Package 'strucchange'

May 9, 2026

Version 1.5-4

Date 2024-09-02

Title Testing, Monitoring, and Dating Structural Changes

Description Testing, monitoring and dating structural changes in (linear) regression models. strucchange features tests/methods from the generalized fluctuation test framework as well as from the F test (Chow test) framework. This includes methods to fit, plot and test fluctuation processes (e.g., CUSUM, MOSUM, recursive/moving estimates) and F statistics, respectively. It is possible to monitor incoming data online using fluctuation processes. Finally, the breakpoints in regression models with structural changes can be estimated together with confidence intervals. Emphasis is always given to methods for visualizing the data.

LazyData yes

Depends R (>= 2.10.0), zoo, sandwich

Suggests stats4, car, dynlm, e1071, foreach, lmtest, mvtnorm, tseries

Imports graphics, stats, utils

License GPL-2 | GPL-3

NeedsCompilation yes

Author Achim Zeileis [aut, cre] (ORCID: <https://orcid.org/0000-0003-0918-3766>), Friedrich Leisch [aut], Kurt Hornik [aut], Christian Kleiber [aut], Bruce Hansen [ctb], Edgar C. Merkle [ctb], Nikolaus Umlauf [ctb]

Maintainer Achim Zeileis <Achim.Zeileis@R-project.org>

Repository CRAN

Date/Publication 2024-09-02 08:00:02 UTC

Contents

`BostonHomicide` *Youth Homicides in Boston*

Description

Data about the number of youth homicides in Boston during the 'Boston Gun Project'—a policing initiative aiming at lowering homicide victimization among young people in Boston.

Usage

```
data("BostonHomicide")
```

Format

A data frame containing 6 monthly time series and two factors coding seasonality and year, respectively.

homicides time series. Number of youth homicides.

population time series. Boston population (aged 25-44), linearly interpolated from annual data.

populationBM time series. Population of black males (aged 15-24), linearly interpolated from annual data.

ahomicides25 time series. Number of adult homicides (aged 25 and older).

ahomicides35 time series. Number of adult homicides (aged 35-44).

unemploy time series. Teen unemployment rate (in percent).

season factor coding the month.

year factor coding the year.

Details

The 'Boston Gun Project' is a policing initiative aiming at lowering youth homicides in Boston. The project began in early 1995 and implemented the so-called 'Operation Ceasefire' intervention which began in the late spring of 1996.

Source

Piehl et al. (2004), Figure 1, Figure 3, and Table 1.

From the table it is not clear how the data should be linearly interpolated. Here, it was chosen to use the given observations for July of the corresponding year and then use `approx` with `rule = 2`.

References

Piehl A.M., Cooper S.J., Braga A.A., Kennedy D.M. (2003), Testing for Structural Breaks in the Evaluation of Programs, *The Review of Economics and Statistics*, **85**(3), 550-558.

Kennedy D.M., Piehl A.M., Braga A.A. (1996), Youth Violence in Boston: Gun Markets, Serious Youth Offenders, and a Use-Reduction Strategy, *Law and Contemporary Problems*, **59**, 147-183.

Examples

```
data("BostonHomicide")
attach(BostonHomicide)

## data from Table 1
tapply(homicides, year, mean)
populationBM[0:6*12 + 7]
tapply(ahomicides25, year, mean)
tapply(ahomicides35, year, mean)
population[0:6*12 + 7]
unemploy[0:6*12 + 7]

## model A
## via OLS
fmA <- lm(homicides ~ populationBM + season)
anova(fmA)
## as GLM
fmA1 <- glm(homicides ~ populationBM + season, family = poisson)
anova(fmA1, test = "Chisq")

## model B & C
fmB <- lm(homicides ~ populationBM + season + ahomicides25)
fmC <- lm(homicides ~ populationBM + season + ahomicides25 + unemploy)

detach(BostonHomicide)
```

---

boundary *Boundary Function for Structural Change Tests*

Description

A generic function computing boundaries for structural change tests

Usage

```
boundary(x, ...)
```

Arguments

x	an object. Use `methods` to see which `class` has a method for boundary.
...	additional arguments affecting the boundary.

Value

an object of class `"ts"` with the same time properties as the time series in `x`

See Also

`boundary.efp`, `boundary.mefp`, `boundary.Fstats`

boundary.efp *Boundary for Empirical Fluctuation Processes*

Description

Computes boundary for an object of class "efp"

Usage

```
## S3 method for class 'efp'
boundary(x, alpha = 0.05, alt.boundary = FALSE,
   functional = "max", ...)
```

Arguments

x	an object of class "efp".
alpha	numeric from interval (0,1) indicating the confidence level for which the boundary of the corresponding test will be computed.
alt.boundary	logical. If set to TRUE alternative boundaries (instead of the standard linear boundaries) will be computed (for Brownian bridge type processes only).
functional	indicates which functional should be applied to the empirical fluctuation process. See also plot.efp.
...	currently not used.

Value

an object of class "ts" with the same time properties as the process in x

See Also

efp, plot.efp

Examples

```
## Load dataset "nhtemp" with average yearly temperatures in New Haven
data("nhtemp")
## plot the data
plot(nhtemp)

## test the model null hypothesis that the average temperature remains constant
## over the years
## compute OLS-CUSUM fluctuation process
temp.cus <- efp(nhtemp ~ 1, type = "OLS-CUSUM")
## plot the process without boundaries
plot(temp.cus, alpha = 0.01, boundary = FALSE)
## add the boundaries in another colour
bound <- boundary(temp.cus, alpha = 0.01)
lines(bound, col=4)
lines(-bound, col=4)
```

boundary.Fstats *Boundary for F Statistics*

Description

Computes boundary for an object of class "Fstats"

Usage

```
## S3 method for class 'Fstats'
boundary(x, alpha = 0.05, pval = FALSE, aveF = FALSE,
    asymptotic = FALSE, ...)
```

Arguments

x	an object of class "Fstats".
alpha	numeric from interval (0,1) indicating the confidence level for which the boundary of the supF test will be computed.
pval	logical. If set to TRUE a boundary for the corresponding p values will be computed.
aveF	logical. If set to TRUE the boundary of the aveF (instead of the supF) test will be computed. The resulting boundary then is a boundary for the mean of the F statistics rather than for the F statistics themselves.
asymptotic	logical. If set to TRUE the asymptotic (chi-square) distribution instead of the exact (F) distribution will be used to compute the p values (only if pval is TRUE).
...	currently not used.

Value

an object of class "ts" with the same time properties as the time series in x

See Also

Fstats, plot.Fstats

Examples

```
## Load dataset "nhtemp" with average yearly temperatures in New Haven
data("nhtemp")
## plot the data
plot(nhtemp)

## test the model null hypothesis that the average temperature remains
## constant over the years for potential break points between 1941
## (corresponds to from = 0.5) and 1962 (corresponds to to = 0.85)
## compute F statistics
fs <- Fstats(nhtemp ~ 1, from = 0.5, to = 0.85)
```

```
## plot the p values without boundary
plot(fs, pval = TRUE, alpha = 0.01)
## add the boundary in another colour
lines(boundary(fs, pval = TRUE, alpha = 0.01), col = 2)
```

---

boundary.mefp *Boundary Function for Monitoring of Structural Changes*

Description

Computes boundary for an object of class "mefp"

Usage

```
## S3 method for class 'mefp'
boundary(x, ...)
```

Arguments

x	an object of class "mefp".
...	currently not used.

Value

an object of class "ts" with the same time properties as the monitored process

See Also

mefp, plot.mefp

Examples

```
df1 <- data.frame(y=rnorm(300))
df1[150:300,"y"] <- df1[150:300,"y"]+1
me1 <- mefp(y~1, data=df1[1:50,,drop=FALSE], type="ME", h=1,
            alpha=0.05)
me2 <- monitor(me1, data=df1)

plot(me2, boundary=FALSE)
lines(boundary(me2), col="green", lty="44")
```

breakdates *Breakdates Corresponding to Breakpoints*

Description

A generic function for computing the breakdates corresponding to breakpoints (and their confidence intervals).

Usage

```
breakdates(obj, format.times = FALSE, ...)
```

Arguments

obj	An object of class "breakpoints", "breakpointsfull" or their confidence intervals as returned by confint.
format.times	logical. If set to TRUE a vector of strings with the formatted breakdates. See details for more information.
...	currently not used.

Details

Breakpoints are the number of observations that are the last in one segment and breakdates are the corresponding points on the underlying time scale. The breakdates can be formatted which enhances readability in particular for quarterly or monthly time series. For example the breakdate 2002.75 of a monthly time series will be formatted to "2002(10)".

Value

A vector or matrix containing the breakdates.

See Also

breakpoints, confint

Examples

```
## Nile data with one breakpoint: the annual flows drop in 1898
## because the first Ashwan dam was built
data("Nile")
plot(Nile)

bp.nile <- breakpoints(Nile ~ 1)
summary(bp.nile)
plot(bp.nile)

## compute breakdates corresponding to the
## breakpoints of minimum BIC segmentation
```

```
breakdates(bp.nile)

## confidence intervals
ci.nile <- confint(bp.nile)
breakdates(ci.nile)
ci.nile

plot(Nile)
lines(ci.nile)
```

`breakfactor` *Factor Coding of Segmentations*

Description

Generates a factor encoding the segmentation given by a set of breakpoints.

Usage

```
breakfactor(obj, breaks = NULL, labels = NULL, ...)
```

Arguments

`obj`	An object of class `"breakpoints"` or `"breakpointsfull"` respectively.
`breaks`	an integer specifying the number of breaks to extract (only if `obj` is of class `"breakpointsfull"`), by default the minimum BIC partition is used.
`labels`	a vector of labels for the returned factor, by default the segments are numbered starting from `"segment1"`.
`...`	further arguments passed to `factor`.

Value

A factor encoding the segmentation.

See Also

`breakpoints`

Examples

```
## Nile data with one breakpoint: the annual flows drop in 1898
## because the first Ashwan dam was built
data("Nile")
plot(Nile)

## compute breakpoints
bp.nile <- breakpoints(Nile ~ 1)
```

```
## fit and visualize segmented and unsegmented model
fm0 <- lm(Nile ~ 1)
fm1 <- lm(Nile ~ breakfactor(bp.nile, breaks = 1))

lines(fitted(fm0), col = 3)
lines(fitted(fm1), col = 4)
lines(bp.nile, breaks = 1)
```

`breakpoints` *Dating Breaks*

Description

Computation of breakpoints in regression relationships. Given a number of breaks the function computes the optimal breakpoints.

Usage

```
## S3 method for class 'formula'
breakpoints(formula, h = 0.15, breaks = NULL,
    data = list(), hpc = c("none", "foreach"), ...)
## S3 method for class 'breakpointsfull'
breakpoints(obj, breaks = NULL, ...)
## S3 method for class 'breakpointsfull'
summary(object, breaks = NULL, sort = NULL,
    format.times = NULL, ...)
## S3 method for class 'breakpoints'
lines(x, breaks = NULL, lty = 2, ...)

## S3 method for class 'breakpointsfull'
coef(object, breaks = NULL, names = NULL, ...)
## S3 method for class 'breakpointsfull'
fitted(object, breaks = NULL, ...)
## S3 method for class 'breakpointsfull'
residuals(object, breaks = NULL, ...)
## S3 method for class 'breakpointsfull'
vcov(object, breaks = NULL, names = NULL,
    het.reg = TRUE, het.err = TRUE, vcov. = NULL, sandwich = TRUE, ...)
```

Arguments

`formula`	a symbolic description for the model in which breakpoints will be estimated.
`h`	minimal segment size either given as fraction relative to the sample size or as an integer giving the minimal number of observations in each segment.
`breaks`	positive integer specifying the maximal number of breaks to be calculated. By default the maximal number allowed by `h` is used.

data	an optional data frame containing the variables in the model. By default the variables are taken from the environment which breakpoints is called from.
hpc	a character specifying the high performance computing support. Default is "none", can be set to "foreach".
...	arguments passed to recresid.
obj, object	an object of class "breakpointsfull".
sort	logical. If set to TRUE summary tries to match the breakpoints from partitions with different numbers of breaks. The default tries to sort if a suitable matching can be found.
format.times	logical. If set to TRUE a vector of strings with the formatted breakdates is printed. See breakdates for more information.
x	an object of class "breakpoints".
lty	line type.
names	a character vector giving the names of the segments. If of length 1 it is taken to be a generic prefix, e.g. "segment".
het.reg	logical. Should heterogeneous regressors be assumed? If set to FALSE the distribution of the regressors is assumed to be homogeneous over the segments.
het.err	logical. Should heterogeneous errors be assumed? If set to FALSE the distribution of the errors is assumed to be homogeneous over the segments.
vcov.	a function to extract the covariance matrix for the coefficients of a fitted model of class "lm".
sandwich	logical. Is the function vcov. the sandwich estimator or only the middle part?

Details

All procedures in this package are concerned with testing or assessing deviations from stability in the classical linear regression model

$$y_i = x_i^\top \beta + u_i$$

In many applications it is reasonable to assume that there are m breakpoints, where the coefficients shift from one stable regression relationship to a different one. Thus, there are $m + 1$ segments in which the regression coefficients are constant, and the model can be rewritten as

$$y_i = x_i^\top \beta_j + u_i \qquad (i = i_{j-1} + 1, \ldots, i_j, \quad j = 1, \ldots, m + 1)$$

where j denotes the segment index. In practice the breakpoints i_j are rarely given exogenously, but have to be estimated. breakpoints estimates these breakpoints by minimizing the residual sum of squares (RSS) of the equation above.

The foundation for estimating breaks in time series regression models was given by Bai (1994) and was extended to multiple breaks by Bai (1997ab) and Bai & Perron (1998). breakpoints implements the algorithm described in Bai & Perron (2003) for simultaneous estimation of multiple breakpoints. The distribution function used for the confidence intervals for the breakpoints is given in Bai (1997b). The ideas behind this implementation are described in Zeileis et al. (2003).

The algorithm for computing the optimal breakpoints given the number of breaks is based on a dynamic programming approach. The underlying idea is that of the Bellman principle. The main computational effort is to compute a triangular RSS matrix, which gives the residual sum of squares for a segment starting at observation i and ending at i' with $i < i'$.

Given a `formula` as the first argument, `breakpoints` computes an object of class `"breakpointsfull"` which inherits from `"breakpoints"`. This contains in particular the triangular RSS matrix and functions to extract an optimal segmentation. A `summary` of this object will give the breakpoints (and associated) breakdates for all segmentations up to the maximal number of breaks together with the associated RSS and BIC. These will be plotted if `plot` is applied and thus visualize the minimum BIC estimator of the number of breakpoints. From an object of class `"breakpointsfull"` an arbitrary number of `breaks` (admissible by the minimum segment size `h`) can be extracted by another application of `breakpoints`, returning an object of class `"breakpoints"`. This contains only the breakpoints for the specified number of breaks and some model properties (number of observations, regressors, time series properties and the associated RSS) but not the triangular RSS matrix and related extractor functions. The set of breakpoints which is associated by default with a `"breakpointsfull"` object is the minimum BIC partition.

Breakpoints are the number of observations that are the last in one segment, it is also possible to compute the corresponding `breakdates` which are the breakpoints on the underlying time scale. The breakdates can be formatted which enhances readability in particular for quarterly or monthly time series. For example the breakdate `2002.75` of a monthly time series will be formatted to `"2002(10)"`. See `breakdates` for more details.

From a `"breakpointsfull"` object confidence intervals for the breakpoints can be computed using the method of `confint`. The breakdates corresponding to the breakpoints can again be computed by `breakdates`. The breakpoints and their confidence intervals can be visualized by `lines`. Convenience functions are provided for extracting the coefficients and covariance matrix, fitted values and residuals of segmented models.

The log likelihood as well as some information criteria can be computed using the methods for the `logLik` and `AIC`. As for linear models the log likelihood is computed on a normal model and the degrees of freedom are the number of regression coefficients multiplied by the number of segments plus the number of estimated breakpoints plus 1 for the error variance. More details can be found on the help page of the method `logLik.breakpoints`.

As the maximum of a sequence of F statistics is equivalent to the minimum OLS estimator of the breakpoint in a 2-segment partition it can be extracted by `breakpoints` from an object of class `"Fstats"` as computed by `Fstats`. However, this cannot be used to extract a larger number of breakpoints.

For illustration see the commented examples below and Zeileis et al. (2003).

Optional support for high performance computing is available, currently using `foreach` for the dynamic programming algorithm. If `hpc = "foreach"` is to be used, a parallel backend should be registered before. See `foreach` for more information.

value

An object of class `"breakpoints"` is a list with the following elements:

breakpoints the breakpoints of the optimal partition with the number of breaks specified (set to `NA` if the optimal 1-segment solution is reported),

RSS the associated RSS,

nobs the number of observations,

nreg the number of regressors,

call the function call,

datatsp the time series properties `tsp` of the data, if any, `c(1/nobs, 1, nobs)` otherwise.

If applied to a `formula` as first argument, `breakpoints` returns an object of class `"breakpointsfull"` (which inherits from `"breakpoints"`), that contains some additional (or slightly different) elements such as:

breakpoints the breakpoints of the minimum BIC partition,

RSS a function which takes two arguments `i,j` and computes the residual sum of squares for a segment starting at observation `i` and ending at `j` by looking up the corresponding element in the triangular RSS matrix `RSS.triang`,

RSS.triang a list encoding the triangular RSS matrix.

References

Bai J. (1994), Least Squares Estimation of a Shift in Linear Processes, *Journal of Time Series Analysis*, **15**, 453-472.

Bai J. (1997a), Estimating Multiple Breaks One at a Time, *Econometric Theory*, **13**, 315-352.

Bai J. (1997b), Estimation of a Change Point in Multiple Regression Models, *Review of Economics and Statistics*, **79**, 551-563.

Bai J., Perron P. (1998), Estimating and Testing Linear Models With Multiple Structural Changes, *Econometrica*, **66**, 47-78.

Bai J., Perron P. (2003), Computation and Analysis of Multiple Structural Change Models, *Journal of Applied Econometrics*, **18**, 1-22.

Zeileis A., Kleiber C., Krämer W., Hornik K. (2003), Testing and Dating of Structural Changes in Practice, *Computational Statistics and Data Analysis*, **44**, 109-123. doi:10.1016/S0167-9473(03)00030-6.

Zeileis A., Shah A., Patnaik I. (2010), Testing, Monitoring, and Dating Structural Changes in Exchange Rate Regimes, *Computational Statistics and Data Analysis*, **54**(6), 1696–1706. doi:10.1016/j.csda.2009.12.005.

Examples

```
## Nile data with one breakpoint: the annual flows drop in 1898
## because the first Ashwan dam was built
data("Nile")
plot(Nile)

## F statistics indicate one breakpoint
fs.nile <- Fstats(Nile ~ 1)
plot(fs.nile)
breakpoints(fs.nile)
lines(breakpoints(fs.nile))

## or
bp.nile <- breakpoints(Nile ~ 1)
```

```
summary(bp.nile)

## the BIC also chooses one breakpoint
plot(bp.nile)
breakpoints(bp.nile)

## fit null hypothesis model and model with 1 breakpoint
fm0 <- lm(Nile ~ 1)
fm1 <- lm(Nile ~ breakfactor(bp.nile, breaks = 1))
plot(Nile)
lines(ts(fitted(fm0), start = 1871), col = 3)
lines(ts(fitted(fm1), start = 1871), col = 4)
lines(bp.nile)

## confidence interval
ci.nile <- confint(bp.nile)
ci.nile
lines(ci.nile)


## UK Seatbelt data: a SARIMA(1,0,0)(1,0,0)_12 model
## (fitted by OLS) is used and reveals (at least) two
## breakpoints - one in 1973 associated with the oil crisis and
## one in 1983 due to the introduction of compulsory
## wearing of seatbelts in the UK.
data("UKDriverDeaths")
seatbelt <- log10(UKDriverDeaths)
seatbelt <- cbind(seatbelt, lag(seatbelt, k = -1), lag(seatbelt, k = -12))
colnames(seatbelt) <- c("y", "ylag1", "ylag12")
seatbelt <- window(seatbelt, start = c(1970, 1), end = c(1984,12))
plot(seatbelt[,"y"], ylab = expression(log[10](casualties)))

## testing
re.seat <- efp(y ~ ylag1 + ylag12, data = seatbelt, type = "RE")
plot(re.seat)

## dating
bp.seat <- breakpoints(y ~ ylag1 + ylag12, data = seatbelt, h = 0.1)
summary(bp.seat)
lines(bp.seat, breaks = 2)

## minimum BIC partition
plot(bp.seat)
breakpoints(bp.seat)
## the BIC would choose 0 breakpoints although the RE and supF test
## clearly reject the hypothesis of structural stability. Bai &
## Perron (2003) report that the BIC has problems in dynamic regressions.
## due to the shape of the RE process of the F statistics choose two
## breakpoints and fit corresponding models
bp.seat2 <- breakpoints(bp.seat, breaks = 2)
fm0 <- lm(y ~ ylag1 + ylag12, data = seatbelt)
fm1 <- lm(y ~ breakfactor(bp.seat2)/(ylag1 + ylag12) - 1, data = seatbelt)
```

```
## plot
plot(seatbelt[,"y"], ylab = expression(log[10](casualties)))
time.seat <- as.vector(time(seatbelt))
lines(time.seat, fitted(fm0), col = 3)
lines(time.seat, fitted(fm1), col = 4)
lines(bp.seat2)

## confidence intervals
ci.seat2 <- confint(bp.seat, breaks = 2)
ci.seat2
lines(ci.seat2)
```

---

catL2BB *Generators for efpFunctionals along Categorical Variables*

Description

Generators for `efpFunctional` objects suitable for aggregating empirical fluctuation processes to test statistics along (ordinal) categorical variables.

Usage

```
catL2BB(freq)
ordL2BB(freq, nproc = NULL, nrep = 1e5, probs = c(0:84/100, 850:1000/1000), ...)
ordwmax(freq, algorithm = mvtnorm::GenzBretz(), ...)
```

Arguments

`freq`	object specifying the category frequencies for the categorical variable to be used for aggregation: either a `gefp` object, a `factor`, or a numeric vector with either absolute or relative category frequencies.
`nproc`	numeric. Number of processes used for simulating from the asymptotic distribution (passed to `efpFunctional`). If `feq` is a `gefp` object, then its number of processes is used by default.
`nrep`	numeric. Number of replications used for simulating from the asymptotic distribution (passed to `efpFunctional`).
`probs`	numeric vector specifying for which probabilities critical values should be tabulated.
`...`	further arguments passed to `efpFunctional`.
`algorithm`	algorithm specification passed to `pmvnorm` for computing the asymptotic distribution.

Details

Merkle, Fan, and Zeileis (2014) discuss three functionals that are suitable for aggregating empirical fluctuation processes along categorical variables, especially ordinal variables. The functions `catL2BB`, `ordL2BB`, and `ordwmax` all require a specification of the relative frequencies within each category (which can be computed from various specifications, see arguments). All of them employ `efpFunctional` (Zeileis 2006) internally to set up an object that can be employed with `gefp` fluctuation processes.

`catL2BB` results in a chi-squared test. This is essentially the LM test counterpart to the likelihood ratio test that assesses a split into unordered categories.

`ordL2BB` is the ordinal counterpart to `supLM` where aggregation is done along the ordered categories (rather than continuously). The asymptotic distribution is non-standard and needs to be simulated for every combination of frequencies and number of processes. Hence, this is somewhat more time-consuming compared to the closed-form solution employed in `catL2BB`. It is also possible to store the result of `ordL2BB` in case it needs to be applied several `gefp` fluctuation processes.

`ordwmax` is a weighted double maximum test based on ideas previously suggested by Hothorn and Zeileis (2008) in the context of maximally selected statistics. The asymptotic distribution is (multivariate) normal and computed by means of `pmvnorm`.

Value

An object of class `efpFunctional`.

References

Hothorn T., Zeileis A. (2008), Generalized Maximally Selected Statistics. *Biometrics*, **64**, 1263–1269.

Merkle E.C., Fan J., Zeileis A. (2014), Testing for Measurement Invariance with Respect to an Ordinal Variable. *Psychometrika*, **79**(4), 569–584. doi:10.1007/S11336-013-9376-7.

Zeileis A. (2006), Implementing a Class of Structural Change Tests: An Econometric Computing Approach. *Computational Statistics & Data Analysis*, **50**, 2987–3008. doi:10.1016/j.csda.2005.07.001.

See Also

`efpFunctional`, `gefp`

Examples

```
## artificial data
set.seed(1)
d <- data.frame(
  x = runif(200, -1, 1),
  z = factor(rep(1:4, each = 50)),
  err = rnorm(200)
)
d$y <- rep(c(0.5, -0.5), c(150, 50)) * d$x + d$err

## empirical fluctuation process
scus <- gefp(y ~ x, data = d, fit = lm, order.by = ~ z)
```

```
## chi-squared-type test (unordered LM-type test)
LMuo <- catL2BB(scus)
plot(scus, functional = LMuo)
sctest(scus, functional = LMuo)

## ordinal maxLM test (with few replications only to save time)
maxLMo <- ordL2BB(scus, nrep = 10000)
plot(scus, functional = maxLMo)
sctest(scus, functional = maxLMo)

## ordinal weighted double maximum test
WDM <- ordwmax(scus)
plot(scus, functional = WDM)
sctest(scus, functional = WDM)
```

`confint.breakpointsfull`

Confidence Intervals for Breakpoints

Description

Computes confidence intervals for breakpoints.

Usage

```
## S3 method for class 'breakpointsfull'
confint(object, parm = NULL, level = 0.95,
   breaks = NULL, het.reg = TRUE, het.err = TRUE, vcov. = NULL, sandwich = TRUE, ...)
## S3 method for class 'confint.breakpoints'
lines(x, col = 2, angle = 90, length = 0.05,
    code = 3, at = NULL, breakpoints = TRUE, ...)
```

Arguments

`object`	an object of class `"breakpointsfull"` as computed by `breakpoints` from a `formula`.
`parm`	the same as `breaks`, only one of the two should be specified.
`level`	the confidence level required.
`breaks`	an integer specifying the number of breaks to be used. By default the breaks of the minimum BIC partition are used.
`het.reg`	logical. Should heterogeneous regressors be assumed? If set to `FALSE` the distribution of the regressors is assumed to be homogeneous over the segments.
`het.err`	logical. Should heterogeneous errors be assumed? If set to `FALSE` the distribution of the errors is assumed to be homogeneous over the segments.
`vcov.`	a function to extract the covariance matrix for the coefficients of a fitted model of class `"lm"`.

sandwich	logical. Is the function vcov. the sandwich estimator or only the middle part?
x	an object of class "confint.breakpoints" as returned by confint.
col, angle, length, code	arguments passed to arrows.
at	position on the y axis, where the confidence arrows should be drawn. By default they are drawn at the bottom of the plot.
breakpoints	logical. If TRUE vertical lines for the breakpoints are drawn.
...	*currently not used.*

Details

As the breakpoints are integers (observation numbers) the corresponding confidence intervals are also rounded to integers.

The distribution function used for the computation of confidence intervals of breakpoints is given in Bai (1997). The procedure, in particular the usage of heterogeneous regressors and/or errors, is described in more detail in Bai & Perron (2003).

The breakpoints should be computed from a formula with breakpoints, then the confidence intervals for the breakpoints can be derived by confint and these can be visualized by lines. For an example see below.

Value

A matrix containing the breakpoints and their lower and upper confidence boundary for the given level.

References

Bai J. (1997), Estimation of a Change Point in Multiple Regression Models, *Review of Economics and Statistics*, **79**, 551-563.

Bai J., Perron P. (2003), Computation and Analysis of Multiple Structural Change Models, *Journal of Applied Econometrics*, **18**, 1-22.

See Also

breakpoints

Examples

```
## Nile data with one breakpoint: the annual flows drop in 1898
## because the first Ashwan dam was built
data("Nile")
plot(Nile)

## dating breaks
bp.nile <- breakpoints(Nile ~ 1)
ci.nile <- confint(bp.nile, breaks = 1)
lines(ci.nile)
```

DJIA *Dow Jones Industrial Average*

Description

Weekly closing values of the Dow Jones Industrial Average.

Usage

```
data("DJIA")
```

Format

A weekly univariate time series of class "zoo" from 1971-07-01 to 1974-08-02.

Source

Appendix A in Hsu (1979).

References

Hsu D. A. (1979), Detecting Shifts of Parameter in Gamma Sequences with Applications to Stock Price and Air Traffic Flow Analysis, *Journal of the American Statistical Association*, **74**, 31–40.

Examples

```
data("DJIA")
## look at log-difference returns
djia <- diff(log(DJIA))
plot(djia)

## convenience functions
## set up a normal regression model which
## explicitely also models the variance
normlm <- function(formula, data = list()) {
  rval <- lm(formula, data = data)
  class(rval) <- c("normlm", "lm")
  return(rval)
}
estfun.normlm <- function(obj) {
  res <- residuals(obj)
  ef <- NextMethod(obj)
  sigma2 <- mean(res^2)
  rval <- cbind(ef, res^2 - sigma2)
  colnames(rval) <- c(colnames(ef), "(Variance)")
  return(rval)
}

## normal model (with constant mean and variance) for log returns
```

```
m1 <- gefp(djia ~ 1, fit = normlm, vcov = meatHAC, sandwich = FALSE)
plot(m1, aggregate = FALSE)
## suggests a clear break in the variance (but not the mean)

## dating
bp <- breakpoints(I(djia^2) ~ 1)
plot(bp)
## -> clearly one break
bp
time(djia)[bp$breakpoints]

## visualization
plot(djia)
abline(v = time(djia)[bp$breakpoints], lty = 2)
lines(time(djia)[confint(bp)$confint[c(1,3)]], rep(min(djia), 2), col = 2, type = "b", pch = 3)
```

---

durab *US Labor Productivity*

Description

US labor productivity in the manufacturing/durables sector.

Usage

```
data("durab")
```

Format

durab is a multivariate monthly time series from 1947(3) to 2001(4) with variables

y growth rate of the Industrial Production Index to average weekly labor hours in the manufacturing/durables sector,

lag lag 1 of the series y,

Source

The data set is available from Bruce Hansen's homepage https://www.ssc.wisc.edu/~bhansen/. For more information see Hansen (2001).

References

Hansen B. (2001), The New Econometrics of Structural Change: Dating Breaks in U.S. Labor Productivity, *Journal of Economic Perspectives*, **15**, 117–128.

Zeileis A., Leisch F., Kleiber C., Hornik K. (2005), Monitoring Structural Change in Dynamic Econometric Models, *Journal of Applied Econometrics*, **20**, 99–121.

Examples

```
data("durab")
## use AR(1) model as in Hansen (2001) and Zeileis et al. (2005)
durab.model <- y ~ lag

## historical tests
## OLS-based CUSUM process
ols <- efp(durab.model, data = durab, type = "OLS-CUSUM")
plot(ols)
## F statistics
fs <- Fstats(durab.model, data = durab, from = 0.1)
plot(fs)


## F statistics based on heteroskadisticy-consistent covariance matrix
fsHC <- Fstats(durab.model, data = durab, from = 0.1,
       vcov = function(x, ...) vcovHC(x, type = "HC", ...))
plot(fsHC)


## monitoring
Durab <- window(durab, start=1964, end = c(1979, 12))
ols.efp <- efp(durab.model, type = "OLS-CUSUM", data = Durab)
newborder <- function(k) 1.723 * k/192
ols.mefp <- mefp(ols.efp, period=2)
ols.mefp2 <- mefp(ols.efp, border=newborder)
Durab <- window(durab, start=1964)
ols.mon <- monitor(ols.mefp)
ols.mon2 <- monitor(ols.mefp2)
plot(ols.mon)
lines(boundary(ols.mon2), col = 2)
## Note: critical value for linear boundary taken from Table III
## in Zeileis et al. 2005: (1.568 + 1.896)/2 = 1.732 is a linear
## interpolation between the values for T = 2 and T = 3 at
## alpha = 0.05. A typo switched 1.732 to 1.723.


## dating
bp <- breakpoints(durab.model, data = durab)
summary(bp)
plot(summary(bp))

plot(ols)
lines(breakpoints(bp, breaks = 1), col = 3)
lines(breakpoints(bp, breaks = 2), col = 4)
plot(fs)
lines(breakpoints(bp, breaks = 1), col = 3)
lines(breakpoints(bp, breaks = 2), col = 4)
```

efp *Empirical Fluctuation Processes*

Description

Computes an empirical fluctuation process according to a specified method from the generalized fluctuation test framework, which includes CUSUM and MOSUM tests based on recursive or OLS residuals, parameter estimates or ML scores (OLS first order conditions).

Usage

```
efp(formula, data, type = , h = 0.15,
    dynamic = FALSE, rescale = TRUE, lrvar = FALSE, vcov = NULL)
```

Arguments

formula	a symbolic description for the model to be tested.
data	an optional data frame containing the variables in the model. By default the variables are taken from the environment which efp is called from.
type	specifies which type of fluctuation process will be computed, the default is "Rec-CUSUM". For details see below.
h	a numeric from interval (0,1) specifying the bandwidth. determines the size of the data window relative to sample size (for MOSUM and ME processes only).
dynamic	logical. If TRUE the lagged observations are included as a regressor.
rescale	logical. If TRUE the estimates will be standardized by the regressor matrix of the corresponding subsample according to Kuan & Chen (1994); if FALSE the whole regressor matrix will be used. (only if type is either "RE" or "ME")
lrvar	logical or character. Should a long-run variance estimator be used for the residuals? By default, the standard OLS variance is employed. Alternatively, lrvar can be used. If lrvar is character ("Andrews" or "Newey-West"), then the corresponding type of long-run variance is used. (The argument is ignored for the score-based tests where gefp should be used instead.)
vcov	a function to extract the covariance matrix for the coefficients of the fitted model (only for "RE" and "ME").

Details

If type is one of "Rec-CUSUM", "OLS-CUSUM", "Rec-MOSUM" or "OLS-MOSUM" the function efp will return a one-dimensional empirical process of sums of residuals. Either it will be based on recursive residuals or on OLS residuals and the process will contain CUmulative SUMs or MOving SUMs of residuals in a certain data window. For the MOSUM and ME processes all estimations are done for the observations in a moving data window, whose size is determined by h and which is shifted over the whole sample.

If type is either "RE" or "ME" a k-dimensional process will be returned, if k is the number of regressors in the model, as it is based on recursive OLS estimates of the regression coefficients

or moving OLS estimates respectively. The recursive estimates test is also called fluctuation test, therefore setting `type` to `"fluctuation"` was used to specify it in earlier versions of strucchange. It still can be used now, but will be forced to `"RE"`.

If `type` is `"Score-CUSUM"` or `"Score-MOSUM"` a *k+1*-dimensional process will be returned, one for each score of the regression coefficients and one for the scores of the variance. The process gives the decorrelated cumulative sums of the ML scores (in a Gaussian model) or first order conditions respectively (in an OLS framework).

If there is a single structural change point t^*, the recursive CUSUM path starts to depart from its mean 0 at t^*. The Brownian bridge type paths will have their respective peaks around t^*. The Brownian bridge increments type paths should have a strong change at t^*.

The function `plot` has a method to plot the empirical fluctuation process; with `sctest` the corresponding test on structural change can be performed.

Value

`efp` returns a list of class `"efp"` with components including:

`process`	the fitted empirical fluctuation process of class `"ts"` or `"mts"` respectively,
`type`	a string with the `type` of the process fitted,
`nreg`	the number of regressors,
`nobs`	the number of observations,
`par`	the bandwidth `h` used.

References

Brown R.L., Durbin J., Evans J.M. (1975), Techniques for testing constancy of regression relationships over time, *Journal of the Royal Statistical Society*, B, **37**, 149-163.

Chu C.-S., Hornik K., Kuan C.-M. (1995), MOSUM tests for parameter constancy, *Biometrika*, **82**, 603-617.

Chu C.-S., Hornik K., Kuan C.-M. (1995), The moving-estimates test for parameter stability, *Econometric Theory*, **11**, 669-720.

Hansen B. (1992), Testing for Parameter Instability in Linear Models, *Journal of Policy Modeling*, **14**, 517-533.

Hjort N.L., Koning A. (2002), Tests for Constancy of Model Parameters Over Time, *Nonparametric Statistics*, **14**, 113-132.

Krämer W., Ploberger W., Alt R. (1988), Testing for structural change in dynamic models, *Econometrica*, **56**, 1355-1369.

Kuan C.-M., Hornik K. (1995), The generalized fluctuation test: A unifying view, *Econometric Reviews*, **14**, 135 - 161.

Kuan C.-M., Chen (1994), Implementing the fluctuation and moving estimates tests in dynamic econometric models, *Economics Letters*, **44**, 235-239.

Ploberger W., Krämer W. (1992), The CUSUM test with OLS residuals, *Econometrica*, **60**, 271-285.

Zeileis A., Leisch F., Hornik K., Kleiber C. (2002), strucchange: An R Package for Testing for Structural Change in Linear Regression Models, *Journal of Statistical Software*, **7**(2), 1-38. doi:10.18637/jss.v007.i02.

Zeileis A. (2005), A Unified Approach to Structural Change Tests Based on ML Scores, F Statistics, and OLS Residuals. *Econometric Reviews*, **24**, 445–466. doi:10.1080/07474930500406053.

Zeileis A. (2006), Implementing a Class of Structural Change Tests: An Econometric Computing Approach. *Computational Statistics & Data Analysis*, **50**, 2987–3008. doi:10.1016/j.csda.2005.07.001.

Zeileis A., Hornik K. (2007), Generalized M-Fluctuation Tests for Parameter Instability, *Statistica Neerlandica*, **61**, 488–508. doi:10.1111/j.14679574.2007.00371.x.

See Also

gefp, plot.efp, print.efp, sctest.efp, boundary.efp

Examples

```
## Nile data with one breakpoint: the annual flows drop in 1898
## because the first Ashwan dam was built
data("Nile")
plot(Nile)

## test the null hypothesis that the annual flow remains constant
## over the years
## compute OLS-based CUSUM process and plot
## with standard and alternative boundaries
ocus.nile <- efp(Nile ~ 1, type = "OLS-CUSUM")
plot(ocus.nile)
plot(ocus.nile, alpha = 0.01, alt.boundary = TRUE)
## calculate corresponding test statistic
sctest(ocus.nile)

## UK Seatbelt data: a SARIMA(1,0,0)(1,0,0)_12 model
## (fitted by OLS) is used and reveals (at least) two
## breakpoints - one in 1973 associated with the oil crisis and
## one in 1983 due to the introduction of compulsory
## wearing of seatbelts in the UK.
data("UKDriverDeaths")
seatbelt <- log10(UKDriverDeaths)
seatbelt <- cbind(seatbelt, lag(seatbelt, k = -1), lag(seatbelt, k = -12))
colnames(seatbelt) <- c("y", "ylag1", "ylag12")
seatbelt <- window(seatbelt, start = c(1970, 1), end = c(1984,12))
plot(seatbelt[,"y"], ylab = expression(log[10](casualties)))

## use RE process
re.seat <- efp(y ~ ylag1 + ylag12, data = seatbelt, type = "RE")
plot(re.seat)
plot(re.seat, functional = NULL)
sctest(re.seat)
```

efpFunctional *Functionals for Fluctuation Processes*

Description

Computes an object for aggregating, plotting and testing empirical fluctuation processes.

Usage

```
efpFunctional(functional = list(comp = function(x) max(abs(x)), time = max),
  boundary = function(x) rep(1, length(x)),
  computePval = NULL, computeCritval = NULL,
  plotProcess = NULL, lim.process = "Brownian bridge",
  nobs = 10000, nrep = 50000, nproc = 1:20, h = 0.5,
  probs = c(0:84/100, 850:1000/1000))
```

Arguments

functional	either a function for aggregating fluctuation processes or a list with two functions names "comp" and "time".
boundary	a boundary function.
computePval	a function for computing p values. If neither computePval nor computeCritval are specified critical values are simulated with settings as specified below.
computeCritval	a function for computing critical values. If neither computePval nor computeCritval are specified critical values are simulated with settings as specified below.
plotProcess	a function for plotting the empirical process, if set to NULL a suitable function is set up.
lim.process	a string specifying the limiting process.
nobs	integer specifying the number of observations of each Brownian motion simulated.
nrep	integer specifying the number of replications.
nproc	integer specifying for which number of processes Brownian motions should be simulated. If set to NULL only nproc = 1 is used and all other values are derived from a Bonferroni correction.
h	bandwidth parameter for increment processes.
probs	numeric vector specifying for which probabilities critical values should be tabulated.

Details

efpFunctional computes an object of class "efpFunctional" which then knows how to do inference based on empirical fluctuation processes (currently only for gefp objects and not yet for efp objects) and how to visualize the corresponding processes.

efpFunctionals for many frequently used test statistics are provided: maxBB for the double maximum statistic, meanL2BB for the Cramer-von Mises statistic, or rangeBB for the range statistic. Furthermore, supLM generates an object of class "efpFunctional" for a certain trimming parameter, see the examples. More details can be found in Zeileis (2006). Based on Merkle, Fan, and Zeileis (2014), further efpFunctional generators for aggregating along (ordered) categorical variables have been added: catL2BB, ordL2BB, ordwmax.

For setting up an efpFunctional, the functions computeStatistic, computePval, and plotProcess need to be supplied. These should have the following interfaces: computeStatistic should take a single argument which is the process itself, i.e., essentially a n x k matrix where n is the number of observations and k the number of processes (regressors). computePval should take two arguments: a scalar test statistic and the number of processes k. plotProcess should take two arguments: an object of class "gefp" and alpha the level of significance for any boundaries or critical values to be visualized.

Value

efpFunctional returns a list of class "efpFunctional" with components including:

plotProcess	a function for plotting empirical fluctuation processes,
computeStatistic	a function for computing a test statistic from an empirical fluctuation process,
computePval	a function for computing the corresponding p value,
computeCritval	a function for computing critical values.

References

Merkle E.C., Zeileis A. (2013), Tests of Measurement Invariance without Subgroups: A Generalization of Classical Methods. *Psychometrika*, **78**(1), 59–82. doi:10.1007/S11336-012-9302-4

Merkle E.C., Fan J., Zeileis A. (2014), Testing for Measurement Invariance with Respect to an Ordinal Variable. *Psychometrika*, **79**(4), 569–584. doi:10.1007/S11336-013-9376-7.

Zeileis A. (2005), A Unified Approach to Structural Change Tests Based on ML Scores, F Statistics, and OLS Residuals. *Econometric Reviews*, **24**, 445–466. doi:10.1080/07474930500406053.

Zeileis A. (2006), Implementing a Class of Structural Change Tests: An Econometric Computing Approach. *Computational Statistics & Data Analysis*, **50**, 2987–3008. doi:10.1016/j.csda.2005.07.001.

Zeileis A., Hornik K. (2007), Generalized M-Fluctuation Tests for Parameter Instability, *Statistica Neerlandica*, **61**, 488–508. doi:10.1111/j.1467-9574.2007.00371.x.

See Also

gefp, supLM, catL2BB, sctest.default

Examples

```
data("BostonHomicide")
gcus <- gefp(homicides ~ 1, family = poisson, vcov = kernHAC,
  data = BostonHomicide)
plot(gcus, functional = meanL2BB)
gcus
```

```
sctest(gcus, functional = meanL2BB)

y <- rnorm(1000)
x1 <- runif(1000)
x2 <- runif(1000)

## supWald statistic computed by Fstats()
fs <- Fstats(y ~ x1 + x2, from = 0.1)
plot(fs)
sctest(fs)

## compare with supLM statistic
scus <- gefp(y ~ x1 + x2, fit = lm)
plot(scus, functional = supLM(0.1))
sctest(scus, functional = supLM(0.1))

## seatbelt data
data("UKDriverDeaths")
seatbelt <- log10(UKDriverDeaths)
seatbelt <- cbind(seatbelt, lag(seatbelt, k = -1), lag(seatbelt, k = -12))
colnames(seatbelt) <- c("y", "ylag1", "ylag12")
seatbelt <- window(seatbelt, start = c(1970, 1), end = c(1984,12))

scus.seat <- gefp(y ~ ylag1 + ylag12, data = seatbelt)

## double maximum test
plot(scus.seat)
## range test
plot(scus.seat, functional = rangeBB)
## Cramer-von Mises statistic (Nyblom-Hansen test)
plot(scus.seat, functional = meanL2BB)
## supLM test
plot(scus.seat, functional = supLM(0.1))
```

`Fstats` *F Statistics*

Description

Computes a series of F statistics for a specified data window.

Usage

```
Fstats(formula, from = 0.15, to = NULL, data = list(), vcov. = NULL)
```

Arguments

`formula`	a symbolic description for the model to be tested

from, to	numeric. If from is smaller than 1 they are interpreted as percentages of data and by default to is taken to be 1 - from. F statistics will be calculated for the observations (n*from):(n*to), when n is the number of observations in the model. If from is greater than 1 it is interpreted to be the index and to defaults to n - from. If from is a vector with two elements, then from and to are interpreted as time specifications like in ts, see also the examples.
data	an optional data frame containing the variables in the model. By default the variables are taken from the environment which Fstats is called from.
vcov.	a function to extract the covariance matrix for the coefficients of a fitted model of class "lm".

Details

For every potential change point in from:to a F statistic (Chow test statistic) is computed. For this an OLS model is fitted for the observations before and after the potential change point, i.e. 2k parameters have to be estimated, and the error sum of squares is computed (ESS). Another OLS model for all observations with a restricted sum of squares (RSS) is computed, hence k parameters have to be estimated here. If n is the number of observations and k the number of regressors in the model, the formula is:

$$F = \frac{(RSS - ESS)}{ESS/(n - 2k)}$$

Note that this statistic has an asymptotic chi-squared distribution with k degrees of freedom and (under the assumption of normality) F/k has an exact F distribution with k and n - 2k degrees of freedom.

Value

Fstats returns an object of class "Fstats", which contains mainly a time series of F statistics. The function plot has a method to plot the F statistics or the corresponding p values; with sctest a supF-, aveF- or expF-test on structural change can be performed.

References

Andrews D.W.K. (1993), Tests for parameter instability and structural change with unknown change point, *Econometrica*, **61**, 821-856.

Hansen B. (1992), Tests for parameter instability in regressions with I(1) processes, *Journal of Business & Economic Statistics*, **10**, 321-335.

Hansen B. (1997), Approximate asymptotic p values for structural-change tests, *Journal of Business & Economic Statistics*, **15**, 60-67.

See Also

plot.Fstats, sctest.Fstats, boundary.Fstats

Examples

```
## Nile data with one breakpoint: the annual flows drop in 1898
## because the first Ashwan dam was built
data("Nile")
plot(Nile)

## test the null hypothesis that the annual flow remains constant
## over the years
fs.nile <- Fstats(Nile ~ 1)
plot(fs.nile)
sctest(fs.nile)
## visualize the breakpoint implied by the argmax of the F statistics
plot(Nile)
lines(breakpoints(fs.nile))

## UK Seatbelt data: a SARIMA(1,0,0)(1,0,0)_12 model
## (fitted by OLS) is used and reveals (at least) two
## breakpoints - one in 1973 associated with the oil crisis and
## one in 1983 due to the introduction of compulsory
## wearing of seatbelts in the UK.
data("UKDriverDeaths")
seatbelt <- log10(UKDriverDeaths)
seatbelt <- cbind(seatbelt, lag(seatbelt, k = -1), lag(seatbelt, k = -12))
colnames(seatbelt) <- c("y", "ylag1", "ylag12")
seatbelt <- window(seatbelt, start = c(1970, 1), end = c(1984,12))
plot(seatbelt[,"y"], ylab = expression(log[10](casualties)))

## compute F statistics for potential breakpoints between
## 1971(6) (corresponds to from = 0.1) and 1983(6) (corresponds to
## to = 0.9 = 1 - from, the default)
## compute F statistics
fs <- Fstats(y ~ ylag1 + ylag12, data = seatbelt, from = 0.1)
## this gives the same result
fs <- Fstats(y ~ ylag1 + ylag12, data = seatbelt, from = c(1971, 6),
             to = c(1983, 6))
## plot the F statistics
plot(fs, alpha = 0.01)
## plot F statistics with aveF boundary
plot(fs, aveF = TRUE)
## perform the expF test
sctest(fs, type = "expF")
```

---

gefp *Generalized Empirical M-Fluctuation Processes*

Description

Computes an empirical M-fluctuation process from the scores of a fitted model.

Usage

```
gefp(..., fit = glm, scores = estfun, vcov = NULL,
  decorrelate = TRUE, sandwich = TRUE, order.by = NULL,
  fitArgs = NULL, parm = NULL, data = list())
```

Arguments

`...`	specification of some model which is passed together with `data` to the `fit` function: `fm <- fit(..., data = data)`. If `fit` is set to `NULL` the first argument `...` is assumed to be already the fitted model `fm` (all other arguments in `...` are ignored and a warning is issued in this case).
`fit`	a model fitting function, typically `lm`, `glm` or `rlm`.
`scores`	a function which extracts the scores or estimating function from the fitted object: `scores(fm)`.
`vcov`	a function to extract the covariance matrix for the coefficients of the fitted model: `vcov(fm, order.by = order.by, data = data)`.
`decorrelate`	logical. Should the process be decorrelated?
`sandwich`	logical. Is the function `vcov` the full sandwich estimator or only the meat?
`order.by`	Either a vector `z` or a formula with a single explanatory variable like `~ z`. The observations in the model are ordered by the size of `z`. If set to `NULL` (the default) the observations are assumed to be ordered (e.g., a time series).
`fitArgs`	List of additional arguments which could be passed to the `fit` function. Usually, this is not needed and `...` will be sufficient to pass arguments to `fit`.
`parm`	integer or character specifying the component of the estimating functions which should be used (by default all components are used).
`data`	an optional data frame containing the variables in the `...` specification and the `order.by` model. By default the variables are taken from the environment which `gefp` is called from.

Value

`gefp` returns a list of class `"gefp"` with components including:

`process`	the fitted empirical fluctuation process of class `"zoo"`,
`nreg`	the number of regressors,
`nobs`	the number of observations,
`fit`	the fit function used,
`scores`	the scores function used,
`fitted.model`	the fitted model.

References

Zeileis A. (2005), A Unified Approach to Structural Change Tests Based on ML Scores, F Statistics, and OLS Residuals. *Econometric Reviews*, **24**, 445–466. doi:10.1080/07474930500406053.

Zeileis A. (2006), Implementing a Class of Structural Change Tests: An Econometric Computing Approach. *Computational Statistics & Data Analysis*, **50**, 2987–3008. doi:10.1016/j.csda.2005.07.001.

Zeileis A., Hornik K. (2007), Generalized M-Fluctuation Tests for Parameter Instability, *Statistica Neerlandica*, **61**, 488–508. doi:10.1111/j.1467-9574.2007.00371.x.

Zeileis A., Shah A., Patnaik I. (2010), Testing, Monitoring, and Dating Structural Changes in Exchange Rate Regimes, *Computational Statistics and Data Analysis*, **54**(6), 1696–1706. doi:10.1016/j.csda.2009.12.005.

See Also

`efp`, `efpFunctional`

Examples

```
data("BostonHomicide")
gcus <- gefp(homicides ~ 1, family = poisson, vcov = kernHAC,
     data = BostonHomicide)
plot(gcus, aggregate = FALSE)
gcus
sctest(gcus)
```

`GermanM1` *German M1 Money Demand*

Description

German M1 money demand.

Usage

```
data("GermanM1")
```

Format

`GermanM1` is a data frame containing 12 quarterly time series from 1961(1) to 1995(4) and two further variables. `historyM1` is the subset of `GermanM1` up to 1990(2), i.e., the data before the German monetary unification on 1990-06-01. `monitorM1` is the complement of `historyM1`, i.e., the data after the unification. All three data frames contain the variables

m time series. Logarithm of real M1 per capita,

p time series. Logarithm of a price index,

y time series. Logarithm of real per capita gross national product,

R time series. Long-run interest rate,

dm time series. First differences of `m`,

dy2 time series. First differences of lag 2 of y,

dR time series. First differences of R,

dR1 time series. First differences of lag 1 of R,

dp time series. First differences of p,

m1 time series. Lag 1 of m,

y1 time series. Lag 1 of y,

R1 time series. Lag 1 of R,

season factor coding the seasonality,

ecm.res vector containing the OLS residuals of the Lütkepohl et al. (1999) model fitted in the history period.

Details

Lütkepohl et al. (1999) investigate the linearity and stability of German M1 money demand: they find a stable regression relation for the time before the monetary union on 1990-06-01 but a clear structural instability afterwards.

Zeileis et al. (2005) use a model with ecm.res instead of m1, y1 and R1, which leads to equivalent results in the history period but slightly different results in the monitoring period. The reason for the replacement is that stationary regressors are needed for the structural change tests. See references and the examples below for more details.

Source

The data is provided by the German central bank and is available online in the data archive of the Journal of Applied Econometrics http://qed.econ.queensu.ca/jae/1999-v14.5/lutkepohl-terasvirta-wolters/.

References

Lütkepohl H., Teräsvirta T., Wolters J. (1999), Investigating Stability and Linearity of a German M1 Money Demand Function, *Journal of Applied Econometrics*, **14**, 511-525.

Zeileis A., Leisch F., Kleiber C., Hornik K. (2005), Monitoring Structural Change in Dynamic Econometric Models, *Journal of Applied Econometrics*, **20**, 99–121.

Examples

```
data("GermanM1")
## Lütkepohl et al. (1999) use the following model
LTW.model <- dm ~ dy2 + dR + dR1 + dp + m1 + y1 + R1 + season
## Zeileis et al. (2005) use
M1.model <- dm ~ dy2 + dR + dR1 + dp + ecm.res + season


## historical tests
ols <- efp(LTW.model, data = GermanM1, type = "OLS-CUSUM")
plot(ols)
re <- efp(LTW.model, data = GermanM1, type = "fluctuation")
plot(re)
```

```
fs <- Fstats(LTW.model, data = GermanM1, from = 0.1)
plot(fs)

## monitoring
M1 <- historyM1
ols.efp <- efp(M1.model, type = "OLS-CUSUM", data = M1)
newborder <- function(k) 1.5778*k/118
ols.mefp <- mefp(ols.efp, period = 2)
ols.mefp2 <- mefp(ols.efp, border = newborder)
M1 <- GermanM1
ols.mon <- monitor(ols.mefp)
ols.mon2 <- monitor(ols.mefp2)
plot(ols.mon)
lines(boundary(ols.mon2), col = 2)

## dating
bp <- breakpoints(LTW.model, data = GermanM1)
summary(bp)
plot(bp)

plot(fs)
lines(confint(bp))
```

`Grossarl` *Marriages, Births and Deaths in Grossarl*

Description

Data about the number of marriages, illegitimate and legitimate births, and deaths in the Austrian Alpine village Grossarl during the 18th and 19th century.

Usage

```
data("Grossarl")
```

Format

`Grossarl` is a data frame containing 6 annual time series (1700 - 1899), 3 factors coding policy interventions and 1 vector with the year (plain numeric).

marriages time series. Number of marriages,

illegitimate time series. Number of illegitimate births,

legitimate time series. Number of legitimate births,

legitimate time series. Number of deaths,

fraction time series. Fraction of illegitimate births,

lag.marriages time series. Number of marriages in the previous year,

politics ordered factor coding 4 different political regimes,

morals ordered factor coding 5 different moral regulations,

nuptiality ordered factor coding 5 different marriage restrictions,

year numeric. Year of observation.

Details

The data frame contains historical demographic data from Grossarl, a village in the Alpine region of Salzburg, Austria, during the 18th and 19th century. During this period, the total population of Grossarl did not vary much on the whole, with the very exception of the period of the protestant emigrations in 1731/32.

Especially during the archbishopric, moral interventions aimed at lowering the proportion of illegitimate baptisms. For details see the references.

Source

Parish registers provide the basic demographic series of baptisms and burials (which is almost equivalent to births and deaths in the study area) and marriages. For more information see Veichtlbauer et al. (2006).

References

Veichtlbauer O., Zeileis A., Leisch F. (2006), The Impact Of Policy Interventions on a Pre-Industrial Population System in the Austrian Alps, forthcoming.

Zeileis A., Veichtlbauer O. (2002), Policy Interventions Affecting Illegitimacy in Preindustrial Austria: A Structural Change Analysis, In R. Dutter (ed.), *Festschrift 50 Jahre Österreichische Statistische Gesellschaft*, 133-146, Österreichische Statistische Gesellschaft.

Examples

```
data("Grossarl")

## time series of births, deaths, marriages
###########################################

with(Grossarl, plot(cbind(deaths, illegitimate + legitimate, marriages),
  plot.type = "single", col = grey(c(0.7, 0, 0)), lty = c(1, 1, 3),
  lwd = 1.5, ylab = "annual Grossarl series"))
legend("topright", c("deaths", "births", "marriages"), col = grey(c(0.7, 0, 0)),
  lty = c(1, 1, 3), bty = "n")

## illegitimate births
######################
## lm + MOSUM
plot(Grossarl$fraction)
fm.min <- lm(fraction ~ politics, data = Grossarl)
fm.ext <- lm(fraction ~ politics + morals + nuptiality + marriages,
  data = Grossarl)
lines(ts(fitted(fm.min), start = 1700), col = 2)
lines(ts(fitted(fm.ext), start = 1700), col = 4)
mos.min <- efp(fraction ~ politics, data = Grossarl, type = "OLS-MOSUM")
```

```
mos.ext <- efp(fraction ~ politics + morals + nuptiality + marriages,
  data = Grossarl, type = "OLS-MOSUM")
plot(mos.min)
lines(mos.ext, lty = 2)

## dating
bp <- breakpoints(fraction ~ 1, data = Grossarl, h = 0.1)
summary(bp)
## RSS, BIC, AIC
plot(bp)
plot(0:8, AIC(bp), type = "b")

## probably use 5 or 6 breakpoints and compare with
## coding of the factors as used by us
##
## politics                    1803      1816 1850
## morals      1736 1753 1771 1803
## nuptiality                  1803 1810 1816      1883
##
## m = 5            1753 1785           1821 1856 1878
## m = 6       1734 1754 1785           1821 1856 1878
##              6    2    5             1    4    3

## fitted models
coef(bp, breaks = 6)
plot(Grossarl$fraction)
lines(fitted(bp, breaks = 6), col = 2)
lines(ts(fitted(fm.ext), start = 1700), col = 4)


## marriages
#############
## lm + MOSUM
plot(Grossarl$marriages)
fm.min <- lm(marriages ~ politics, data = Grossarl)
fm.ext <- lm(marriages ~ politics + morals + nuptiality, data = Grossarl)
lines(ts(fitted(fm.min), start = 1700), col = 2)
lines(ts(fitted(fm.ext), start = 1700), col = 4)
mos.min <- efp(marriages ~ politics, data = Grossarl, type = "OLS-MOSUM")
mos.ext <- efp(marriages ~ politics + morals + nuptiality, data = Grossarl,
  type = "OLS-MOSUM")
plot(mos.min)
lines(mos.ext, lty = 2)

## dating
bp <- breakpoints(marriages ~ 1, data = Grossarl, h = 0.1)
summary(bp)
## RSS, BIC, AIC
plot(bp)
plot(0:8, AIC(bp), type = "b")

## probably use 3 or 4 breakpoints and compare with
## coding of the factors as used by us
```

```
##
## politics                    1803      1816 1850
## morals      1736 1753 1771 1803
## nuptiality                  1803 1810 1816      1883
##
## m = 3       1738                      1813      1875
## m = 4       1738      1794            1814      1875
##              2         4               1         3

## fitted models
coef(bp, breaks = 4)
plot(Grossarl$marriages)
lines(fitted(bp, breaks = 4), col = 2)
lines(ts(fitted(fm.ext), start = 1700), col = 4)
```

`logLik.breakpoints` *Log Likelihood and Information Criteria for Breakpoints*

Description

Computation of log likelihood and AIC type information criteria for partitions given by breakpoints.

Usage

```
## S3 method for class 'breakpointsfull'
logLik(object, breaks = NULL, ...)
## S3 method for class 'breakpointsfull'
AIC(object, breaks = NULL, ..., k = 2)
```

Arguments

`object`	an object of class `"breakpoints"` or `"breakpointsfull"`.
`breaks`	if `object` is of class `"breakpointsfull"` the number of breaks can be specified.
`...`	*currently not used.*
`k`	the penalty parameter to be used, the default `k = 2` is the classical AIC, `k = log(n)` gives the BIC, if `n` is the number of observations.

Details

As for linear models the log likelihood is computed on a normal model and the degrees of freedom are the number of regression coefficients multiplied by the number of segments plus the number of estimated breakpoints plus 1 for the error variance.

If `AIC` is applied to an object of class `"breakpointsfull"` `breaks` can be a vector of integers and the AIC for each corresponding partition will be returned. By default the maximal number of breaks stored in the `object` is used. See below for an example.

Value

An object of class "logLik" or a simple vector containing the AIC respectively.

See Also

breakpoints

Examples

```
## Nile data with one breakpoint: the annual flows drop in 1898
## because the first Ashwan dam was built
data("Nile")
plot(Nile)

bp.nile <- breakpoints(Nile ~ 1)
summary(bp.nile)
plot(bp.nile)

## BIC of partitions with0 to 5 breakpoints
plot(0:5, AIC(bp.nile, k = log(bp.nile$nobs)), type = "b")
## AIC
plot(0:5, AIC(bp.nile), type = "b")

## BIC, AIC, log likelihood of a single partition
bp.nile1 <- breakpoints(bp.nile, breaks = 1)
AIC(bp.nile1, k = log(bp.nile1$nobs))
AIC(bp.nile1)
logLik(bp.nile1)
```

---

mefp *Monitoring of Empirical Fluctuation Processes*

Description

Online monitoring of structural breaks in a linear regression model. A sequential fluctuation test based on parameter estimates or OLS residuals signals structural breaks.

Usage

```
mefp(obj, ...)

## S3 method for class 'formula'
mefp(formula, type = c("OLS-CUSUM", "OLS-MOSUM", "RE", "ME",
    "fluctuation"), data, h = 1, alpha = 0.05,
    functional = c("max", "range"), period = 10,
    tolerance = .Machine$double.eps^0.5, CritvalTable = NULL,
    rescale = NULL, border = NULL, ...)
```

```
## S3 method for class 'efp'
mefp(obj, alpha=0.05, functional = c("max", "range"),
    period = 10, tolerance = .Machine$double.eps^0.5,
    CritvalTable = NULL, rescale = NULL, border = NULL, ...)

monitor(obj, data = NULL, verbose = TRUE)
```

Arguments

formula	a symbolic description for the model to be tested.
data	an optional data frame containing the variables in the model. By default the variables are taken from the environment which efp is called from.
type	specifies which type of fluctuation process will be computed.
h	(only used for MOSUM/ME processes). A numeric scalar from interval (0,1) specifying the size of the data window relative to the sample size.
obj	Object of class "efp" (for mefp) or "mefp" (for monitor).
alpha	Significance level of the test, i.e., probability of type I error.
functional	Determines if maximum or range of parameter differences is used as statistic.
period	(only used for MOSUM/ME processes). Maximum time (relative to the history period) that will be monitored. Default is 10 times the history period.
tolerance	Tolerance for numeric == comparisons.
CritvalTable	Table of critical values, this table is interpolated to get critical values for arbitrary alphas. The default depends on the type of fluctuation process (precomputed tables are available for all types). *This argument is under development.*
rescale	If TRUE the estimates will be standardized by the regressor matrix of the corresponding subsample similar to Kuan & Chen (1994); if FALSE the historic regressor matrix will be used. The default is to rescale the monitoring processes of type "ME" but not of "RE".
border	An optional user-specified border function for the empirical process. *This argument is under development.*
verbose	If TRUE, signal breaks by text output.
...	Currently not used.

Details

mefp creates an object of class "mefp" either from a model formula or from an object of class "efp". In addition to the arguments of efp, the type of statistic and a significance level for the monitoring must be specified. The monitoring itself is performed by monitor, which can be called arbitrarily often on objects of class "mefp". If new data have arrived, then the empirical fluctuation process is computed for the new data. If the process crosses the boundaries corresponding to the significance level alpha, a structural break is detected (and signaled).

The typical usage is to initialize the monitoring by creation of an object of class "mefp" either using a formula or an "efp" object. Data available at this stage are considered the *history sample*, which is kept fixed during the complete monitoring process, and may not contain any structural changes.

Subsequent calls to `monitor` perform a sequential test of the null hypothesis of no structural change in new data against the general alternative of changes in one or more of the coefficients of the regression model.

The recursive estimates test is also called fluctuation test, therefore setting `type` to `"fluctuation"` was used to specify it in earlier versions of strucchange. It still can be used now, but will be forced to `"RE"`

References

Leisch F., Hornik K., Kuan C.-M. (2000), Monitoring Structural Changes with the Generalized Fluctuation Test, *Econometric Theory*, **16**, 835–854.

Zeileis A., Leisch F., Kleiber C., Hornik K. (2005), Monitoring Structural Change in Dynamic Econometric Models, *Journal of Applied Econometrics*, **20**, 99–121. doi:10.1002/jae.776.

Zeileis A. (2005), A Unified Approach to Structural Change Tests Based on ML Scores, F Statistics, and OLS Residuals. *Econometric Reviews*, **24**, 445–466. doi:10.1080/07474930500406053.

Zeileis A., Shah A., Patnaik I. (2010), Testing, Monitoring, and Dating Structural Changes in Exchange Rate Regimes, *Computational Statistics and Data Analysis*, **54**(6), 1696–1706. doi:10.1016/j.csda.2009.12.005.

See Also

`plot.mefp`, `boundary.mefp`

Examples

```
df1 <- data.frame(y=rnorm(300))
df1[150:300,"y"] <- df1[150:300,"y"]+1

## use the first 50 observations as history period
e1 <- efp(y~1, data=df1[1:50,,drop=FALSE], type="ME", h=1)
me1 <- mefp(e1, alpha=0.05)

## the same in one function call
me1 <- mefp(y~1, data=df1[1:50,,drop=FALSE], type="ME", h=1,
             alpha=0.05)

## monitor the 50 next observations
me2 <- monitor(me1, data=df1[1:100,,drop=FALSE])
plot(me2)

# and now monitor on all data
me3 <- monitor(me2, data=df1)
plot(me3)


## Load dataset "USIncExp" with income and expenditure in the US
## and choose a suitable subset for the history period
data("USIncExp")
USIncExp3 <- window(USIncExp, start=c(1969,1), end=c(1971,12))
## initialize the monitoring with the formula interface
me.mefp <- mefp(expenditure~income, type="ME", rescale=TRUE,
```

```
                    data=USIncExp3, alpha=0.05)

## monitor the new observations for the year 1972
USIncExp3 <- window(USIncExp, start=c(1969,1), end=c(1972,12))
me.mefp <- monitor(me.mefp)

## monitor the new data for the years 1973-1976
USIncExp3 <- window(USIncExp, start=c(1969,1), end=c(1976,12))
me.mefp <- monitor(me.mefp)
plot(me.mefp, functional = NULL)
```

`PhillipsCurve` *UK Phillips Curve Equation Data*

Description

Macroeconomic time series from the United Kingdom with variables for estimating the Phillips curve equation.

Usage

```
data("PhillipsCurve")
```

Format

A multivariate annual time series from 1857 to 1987 with the columns

p Logarithm of the consumer price index,

w Logarithm of nominal wages,

u Unemployment rate,

dp First differences of `p`,

dw First differences of `w`,

du First differences of `u`

u1 Lag 1 of `u`,

dp1 Lag 1 of `dp`.

Source

The data is available online in the data archive of the Journal of Applied Econometrics http://qed.econ.queensu.ca/jae/2003-v18.1/bai-perron/.

References

Alogoskoufis G.S., Smith R. (1991), The Phillips Curve, the Persistence of Inflation, and the Lucas Critique: Evidence from Exchange Rate Regimes, *American Economic Review*, **81**, 1254-1275.

Bai J., Perron P. (2003), Computation and Analysis of Multiple Structural Change Models, *Journal of Applied Econometrics*, **18**, 1-22.

Examples

```
## load and plot data
data("PhillipsCurve")
uk <- window(PhillipsCurve, start = 1948)
plot(uk[, "dp"])

## AR(1) inflation model
## estimate breakpoints
bp.inf <- breakpoints(dp ~ dp1, data = uk, h = 8)
plot(bp.inf)
summary(bp.inf)

## fit segmented model with three breaks
fac.inf <- breakfactor(bp.inf, breaks = 2, label = "seg")
fm.inf <- lm(dp ~ 0 + fac.inf/dp1, data = uk)
summary(fm.inf)

## Results from Table 2 in Bai & Perron (2003):
## coefficient estimates
coef(bp.inf, breaks = 2)
## corresponding standard errors
sqrt(sapply(vcov(bp.inf, breaks = 2), diag))
## breakpoints and confidence intervals
confint(bp.inf, breaks = 2)

## Phillips curve equation
## estimate breakpoints
bp.pc <- breakpoints(dw ~ dp1 + du + u1, data = uk, h = 5, breaks = 5)
## look at RSS and BIC
plot(bp.pc)
summary(bp.pc)

## fit segmented model with three breaks
fac.pc <- breakfactor(bp.pc, breaks = 2, label = "seg")
fm.pc <- lm(dw ~ 0 + fac.pc/dp1 + du + u1, data = uk)
summary(fm.pc)

## Results from Table 3 in Bai & Perron (2003):
## coefficient estimates
coef(fm.pc)
## corresponding standard errors
sqrt(diag(vcov(fm.pc)))
## breakpoints and confidence intervals
confint(bp.pc, breaks = 2, het.err = FALSE)
```

`plot.efp` *Plot Empirical Fluctuation Process*

Description

Plot and lines method for objects of class `"efp"`

Usage

```
## S3 method for class 'efp'
plot(x, alpha = 0.05, alt.boundary = FALSE, boundary = TRUE,
    functional = "max", main = NULL,  ylim = NULL,
    ylab = "Empirical fluctuation process", ...)
## S3 method for class 'efp'
lines(x, functional = "max", ...)
```

Arguments

x	an object of class "efp".
alpha	numeric from interval (0,1) indicating the confidence level for which the boundary of the corresponding test will be computed.
alt.boundary	logical. If set to TRUE alternative boundaries (instead of the standard linear boundaries) will be plotted (for CUSUM processes only).
boundary	logical. If set to FALSE the boundary will be computed but not plotted.
functional	indicates which functional should be applied to the process before plotting and which boundaries should be used. If set to NULL a multiple process with boundaries for the "max" functional is plotted. For more details see below.
main, ylim, ylab, ...	high-level plot function parameters.

Details

Plots are available for the "max" functional for all process types. For Brownian bridge type processes the maximum or mean squared Euclidean norm ("maxL2" and "meanL2") can be used for aggregating before plotting. No plots are available for the "range" functional.

Alternative boundaries that are proportional to the standard deviation of the corresponding limiting process are available for processes with Brownian motion or Brownian bridge limiting processes.

Value

efp returns an object of class "efp" which inherits from the class "ts" or "mts" respectively. The function plot has a method to plot the empirical fluctuation process; with sctest the corresponding test for structural change can be performed.

References

Brown R.L., Durbin J., Evans J.M. (1975), Techniques for testing constancy of regression relationships over time, *Journal of the Royal Statistical Society*, B, **37**, 149-163.

Chu C.-S., Hornik K., Kuan C.-M. (1995), MOSUM tests for parameter constancy, *Biometrika*, **82**, 603-617.

Chu C.-S., Hornik K., Kuan C.-M. (1995), The moving-estimates test for parameter stability, *Econometric Theory*, **11**, 669-720.

Krämer W., Ploberger W., Alt R. (1988), Testing for structural change in dynamic models, *Econometrica*, **56**, 1355-1369.

Kuan C.-M., Hornik K. (1995), The generalized fluctuation test: A unifying view, *Econometric Reviews*, **14**, 135 - 161.

Kuan C.-M., Chen (1994), Implementing the fluctuation and moving estimates tests in dynamic econometric models, *Economics Letters*, **44**, 235-239.

Ploberger W., Krämer W. (1992), The CUSUM test with OLS residuals, *Econometrica*, **60**, 271-285.

Zeileis A., Leisch F., Hornik K., Kleiber C. (2002), strucchange: An R Package for Testing for Structural Change in Linear Regression Models, *Journal of Statistical Software*, **7**(2), 1-38. doi:10.18637/jss.v007.i02.

Zeileis A. (2004), Alternative Boundaries for CUSUM Tests, *Statistical Papers*, **45**, 123–131.

See Also

efp, boundary.efp, sctest.efp

Examples

```
## Load dataset "nhtemp" with average yearly temperatures in New Haven
data("nhtemp")
## plot the data
plot(nhtemp)

## test the model null hypothesis that the average temperature remains
## constant over the years
## compute Rec-CUSUM fluctuation process
temp.cus <- efp(nhtemp ~ 1)
## plot the process
plot(temp.cus, alpha = 0.01)
## and calculate the test statistic
sctest(temp.cus)

## compute (recursive estimates) fluctuation process
## with an additional linear trend regressor
lin.trend <- 1:60
temp.me <- efp(nhtemp ~ lin.trend, type = "fluctuation")
## plot the bivariate process
plot(temp.me, functional = NULL)
## and perform the corresponding test
sctest(temp.me)
```

---

plot.Fstats *Plot F Statistics*

Description

Plotting method for objects of class "Fstats"

Usage

```
## S3 method for class 'Fstats'
plot(x, pval = FALSE, asymptotic = FALSE, alpha = 0.05,
    boundary = TRUE, aveF = FALSE, xlab = "Time", ylab = NULL,
    ylim = NULL, ...)
```

Arguments

x	an object of class "Fstats".
pval	logical. If set to TRUE the corresponding p values instead of the original F statistics will be plotted.
asymptotic	logical. If set to TRUE the asymptotic (chi-square) distribution instead of the exact (F) distribution will be used to compute the p values (only if pval is TRUE).
alpha	numeric from interval (0,1) indicating the confidence level for which the boundary of the supF test will be computed.
boundary	logical. If set to FALSE the boundary will be computed but not plotted.
aveF	logical. If set to TRUE the boundary of the aveF test will be plotted. As this is a boundary for the mean of the F statistics rather than for the F statistics themselves a dashed line for the mean of the F statistics will also be plotted.
xlab, ylab, ylim, ...	high-level plot function parameters.

References

Andrews D.W.K. (1993), Tests for parameter instability and structural change with unknown change point, *Econometrica*, **61**, 821-856.

Hansen B. (1992), Tests for parameter instability in regressions with I(1) processes, *Journal of Business & Economic Statistics*, **10**, 321-335.

Hansen B. (1997), Approximate asymptotic p values for structural-change tests, *Journal of Business & Economic Statistics*, **15**, 60-67.

See Also

Fstats, boundary.Fstats, sctest.Fstats

Examples

```
## Load dataset "nhtemp" with average yearly temperatures in New Haven
data("nhtemp")
## plot the data
plot(nhtemp)

## test the model null hypothesis that the average temperature remains
## constant over the years for potential break points between 1941
## (corresponds to from = 0.5) and 1962 (corresponds to to = 0.85)
## compute F statistics
fs <- Fstats(nhtemp ~ 1, from = 0.5, to = 0.85)
```

```
## plot the F statistics
plot(fs, alpha = 0.01)
## and the corresponding p values
plot(fs, pval = TRUE, alpha = 0.01)
## perform the aveF test
sctest(fs, type = "aveF")
```

`plot.mefp` *Plot Methods for mefp Objects*

Description

This is a method of the generic `plot` function for for `"mefp"` objects as returned by `mefp` or `monitor`. It plots the empirical fluctuation process (or a functional thereof) as a time series plot, and includes boundaries corresponding to the significance level of the monitoring procedure.

Usage

```
## S3 method for class 'mefp'
plot(x, boundary = TRUE, functional = "max", main = NULL,
    ylab = "Empirical fluctuation process", ylim = NULL, ...)
```

Arguments

`x`	an object of class `"mefp"`.
`boundary`	if `FALSE`, plotting of boundaries is suppressed.
`functional`	indicates which functional should be applied to a multivariate empirical process. If set to `NULL` all dimensions of the process (one process per coefficient in the linear model) are plotted.
`main`, `ylab`, `ylim`, `...`	high-level `plot` function parameters.

See Also

`mefp`

Examples

```
df1 <- data.frame(y=rnorm(300))
df1[150:300,"y"] <- df1[150:300,"y"]+1
me1 <- mefp(y~1, data=df1[1:50,,drop=FALSE], type="ME", h=1,
             alpha=0.05)
me2 <- monitor(me1, data=df1)

plot(me2)
```

RealInt *US Ex-post Real Interest Rate*

Description

US ex-post real interest rate: the three-month treasury bill deflated by the CPI inflation rate.

Usage

```
data("RealInt")
```

Format

A quarterly time series from 1961(1) to 1986(3).

Source

The data is available online in the data archive of the Journal of Applied Econometrics http://qed.econ.queensu.ca/jae/2003-v18.1/bai-perron/.

References

Bai J., Perron P. (2003), Computation and Analysis of Multiple Structural Change Models, *Journal of Applied Econometrics*, **18**, 1-22.

Zeileis A., Kleiber C. (2005), Validating Multiple Structural Change Models - A Case Study. Journal of Applied Econometrics, **20**, 685-690.

Examples

```
## load and plot data
data("RealInt")
plot(RealInt)

## estimate breakpoints
bp.ri <- breakpoints(RealInt ~ 1, h = 15)
plot(bp.ri)
summary(bp.ri)

## fit segmented model with three breaks
fac.ri <- breakfactor(bp.ri, breaks = 3, label = "seg")
fm.ri <- lm(RealInt ~ 0 + fac.ri)
summary(fm.ri)

## setup kernel HAC estimator
vcov.ri <- function(x, ...) kernHAC(x, kernel = "Quadratic Spectral",
  prewhite = 1, approx = "AR(1)", ...)

## Results from Table 1 in Bai & Perron (2003):
## coefficient estimates
```

```
coef(bp.ri, breaks = 3)
## corresponding standard errors
sapply(vcov(bp.ri, breaks = 3, vcov = vcov.ri), sqrt)
## breakpoints and confidence intervals
confint(bp.ri, breaks = 3, vcov = vcov.ri)

## Visualization
plot(RealInt)
lines(as.vector(time(RealInt)), fitted(fm.ri), col = 4)
lines(confint(bp.ri, breaks = 3, vcov = vcov.ri))
```

`recresid` *Recursive Residuals*

Description

A generic function for computing the recursive residuals (standardized one step prediction errors) of a linear regression model.

Usage

```
## Default S3 method:
recresid(x, y, start = ncol(x) + 1, end = nrow(x),
  tol = sqrt(.Machine$double.eps)/ncol(x), qr.tol = 1e-7,
  engine = c("R", "C"), ...)
## S3 method for class 'formula'
recresid(formula, data = list(), ...)
## S3 method for class 'lm'
recresid(x, data = list(), ...)
```

Arguments

`x`, `y`, `formula`	specification of the linear regression model: either by a regressor matrix `x` and a response variable `y`, or by a `formula` or by a fitted object `x` of class `"lm"`.
`start`, `end`	integer. Index of the first and last observation, respectively, for which recursive residuals should be computed. By default, the maximal range is selected.
`tol`	numeric. A relative tolerance for precision of recursive coefficient estimates, see details.
`qr.tol`	numeric. The `tol`erance passed to `lm.fit` for detecting linear dependencies.
`engine`	character. In addition to the R implementation of the default method, there is also a faster C implementation (see below for further details).
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which `recresid` is called from. Specifying `data` might also be necessary when applying `recresid` to a fitted model of class `"lm"` if this does not contain the regressor matrix and the response.
`...`	*currently not used.*

Details

Recursive residuals are standardized one-step-ahead prediction errors. Under the usual assumptions for the linear regression model they are (asymptotically) normal and i.i.d. (see Brown, Durbin, Evans, 1975, for details).

The default method computes the initial coefficient estimates via QR decomposition, using `lm.fit`. In subsequent steps, the updating formula provided by Brown, Durbin, Evans (1975) is employed. To avoid numerical instabilities in the first steps (with typically small sample sizes), the QR solution is computed for comparison. When the relative difference (assessed bey `all.equal`) between the two solutions falls below `tol`, only the updating formula is used in subsequent steps.

In large data sets, the R implementation can become rather slow. Hence, a C implementation is also available. This is not the default, yet, because it should receive more testing in numerically challenging cases. In addition to the R and C implementation, there is also an Armadillo-based C++ implementation available on R-Forge in package strucchangeArmadillo. For models with about 10 parameters, the C and C++ version perform similarly. For larger models, the C++ implementation seems to scale better.

Value

A vector containing the recursive residuals.

References

Brown R.L., Durbin J., Evans J.M. (1975), Techniques for testing constancy of regression relationships over time, *Journal of the Royal Statistical Society*, B, **37**, 149-163.

See Also

`efp`

Examples

```
x <- rnorm(100) + rep(c(0, 2), each = 50)
rr <- recresid(x ~ 1)
plot(cumsum(rr), type = "l")

plot(efp(x ~ 1, type = "Rec-CUSUM"))
```

`root.matrix` *Root of a Matrix*

Description

Computes the root of a symmetric and positive semidefinite matrix.

Usage

```
root.matrix(X)
```

Arguments

X	a symmetric and positive semidefinite matrix

Value

a symmetric matrix of same dimensions as X

Examples

```
X <- matrix(c(1,2,2,8), ncol=2)
test <- root.matrix(X)
## control results
X
test %*% test
```

---

scPublications *Structural Change Publications*

Description

Bibliographic information about papers related to structural change and changepoints published in 27 different econometrics and statistics journals.

Usage

```
data("scPublications")
```

Format

A data frame containing information on 835 structural change papers in 9 variables.

author character. Author(s) of the paper.

title character. Title of the paper.

journal factor. In which journal was the paper published?

year numeric. Year of publication.

volume numeric. Journal volume.

issue character. Issue within the journal volume.

bpage numeric. Page on which the paper begins.

epage numeric. Page on which the paper ends.

type factor. Is the journal an econometrics or statistics journal?

Details

The data set scPublications includes bibliographic information about publications related to structural change and obtained from the 'ISI Web of Science'. The query was based on the 'Science Citation Index Expanded' and 'Social Sciences Citation Index' (for the full range of years available: 1900-2006 and 1956-2006, respectively). The 'Source Title' was restricted to the 27 journals in the data frame and the 'Topic' to be one of the following: structural change, structural break, structural stability, structural instability, parameter instability, parameter stability, parameter constancy, change point, changepoint, change-point, breakpoint, break-point, break point, CUSUM, MOSUM. Additionally, the famous CUSUM paper of Brown, Durbin and Evans (1975) was added manually to scPublications (because it did not match the query above).

Source

ISI Web of Science at https://www.webofknowledge.com/. Queried by James Bullard.

Examples

```
## construct time series:
## number of sc publications in econometrics/statistics
data("scPublications")

## select years from 1987 and
## `most important' journals
pub <- scPublications
pub <- subset(pub, year > 1986)
tab1 <- table(pub$journal)
nam1 <- names(tab1)[as.vector(tab1) > 9] ## at least 10 papers
tab2 <- sapply(levels(pub$journal), function(x) min(subset(pub, journal == x)$year))
nam2 <- names(tab2)[as.vector(tab2) < 1991] ## started at least in 1990
nam <- nam1[nam1 %in% nam2]
pub <- subset(pub, as.character(journal) %in% nam)
pub$journal <- factor(pub$journal)
pub_data <- pub

## generate time series
pub <- with(pub, tapply(type, year, table))
pub <- zoo(t(sapply(pub, cbind)), 1987:2006)
colnames(pub) <- levels(pub_data$type)

## visualize
plot(pub, ylim = c(0, 35))
```

---

sctest *Structural Change Tests*

Description

Generic function for performing structural change tests.

Usage

```
sctest(x, ...)
```

Arguments

x	an object.
...	arguments passed to methods.

Details

sctest is a generic function for performing/extracting structural change tests based on various types of objects. The strucchange package provides various types of methods.

First, structural change tests based on F statistics in linear regression models (Fstats), empirical fluctuation processes in linear regression models (efp), and generalized empirical fluctuation processes in parametric models (gefp) are available in the corresponding sctest methods.

Second, convenience interfaces for carrying out structural change tests in linear regression models and general parametric models are provided in sctest.formula and sctest.default, respectively.

Value

An object of class "htest" containing:

statistic	the test statistic,
p.value	the corresponding p value,
method	a character string with the method used,
data.name	a character string with the data name.

References

Zeileis A., Leisch F., Hornik K., Kleiber C. (2002), strucchange: An R Package for Testing for Structural Change in Linear Regression Models, *Journal of Statistical Software*, **7**(2), 1-38. doi:10.18637/jss.v007.i02.

Zeileis A. (2006), Implementing a Class of Structural Change Tests: An Econometric Computing Approach. *Computational Statistics & Data Analysis*, **50**, 2987–3008. doi:10.1016/j.csda.2005.07.001.

See Also

sctest.formula, sctest.default, sctest.Fstats, sctest.efp, sctest.gefp

`sctest.default` *Structural Change Tests in Parametric Models*

Description

Performs model-based tests for structural change (or parameter instability) in parametric models.

Usage

```
## Default S3 method:
sctest(x, order.by = NULL, functional = maxBB,
  vcov = NULL, scores = estfun, decorrelate = TRUE, sandwich = TRUE,
  parm = NULL, plot = FALSE, from = 0.1, to = NULL, nobs = NULL,
  nrep = 50000, width = 0.15, xlab = NULL, ...)
```

Arguments

`x`	a model object. The model class can in principle be arbitrary but needs to provide suitable methods for extracting the `scores` and associated variance-covariance matrix `vcov`.
`order.by`	either a vector `z` or a formula with a single explanatory variable like `~ z`. The observations in the model are ordered by the size of `z`. If set to `NULL` (the default) the observations are assumed to be ordered (e.g., a time series).
`functional`	either a character specification of the functional to be used or an `efpFunctional` object. For a list of functionals see the details.
`vcov`	a function to extract the covariance matrix for the coefficients of the fitted model: `vcov(x, order.by = order.by, data = data)`. Alternatively, the character string `"info"`, for details see below.
`scores`	a function which extracts the scores or estimating function from the fitted object: `scores(x)`, by default this is `estfun`.
`decorrelate`	logical. Should the process be decorrelated?
`sandwich`	logical. Is the function `vcov` the full sandwich estimator or only the meat?
`parm`	integer or character specifying the component of the estimating functions which should be used (by default all components are used).
`plot`	logical. Should the result of the test also be visualized?
`from, to`	numeric. In case the `functional` is `"supLM"` (or equivalently `"maxLM"`), `from` and `to` can be passed to the `supLM` functional.
`nobs, nrep`	numeric. In case the `functional` is `"maxLMo"`, `nobs` and `nrep` are passed to the `catL2BB` functional.
`width`	numeric. In case the `functional` is `"MOSUM"`, the bandwidth `width` is passed to the `maxMOSUM` functional.
`xlab, ...`	graphical parameters passed to the plot method (in case `plot = TRUE`).

Details

sctest.default is a convenience interface to gefp for structural change tests (or parameter instability tests) in general parametric models. It proceeds in the following steps:

1. The generalized empirical fluctuation process (or score-based CUSUM process) is computed via scus <- gefp(x, fit = NULL, ...) where ... comprises the arguments order.by, vcov, scores, decorrelate, sandwich, parm that are simply passed on to gefp.
2. The empirical fluctuation process is visualized (if plot = TRUE) via plot(scus, functional = functional, ...).
3. The empirical fluctuation is assessed by the corresponding significance test via sctest(scus, functional = functional).

The main motivation for prociding the convenience interface is that these three steps can be easily carried out in one go along with a two convenience options:

1. By default, the covariance is computed by an outer-product of gradients estimator just as in gefp. This is always available based on the scores. Additionally, by setting vcov = "info", the corresponding information matrix can be used. Then the average information is assumed to be provided by the vcov method for the model class. (Note that this is only sensible for models estimated by maximum likelihood.)
2. Instead of providing the functional by an efpFunctional object, the test labels employed by Merkle and Zeileis (2013) and Merkle, Fan, and Zeileis (2013) can be used for convenience. Namely, for continuous numeric orderings, the following functionals are available: functional = "DM" or "dmax" provides the double-maximum test (maxBB). "CvM" is the Cramer-von Mises functional meanL2BB. "supLM" or equivalently "maxLM" is Andrews' supLM test (supLM). "MOSUM" or "maxMOSUM" is the MOSUM functional (maxMOSUM), and "range" is the range functional rangeBB. Furthermore, several functionals suitable for (ordered) categorical order.by variables are provided: "LMuo" is the unordered LM test (catL2BB), "WDMo" is the weighted double-maximum test for ordered variables (ordwmax), and "maxLMo" is the maxLM test for ordered variables (ordL2BB).

The theoretical model class is introduced in Zeileis and Hornik (2007) with a unifying view in Zeileis (2005), especially from an econometric perspective. Zeileis (2006) introduces the underling computational tools gefp and efpFunctional.

Merkle and Zeileis (2013) discuss the methods in the context of measurement invariance which is particularly relevant to psychometric models for cross section data. Merkle, Fan, and Zeileis (2014) extend the results to ordered categorical variables.

Zeileis, Shah, and Patnaik (2013) provide a unifying discussion in the context of time series methods, specifically in financial econometrics.

Value

An object of class "htest" containing:

statistic	the test statistic,
p.value	the corresponding p value,
method	a character string with the method used,
data.name	a character string with the data name.

References

Merkle E.C., Zeileis A. (2013), Tests of Measurement Invariance without Subgroups: A Generalization of Classical Methods. *Psychometrika*, **78**(1), 59–82. doi:10.1007/S11336-012-9302-4

Merkle E.C., Fan J., Zeileis A. (2014), Testing for Measurement Invariance with Respect to an Ordinal Variable. *Psychometrika*, **79**(4), 569–584. doi:10.1007/S11336-013-9376-7.

Zeileis A. (2005), A Unified Approach to Structural Change Tests Based on ML Scores, F Statistics, and OLS Residuals. *Econometric Reviews*, **24**, 445–466. doi:10.1080/07474930500406053.

Zeileis A. (2006), Implementing a Class of Structural Change Tests: An Econometric Computing Approach. *Computational Statistics & Data Analysis*, **50**, 2987–3008. doi:10.1016/j.csda.2005.07.001.

Zeileis A., Hornik K. (2007), Generalized M-Fluctuation Tests for Parameter Instability, *Statistica Neerlandica*, **61**, 488–508. doi:10.1111/j.1467-9574.2007.00371.x.

Zeileis A., Shah A., Patnaik I. (2010), Testing, Monitoring, and Dating Structural Changes in Exchange Rate Regimes, *Computational Statistics and Data Analysis*, **54**(6), 1696–1706. doi:10.1016/j.csda.2009.12.005.

See Also

gefp, efpFunctional

Examples

```
## Zeileis and Hornik (2007), Section 5.3, Figure 6
data("Grossarl")
m <- glm(cbind(illegitimate, legitimate) ~ 1, family = binomial, data = Grossarl,
  subset = time(fraction) <= 1800)
sctest(m, order.by = 1700:1800, functional = "CvM")
```

---

sctest.efp *Generalized Fluctuation Tests*

Description

Performs a generalized fluctuation test.

Usage

```
## S3 method for class 'efp'
sctest(x, alt.boundary = FALSE,
    functional = c("max", "range", "maxL2", "meanL2"), ...)
```

Arguments

x	an object of class "efp".
alt.boundary	logical. If set to TRUE alternative boundaries (instead of the standard linear boundaries) will be used (for CUSUM processes only).
functional	indicates which functional should be applied to the empirical fluctuation process.
...	currently not used.

Details

The critical values for the MOSUM tests and the ME test are just tabulated for confidence levels between 0.1 and 0.01, thus the p value approximations will be poor for other p values. Similarly the critical values for the maximum and mean squared Euclidean norm (`"maxL2"` and `"meanL2"`) are tabulated for confidence levels between 0.2 and 0.005.

Value

An object of class `"htest"` containing:

`statistic`	the test statistic,
`p.value`	the corresponding p value,
`method`	a character string with the method used,
`data.name`	a character string with the data name.

References

Brown R.L., Durbin J., Evans J.M. (1975), Techniques for testing constancy of regression relationships over time, *Journal of the Royal Statistical Society*, B, **37**, 149-163.

Chu C.-S., Hornik K., Kuan C.-M. (1995), MOSUM tests for parameter constancy, *Biometrika*, **82**, 603-617.

Chu C.-S., Hornik K., Kuan C.-M. (1995), The moving-estimates test for parameter stability, *Econometric Theory*, **11**, 669-720.

Krämer W., Ploberger W., Alt R. (1988), Testing for structural change in dynamic models, *Econometrica*, **56**, 1355-1369.

Kuan C.-M., Hornik K. (1995), The generalized fluctuation test: A unifying view, *Econometric Reviews*, **14**, 135 - 161.

Kuan C.-M., Chen (1994), Implementing the fluctuation and moving estimates tests in dynamic econometric models, *Economics Letters*, **44**, 235-239.

Ploberger W., Krämer W. (1992), The CUSUM Test with OLS Residuals, *Econometrica*, **60**, 271-285.

Zeileis A., Leisch F., Hornik K., Kleiber C. (2002), `strucchange`: An R Package for Testing for Structural Change in Linear Regression Models, *Journal of Statistical Software*, **7**(2), 1-38. doi:10.18637/jss.v007.i02.

Zeileis A. (2004), Alternative Boundaries for CUSUM Tests, *Statistical Papers*, **45**, 123–131.

See Also

`efp`, `plot.efp`

Examples

```
## Load dataset "nhtemp" with average yearly temperatures in New Haven
data("nhtemp")
## plot the data
plot(nhtemp)
```

```
## test the model null hypothesis that the average temperature remains
## constant over the years compute OLS-CUSUM fluctuation process
temp.cus <- efp(nhtemp ~ 1, type = "OLS-CUSUM")
## plot the process with alternative boundaries
plot(temp.cus, alpha = 0.01, alt.boundary = TRUE)
## and calculate the test statistic
sctest(temp.cus)

## compute moving estimates fluctuation process
temp.me <- efp(nhtemp ~ 1, type = "ME", h = 0.2)
## plot the process with functional = "max"
plot(temp.me)
## and perform the corresponding test
sctest(temp.me)
```

`sctest.formula` *Structural Change Tests in Linear Regression Models*

Description

Performs tests for structural change in linear regression models.

Usage

```
## S3 method for class 'formula'
sctest(formula, type = , h = 0.15,
    alt.boundary = FALSE, functional = c("max", "range",
    "maxL2", "meanL2"), from = 0.15, to = NULL, point = 0.5,
    asymptotic = FALSE, data, ...)
```

Arguments

`formula`	a formula describing the model to be tested.
`type`	a character string specifying the structural change test that is to be performed, the default is `"Rec-CUSUM"`. Besides the test types described in `efp` and `sctest.Fstats` the Chow test and the Nyblom-Hansen test can be performed by setting type to `"Chow"` or `"Nyblom-Hansen"`, respectively.
`h`	numeric from interval (0,1) specifying the bandwidth. Determines the size of the data window relative to the sample size (for MOSUM and ME tests only).
`alt.boundary`	logical. If set to `TRUE` alternative boundaries (instead of the standard linear boundaries) will be used (for CUSUM processes only).
`functional`	indicates which functional should be used to aggregate the empirical fluctuation processes to a test statistic.

from, to	numeric. If from is smaller than 1 they are interpreted as percentages of data and by default to is taken to be the 1 - from. F statistics will be calculated for the observations (n*from):(n*to), when n is the number of observations in the model. If from is greater than 1 it is interpreted to be the index and to defaults to n - from. (for F tests only)
point	parameter of the Chow test for the potential change point. Interpreted analogous to the from parameter. By default taken to be floor(n*0.5) if n is the number of observations in the model.
asymptotic	logical. If TRUE the asymptotic (chi-square) distribution instead of the exact (F) distribution will be used to compute the p value (for Chow test only).
data	an optional data frame containing the variables in the model. By default the variables are taken from the environment which sctest is called from.
...	further arguments passed to efp or Fstats.

Details

sctest.formula is a convenience interface for performing structural change tests in linear regression models based on efp and Fstats. It is mainly a wrapper for sctest.efp and sctest.Fstats as it fits an empirical fluctuation process first or computes the F statistics respectively and subsequently performs the corresponding test. The Chow test and the Nyblom-Hansen test are available explicitly here.

An alternative convenience interface for performing structural change tests in general parametric models (based on gefp) is available in sctest.default.

Value

An object of class "htest" containing:

statistic	the test statistic,
p.value	the corresponding p value,
method	a character string with the method used,
data.name	a character string with the data name.

See Also

sctest.efp, sctest.Fstats, sctest.default

Examples

```
## Example 7.4 from Greene (1993), "Econometric Analysis"
## Chow test on Longley data
data("longley")
sctest(Employed ~ Year + GNP.deflator + GNP + Armed.Forces, data = longley,
  type = "Chow", point = 7)

## which is equivalent to segmenting the regression via
fac <- factor(c(rep(1, 7), rep(2, 9)))
fm0 <- lm(Employed ~ Year + GNP.deflator + GNP + Armed.Forces, data = longley)
```

```
fm1 <- lm(Employed ~ fac/(Year + GNP.deflator + GNP + Armed.Forces), data = longley)
anova(fm0, fm1)

## estimates from Table 7.5 in Greene (1993)
summary(fm0)
summary(fm1)
```

`sctest.Fstats` *supF-, aveF- and expF-Test*

Description

Performs the supF-, aveF- or expF-test

Usage

```
## S3 method for class 'Fstats'
sctest(x, type = c("supF", "aveF", "expF"),
    asymptotic = FALSE, ...)
```

Arguments

`x`	an object of class `"Fstats"`.
`type`	a character string specifying which test will be performed.
`asymptotic`	logical. Only necessary if `x` contains just a single F statistic and type is `"supF"` or `"aveF"`. If then set to `TRUE` the asymptotic (chi-square) distribution instead of the exact (F) distribution will be used to compute the p value.
`...`	currently not used.

Details

If `x` contains just a single F statistic and type is `"supF"` or `"aveF"` the Chow test will be performed.

The original GAUSS code for computing the p values of the supF-, aveF- and expF-test was written by Bruce Hansen and is available from https://www.ssc.wisc.edu/~bhansen/. R port by Achim Zeileis.

Value

An object of class `"htest"` containing:

`statistic`	the test statistic,
`p.value`	the corresponding p value,
`method`	a character string with the method used,
`data.name`	a character string with the data name.

References

Andrews D.W.K. (1993), Tests for parameter instability and structural change with unknown change point, *Econometrica*, **61**, 821-856.

Andrews D.W.K., Ploberger W. (1994), Optimal tests when a nuisance parameter is present only under the alternative, *Econometrica*, **62**, 1383-1414.

Hansen B. (1992), Tests for parameter instability in regressions with I(1) processes, *Journal of Business & Economic Statistics*, **10**, 321-335.

Hansen B. (1997), Approximate asymptotic p values for structural-change tests, *Journal of Business & Economic Statistics*, **15**, 60-67.

See Also

`Fstats`, `plot.Fstats`

Examples

```
## Load dataset "nhtemp" with average yearly temperatures in New Haven
data(nhtemp)
## plot the data
plot(nhtemp)

## test the model null hypothesis that the average temperature remains
## constant over the years for potential break points between 1941
## (corresponds to from = 0.5) and 1962 (corresponds to to = 0.85)
## compute F statistics
fs <- Fstats(nhtemp ~ 1, from = 0.5, to = 0.85)
## plot the F statistics
plot(fs, alpha = 0.01)
## and the corresponding p values
plot(fs, pval = TRUE, alpha = 0.01)
## perform the aveF test
sctest(fs, type = "aveF")
```

---

solveCrossprod *Inversion of X'X*

Description

Computes the inverse of the cross-product of a matrix X.

Usage

```
solveCrossprod(X, method = c("qr", "chol", "solve"))
```

Arguments

`X`	a matrix, typically a regressor matrix.
`method`	a string indicating whether the QR decomposition, the Cholesky decomposition or `solve` should be used.

Details

Using the Cholesky decomposition of X'X (as computed by `crossprod(X)`) is computationally faster and preferred to `solve(crossprod(X))`. Using the QR decomposition of X is slower but should be more accurate.

Value

a matrix containing the inverse of `crossprod(X)`.

Examples

```
X <- cbind(1, rnorm(100))
solveCrossprod(X)
solve(crossprod(X))
```

`SP2001` *S&P 500 Stock Prices*

Description

A multivariate series of all S&P 500 stock prices in the second half of the year 2001, i.e., before and after the terrorist attacks of 2001-09-11.

Usage

```
data("SP2001")
```

Format

A multivariate daily `"zoo"` series with `"Date"` index from 2001-07-31 to 2001-12-31 (103 observations) of all 500 S&P stock prices.

Source

Yahoo! Finance: https://finance.yahoo.com/.

References

Zeileis A., Leisch F., Kleiber C., Hornik K. (2005), Monitoring Structural Change in Dynamic Econometric Models, *Journal of Applied Econometrics*, **20**, 99–121.

See Also

`get.hist.quote`

Examples

```
## load and transform data
## (DAL: Delta Air Lines, LU: Lucent Technologies)
data("SP2001")
stock.prices <- SP2001[, c("DAL", "LU")]
stock.returns <- diff(log(stock.prices))

## price and return series
plot(stock.prices, ylab = c("Delta Air Lines", "Lucent Technologies"), main = "")
plot(stock.returns, ylab = c("Delta Air Lines", "Lucent Technologies"), main = "")

## monitoring of DAL series
myborder <- function(k) 1.939*k/28
x <- as.vector(stock.returns[, "DAL"][1:28])
dal.cusum <- mefp(x ~ 1, type = "OLS-CUSUM", border = myborder)
dal.mosum <- mefp(x ~ 1, type = "OLS-MOSUM", h = 0.5, period = 4)
x <- as.vector(stock.returns[, "DAL"])
dal.cusum <- monitor(dal.cusum)
dal.mosum <- monitor(dal.mosum)

## monitoring of LU series
x <- as.vector(stock.returns[, "LU"][1:28])
lu.cusum <- mefp(x ~ 1, type = "OLS-CUSUM", border = myborder)
lu.mosum <- mefp(x ~ 1, type = "OLS-MOSUM", h = 0.5, period = 4)
x <- as.vector(stock.returns[, "LU"])
lu.cusum <- monitor(lu.cusum)
lu.mosum <- monitor(lu.mosum)

## pretty plotting
## (needs some work because lm() does not keep "zoo" attributes)
cus.bound <- zoo(c(rep(NA, 27), myborder(28:102)), index(stock.returns))
mos.bound <- as.vector(boundary(dal.mosum))
mos.bound <- zoo(c(rep(NA, 27), mos.bound[1], mos.bound), index(stock.returns))

## Lucent Technologies: CUSUM test
plot(zoo(c(lu.cusum$efpprocess, lu.cusum$process), index(stock.prices)),
 ylim = c(-1, 1) * coredata(cus.bound)[102], xlab = "Time", ylab = "empirical fluctuation process")
abline(0, 0)
abline(v = as.Date("2001-09-10"), lty = 2)
lines(cus.bound, col = 2)
lines(-cus.bound, col = 2)

## Lucent Technologies: MOSUM test
plot(zoo(c(lu.mosum$efpprocess, lu.mosum$process), index(stock.prices)[-(1:14)]),
 ylim = c(-1, 1) * coredata(mos.bound)[102], xlab = "Time", ylab = "empirical fluctuation process")
abline(0, 0)
abline(v = as.Date("2001-09-10"), lty = 2)
lines(mos.bound, col = 2)
lines(-mos.bound, col = 2)

## Delta Air Lines: CUSUM test
plot(zoo(c(dal.cusum$efpprocess, dal.cusum$process), index(stock.prices)),
```

```
 ylim = c(-1, 1) * coredata(cus.bound)[102], xlab = "Time", ylab = "empirical fluctuation process")
abline(0, 0)
abline(v = as.Date("2001-09-10"), lty = 2)
lines(cus.bound, col = 2)
lines(-cus.bound, col = 2)

## Delta Air Lines: MOSUM test
plot(zoo(c(dal.mosum$efpprocess, dal.mosum$process), index(stock.prices)[-(1:14)]),
  ylim = range(dal.mosum$process), xlab = "Time", ylab = "empirical fluctuation process")
abline(0, 0)
abline(v = as.Date("2001-09-10"), lty = 2)
lines(mos.bound, col = 2)
lines(-mos.bound, col = 2)
```

---

supLM *Generators for efpFunctionals along Continuous Variables*

Description

Generators for `efpFunctional` objects suitable for aggregating empirical fluctuation processes to test statistics along continuous variables (i.e., along time in time series applications).

Usage

```
supLM(from = 0.15, to = NULL)

maxMOSUM(width = 0.15)
```

Arguments

`from`, `to`	numeric from interval (0, 1) specifying start and end of trimmed sample period. By default, `to` is `1 - from`, i.e., with the default `from = 0.15` the first and last 15 percent of observations are trimmed.
`width`	a numeric from interval (0,1) specifying the bandwidth. Determines the size of the moving data window relative to sample size.

Details

`supLM` and `maxMOSUM` generate `efpFunctional` objects for Andrews' supLM test and a (maximum) MOSUM test, respectively, with the specified optional parameters (`from` and `to`, and `width`, respectively). The resulting objects can be used in combination with empirical fluctuation processes of class `gefp` for significance testing and visualization. The corresponding statistics are useful for carrying out structural change tests along a continuous variable (i.e., along time in time series applications). Further typical `efpFunctional`s for this setting are the double-maximum functional `maxBB` and the Cramer-von Mises functional `meanL2BB`.

Value

An object of class `efpFunctional`.

References

Merkle E.C., Zeileis A. (2013), Tests of Measurement Invariance without Subgroups: A Generalization of Classical Methods. *Psychometrika*, **78**(1), 59–82. doi:10.1007/S11336-012-9302-4

Zeileis A. (2005), A Unified Approach to Structural Change Tests Based on ML Scores, F Statistics, and OLS Residuals. *Econometric Reviews*, **24**, 445–466. doi:10.1080/07474930500406053.

Zeileis A. (2006), Implementing a Class of Structural Change Tests: An Econometric Computing Approach. *Computational Statistics & Data Analysis*, **50**, 2987–3008. doi:10.1016/j.csda.2005.07.001.

Zeileis A., Hornik K. (2007), Generalized M-Fluctuation Tests for Parameter Instability, *Statistica Neerlandica*, **61**, 488–508. doi:10.1111/j.1467-9574.2007.00371.x.

See Also

efpFunctional, gefp

Examples

```
## seatbelt data
data("UKDriverDeaths")
seatbelt <- log10(UKDriverDeaths)
seatbelt <- cbind(seatbelt, lag(seatbelt, k = -1), lag(seatbelt, k = -12))
colnames(seatbelt) <- c("y", "ylag1", "ylag12")
seatbelt <- window(seatbelt, start = c(1970, 1), end = c(1984,12))

## empirical fluctuation process
scus.seat <- gefp(y ~ ylag1 + ylag12, data = seatbelt)

## supLM test
plot(scus.seat, functional = supLM(0.1))
## MOSUM test
plot(scus.seat, functional = maxMOSUM(0.25))
## double maximum test
plot(scus.seat)
## range test
plot(scus.seat, functional = rangeBB)
## Cramer-von Mises statistic (Nyblom-Hansen test)
plot(scus.seat, functional = meanL2BB)
```

USIncExp *Income and Expenditures in the US*

Description

Personal income and personal consumption expenditures in the US between January 1959 and February 2001 (seasonally adjusted at annual rates).

Usage

```
data("USIncExp")
```

Format

A multivariate monthly time series from 1959(1) to 2001(2) with variables

income monthly personal income (in billion US dollars),

expenditure monthly personal consumption expenditures (in billion US Dollars).

Source

https://www.economagic.com/

References

A. Zeileis, F. Leisch, K. Hornik, C. Kleiber (2002), strucchange: An R Package for Testing for Structural Change in Linear Regression Models. *Journal of Statistical Software* **7**(2), 1–38.

Examples

```
## These example are presented in the vignette distributed with this
## package, the code was generated by Stangle("strucchange-intro.Rnw")

###################################################
### chunk number 1: data
###################################################
library("strucchange")
data("USIncExp")
plot(USIncExp, plot.type = "single", col = 1:2, ylab = "billion US$")
legend(1960, max(USIncExp), c("income", "expenditures"),
       lty = c(1,1), col = 1:2, bty = "n")


###################################################
### chunk number 2: subset
###################################################
library("strucchange")
data("USIncExp")
USIncExp2 <- window(USIncExp, start = c(1985,12))


###################################################
### chunk number 3: ecm-setup
###################################################
coint.res <- residuals(lm(expenditure ~ income, data = USIncExp2))
coint.res <- lag(ts(coint.res, start = c(1985,12), freq = 12), k = -1)
USIncExp2 <- cbind(USIncExp2, diff(USIncExp2), coint.res)
USIncExp2 <- window(USIncExp2, start = c(1986,1), end = c(2001,2))
colnames(USIncExp2) <- c("income", "expenditure", "diff.income",
                         "diff.expenditure", "coint.res")
ecm.model <- diff.expenditure ~ coint.res + diff.income


###################################################
### chunk number 4: ts-used
```

```
####################################################
plot(USIncExp2[,3:5], main = "")


####################################################
### chunk number 5: efp
####################################################
ocus <- efp(ecm.model, type="OLS-CUSUM", data=USIncExp2)
me <- efp(ecm.model, type="ME", data=USIncExp2, h=0.2)


####################################################
### chunk number 6: efp-boundary
####################################################
bound.ocus <- boundary(ocus, alpha=0.05)


####################################################
### chunk number 7: OLS-CUSUM
####################################################
plot(ocus)


####################################################
### chunk number 8: efp-boundary2
####################################################
plot(ocus, boundary = FALSE)
lines(bound.ocus, col = 4)
lines(-bound.ocus, col = 4)


####################################################
### chunk number 9: ME-null
####################################################
plot(me, functional = NULL)


####################################################
### chunk number 10: efp-sctest
####################################################
sctest(ocus)


####################################################
### chunk number 11: efp-sctest2
####################################################
sctest(ecm.model, type="OLS-CUSUM", data=USIncExp2)


####################################################
### chunk number 12: Fstats
####################################################
fs <- Fstats(ecm.model, from = c(1990, 1), to = c(1999,6), data = USIncExp2)
```

```
###################################################
### chunk number 13: Fstats-plot
###################################################
plot(fs)


###################################################
### chunk number 14: pval-plot
###################################################
plot(fs, pval=TRUE)


###################################################
### chunk number 15: aveF-plot
###################################################
plot(fs, aveF=TRUE)


###################################################
### chunk number 16: Fstats-sctest
###################################################
sctest(fs, type="expF")


###################################################
### chunk number 17: Fstats-sctest2
###################################################
sctest(ecm.model, type = "expF", from = 49, to = 162, data = USIncExp2)


###################################################
### chunk number 18: mefp
###################################################
USIncExp3 <- window(USIncExp2, start = c(1986, 1), end = c(1989,12))
me.mefp <- mefp(ecm.model, type = "ME", data = USIncExp3, alpha = 0.05)


###################################################
### chunk number 19: monitor1
###################################################
USIncExp3 <- window(USIncExp2, start = c(1986, 1), end = c(1990,12))
me.mefp <- monitor(me.mefp)


###################################################
### chunk number 20: monitor2
###################################################
USIncExp3 <- window(USIncExp2, start = c(1986, 1))
me.mefp <- monitor(me.mefp)
me.mefp
```

```
###################################################
### chunk number 21: monitor-plot
###################################################
plot(me.mefp)


###################################################
### chunk number 22: mefp2
###################################################
USIncExp3 <- window(USIncExp2, start = c(1986, 1), end = c(1989,12))
me.efp <- efp(ecm.model, type = "ME", data = USIncExp3, h = 0.5)
me.mefp <- mefp(me.efp, alpha=0.05)


###################################################
### chunk number 23: monitor3
###################################################
USIncExp3 <- window(USIncExp2, start = c(1986, 1))
me.mefp <- monitor(me.mefp)


###################################################
### chunk number 24: monitor-plot2
###################################################
plot(me.mefp)
```

Index