

Package ‘disaggR’

February 21, 2023

Type Package

Title Two-Steps Benchmarks for Time Series Disaggregation

Version 1.0.5

Description The twoStepsBenchmark() and threeRuleSmooth() functions allow you to disaggregate a low-frequency time series with higher frequency time series, using the French National Accounts methodology. The aggregated sum of the resulting time series is strictly equal to the low-frequency time series within the benchmarking window. Typically, the low-frequency time series is an annual one, unknown for the last year, and the high frequency one is either quarterly or monthly. See “Methodology of quarterly national accounts”, Insee Méthodes N°126, by Insee (2012, ISBN:978-2-11-068613-8, <<https://www.insee.fr/en/information/2579410>>).

Imports graphics, grDevices, methods, RColorBrewer (>= 1.1-2), stats, utils

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Suggests knitr, ggplot2 (>= 3.0.0), rmarkdown (>= 2.0.0), shiny (>= 1.5.0), shinytest2 (>= 0.1.0), testthat (>= 3.0.0), vdiff (>= 1.0.0)

Depends R (>= 3.6.0)

BugReports <https://github.com/InseeFr/disaggR/issues>

LazyData yes

Collate 'bflSmooth.R' 'data.R' 'disaggR.R' 'utils.R' 'in.R' 's4register.R' 'twoStepsBenchmark.R' 'methods.R' 'plot.R' 'praislm.R' 'reView.R' 'threeRuleSmooth.R'

Config/testthat/edition 3

VignetteBuilder knitr

URL <https://insee.fr.github.io/disaggR/>

NeedsCompilation no

Author Arnaud Feldmann [aut] (<<https://orcid.org/0000-0003-0109-7505>>, Author, creator and maintener of the package until the version 1.0.2),
 Franck Arnaud [ctb] (barplot base graphics method for the mts class),
 Thomas Laurent [cre],
 Institut national de la statistique et des études économiques [cph]
 (<https://www.insee.fr/>)

Maintainer Thomas Laurent <thomas.laurent@insee.fr>

Repository CRAN

Date/Publication 2023-02-21 00:10:02 UTC

R topics documented:

bf1Smooth	2
distance	3
in_disaggr	4
in_revisions	5
in_sample	6
in_scatter	7
plot.twoStepsBenchmark	8
rePort	11
reUseBenchmark	12
reView	13
threeRuleSmooth	14
twoStepsBenchmark	16

Index	20
--------------	-----------

bf1Smooth	<i>Smooth a time series</i>
-----------	-----------------------------

Description

bf1Smooth smoothes a time series into a time series of a higher frequency that exactly aggregates into the higher one. The process followed is Boot, Feibes and Lisman, which minimizes the squares of the variations.

Usage

```
bf1Smooth(lfserie, nfrequency, weights = NULL, lfserie.is.rate = FALSE)
```

Arguments

lfserie	a time series to be smoothed
nfrequency	the new high frequency. It must be a multiple of the low frequency.
weights	NULL or a time series of the same size than the expected high-frequency serie.
lfserie.is.rate	TRUE or FALSE. Only taken into account if weights isn't NULL.

Details

If `weights` isn't NULL the results depends of `lfserie.is.rate` :

- if FALSE the rate output/weights is smoothed with the constraint that the aggregated output is equal to the input `lfserie`.
- if TRUE the input `lfserie` is the rate to be smoothed, with the constraint that the low-frequency weighted means of the output are equal to `lfserie`.

Value

A time series of frequency `nfrequency`

distance	<i>Distance computation for disaggregations</i>
----------	---

Description

This function `distance` computes the Minkowski distance of exponent `p`, related to a `tscomparison` object, produced with `in_sample`, `in_disaggr` or `in_revisions`

Usage

```
distance(x, p = 2)
```

Arguments

<code>x</code>	an object of class <code>tscomparison</code>
<code>p</code>	an integer greater than 1L, or Inf.

Details

The meaning depends on the `tscomparison` function :

- `in_sample` will produce the low-frequency distance between the predicted value and the response, on the coefficient calculation window.
- `in_disaggr` will produce the high-frequency distance between the inputs (eventually, the sum of its contributions) and the benchmarked series.
- `in_revisions` will produce the high-frequency distance between the two benchmarked series (contributions distance isn't permitted).

Value

a numeric of length 1, the distance.

See Also

`in_sample` `in_disaggr` `in_revisions`

Examples

```
benchmark <- twoStepsBenchmark(turnover,construction,include.rho = TRUE)
distance(in_sample(benchmark,type="changes"))
distance(in_disaggr(benchmark,type="contributions"),p=1L)
distance(in_disaggr(benchmark,type="changes"),p=Inf)
```

in_disaggr

Comparing a disaggregation with the high-frequency input

Description

The function `in_disaggr` takes a [twoStepsBenchmark](#) or a [threeRuleSmooth](#) object as an input. It produces a comparison between the benchmarked time series and the high-frequency input.

Usage

```
in_disaggr(object, type = "changes")
```

Arguments

object	an object of class "twoStepsBenchmark" or "threeRuleSmooth".
type	"levels","levels-rebased", "changes" or "contributions". This defines the type of output.

Details

The functions `plot` and `autoplot` can be used on this object to produce graphics.

Value

a named matrix time series of two columns, one for the response and the other for the input. A `tscomparison` class is added to the object.

See Also

[in_sample](#) [in_revisions](#) [in_scatter](#) [plot.tscomparison](#)

Examples

```
benchmark <- twoStepsBenchmark(turnover,construction,include.rho = TRUE)
plot(in_disaggr(benchmark))
```

in_revisions	<i>Comparing two disaggregations together</i>
--------------	---

Description

The function `in_revisions` takes two inputs, [twoStepsBenchmark](#) or a [threeRuleSmooth](#), and produces a comparison between those.

Usage

```
in_revisions(object, object_old, type = "changes")
```

Arguments

<code>object</code>	an object of class "twoStepsBenchmark" or "threeRuleSmooth".
<code>object_old</code>	an object of class "twoStepsBenchmark" or "threeRuleSmooth".
<code>type</code>	"levels", "levels-rebased", "changes" or "contributions". This defines the type of output.

Details

The functions `plot` and `autoplot` can be used on this object to produce graphics.

Value

a named matrix time series of two columns, one for the response and the other for the predicted value. A `tscomparison` class is added to the object.

See Also

[in_sample](#) [in_disaggr](#) [in_scatter](#) [plot.tscomparison](#)

Examples

```
benchmark <- twoStepsBenchmark(turnover, construction, include.rho = TRUE)
benchmark2 <- twoStepsBenchmark(turnover, construction, include.differentiation = TRUE)
plot(in_revisions(benchmark, benchmark2))
```

`in_sample`*Producing the in sample predictions of a prais-lm regression*

Description

The function `in_sample` returns in-sample predictions from a [praislm](#) or a [twoStepsBenchmark](#) object.

Usage

```
in_sample(object, type = "changes")
```

Arguments

`object` an object of class "praislm" or "twoStepsBenchmark".
`type` "changes" or "levels". The results are either returned in changes or in levels.

Details

The functions `plot` and `autoplot` can be used on this object to produce graphics.

The predicted values are different from the fitted values :

- they are eventually reintegrated.
- they contain the autocorrelated part of the residuals.

Besides, changes are relative to the latest benchmark value, not the latest predicted value.

Value

a named matrix time series of two columns, one for the response and the other for the predicted value. A "tscomparison" class is added to the object.

See Also

[in_disagr](#) [in_revisions](#) [in_scatter](#) [plot.tscomparison](#)

Examples

```
benchmark <- twoStepsBenchmark(turnover,construction,include.rho = TRUE)  
plot(in_sample(benchmark))
```

`in_scatter`*Comparing the inputs of a praislm regression*

Description

The function `in_scatter` returns low-frequency comparisons of the inputs from a [praislm](#), a [twoStepsBenchmark](#) or [threeRuleSmooth](#).

Usage

```
in_scatter(  
  object,  
  type = if (model.list(object)$include.differentiation) "changes" else "levels"  
)
```

Arguments

<code>object</code>	an object of class "praislm", "twoStepsBenchmark" or "threeRuleSmooth".
<code>type</code>	"levels" or "changes". This defines the type of output. A differenced model can't have a scatterplot in levels.

Details

The functions `plot` and `autoplot` can be used on this object to produce graphics.

Value

a named matrix time series of two or three columns, one for the low-frequency serie and the others for the high-frequency series (eventually differentiated if `include.differentiation` is TRUE). A `tscomparison` class is added to the object. For a `twoStepsBenchmark` object, this matrix has three columns, for the low-frequency series, the high-frequency on the regression span and the high-frequency series on the benchmark span.

If outlier effects are estimated, the contributions of the outliers are substracted from the low-frequency series.

See Also

[in_sample](#) [in_disaggr](#) [in_revisions](#) [plot.tscomparison](#)

Examples

```
benchmark <- twoStepsBenchmark(turnover,construction,include.rho = TRUE)  
plot(in_scatter(benchmark))
```

plot.twoStepsBenchmark

Plotting disaggR objects

Description

Plot methods for objects of class "tscomparison", [threeRuleSmooth](#) and [twoStepsBenchmark](#). :

- plot draws a plot with base graphics
- autoplot produces a ggplot object

Objects of class tscomparison can be produced with the functions [in_sample](#), [in_scatter](#), [in_revisions](#), [in_disaggr](#).

Usage

```
## S3 method for class 'twoStepsBenchmark'
plot(
  x,
  xlab = NULL,
  ylab = NULL,
  start = NULL,
  end = NULL,
  col = default_col_pal(x),
  lty = default_lty_pal(x),
  show.legend = TRUE,
  main = NULL,
  mar = default_margins(main, xlab, ylab),
  ...
)

## S3 method for class 'threeRuleSmooth'
plot(
  x,
  xlab = NULL,
  ylab = NULL,
  start = NULL,
  end = NULL,
  col = default_col_pal(x),
  lty = default_lty_pal(x),
  show.legend = TRUE,
  main = NULL,
  mar = default_margins(main, xlab, ylab),
  ...
)

## S3 method for class 'tscomparison'
```



```
plot(  
  x,  
  xlab = NULL,  
  ylab = NULL,  
  start = NULL,  
  end = NULL,  
  col = default_col_pal(x),  
  lty = default_lty_pal(x),  
  show.legend = TRUE,  
  main = NULL,  
  mar = default_margins(main, xlab, ylab),  
  ...  
)  
  
autoplot.twoStepsBenchmark(  
  object,  
  xlab = NULL,  
  ylab = NULL,  
  start = NULL,  
  end = NULL,  
  col = default_col_pal(object),  
  lty = default_lty_pal(object),  
  show.legend = TRUE,  
  main = NULL,  
  mar = NULL,  
  theme = default_theme_ggplot(object, start, end, show.legend, xlab, ylab, mar),  
  ...  
)  
  
autoplot.threeRuleSmooth(  
  object,  
  xlab = NULL,  
  ylab = NULL,  
  start = NULL,  
  end = NULL,  
  col = default_col_pal(object),  
  lty = default_lty_pal(object),  
  show.legend = TRUE,  
  main = NULL,  
  mar = NULL,  
  theme = default_theme_ggplot(object, start, end, show.legend, xlab, ylab, mar),  
  ...  
)  
  
autoplot.tscomparison(  
  object,  
  xlab = NULL,  
  ylab = NULL,
```

```

    start = NULL,
    end = NULL,
    col = default_col_pal(object),
    lty = default_lty_pal(object),
    show.legend = TRUE,
    main = NULL,
    mar = NULL,
    theme = default_theme_ggplot(object, start, end, show.legend, xlab, ylab, mar),
    ...
  )

```

Arguments

<code>x</code>	(for the plot method) a <code>tscomparison</code> , a <code>twoStepsBenchmark</code> or a <code>threeRuleSmooth</code> .
<code>xlab</code>	the title for the x axis
<code>ylab</code>	the title for the y axis
<code>start</code>	a numeric of length 1 or 2. The start of the plot.
<code>end</code>	a numeric of length 1 or 2. The end of the plot.
<code>col</code>	the color scale applied on the plot. Could be a vector of colors, or a function from <code>n</code> to a color vector of size <code>n</code> .
<code>lty</code>	the linetype scales applied on the plot. Could be a vector of linetypes, or a function from <code>n</code> to a linetypes vector of size <code>n</code> .
<code>show.legend</code>	TRUE or FALSE. Should an automatic legend be added to the plot.
<code>main</code>	a character of length 1, the title of the plot
<code>mar</code>	a numeric of length 4, the margins of the plot specified in the form <code>c(bottom, left, top, right)</code> .
<code>...</code>	other arguments passed either to <code>ggplot</code> or <code>plot</code>
<code>object</code>	(for the <code>autoplot</code> method) a <code>tscomparison</code> , a <code>twoStepsBenchmark</code> or a <code>threeRuleSmooth</code> .
<code>theme</code>	a <code>ggplot</code> theme object to replace the default one (only for <code>autoplot</code> methods)

Value

NULL for the plot methods, the `ggplot` object for the `autoplot` methods

Examples

```

benchmark <- twoStepsBenchmark(turnover, construction, include.rho = TRUE)
plot(benchmark)
plot(in_sample(benchmark))
if(require("ggplot2")) {
  autoplot(in_disaggr(benchmark, type="changes"),
           start=c(2015,1),
           end=c(2020,12))
}
plot(in_scatter(benchmark), xlab="title x", ylab="title y")

```

`rePort`*Producing a report*

Description

This function takes an output of the [reView shiny](#) application and produces an html report with the same outputs than in shiny.

Usage

```
rePort(  
  object,  
  output_file = NULL,  
  launch.browser = if (is.null(output_file)) TRUE else FALSE,  
  hfserie_name = NULL,  
  lfserie_name = NULL,  
  ...  
)
```

Arguments

<code>object</code>	a <code>twoStepsBenchmark</code> with an univariate <code>hfserie</code> , a <code>reViewOutput</code> , or a character of length 1 with the path of their RDS file. If a <code>reViewOutput</code> is chosen, the former new benchmark is taken as the old one.
<code>output_file</code>	The file in which the html should be saved. If <code>NULL</code> the file is temporary, and opened in a tab of the default browser.
<code>launch.browser</code>	<code>TRUE</code> or <code>FALSE</code> . If <code>TRUE</code> , the output is opened in the browser. Defaults to <code>TRUE</code> if <code>output_file</code> is <code>NULL</code> .
<code>hfserie_name</code>	a language object or a character of length 1. The name of the <code>hfserie</code> , eventually its expression.
<code>lfserie_name</code>	a language object or a character of length 1. The name of the <code>lfserie</code> , eventually its expression.
<code>...</code>	other arguments passed to <code>rmarkdown::render</code>

Details

It can also directly take a [twoStepsBenchmark](#) as an input.

See Also

[reView](#)

reUseBenchmark	<i>Using an estimated benchmark model on another time series</i>
----------------	--

Description

This function reapplies the coefficients and parameters of a benchmark on new time series.

Usage

```
reUseBenchmark(hfserie,benchmark,reeval.smoothed.part=FALSE)
```

Arguments

hfserie	the bended time series. If it is a matrix time series, it has to have the same column names than the hfserie used for the benchmark.
benchmark	a twoStepsBenchmark object, from which the parameters and coefficients are taken.
reeval.smoothed.part	a boolean of length 1. If TRUE, the smoothed part is reevaluated, hence the aggregated benchmarked series is equal to the low-frequency series.

Details

reUseBenchmark is primarily meant to be used on a series that is derived from the previous one, after some modifications that would bias the estimation otherwise. Working-day adjustment is a good example. Hence, by default, the smoothed part of the first model isn't reevaluated ; the aggregated benchmarked series isn't equal to the low-frequency series.

Value

reUseBenchmark returns an object of class [twoStepsBenchmark](#).

Examples

```
benchmark <- twoStepsBenchmark(turnover,construction)
turnover_modif <- turnover
turnover_modif[2] <- turnover[2]+2
benchmark2 <- reUseBenchmark(turnover_modif,benchmark)
```

`reView`*A shiny app to reView and modify twoStepsBenchmarks*

Description

reView allows the user to easily access diverse outputs in order to review a benchmark object, made with [twoStepsBenchmark](#).

The `hfserie_name` and `lfserie_name` define :

Usage

```
reView(object, hfserie_name = NULL, lfserie_name = NULL, compare = TRUE)
```

Arguments

<code>object</code>	a <code>twoStepsBenchmark</code> with an univariate <code>hfserie</code> , a <code>reViewOutput</code> , or a character of length 1 with the path of their RDS file. If a <code>reViewOutput</code> is chosen, the former new benchmark is taken as the old one.
<code>hfserie_name</code>	a language object or a character of length 1. The name of the <code>hfserie</code> , eventually its expression.
<code>lfserie_name</code>	a language object or a character of length 1. The name of the <code>lfserie</code> , eventually its expression.
<code>compare</code>	a boolean of length 1, that tells if the outputs of the old benchmark should be displayed.

Details

- the default file name of the RDS file
- the names of the series in the output `call` element

By default, these are set as defined in their `call` element.

The app is made of **shiny** modules in order to make it easy to integrate it into a wider application. In the module part, every input are defined as reactive variables.

Value

a list, of class `reViewOutput`, containing the new benchmark, the old one, the names of the series and the boolean `compare`. This object can also be saved in RDS format through the app. The `reViewOutput` object can be displayed as a html report with the same informations than in shiny, with the [rePort](#) method.

See Also

[rePort](#)

Examples

```
## Not run:
reView(twoStepsBenchmark(turnover, construction))

## End(Not run)
```

threeRuleSmooth	<i>Bends a time series with a lower frequency one by smoothing their rate</i>
-----------------	---

Description

threeRuleSmooth bends a time series with a time series of a lower frequency. The procedure involved is a proportional Denton benchmark.

Therefore, the resulting time series is the product of the high frequency input with a smoothed rate. This latter is extrapolated through an arithmetic sequence.

The resulting time series is equal to the low-frequency series after aggregation within the benchmark window.

Usage

```
threeRuleSmooth(
  hfserie,
  lfserie,
  start.benchmark = NULL,
  end.benchmark = NULL,
  start.domain = NULL,
  end.domain = NULL,
  start.delta.rate = NULL,
  end.delta.rate = NULL,
  set.delta.rate = NULL,
  ...
)
```

Arguments

hfserie	the bended time series. It can be a matrix time series.
lfserie	a time series whose frequency divides the frequency of hfserie.
start.benchmark	an optional start for lfserie to bend hfserie. Should be a numeric of length 1 or 2, like a window for lfserie. If NULL, the start is defined by lfserie's window.
end.benchmark	an optional end for lfserie to bend hfserie. Should be a numeric of length 1 or 2, like a window for lfserie. If NULL, the start is defined by lfserie's window.

<code>start.domain</code>	an optional start of the output high-frequency series. It also defines the smoothing window : The low-frequency residuals will be extrapolated until they contain the smallest low-frequency window that is around the high-frequency domain window. Should be a numeric of length 1 or 2, like a window for <code>hfserie</code> . If NULL, the start is defined by <code>hfserie</code> 's window.
<code>end.domain</code>	an optional end of the output high-frequency series. It also defines the smoothing window : The low-frequency residuals will be extrapolated until they contain the smallest low-frequency window that is around the high-frequency domain window.
<code>start.delta.rate</code>	an optional start for the mean of the rate difference. It is required as a common difference for the arithmetical extrapolation of the rate. Should be a numeric of length 1 or 2, like a window for <code>lfserie</code> . If NULL, the start is defined by <code>lfserie</code> 's window.
<code>end.delta.rate</code>	an optional end for the mean of the rate difference. It is required as a common difference for the arithmetical extrapolation of the rate. Should be a numeric of length 1 or 2, like a window for <code>lfserie</code> . If NULL, the end is defined by <code>lfserie</code> 's window.
<code>set.delta.rate</code>	an optional double, that allows the user to set the delta mean instead of using a mean.
<code>...</code>	if the dots contain a <code>cl</code> item, its value overwrites the value of the returned call. This feature allows to build wrappers.

Details

In order to smooth the rate, `threeRuleSmooth` calls `bfiSmooth` and uses a modified and extrapolated version of `hfserie` as weights :

- only the full cycles are kept
- the first and last full cycles are replicated respectively backwards and forwards to fill the domain window.

Value

`threeRuleSmooth` returns an object of class "threeRuleSmooth".

The functions `plot` and `autoplot` (the generic from **ggplot2**) produce graphics of the benchmarked series and the bending series. The functions `in_disaggr`, `in_revisions`, `in_scatter` produce various comparisons on which `plot` and `autoplot` can also be used.

The generic accessor functions `as.ts`, `model.list`, `smoothed.rate` extract various useful features of the returned value.

An object of class "threeRuleSmooth" is a list containing the following components :

<code>benchmarked.serie</code>	a time series, that is the result of the benchmark.
<code>lfrate</code>	a time series, that is the low-frequency rate of the <code>threeRuleSmooth</code> .
<code>smoothed.rate</code>	the smoothed rate of the <code>threeRuleSmooth</code> .

<code>hfserie.as.weights</code>	the modified and extrapolated <code>hfserie</code> (see details).
<code>delta.rate</code>	the low-frequency delta of the rate, used to extrapolate the low-frequency rate time series. It is estimated as the mean value in the specified window.
<code>model.list</code>	a list containing all the arguments submitted to the function.
<code>call</code>	the matched call.

Examples

```
## How to use threeRuleSmooth

smooth <- threeRuleSmooth(hfserie = turnover,
                          lfserie = construction)

as.ts(smooth)
coef(smooth)
summary(smooth)
library(ggplot2)
autoplot(in_disaggr(smooth))
```

`twoStepsBenchmark` *Regress and bends a time series with a lower frequency one*

Description

`twoStepsBenchmark` bends a time series with a time series of a lower frequency. The procedure involved is a Prais-Winsten regression, then an additive Denton benchmark.

Therefore, the resulting time series is the sum of a regression fit and of a smoothed part. The smoothed part minimizes the sum of squares of its differences.

The resulting time series is equal to the low-frequency series after aggregation within the benchmark window.

Usage

```
twoStepsBenchmark(hfserie, lfserie, include.differentiation=FALSE, include.rho=FALSE,
                  set.coeff=NULL, set.const=NULL,
                  start.coeff.calc=NULL, end.coeff.calc=NULL,
                  start.benchmark=NULL, end.benchmark=NULL,
                  start.domain=NULL, end.domain=NULL, outliers=NULL,
                  ...)
```

```
annualBenchmark(hfserie, lfserie,
                include.differentiation=FALSE, include.rho=FALSE,
                set.coeff=NULL, set.const=NULL,
                start.coeff.calc=start(lfserie)[1L],
```



```

end.coeff.calc=end(lfserie)[1L],
start.benchmark=start(lfserie)[1L],
end.benchmark=end.coeff.calc[1L]+1L,
start.domain=start(hfserie),
end.domain=c(end.benchmark[1L]+2L,frequency(hfserie)),
outliers=NULL)

```

Arguments

<code>hfserie</code>	the bended time series. It can be a matrix time series.
<code>lfserie</code>	a time series whose frequency divides the frequency of <code>hfserie</code> .
<code>include.differentiation</code>	a boolean of length 1. If TRUE, <code>lfserie</code> and <code>hfserie</code> are differentiated before the estimation of the regression.
<code>include.rho</code>	a boolean of length 1. If TRUE, the regression includes an autocorrelation parameter for the residuals. The applied procedure is a Prais-Winsten estimation.
<code>set.coeff</code>	an optional numeric, that allows the user to set the regression coefficients instead of evaluating them. If <code>hfserie</code> is not a matrix, <code>set.coeff</code> can be an unnamed numeric of length 1. Otherwise, <code>set.coeff</code> has to be a named numeric, which will set the corresponding coefficients instead of evaluating them. Each column name of <code>hfserie</code> and each outlier set with the <code>outlier</code> arg initialize a coefficient with the same name, that can be set through <code>set.coeff</code> . The default name for a non-matrix time series is then "hfserie", By example, a LS2003 and the time series can be set using <code>set.coeff=c(hfserie=3,LS2003=1)</code> .
<code>set.const</code>	an optional numeric of length 1, that sets the regression constant. The constant is actually an automatically added column to <code>hfserie</code> . Using <code>set.constant=3</code> is equivalent to using <code>set.coeff=c(constant=3)</code> .
<code>start.coeff.calc</code>	an optional start for the estimation of the coefficients of the regression. Should be a numeric of length 1 or 2, like a window for <code>lfserie</code> . If NULL, the start is defined by <code>lfserie</code> 's window.
<code>end.coeff.calc</code>	an optional end for the estimation of the coefficients of the regression. Should be a numeric of length 1 or 2, like a window for <code>lfserie</code> . If NULL, the end is defined by <code>lfserie</code> 's window.
<code>start.benchmark</code>	an optional start for <code>lfserie</code> to bend <code>hfserie</code> . Should be a numeric of length 1 or 2, like a window for <code>lfserie</code> . If NULL, the start is defined by <code>lfserie</code> 's window.
<code>end.benchmark</code>	an optional end for <code>lfserie</code> to bend <code>hfserie</code> . Should be a numeric of length 1 or 2, like a window for <code>lfserie</code> . If NULL, the start is defined by <code>lfserie</code> 's window.
<code>start.domain</code>	an optional for the output high-frequency series. It also defines the smoothing window : The low-frequency residuals will be extrapolated until they contain the smallest low-frequency window that is around the high-frequency domain window. Should be a numeric of length 1 or 2, like a window for <code>hfserie</code> . If NULL, the start is defined by <code>hfserie</code> 's window.

<code>end.domain</code>	an optional end for the output high-frequency series. It also defines the smoothing window : The low-frequency residuals will be extrapolated until they contain the smallest low-frequency window that is around the high-frequency domain window. Should be a numeric of length 1 or 2, like a window for <code>hfserie</code> . If NULL, the start is defined by <code>hfserie</code> 's window.
<code>outliers</code>	an optional named list of numeric vectors, whose pattern is like <code>list(A02008T2=c(0,0,3,2),LS2002=c(</code> where : <ul style="list-style-type: none"> • "A0" stands for additive outlier or "LS" for level shift • The integer that follows stands for the outlier starting year • an optional integer, preceded by the letter T, stands for the low-frequency cycle of the outlier start. • The numeric vector values stands for the disaggregated value of the outlier and its length must be a multiple of <code>hf / lf</code> <p>The outliers coefficients are evaluated though the regression process, like any coefficient. Therefore, if any outlier is outside of the coefficient calculation window, it should be fixed using <code>set.coeff</code>.</p>
<code>...</code>	if the dots contain a <code>cl</code> item, its value overwrites the value of the returned call. This feature allows to build wrappers.

Details

`annualBenchmark` is a wrapper of the main function, that applies more specifically to annual series, and changes the default window parameters to the ones that are commonly used by quarterly national accounts.

Value

`twoStepsBenchmark` returns an object of class "twoStepsBenchmark".

The function `summary` can be used to obtain and print a summary of the regression used by the benchmark. The functions `plot` and `autoplot` (the generic from **ggplot2**) produce graphics of the benchmarked serie and the bending serie. The functions `in_disaggr`, `in_revisions`, `in_scatter` produce comparisons on which `plot` and `autoplot` can also be used.

The generic accessor functions `as.ts`, `prais`, `coefficients`, `residuals`, `fitted.values`, `model.list`, `se`, `rho` extract various useful features of the returned value.

An object of class "twoStepsBenchmark" is a list containing the following components :

<code>benchmarked.serie</code>	a time series, that is the result of the benchmark. It is equal to <code>fitted.values + smoothed.part</code> .
<code>fitted.values</code>	a time series, that is the high-frequency series as it is after having applied the regression coefficients. Compared to the fitted values of the regression, which can be retrieved inside the regression component, it has a high-frequency time series and can eventually be integrated if <code>include.differentiation</code> is TRUE.
<code>regression</code>	an object of class <code>praislm</code> , it is the regression on which relies the benchmark. It can be extracted with the function <code>prais</code>

Index

annualBenchmark (twoStepsBenchmark), 16
autoplot.threeRuleSmooth
 (plot.twoStepsBenchmark), 8
autoplot.tscomparison
 (plot.twoStepsBenchmark), 8
autoplot.twoStepsBenchmark
 (plot.twoStepsBenchmark), 8

bf1Smooth, 2, 15

distance, 3

in_disaggr, 4, 5–8, 15, 18
in_revisions, 4, 5, 6–8, 15, 18
in_sample, 4, 5, 6, 7, 8
in_scatter, 4–6, 7, 8, 15, 18

plot.threeRuleSmooth
 (plot.twoStepsBenchmark), 8
plot.tscomparison, 4–7
plot.tscomparison
 (plot.twoStepsBenchmark), 8
plot.twoStepsBenchmark, 8
prais, 18
praislm, 6, 7

rePort, 11, 13
reUseBenchmark, 12
reView, 11, 13

threeRuleSmooth, 4, 5, 7, 8, 14
twoStepsBenchmark, 4–8, 11–13, 16