUE,
  parse = c(FALSE, TRUE),
  skip = c("empty", "expand"),
  levels = NULL,
  fill = "grey88",
  border = "grey30",
  level.labels = FALSE,
  level.font = 1,
  main = NULL,
  ...
)
```

Arguments

S	summing matrix ($n \times m$). Non-zero entries indicate that the bottom level series in that column is aggregated by the node in that row. This can describe cross-sectional, grouped or temporal hierarchies.
view	type of plot: • "tree": nodes connected with edges, see plotStree. • "block": each node is a block spanning the bottom level series it aggregates, see plotSblock. This is cleaner for large hierarchies.
labels	TRUE to use the row names of S as node labels (no labels if S has no row names), FALSE for no labels, or one label per row of S. Labels can be a character vector, an expression vector, or a list mixing strings, quoted calls and expressions.
parse	if TRUE and labels=TRUE, row names are parsed as plotmath, e.g. "H[1]" is plotted as H with subscript 1.
skip	how to plot nodes that skip a level: "empty" or "expand". See plotStree and plotSblock for details.
levels	optional vector with the level of each row of S. If NULL, levels are found automatically.
fill	fill colour of the nodes. Either a single colour or one per row of S.

border	border colour of the nodes.
level.labels	FALSE for none, TRUE for "L1", "L2", ..., or one label per level, starting from the top. These are plotted in the left margin.
level.font	font of the level labels, as in par("font"): 1 plain, 2 bold, 3 italic, 4 bold italic.
main	optional plot title.
...	additional arguments passed to plotStree or plotSblock, depending on view.

Value

Invisibly, the output of plotStree or plotSblock.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

See Also

plotStree, plotSblock, plotThief.

Examples

```
# Small hierarchy: A = H1 + H2, H1 = Q1 + Q2, H2 = Q3 + Q4
S <- rbind(c(1,1,1,1),c(1,1,0,0),c(0,0,1,1),diag(4))
rownames(S) <- c("A","H1","H2","Q1","Q2","Q3","Q4")
plotS(S)
plotS(S,view="block")

# Pass additional arguments to plotStree
plotS(S,edges="elbow",shape="circle",level.labels=TRUE)
```

---

plotSblock *Plot hierarchy as blocks*

Description

Plots a hierarchy from its summing matrix S, with each node (row of S) shown as the block of bottom level series it aggregates. Levels are arranged from the top to the bottom of the hierarchy. The horizontal axis has one unit per bottom level series (columns of S). This view remains clean for large hierarchies. For the more traditional view use plotStree.

Usage

```
plotSblock(
  S,
  labels = TRUE,
  parse = c(FALSE, TRUE),
  skip = c("empty", "expand"),
  levels = NULL,
  fill = "grey88",
  border = "grey30",
  hgap = 0.08,
  vgap = 0.35,
  cex = 1,
  min.cex = 0.35,
  level.labels = FALSE,
  level.font = 1,
  main = NULL
)
```

Arguments

S	summing matrix ($n \times m$). Non-zero entries indicate that the bottom level series in that column is aggregated by the node in that row. This can describe cross-sectional, grouped or temporal hierarchies.
labels	TRUE to use the row names of S as block labels (no labels if S has no row names), FALSE for no labels, or one label per row of S. Labels can be a character vector, an expression vector, or a list mixing strings, quoted calls and expressions.
parse	if TRUE and labels=TRUE, row names are parsed as plotmath, e.g. "H[1]" is plotted as H with subscript 1. Names that are not valid plotmath are plotted as text, with a warning.
skip	how to treat the space left by a node that skips a level: • "empty": leave it empty. • "expand": extend the block upwards through every level above that covers none of its columns. This stops at the first level that covers any of them.
levels	optional vector with the level of each row of S. Levels are ordered top to bottom in order of appearance. If NULL, levels are found automatically. See details.
fill	fill colour of the blocks. Either a single colour or one per row of S.
border	border colour of the blocks.
hgap	horizontal space between blocks, as a fraction of a column width.
vgap	vertical space between levels, as a fraction of the height of a level.
cex	maximum label size.
min.cex	labels that would need to be smaller than this to fit in their block are not plotted.
level.labels	FALSE for none, TRUE for "L1", "L2", ..., or one label per level, starting from the top, as a character or expression vector. These are plotted in the left margin.
level.font	font of the level labels, as in par("font"): 1 plain, 2 bold, 3 italic, 4 bold italic.
main	optional plot title.

Details

By default, rows are assigned to levels by scanning `S` from the top and starting a new level whenever a row overlaps with a row already in the current level. Therefore, rows within a level aggregate different bottom level series. This works with the usual ordering of `S` for cross-sectional, grouped and temporal hierarchies. For unusual row orders, provide the level of each row with `levels`.

In grouped hierarchies some nodes aggregate bottom level series that are not next to each other (e.g., a product node in a region x product grouping). These are plotted as several blocks, all with the same label and colour.

Value

Invisibly, a data frame with one row per block and columns:

- `node`: row name of `S`.
- `row`: row index in `S`.
- `level`: level of the node.
- `top`: highest level reached, after any expansion.
- `first`, `last`: first and last column of `S` covered by the block.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

For the design and explanation of this plot see arXiv:2609.23223.

See Also

`plotStree`, `plotS`, `plotThief`.

Examples

```
# Helper to construct S from a list of aggregates
makeS <- function(aggs,bottom){
  A <- t(sapply(aggs,function(a){as.numeric(bottom %in% a)}))
  S <- rbind(A,diag(length(bottom)))
  dimnames(S) <- list(c(names(aggs),bottom),bottom)
  return(S)
}

# Simple hierarchy
S1 <- makeS(list(A=c("Q1","Q2","Q3","Q4"),H1=c("Q1","Q2"),H2=c("Q3","Q4")),
            c("Q1","Q2","Q3","Q4"))
plotSblock(S1)

# Row names parsed as plotmath and level labels
rownames(S1) <- c("A","H[1]","H[2]","Q[1]","Q[2]","Q[3]","Q[4]")
plotSblock(S1,parse=TRUE,level.labels=expression(bold(S)[4],bold(S)[2],bold(S)[1]))
```

```
# Labels mixing text and expressions
plotSblock(S1,labels=list("Total",quote(H[1]),quote(H[2]),"Q1","Q2","Q3",quote(Q[4]^"*")))

# Q5 skips a level and connects directly to A
S2 <- makeS(list(A=c("Q1","Q2","Q3","Q4","Q5"),H1=c("Q1","Q2"),H2=c("Q3","Q4")),
            c("Q1","Q2","Q3","Q4","Q5"))
plotSblock(S2,skip="empty")
plotSblock(S2,skip="expand")

# Temporal hierarchy for monthly data
plotSblock(Sthief(12),labels=FALSE,level.labels=paste("k =",c(12,6,4,3,2,1)))

# Grouped hierarchy: region x product
bottom <- c("R1P1","R1P2","R2P1","R2P2")
S3 <- makeS(list(Total=bottom,R1=c("R1P1","R1P2"),R2=c("R2P1","R2P2"),
                 P1=c("R1P1","R2P1"),P2=c("R1P2","R2P2")),bottom)
cmp <- RColorBrewer::brewer.pal(5,"Pastel1")
plotSblock(S3,fill=c("grey88",cmp[2],cmp[2],cmp[5],cmp[5],rep("grey88",4)))

# Larger hierarchy without labels
b <- paste0("b",1:60)
S4 <- makeS(c(list(Total=b),
              setNames(lapply(0:4,function(g){b[g*12+1:12]}),paste0("G",1:5)),
              setNames(lapply(0:19,function(g){b[g*3+1:3]}),paste0("SG",1:20))),b)
plotSblock(S4,labels=FALSE)
```

`plotSthief` *Plot temporal hierarchy*

Description

Plots the temporal hierarchy for a given time series of seasonal periodicity.

Usage

```
plotSthief(y, labels = c(TRUE, FALSE), ...)
```

Arguments

`y`	input time series (a `ts` object) or an integer.
`labels`	if `TRUE` labels will be added for the temporal aggregation levels if the seasonal period is 4 (quarters), 7 (days in a week), 12 (months), 24 (hours), 48 (half-hours), 52 (weeks) or 364 (days).
`...`	additional arguments passed to the plotting function.

Value

Produces a plot of the temporal hierarchy.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

Athanasopoulos, G., Hyndman, R. J., Kourentzes, N., & Petropoulos, F. (2017). Forecasting with temporal hierarchies. European Journal of Operational Research, 262(1), 60-74.

Examples

```
plotSthief(AirPassengers)
```

---

plotStree *Plot hierarchy as a tree*

Description

Plots a hierarchy from its summing matrix `S` as the familiar tree of nodes and edges. Each node is a row of `S`, levels are arranged from the top to the bottom of the hierarchy, and each node is connected to the nodes directly below it. For large hierarchies `plotSblock` is usually cleaner.

Usage

```
plotStree(
  S,
  labels = TRUE,
  parse = c(FALSE, TRUE),
  skip = c("empty", "expand"),
  levels = NULL,
  layout = c("span", "even"),
  edges = c("straight", "elbow"),
  shape = c("box", "circle"),
  fill = "grey88",
  border = "grey30",
  edge.col = NULL,
  edge.lwd = 1.5,
  hgap = 0.2,
  vgap = 0.5,
  node.size = 0.2,
  cex = 1,
  min.cex = 0.35,
  level.labels = FALSE,
  level.font = 1,
  main = NULL
)
```

Arguments

S	summing matrix ($n \times m$). Non-zero entries indicate that the bottom level series in that column is aggregated by the node in that row. This can describe cross-sectional, grouped or temporal hierarchies.
labels	TRUE to use the row names of S as node labels (no labels if S has no row names), FALSE for no labels, or one label per row of S. Labels can be a character vector, an expression vector, or a list mixing strings, quoted calls and expressions.
parse	if TRUE and labels=TRUE, row names are parsed as plotmath, e.g. "H[1]" is plotted as H with subscript 1. Names that are not valid plotmath are plotted as text, with a warning.
skip	how to plot edges that skip levels: • "empty": the edge goes directly from the parent to the child. • "expand": the child gets a vertical stem that rises through every level above it that covers none of its bottom level series, as in plotSblock. The edge joins the top of the stem. With edges="elbow" this sets which end has the long vertical segment: the parent ("empty") or the child ("expand").
levels	optional vector with the level of each row of S. Levels are ordered top to bottom in order of appearance. If NULL, levels are found automatically, as in plotSblock.
layout	horizontal position of the nodes: • "span": each node is centred over the bottom level series it aggregates, so that the plot lines up with the columns of S and with plotSblock. Nodes closer than one column apart are pushed apart. • "even": nodes of each level are spread evenly, keeping their order.
edges	edge style: "straight" or "elbow". Elbows give an organisation chart look that suits hierarchies where each node has a single parent. When nodes have several parents (grouped and temporal hierarchies) straight edges are clearer.
shape	node shape: "box" or "circle".
fill	fill colour of the nodes. Either a single colour or one per row of S.
border	border colour of the nodes.
edge.col	edge colour. Either a single colour, or one per level, in which case the edge takes the colour of the level of its parent. If NULL, shades of grey from dark (top) to light (bottom).
edge.lwd	edge line width.
hgap	horizontal space between nodes, as a fraction of the distance between neighbouring nodes in a level.
vgap	vertical space between levels, as a fraction of the height of a level.
node.size	size of nodes without labels, in inches. Nodes with labels are sized to fit them. In both cases nodes are not larger than hgap and vgap allow.
cex	maximum label size.
min.cex	labels that would need to be smaller than this to fit in their node are not plotted.

level.labels	FALSE for none, TRUE for "L1", "L2", ..., or one label per level, starting from the top, as a character or expression vector. These are plotted in the left margin.
level.font	font of the level labels, as in par("font"): 1 plain, 2 bold, 3 italic, 4 bold italic.
main	optional plot title.

Details

A node is connected to another node below it if it aggregates all the bottom level series of that node, and there is no other node in between that does so as well. Therefore, each node is connected only to its nearest parents. In temporal hierarchies this gives the usual lattice, e.g., a monthly node is connected to both its 2-monthly and its quarterly node, but not to the annual node. In grouped hierarchies each node has one parent per grouping. Edges are shaded by the level of their parent, which helps to follow the different groupings.

Value

Invisibly, a list with:

- nodes: data frame with one row per node and columns node (row name of S), row (row index in S), level, x, y (centre of the node), width and height. Positions and sizes are in plot coordinates, with one unit per column of S and per level.
- edges: data frame with one row per edge and columns from, to (row names of S), from.row and to.row (row indices in S).

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

See Also

plotSblock, plotS, plotThief.

Examples

```
# Helper to construct S from a list of aggregates
makeS <- function(aggs,bottom){
  A <- t(sapply(aggs,function(a){as.numeric(bottom %in% a)}))
  S <- rbind(A,diag(length(bottom)))
  dimnames(S) <- list(c(names(aggs),bottom),bottom)
  return(S)
}

# Simple hierarchy
S1 <- makeS(list(A=c("Q1","Q2","Q3","Q4"),H1=c("Q1","Q2"),H2=c("Q3","Q4")),
            c("Q1","Q2","Q3","Q4"))
plotStree(S1)
plotStree(S1,edges="elbow")

# Row names parsed as plotmath, circles and level labels
rownames(S1) <- c("A","H[1]","H[2]","Q[1]","Q[2]","Q[3]","Q[4]")
```

```
plotStree(S1,parse=TRUE,shape="circle",
          level.labels=expression(bold(S)[4],bold(S)[2],bold(S)[1]))

# Q5 skips a level and connects directly to A
S2 <- makeS(list(A=c("Q1","Q2","Q3","Q4","Q5"),H1=c("Q1","Q2"),H2=c("Q3","Q4")),
            c("Q1","Q2","Q3","Q4","Q5"))
plotStree(S2,skip="empty")
tr <- plotStree(S2,skip="expand")
tr$edges

# Temporal hierarchy for monthly data
plotStree(Sthief(12),labels=FALSE,shape="circle",level.labels=paste("k =",c(12,6,4,3,2,1)))

# Grouped hierarchy: region x product, with colours for each grouping
bottom <- c("R1P1","R1P2","R2P1","R2P2")
S3 <- makeS(list(Total=bottom,R1=c("R1P1","R1P2"),R2=c("R2P1","R2P2"),
                 P1=c("R1P1","R2P1"),P2=c("R1P2","R2P2")),bottom)
cmp.fill <- RColorBrewer::brewer.pal(5,"Pastel1")
cmp.edge <- RColorBrewer::brewer.pal(5,"Set1")
plotStree(S3,fill=c("grey88",cmp.fill[2],cmp.fill[2],cmp.fill[5],cmp.fill[5],rep("grey88",4)),
          edge.col=c("grey20",cmp.edge[2],cmp.edge[5]))

# Larger hierarchy without labels
b <- paste0("b",1:60)
S4 <- makeS(c(list(Total=b),
              setNames(lapply(0:4,function(g){b[g*12+1:12]}),paste0("G",1:5)),
              setNames(lapply(0:19,function(g){b[g*3+1:3]}),paste0("SG",1:20))),b)
plotStree(S4,labels=FALSE,shape="circle",fill=RColorBrewer::brewer.pal(3,"Pastel1")[2])
plotStree(S4,labels=FALSE,edges="elbow")
```

`plotThief` *Plot temporal hierarchy*

Description

Plots a temporal hierarchy, either as a tree or as blocks, and names its levels descriptively, e.g., "Monthly", "2-monthly", "Quarterly", "Yearly", instead of by their temporal aggregation level k. The plots are produced by `plotS`, which calls `plotStree` or `plotSblock`.

Usage

```
plotThief(
  y,
  freq = NULL,
  k = NULL,
  view = c("tree", "block"),
  labels = FALSE,
  level.labels = TRUE,
```

```
  show.k = c(FALSE, TRUE),
  shape = c("circle", "box"),
  node.size = 0.5,
  fill = RColorBrewer::brewer.pal(3, "Pastel1")[2],
  border = RColorBrewer::brewer.pal(3, "Set1")[2],
  ...
)
```

Arguments

y	input time series (a ts object), an integer, or a temporal hierarchy summing matrix. For a time series or an integer the summing matrix is constructed with Sthief, using its frequency or the integer as the seasonal periodicity.
freq	frequency of the bottom level series, see details. If NULL, this is inferred from the size of the top level.
k	optional vector of temporal aggregation levels to include, passed to Sthief. Ignored if y is a summing matrix.
view	type of plot: "tree" (see plotStree) or "block" (see plotSblock). The block view is cleaner for large hierarchies, such as daily data within a year.
labels	FALSE for no labels in the nodes, TRUE to number the nodes within each level, or one label per row of the summing matrix.
level.labels	TRUE for the descriptive names of the levels, FALSE for none, or one label per level, starting from the top.
show.k	if TRUE, the temporal aggregation level is added to the level names, e.g., "Quarterly (k = 3)".
shape	node shape for the tree view: "circle" or "box". Ignored for the block view.
node.size	size of nodes without labels for the tree view, in inches. This is larger than the default of plotStree. Nodes are not made larger than the space available in the plot. Ignored for the block view.
fill	fill colour of the nodes. Either a single colour or one per row of the summing matrix.
border	border colour of the nodes.
...	additional arguments passed to plotStree or plotSblock, depending on view. For example, edges, skip, level.font or main.

Details

Rows of the summing matrix are grouped into levels by the number of bottom level periods they aggregate (k), with the largest k at the top. Therefore, the order of the rows does not matter.

The level names depend on the frequency of the bottom level. If freq=NULL, this is inferred from the number of bottom level periods in the top level, assuming that this is the usual seasonal cycle:

- 4, 12, 13, 52: quarterly, monthly, 4-weekly and weekly data, with yearly top level.
- 260, 364, 365: business daily and daily data, with yearly top level.
- 5, 7, 168, 336: business daily, daily, hourly and half-hourly data, with weekly top level.

- 24, 48, 96, 144, 288, 1440: hourly, half-hourly, 15-minute, 10-minute, 5-minute and minutely data, with daily top level.

For other cases, or when the hierarchy does not end in the usual cycle (e.g., 24 months with a 2-yearly top level, which would otherwise be taken as hourly data), use `freq`. This can be `"secondly"`, `"minutely"`, `"quarter-hourly"`, `"half-hourly"`, `"hourly"`, `"daily"`, `"business-daily"`, `"weekly"`, `"fortnightly"`, `"half-monthly"`, `"monthly"`, `"quarterly"`, `"half-yearly"` or `"yearly"`, optionally with a multiple, e.g., `"15-minute"`, `"2-hourly"` or `"4-weekly"`. Case, spaces, hyphens and underscores are ignored and nouns such as `"minute"` or `"month"` work as well.

Each level is named using the largest time unit that divides it exactly, e.g., 120 minutes is "2-hourly" and 14 days is "2-weekly". The following conventions are used: 7 days or 5 business days are a week; 13, 26 and 52 weeks are "Quarterly", "Half-yearly" and "Yearly"; and 365 days are a year. Levels that do not fit any of these keep the unit of the bottom level, e.g., "26-daily".

Value

Invisibly, a list with the output of `plotStree` (`nodes` and `edges`) or of `plotSblock` (`blocks`), with row indices referring to the rows of the summing matrix as given, and:

- `levels`: data frame with the `level`, its temporal aggregation level `k` and its `name`.
- `freq`: name of the frequency of the bottom level.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

Athanasopoulos, G., Hyndman, R. J., Kourentzes, N., & Petropoulos, F. (2017). Forecasting with temporal hierarchies. European Journal of Operational Research, 262(1), 60-74.

See Also

`Sthief`, `plotStree`, `plotSblock`, `plotS`.

Examples

```
# Full temporal hierarchies
plotThief(AirPassengers)
plotThief(4,show.k=TRUE)
plotThief(24)
plotThief(7,labels=TRUE)

# Plain level labels and smaller nodes
plotThief(12,level.font=1,node.size=0.25)

# Same structure, but half-monthly data
plotThief(24,freq="half-monthly")

# Hourly data aggregated to days and weeks, but not years
plotThief(168,k=c(168,24,12))
```

```
# Two years of monthly data
plotThief(24,k=c(24,12,6),freq="monthly")

# 15-minute data, with numbered boxes and elbow edges
plotThief(4,freq="15-minute",labels=TRUE,shape="box",edges="elbow")

# Block view, better for large hierarchies
plotThief(12,view="block")
plotThief(364,view="block",main="Daily data")
```

`referrals` *NHS A&E Referrals*

Description

Monthly Accident & Emergency referrals for England and Wales.

References

https://www.england.nhs.uk/statistics/statistical-work-areas/ae-waiting-times-and-activity/

`residout` *Residuals control chart*

Description

Create a control chart of residuals and identify outliers.

Usage

```
residout(resid, t = 2, outplot = c(TRUE, FALSE))
```

Arguments

`resid`	vector of residuals.
`t`	threshold value over which standardised residuals are regarded as outliers.
`outplot`	if `TRUE`, then a control chart of the standardised residuals is plotted.

Value

A list containing:

- `location`: locations of outliers.
- `outliers`: values of outliers.
- `residuals`: standardised residuals.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

Examples

```
residout(rnorm(50), outplot=TRUE)
```

Saugment *Augment hierarchy with discrepancy series*

Description

Augments a hierarchy whose observed series are not coherent with additional bottom level series that capture the observed discrepancies. The augmented hierarchy is coherent, so that standard forecast reconciliation methods can be applied to it without modification.

Usage

```
Saugment(
  S,
  y,
  tol = sqrt(.Machine$double.eps) * max(abs(y)),
  nodes = NULL,
  D = NULL,
  outplot = c(FALSE, TRUE),
  ...
)
```

Arguments

S	summing matrix ($n \times m$). Non-zero entries indicate that the bottom level series in that column is aggregated by the node in that row. Rows of S correspond to columns of y.
y	matrix or ts matrix ($T \times n$) of observed series. If both rownames(S) and colnames(y) exist, the columns of y are matched to the rows of S by name. Otherwise the columns of y must follow the rows of S. Missing values are not allowed.
tol	absolute tolerance that separates genuine discrepancies from rounding. Set this to the rounding precision of the data, e.g., 0.01 for data published with three decimals. The default is only suitable for data without rounding.
nodes	optional vector of names or row indices of aggregate nodes that must receive their own discrepancy series. See details.

`D`	optional matrix with the known structure of the discrepancies, which replaces their classification. Columns are named by the aggregate nodes that receive their own discrepancy series, and rows by aggregate nodes; each row gives the discrepancy of that node as a combination of the discrepancy series. Aggregate nodes not in the rows are coherent. See details.
`outplot`	if `TRUE`, then plot the augmented hierarchy using `plotS`, highlighting the discrepancy series.
`...`	additional arguments passed to `plotS`, when `outplot=TRUE`.

Details

Bottom level series are identified as the rows of `S` that are unit vectors. All other rows are aggregate nodes. For aggregate node i the discrepancy is measured against the bottom level, $\varepsilon_{i,t} = y_{i,t} - s_i' b_t$, where s_i is the row of `S` and b_t the bottom level series.

Aggregate nodes are processed from the smallest to the largest, in terms of the number of bottom level series they aggregate, and are classified as:

- `"coherent"`: the discrepancy is within `tol` of zero.
- `"implied"`: the node is a linear combination of nodes that already have their own discrepancy series, both in `S` and, within `tol`, in its discrepancy. For example, net exports defined as exports minus imports. The coefficients are taken from `S` and stored in the corresponding row of `D`.
- `"own"`: the node receives its own discrepancy series.

When discrepancies are linearly dependent, which nodes receive their own discrepancy series is not unique. Use `nodes` to choose them.

Implied discrepancies must follow from the structure of the hierarchy: the coefficients are obtained from the rows of `S`, and the data are only used to check that the discrepancy follows them. Estimating the coefficients from the data instead is not reliable when the discrepancy series have few observations or smooth histories, e.g., in temporal hierarchies where each observation is a complete seasonal cycle, as a few discrepancy series can then reproduce any other one exactly with arbitrary coefficients.

When the structure of the discrepancies is known, it can be given with `D`, and no classification is done. The function checks that the discrepancies agree with `D` within `tol`.

Value

A list with:

- `S`: augmented summing matrix ($(n + k) \times (m + k)$). Rows are the rows of the input `S`, in the same order, followed by the k discrepancy series. Columns are the bottom level series followed by the discrepancy series.
- `y`: input series with the discrepancy series appended. This is a `ts` object if `y` is one.
- `D`: matrix ($n_a \times k$) that maps the discrepancy series to the aggregate nodes.
- `d`: the discrepancy series ($T \times k$).
- `eps`: discrepancies of all aggregate nodes ($T \times n_a$).

- `status`: classification of each aggregate node.
- `error`: maximum absolute incoherence of the augmented series, i.e., `max(abs(y - B %*% t(S)))`, with `B` the augmented bottom level series. This should be within `tol`.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

See Also

`plotS`, `Sthief`.

Examples

```
# Small hierarchy: Total = A + B, A = A1 + A2, B = B1 + B2
S <- rbind(c(1,1,1,1),c(1,1,0,0),c(0,0,1,1),diag(4))
rownames(S) <- c("Total","A","B","A1","A2","B1","B2")
set.seed(1)
b <- matrix(rnorm(40,10),10,4)
y <- b %*% t(S)
colnames(y) <- rownames(S)

# Add discrepancies at Total and A
y[,"Total"] <- y[,"Total"] + rnorm(10)
y[,"A"] <- y[,"A"] + rnorm(10)
aug <- Saugment(S,ts(y,frequency=4),outplot=TRUE)
aug$status
aug$S

# Block view, passing arguments to plotS
aug <- Saugment(S,y,outplot=TRUE,view="block")

# Total is published as A + B, while A and B differ from their bottom level
# series. The discrepancy of Total is then the sum of those of A and B.
y <- b %*% t(S)
colnames(y) <- rownames(S)
y[,"A"] <- y[,"A"] + rnorm(10)
y[,"B"] <- y[,"B"] + rnorm(10)
y[,"Total"] <- y[,"A"] + y[,"B"]

# By default A and B get their own discrepancy series and Total is implied
Saugment(S,y)$status

# Use nodes to give Total its own discrepancy series instead. B is now implied
# as the difference between the discrepancies of Total and A
aug <- Saugment(S,y,nodes="Total")
aug$status
aug$D

# Short sample: a temporal hierarchy of a seasonal series together with its
# seasonally adjusted version, with only 5 years of data. The discrepancies of the
# adjusted aggregates are sums of those of the adjusted months, following S.
```

```
y <- ts(100 + 10*sin(2*pi*(1:60)/12) + rnorm(60),frequency=12)
s <- stl(y,s.window=7)$time.series[,"seasonal"]
Sn <- Sthief(y)
rownames(Sn) <- paste0("n",1:nrow(Sn))
S2 <- rbind(Sn,Sn)
rownames(S2) <- c(paste0("NSA.",rownames(Sn)),paste0("SA.",rownames(Sn)))
stack <- function(x){matrix(x,ncol=12,byrow=TRUE) %*% t(Sn)}
Y <- cbind(stack(y),stack(y-s))
colnames(Y) <- rownames(S2)
aug <- Saugment(S2,Y)
table(aug$status)

# The same structure given explicitly with D
D <- Sn
dimnames(D) <- list(paste0("SA.",rownames(Sn)),paste0("SA.n",17:28))
aug <- Saugment(S2,Y,D=D)
table(aug$status)
```

`seasdummy` *Create seasonal dummy variables.*

Description

Create binary or trigonometric seasonal dummies.

Usage

```
seasdummy(n, m = NULL, y = NULL, type = c("bin", "trg"), full = c(FALSE, TRUE))
```

Arguments

`n`	number of observations to create.
`m`	seasonal periodicity. If `NULL` it will take the information from the provided time series (`y` argument). See notes.
`y`	this is an optional time series input that can be used to get seasonal periodicity (`m`) and the start point.
`type`	type of seasonal dummies to create. • `"bin}: binary dummies. \item \code{"trg`: trigonometric dummies. See notes.
`full`	If full is `TRUE`, then keeps the m-th dummy that is co-linear to the rest. See notes.

Value

An array with seasonal dummies, where rows correspond observations and columns to dummy variables.

Note

If the seasonal periodicity is fractional then the the type will be overriden to trigonometric and only two seasonal dummies with be produced. One cosine and one sine.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

Examples

```
seasdummy(24,12)
```

---

seasplot *Seasonal plots with simplistic trend/season tests*

Description

Construct seasonal plots of various styles for a given time series. The series can be automatically detrended as needed.

Usage

```
seasplot(
  y,
  m = NULL,
  s = NULL,
  trend = NULL,
  colour = NULL,
  alpha = 0.05,
  outplot = c(1, 0, 2, 3, 4, 5),
  decomposition = c("multiplicative", "additive", "auto"),
  cma = NULL,
  labels = NULL,
  legend = TRUE,
  ...
)
```

Arguments

y	input time series. Can be ts object.
m	seasonal period. If y is a ts object then the default is its frequency.
s	starting period in the season. If y is a ts object then this is picked up from y.
trend	if TRUE, then presence of trend is assumed and removed. If FALSE no trend is assumed. Use NULL to identify automatically.
colour	colour override for plots. A single colour, or a vector of colours, used as follows:

- `outplot=1`: colours of the seasons, from oldest to newest. Two or more colours (other than one per season) are interpolated into a gradient.
- `outplot=2`: colours of the boxes, recycled across seasonal periods.
- `outplot=3`: colours of the medians and of the subseries, in this order. A single colour is used for both.
- `outplot=4`: colour of the median. Optionally, three more colours for the 25%-75%, 10%-90% and min-max bands. Otherwise, the bands are lighter shades of the median colour.
- `outplot=5`: colour of the highest density. Two or more colours are interpolated into the density palette.

`alpha` significance level for statistical tests.

`outplot` type of seasonal plot

- `0`: none.
- `1`: seasonal diagram.
- `2`: seasonal boxplots.
- `3`: seasonal subseries.
- `4`: seasonal distribution.
- `5`: seasonal density.

`decomposition` type of seasonal decomposition. This can be `"multiplicative"`, `"additive"` or `"auto"`. If `y` contains non-positive values then this is forced to `"additive"`.

`cma` input precalculated level/trend for the analysis. If `trend=TRUE` it is removed from `y` without testing, otherwise the presence of trend is tested on `cma` (overriding `trend=FALSE`).

`labels` external labels for the seasonal periods. Use `NULL` for none. If `length(labels) < m`, then this input is ignored.

`legend` if `TRUE` a legend is added to the plot, where applicable (`outplot=1` and `4`). Use `FALSE` to omit it, or a keyword such as `"topright"` to change its position.

`...` additional arguments passed to plotting functions. Arguments that refer to the plotted data, `col`, `lwd`, `lty`, `pch`, `cex`, `bg` and `type`, are applied to the seasonal lines, boxes or points (`col` acts as `colour`). Axis arguments (`las`, `cex.axis`, `col.axis`, `font.axis`, `tck`, `tcl`, `mgp`) are also applied to the seasonal period axis. Everything else, such as `main`, `xlab`, `ylab`, `xlim`, `ylim`, `xaxt="n"` or `axes=FALSE`, is passed to the function that sets up the plot (`plot`, `boxplot` or `image`).

Details

The seasonality test is the same as in `seastest`, which can be used to run it without producing a plot.

Value

An object of class `seasexpl` containing:

- `season`: matrix of (detrended) seasonal elements.

- season.exist: TRUE/FALSE results of seasonality test, or NULL if not tested.
- season.pval: p-value of seasonality test (Friedman test), or NULL if not tested.
- trend: CMA estimate (using cmav) or NULL if no trend was removed.
- trend.exist: TRUE/FALSE results of trend test.
- trend.pval: p-value of trend test (Cox-Stuart).
- decomposition: type of decomposition used. "none" if no trend was removed.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

See Also

seastest, decomp.

Examples

```
seasplot(referrals,outplot=1)
seasplot(referrals,outplot=1,colour=c("grey","red"),lwd=2,legend="bottomright")
seasplot(referrals,outplot=3,labels=month.abb,las=2,main="Referrals",ylab="Index")
```

---

seastest *Seasonality test*

Description

Test a time series for deterministic seasonality using the Friedman test on the distributions of the (detrended) seasonal periods. The series can be automatically detrended as needed. This is the test reported by seasplot.

Usage

```
seastest(
  y,
  m = NULL,
  s = NULL,
  trend = NULL,
  alpha = 0.05,
  decomposition = c("multiplicative", "additive", "auto"),
  cma = NULL
)
```

Arguments

y	input time series. Can be ts object.
m	seasonal period. If y is a ts object then the default is its frequency.
s	starting period in the season. If y is a ts object then this is picked up from y.
trend	if TRUE, then presence of trend is assumed and removed. If FALSE no trend is assumed. Use NULL to identify automatically.
alpha	significance level for statistical tests.
decomposition	type of seasonal decomposition. This can be "multiplicative", "additive" or "auto". If y contains non-positive values then this is forced to "additive".
cma	input precalculated level/trend for the analysis. If trend=TRUE it is removed from y without testing, otherwise the presence of trend is tested on cma (overriding trend=FALSE).

Details

The series is first detrended (if needed) using a centred moving average (see cmav), with the presence of trend tested using the Cox-Stuart test (see coxstuart) when trend=NULL. The detrended series is then arranged in a matrix, with one row per season and one column per seasonal period, and the Friedman test is used to test whether the seasonal periods come from the same distribution. Only complete seasons are used for the test, so at least two complete seasons are needed.

Value

An object of class seastest containing:

- is.seasonal: TRUE/FALSE result of the seasonality test. NA if the test could not be performed.
- statistic: Friedman test statistic.
- pvalue: p-value of the seasonality test.
- alpha: significance level used.
- season: matrix of (detrended) seasonal elements.
- trend: CMA estimate (using cmav) or NULL if no trend was removed.
- trend.exist: TRUE/FALSE results of trend test, or NULL if not tested.
- trend.pval: p-value of trend test (Cox-Stuart), or NULL if not tested.
- decomposition: type of decomposition used. "none" if no trend was removed.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

See Also

seasplot, mseastest, trendtest.

Examples

```
seastest(referrals)
seastest(referrals,trend=TRUE,decomposition="auto")$pvalue
```

Sgroup *Grouped hierarchy S matrix from dimensions*

Description

Construct the summing matrix S of a grouped hierarchy from the labels of the bottom level series on one or more dimensions, e.g., product type, region, and organic. Redundant aggregate nodes are removed.

Usage

```
Sgroup(
  x,
  names = NULL,
  sep = "_",
  cross = c("all", "none"),
  na = c("exclude", "group"),
  total = "Total",
  sign = NULL,
  sort.levels = c(FALSE, TRUE),
  outplot = c(FALSE, TRUE),
  ...
)
```

Arguments

x	data frame or matrix of labels, with one row per bottom level series and one column per dimension. Column names are used as dimension names.
names	optional names of the bottom level series. If NULL, the row names of x are used, or else the labels of each series joined with sep.
sep	separator used to construct the names of the nodes.
cross	which combinations of dimensions to include, in addition to the total and each dimension on its own: • "all": all combinations of two or more dimensions. • "none": no combinations. • a vector of specific combinations, each with the dimension names separated by ":", e.g., c("product:region","region:organic"). A list of character vectors, e.g., list(c("product","region")), is also accepted.
na	how to treat missing labels:

- `"exclude"`: a series with a missing label on a dimension is not part of the nodes of that dimension, but it is still part of the total and of the nodes of the other dimensions.
- `"group"`: missing labels form their own group, named `"NA"`.

`total` name of the top level node.

`sign` optional vector of signs (`1` or `-1`), one per series, with which they enter aggregate nodes, e.g., `-1` for imports when net exports are needed. See details.

`sort.levels` if `TRUE`, then reorder the levels so that levels with fewer nodes are placed higher in the S matrix. Levels with the same number of nodes keep their original order. The bottom level is always last.

`outplot` if `TRUE`, then plot the hierarchy using `plotS`, with the names of the levels in the left margin.

`...` additional arguments passed to `plotS`, when `outplot=TRUE`.

Details

For the total, each dimension on its own, and each combination of dimensions requested in `cross`, the series are grouped by their labels, giving one aggregate node per combination of values. A combination of dimensions forms one level of the hierarchy. Levels are ordered from the total, to single dimensions, to combinations of two dimensions, and so on, with the bottom level series last.

With `sign`, series enter aggregate nodes with their sign. Nodes whose series are all negative are gross totals and are kept positive, e.g., total imports, while nodes that mix signs are net values, e.g., net exports. Bottom level series are always positive.

Aggregate nodes are redundant and removed when they:

- aggregate a single series, as they repeat a bottom level series;
- aggregate the same series as a node higher up, e.g., a dimension with a single value repeats the total, or a combination of nested dimensions (country within region) repeats the finer dimension. The first, higher, node is kept.

Levels left without any nodes disappear. If `sort.levels=TRUE`, the levels are reordered by their number of nodes, after removing the redundant nodes. Levels are named by their dimensions, e.g., `"product x region"`, and the bottom level is named `"Bottom"`. Nodes are named by their values joined with `sep`. If the same name arises from different dimensions, the dimension name is added, e.g., `"organic:yes"`.

Value

S matrix ($n \times m$), with row and column names. The attribute `"levels"` gives the level (combination of dimensions) of each row, which can be passed to `plotS`. The attribute `"removed"` gives the names of the removed redundant nodes.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

See Also

plotS, Saugment, Sthief.

Examples

```
# Six products in five regions, organic or not
x <- expand.grid(organic=c("no","yes"),
                 region=c("denmark","sweden","norway","iceland","finland"),
                 product=c("milk","yoghurt","cheese","quark","whey","casein"),
                 stringsAsFactors=FALSE)[,3:1]
S <- Sgroup(x,outplot=TRUE)
dim(S)
table(attr(S,"levels"))

# Only the total and each dimension on its own
S <- Sgroup(x,cross="none",outplot=TRUE)

# Add only the product by region combination
S <- Sgroup(x,cross="product:region",outplot=TRUE)

# Levels with fewer nodes higher up: organic, region, product, ...
S <- Sgroup(x,cross="product:region",sort.levels=TRUE,outplot=TRUE)
unique(attr(S,"levels"))

# Block view, passing arguments to plotS
S <- Sgroup(x,cross="product:region",outplot=TRUE,view="block")

# Redundant dimensions: market has a single value and union is nested in region
x$market <- "nordics"
x$union <- ifelse(x$region %in% c("norway","iceland"),"non-eu","eu")
S <- Sgroup(x)
head(attr(S,"removed"))

# Missing labels
x <- x[,1:3]
x$region[1:4] <- NA
S <- Sgroup(x,cross="none")
S[c("Total","denmark"),1:6]

# Net values: exports and imports of goods and services
x <- data.frame(flow=c("exports","exports","imports","imports"),
                type=c("goods","services","goods","services"),
                row.names=c("X.goods","X.services","M.goods","M.services"))
S <- Sgroup(x,cross="none",sign=ifelse(x$flow=="imports",-1,1),total="NetExports")
S
# NetExports, goods and services are net values, exports and imports are gross totals
```

Smean *Mean aggregation S matrix*

Description

Converts a summing matrix `S`, where aggregate nodes are sums of bottom level series, to one where aggregate nodes are means of the bottom level series they aggregate. The means can use equal weights or a vector of bottom level weights. Negative entries, e.g., imports in net exports, are allowed.

Usage

```
Smean(S, w = NULL)
```

Arguments

`S`	summing matrix ($n \times m$). Negative entries indicate series that enter a node with a negative sign, e.g., from `Sgroup` with argument `sign`.
`w`	optional vector of non-negative bottom level weights, one per column of `S`. If named, it is matched to the column names of `S`. If `NULL`, all bottom level series have equal weight, giving simple means.

Details

Each aggregate row i of `S` is replaced by $s_{ij}w_j / \sum_k |s_{ik}|w_k$, so that every aggregate node is a convex combination of the bottom level series it aggregates, entering with their sign. For example, net exports with equal weights become $(X_g + X_s - M_g - M_s)/4$. For non-negative `S` the normalisation is the usual weighted sum. The weights do not need to sum to one, as they are normalised within each node. Bottom level rows (unit rows) are not changed, so a bottom level series with zero weight is excluded from the means but remains in the hierarchy. Entries of `S` other than one, e.g., from `Sgroup` or temporal hierarchies, are kept as relative weights.

The result is still a linear aggregation, $y = Sb$, so it can be used in place of the summing matrix for forecast reconciliation, e.g., for series that are rates, prices, or stocks rather than flows.

If structural scaling is used with the resulting `S`, it should be derived from its assumptions rather than computed from `S`: with uncorrelated bottom level errors of equal variance, the error variance of node i is proportional to $\sum_j s_{ij}^2 w_j^2 / (\sum_j |s_{ij}|w_j)^2$, which is $1/k$ for an equal weight mean over k series. Using `diag(rowSums(S))`, as for sums, gives the identity matrix, which corresponds to OLS rather than structural scaling.

Value

S matrix with weighted mean aggregation, with the same dimensions, names and attributes as `S`.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

See Also

`Sgroup`, `Sthief`, `Saugment`.

Examples

```
# Small hierarchy: A = H1 + H2, H1 = Q1 + Q2, H2 = Q3 + Q4
S <- rbind(c(1,1,1,1),c(1,1,0,0),c(0,0,1,1),diag(4))
rownames(S) <- c("A","H1","H2","Q1","Q2","Q3","Q4")

# Equal weight means
Smean(S)

# Weighted means, e.g., by sales volume of each bottom level series
Smean(S,w=c(10,30,20,40))

# Temporal hierarchy for quarterly data where the annual value is the average
Smean(Sthief(4))

# Net exports: exports and imports of goods and services
S <- rbind(NetExports=c(1,1,-1,-1),Exports=c(1,1,0,0),Imports=c(0,0,1,1),diag(4))
colnames(S) <- rownames(S)[4:7] <- c("X.goods","X.services","M.goods","M.services")
Smean(S)

# Structural scaling for the mean hierarchy, derived from its assumptions
Sm <- Smean(S)
W <- diag(rowSums(Sm^2)/rowSums(abs(Sm))^2)
diag(W)
```

Sthief *Temporal hierarchy S matrix*

Description

Calculate the temporal hierarchy summing matrix S for a given time series of seasonal periodicity.

Usage

```
Sthief(y, k = NULL)
```

Arguments

y	input time series (a ts object) or an integer.
k	optional vector of temporal aggregation levels to include. These must be factors of the seasonal periodicity, i.e., frequency(y) or y if it is an integer. The bottom level (k=1) is always included. If NULL, all feasible aggregation levels are used.

Value

S matrix. Rows are ordered from the top level (largest k) to the bottom level (k=1). The top level does not need to be a single node, for example Sthief(24,k=c(12,1)) gives two annual nodes over 24 months.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

Athanasopoulos, G., Hyndman, R. J., Kourentzes, N., & Petropoulos, F. (2017). Forecasting with temporal hierarchies. European Journal of Operational Research, 262(1), 60-74.

See Also

plotThief.

Examples

```
Sthief(AirPassengers)

# Only some aggregation levels: monthly, quarterly, and annual
Sthief(12,k=c(12,3))

# Two years of monthly data, with half-yearly, annual and 2-yearly levels
S <- Sthief(24,k=c(24,12,6))
plotThief(S,freq="monthly")
```

---

theta *Theta method*

Description

Estimate Theta method.

Forecast with fitted Theta method.

Produce a plot of the fitted Theta method.

Usage

```
theta(
  y,
  m = NULL,
  sign.level = 0.05,
  cost0 = c("MSE", "MdSE", "MAE", "MdAE"),
  cost2 = c("MSE", "MdSE", "MAE", "MdAE"),
  costs = c("MSE", "MdSE", "MAE", "MdAE"),
  season = c("multiplicative", "additive", "auto"),
  cma = NULL,
  outliers = NULL,
  multiplicative = NULL
)
```

```
## S3 method for class 'theta'
forecast(object, h = NULL, ...)

## S3 method for class 'theta'
plot(x, thetalines = c(TRUE, FALSE), h = 0, legend = TRUE, ...)

theta.thief(y, h = NULL, ...)
```

Arguments

y	input time series. Can be ts object.
m	seasonal period. If y is a ts object then the default is its frequency.
sign.level	significance level for trend and seasonality tests.
cost0	cost function of theta0 line. Can be: • "MSE": mean squared error. • "MdSE": median squared error. • "MAE": mean absolute error. • "MdAE": median absolute error.
cost2	cost function of theta2 line. Same options as cost0.
costs	cost function of seasonal element. Same options as cost0.
season	type of seasonal decomposition. This can be "multiplicative", "additive" or "auto". If y contains non-positive values then this is forced to "additive". With "auto" the type is chosen with mseastest. Whether seasonality is present at all is always tested (see details).
cma	input precalculated level/trend for the analysis. Use NULL to estimate internally.
outliers	provide vector of location of observations that are considered outliers (see residout). These will be considered in the estimation of theta0. For no outliers use NULL.
multiplicative	deprecated, use season instead. Accepts the same values as season. It is kept for backwards compatibility, gives a warning when used, and will be removed in a future version.
object	object of class theta.
h	forecast horizon. If h is NULL, then the horizon is set equal to the the seasonal frequency.
...	additional arguments passed to functions.
x	object of class theta.
thetalines	if TRUE, then theta lines are included in the plot.
legend	if TRUE a legend is added to the plot. Use FALSE to omit it, or a keyword such as "bottomleft" to change its position.

Details

This implementation of the Theta method tests automatically for seasonality and trend. Seasonal decomposition can be done either additively or multiplicatively and the seasonality is treated as a pure seasonal model. The various Theta components can be optimised using different cost functions. The originally proposed Theta method always assumed multiplicative seasonality and presence of trend, while all theta lines were optimised using MSE and seasonality was estimated using classical decomposition.

The argument `season` replaces the argument `multiplicative` of earlier versions. Calls that use `multiplicative` still work, but give a deprecation warning. The output element `multiplicative` is unchanged.

For `plot.theta`, if `h > 0` the forecasts are added to the plot, together with the forecasts of the theta lines if `thetalines=TRUE`. Outliers provided in `outliers` are marked on the data. Arguments in `...` control the plot: `col` sets the colours of the data, the fit and forecast, theta0, theta2 and the outliers, in this order (recycled as needed); `lwd` and `lty` the lines; everything else, such as `main`, `xlab`, `ylab`, `xlim` or `ylim`, is passed to `plot`. The default colours are from the `Set1` palette of `brewer.pal`.

Value

An object of class `theta`, containing:

- `"method"`: "Theta".
- `"y"`: the input time series.
- `"m"`: seasonal periods.
- `"exist"`: Statistical testing results, `exist[1]` is the result for trend, `exist[2]` is for season.
- `"multiplicative"`: If `TRUE`, then seasonality is modelled multiplicatively.
- `"theta0"`: fitted theta0 line values.
- `"theta2"`: fitted theta2 line values.
- `"season"`: fitted season values.
- `"x.out"`: modelled outliers.
- `"cost"`: cost functions for theta0, theta2 and season components.
- `"a"`: SES parameters of theta2.
- `"b"`: regression parameters of theta0.
- `"p"`: coefficients of outliers from theta0 and theta2 estimation.
- `"g"`: pure seasonal exponential smoothing parameters.
- `"fitted"`: fitted values.
- `"residuals"`: in-sample residuals.
- `"MSE"`: in-sample Mean Squared Error.
- `"outliers"`: location of outliers provided in `outliers`.

Functions

- `theta.thief()`: Wrapper function to use Theta with `thief`.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

- The original Theta method was proposed by: Assimakopoulos, V., & Nikolopoulos, K. (2000). The theta model: a decomposition approach to forecasting. International journal of forecasting, 16(4), 521-530. See details in how the implementation here differs.
- The THieF forecasting methodology used for `theta.thief` is proposed by: Athanasopoulos, G., Hyndman, R. J., Kourentzes, N., & Petropoulos, F. (2017). Forecasting with temporal hierarchies. European Journal of Operational Research, 262(1), 60-74.

Examples

```
fit <- theta(referrals)
plot(fit)

forecast.theta(fit,h=12) # Or simply use forecast(fit)

plot(fit,h=24)
plot(fit,thetalines=FALSE,h=12,main="Referrals",legend="bottomright")

## Not run:
library(thief)
thief(referrals,forecastfunction=theta.thief)

## End(Not run)
```

---

trendtest *Test a time series for trend*

Description

Test a time series for trend by either fitting exponential smoothing models and comparing then using the AICc, or by using the non-parametric Cox-Stuart test. The tests can be augmented by using multiple temporal aggregation.

Usage

```
trendtest(
  y,
  extract = c("FALSE", "TRUE"),
  type = c("aicc", "cs"),
  mta = c(FALSE, TRUE)
)
```

Arguments

y	a time series that must be of either ts or msts class.
extract	if TRUE then the centred moving average of the time series is calculated and the test is performed on that. Otherwise, the test is performed on the raw data.
type	type of test. Can be: • "aicc": test by comparing the AICc of exponential smoothing models. See details. • "cs": test by using the Cox-Stuart test. See details.<
mta	If TRUE augment testing by using Multiple Temporal Aggregation.

Details

All tests are performed at 5

Value

The function returns TRUE when there is evidence of trend and FALSE otherwise.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

The multiple temporal aggregation follows the construction approach suggested by Kourentzes, N., Petropoulos, F., & Trapero, J. R. (2014). Improving forecasting by estimating time series structural components across multiple frequencies. International Journal of Forecasting, 30(2), 291-302.

Examples

```
trendtest(referrals,TRUE)
```

---

tsutils *tsutils: Time Series Exploration, Modelling and Forecasting*

Description

The **tsutils** package provides functions to support various aspects of time series and forecasting modelling. In particular this package includes: (i) tests and visualisations that can help the modeller explore time series components and perform decomposition; (ii) modelling shortcuts, such as functions to construct lagmatrices and seasonal dummy variables of various forms; (iii) an implementation of the Theta method; (iv) tools to facilitate the design of the forecasting process, such as ABC-XYZ analyses; and (v) "quality of life" tools, such as treating time series for trailing and leading values.

Time series exploration

- cmav: centred moving average.
- coxstuart: Cox-Stuart test for location/dispersion.
- decomp: classical time series decomposition.
- seasplot: construct seasonal plots.
- seastest: seasonality test based on the Friedman test.
- trendtest: test a time series for trend.

Time series modelling

- getOptK: optimal temporal aggregation level for AR(1), MA(1), ARMA(1,1).
- lagmatrix: create leads/lags of variable.
- nemenyi: nonparametric multiple comparisons.
- residout: construct control chart of residuals.
- seasdummy: create seasonal dummies.
- theta: Theta method.

Hierarchical time series

- plotSthief: plot temporal hierarchy S matrix. Obsolete, replaced by plotThief.
- plotS: general plotting function for S-matrices. Calls plotSblock or plotStree.
- plotSblock: plot an S-matrix using the block representation.
- plotStree: plot an S-matrix using the standard tree representation.
- plotThief: plot temporal hierarchy S matrix. Calls plotS and handles temporal names automatically.
- Saugment: augment an incoherent hierarchy with discrepancy series.
- Sgroup: S matrix from dimensions.
- Smean: Mean (equal or unequal weights) aggregation S matrix.
- Sthief: temporal hierarchy S matrix.

Forecasting process modelling

- abc: ABC analysis.
- xyz: XYZ analysis.
- abcxyz: ABC-XYZ analyses visualisation.

Quality of life

- geomean: geometric mean.
- lambdaseq: generate sequence of lambda for LASSO regression.
- leadtrail: remove leading/training zeros/NAs.
- wins: winsorisation, including vectorised versions colWins and rowWins.

Time series data

- referrals: A&E monthly referrals.
- gdpchain: Hierarchy of US real GDP in chained dollars.

`wins` *Winsorise*

Description

Winsorise either by number or percentage of observations.

Usage

```
wins(x, p = 0.05)

colWins(x, p = 0.05)

rowWins(x, p = 0.05)
```

Arguments

x	input data. NAs will be removed.
p	percentage or number of observations to be winsorised. If value is <1 then it is used as a percentages. Otherwise it is the number of observations to winsorise. If the resulting p > floor((length(x)-1)/2), then it is set equal to floor((length(x)-1)/2).

Value

Winsorised vector.

Functions

- `colWins()`: Vectorised version of wins by columns.
- `rowWins()`: Vectorised version of wins by rows.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

Examples

```
x <- rnorm(100,mean=0,sd=1)
xW <- wins(x)
```

xyz *XYZ analysis*

Description

Perform XYZ analysis on a set of time series.

Usage

```
xyz(x, m = NULL, prc = c(0.2, 0.3, 0.5), type = c("naive", "ets", "cv"))
```

Arguments

x	this can either be an array, where each column is a series, or a vector of values. If x is a vector of values forecastability is not calculated and the input is used as such.
m	seasonal length for time series. Required when type is "naive" or "ets".
prc	a vector of percentages indicating how many items are included in each class. By default this is c(0.2,0.3,0.5), but any set of percentage values can be used as long as 0<=prc[i]<=1 and sum(prc)==1.
type	the type of forecastability calculation. This can be: • "naive": fit naive and seasonal naive and calculate forecastability using RMSE/mean level. • "ets": fit ets and calculate and calculate forecastability using RMSE/mean level. • "cv": use coefficient of variation as a proxy of forecastability.

Value

Return object of class abc and contains:

- value: a vector containing the forecastability value of each series.
- class: a vector containing the class membership of each series.
- rank: a vector containing the rank of each series, with 1 being the lowest forecastability series.
- conc: the forecastability concentration of each class, as percentage of total value.
- model: fitted model for each series.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>.

References

Ord K., Fildes R., Kourentzes N. (2017) Principles of Business Forecasting, 2e. *Wessex Press Publishing Co.*, p.515-518.

See Also

abc, plot.abc, abcxyz.

Examples

```
x <- abs(matrix(cumsum(rnorm(5400,0,1)),36,150))
z <- xyz(x,m=12)
print(z)
plot(z)
```

Index

∗ **Regression**
  lambdaseq, 14
∗ **cma**
  cmav, 5
∗ **colWins**
  wins, 53
∗ **data**
  gdpchain, 10
  referrals, 33
∗ **geomean**
  geomean, 11
∗ **htest**
  coxstuart, 7
  getOptK, 12
  nemenyi, 16
  nemenyi.sens, 19
  seastest, 40
  trendtest, 50
∗ **rowWins**
  wins, 53
∗ **ts**
  abc, 2
  abcxyz, 4
  cmav, 5
  decomp, 8
  lambdaseq, 14
  plotS, 22
  plotSblock, 23
  plotSthief, 26
  plotStree, 27
  plotThief, 30
  residout, 33
  Saugment, 34
  seasplot, 38
  seastest, 40
  Sgroup, 42
  Smean, 44
  Sthief, 46
  theta, 47
  xyz, 54
∗ **wins**
  wins, 53

abc, 2, *4*, *52*, *55*
abcxyz, *3*, 4, *52*, *55*
adjustcolor, *6*

brewer.pal, *49*

cmav, 5, *9*, *40, 41*, *52*
col2rgb, *6*
colour.blend, 6
colWins *(*wins*)*, 53
coxstuart, 7, *41*, *52*

decomp, 8, *40*, *52*

forecast.theta *(*theta*)*, 47

gdpchain, 10, *53*
geomean, 11, *52*
getOptK, 12, *52*

lagmatrix, 13, *52*
lambdaseq, 14, *52*
leadtrail, 15, *52*

mseastest, *41*, *48*

nemenyi, 16, *20, 21*, *52*
nemenyi.sens, 19

plot.abc, *55*
plot.abc *(*abc*)*, 2
plot.theta *(*theta*)*, 47
plotmath, *22*, *24*, *28*
plotS, *10, 11*, 22, *25*, *29, 30*, *32*, *35, 36*, *43, 44*, *52*
plotSblock, *22, 23*, 23, *27–32*, *52*
plotSthief, 26, *52*

plotStree, *22, 23*, *25*, 27, *30–32*, *52*
plotThief, *23*, *25*, *29*, 30, *47*, *52*

referrals, 33, *53*
residout, 33, *48*, *52*
rowWins *(*wins*)*, 53

Saugment, *10, 11*, 34, *44, 45*, *52*
seasdummy, 37, *52*
seasplot, *9*, 38, *40, 41*, *52*
seastest, *9*, *39, 40*, 40, *52*
Sgroup, 42, *45*, *52*
Smean, *11*, 44, *52*
Sthief, *31, 32*, *36*, *44, 45*, 46, *52*

theta, 47, *52*
thief, *49*
trendtest, *41*, 50, *52*
tsutils, 51

wins, *52*, 53

xyz, *3, 4*, *52*, 54