Package ‘sandwich’

April 7, 2019

Version 2.5-1
Date 2019-04-06
Title Robust Covariance Matrix Estimators
Description Model-robust standard error estimators for cross-sectional, time series, clustered, panel, and longitudinal data.
Depends R (>= 2.10.0)
Imports stats, utils, zoo
Suggests AER, car, geepack, lattice, lmtest, MASS, multiwayvcov, parallel, pcse, plm, pscl, scatterplot3d, stats4, strucchange, survival
License GPL-2 | GPL-3
NeedsCompilation no
Author Achim Zeileis [aut, cre] (<https://orcid.org/0000-0003-0918-3766>),
Thomas Lumley [aut],
Susanne Berger [ctb],
Nathaniel Graham [ctb]
Maintainer Achim Zeileis <Achim.Zeileis@R-project.org>
Repository CRAN
Date/Publication 2019-04-06 21:59:16 UTC

R topics documented:

bread ................................................................. 2
estfun .............................................................. 3
InstInnovation ..................................................... 4
Investment ......................................................... 6
isoacf ............................................................... 8
kweights ............................................................ 9
lrvar ............................................................... 10
meat ............................................................... 11
NeweyWest ......................................................... 12
PetersenCL ......................................................... 14
bread

**Description**

Generic function for extracting an estimator for the bread of sandwiches.

**Usage**

```r
bread(x, ...)  
```

**Arguments**

- `x` : a fitted model object.
- `...` : arguments passed to methods.

**Value**

A matrix containing an estimator for the expectation of the negative derivative of the estimating functions, usually the Hessian. Typically, this should be an \( k \times k \) matrix corresponding to \( k \) parameters. The rows and columns should be named as in `coef` or `terms`, respectively.

The default method tries to extract `vcov` and `nobs` and simply computes their product.

**References**


**See Also**

`lm`, `glm`
Examples

```r
## linear regression
x <- sin(1:10)
y <- rnorm(10)
fm <- lm(y ~ x)

## bread: n * (x'x)^{-1}
bread(fm)
solve(crossprod(cbind(1, x))) * 10
```

---

**estfun**

*Extract Empirical Estimating Functions*

**Description**

Generic function for extracting the empirical estimating functions of a fitted model.

**Usage**

```r
estfun(x, ...)
```

**Arguments**

- `x` a fitted model object.
- `...` arguments passed to methods.

**Value**

A matrix containing the empirical estimating functions. Typically, this should be an \( n \times k \) matrix corresponding to \( n \) observations and \( k \) parameters. The columns should be named as in `coef` or `terms`, respectively.

The estimating function (or score function) for a model is the derivative of the objective function with respect to the parameter vector. The empirical estimating functions is the evaluation of the estimating function at the observed data (\( n \) observations) and the estimated parameters (of dimension \( k \)).

**References**


**See Also**

`lm`, `glm`
**Examples**

```r
## linear regression
x <- sin(1:10)
y <- rnorm(10)
f <- lm(y ~ x)

## estimating function: (y - x*beta) * x
estfun(fm)
residuals(fm) * cbind(1, x)
```

---

**InstInnovation**

**Innovation and Institutional Ownership**

**Description**

Firm-level panel data on innovation and institutional ownership from 1991 to 1999 over 803 firms. The observations refer to different firms over different years.

**Usage**

```r
data("InstInnovation")
```

**Format**

A data frame containing 6208 observations on 25 variables.

- **company** factor. Company names.
- **sales** numeric. Sales (in millions of dollars).
- **acompetition** numeric. Constant inverse Lerner index.
- **competition** numeric. Varying inverse Lerner index.
- **capital** numeric. Net stock of property, plant, and equipment.
- **cites** integer. Future cite-weighted patents.
- **precites** numeric. Presample average of cite-weighted patents.
- **dprecites** factor. Indicates zero precites.
- **patents** integer. Granted patents.
- **drandd** factor. Indicates a zero R&D stock.
- **randd** numeric. R&D stock (in millions of dollars).
- **employment** numeric. Employment (in 1000s).
- **sp500** factor. Membership of firms in the S&P500 index.
- **tobing** numeric. Tobin’s q.
- **value** numeric. Stock market value.
- **institutions** numeric. Proportion of stock owned by institutions.
- **industry** factor. Four-digit industry code.
year factor. Estimation period.

top1 numeric. Share of the largest institution.

quasiindexed numeric. Share of "quasi-indexed" institutional owners.

nonquasiindexed numeric. Share of "non-quasi-indexed" institutional owners.

transient numeric. Share of "transient" institutional owners.

dedicated numeric. Share of "dedicated" institutional owners.

competition4 numeric. Varying inverse Lerner index in the firm’s four-digit industry.

subsample factor. Subsample for the replication of columns 1–5 from Table 4 in Aghion et al. (2013).

Details

Aghion et al. (2013) combine several firm level panel datasets (e.g., USPTO, SEC and Compustat) to examine the role of institutional investors in the governance of innovation. Their baseline to model innovation is the Poisson model, but they also consider negative binomial models. Berger et al. (2017) argue that nonlinearities in the innovation process emerge in case that the first innovation is especially hard to obtain in comparison to succeeding innovations. Then, hurdle models offer a useful way that allows for a distinction between these two processes. Berger et al. (2017) show that an extended analysis with negative binomial hurdle models differs materially from the outcomes of the single-equation Poisson approach of Aghion et al. (2013).

Institutional ownership (institutions) is defined as the proportion of stock owney by institutions. According to Aghion et al. (2013), an institutional owner is defined as an institution that files a Form 13-F with the Securities and Exchange Commission (SEC).

Future cite-weighted patents (cites) are used as a proxy for innovation. They are calculated using ultimately granted patent, dated by year of application, and weight these by future citations through 2002 (see Aghion et al. (2013)).

The presample average of cite-weighted patents (precites) is used by Aghion et al. (2013) as a proxy for unobserved heterogeneity, employing the "presample mean scaling" method of Blundell et al. (1999).

The inverse Lerner index in the firm’s three-digit industry is used as a time-varying measure for product market competition (competition), where the Lerner is calculated as the median gross margin from the entire Compustat database in the firm’s three-digit industry (see Aghion et al. (2013)). A time-invariant measure for competition (acompetition) is constructed by averaging the Lerner over the sample period.

The classification of institutions into "quasiindexed", "transient" and "dedicated" follows Bushee (1998) and distinguishes between institutional investors based on their type of investing. Quasi-indexed institutions are do not trade much and are widely diversified, dedicated institution do not trade much and have more concentrated holdings, and transient institutions often trade and have diversified holdings (see Aghion et al. (2013) and Bushee (1998)).

Source

Data and online appendix of Aghion et al. (2013).
References


Examples

```r
## Poisson models from Table I in Aghion et al. (2013)

## load data set
data("InstInnovation", package = "sandwich")

## log-scale variable
InstInnovation$lograndd <- log(InstInnovation$randd)
InstInnovation$lograndd[InstInnovation$lograndd == -Inf] <- 0

## regression formulas
f1 <- cites ~ institutions + log(capital/employment) + log(sales) + industry + year
f2 <- cites ~ institutions + log(capital/employment) + log(sales) +
     industry + year + lograndd + drandd
f3 <- cites ~ institutions + log(capital/employment) + log(sales) +
     industry + year + lograndd + drandd + dprecites + log(precites)

## Poisson models
tab_I_3_pois <- glm(f1, data = InstInnovation, family = poisson)
tab_I_4_pois <- glm(f2, data = InstInnovation, family = poisson)
tab_I_5_pois <- glm(f3, data = InstInnovation, family = poisson)

## one-way clustered covariances
vCL_I_3 <- vcovCL(tab_I_3_pois, cluster = ~ company)
vCL_I_4 <- vcovCL(tab_I_4_pois, cluster = ~ company)
vCL_I_5 <- vcovCL(tab_I_5_pois, cluster = ~ company)

## replication of columns 3 to 5 from Table I in Aghion et al. (2013)
cbind(coef(tab_I_3_pois), sqrt(diag(vCL_I_3)))[2:4,]
cbind(coef(tab_I_4_pois), sqrt(diag(vCL_I_4)))[c(2:4, 148),]
cbind(coef(tab_I_5_pois), sqrt(diag(vCL_I_5)))[c(2:4, 148),]
```

<table>
<thead>
<tr>
<th>Investment</th>
<th>US Investment Data</th>
</tr>
</thead>
</table>

---
Description

US data for fitting an investment equation.

Usage

data(Investment)

Format

An annual time series from 1963 to 1982 with 7 variables.

- **GNP** nominal gross national product (in billion USD),
- **Investment** nominal gross private domestic investment (in billion USD),
- **Price** price index, implicit price deflator for GNP,
- **Interest** interest rate, average yearly discount rate charged by the New York Federal Reserve Bank,
- **RealGNP** real GNP (= GNP/Price),
- **RealInv** real investment (= Investment/Price),
- **RealInt** approximation to the real interest rate (= Interest - 100 * diff(Price)/Price).

Source

Table 15.1 in Greene (1993)

References


Examples

```r
## William H. Greene, Econometric Analysis, 2nd Ed.
## Chapter 15
## load data set, p. 411, Table 15.1
data(Investment)

## fit linear model, p. 412, Table 15.2
fm <- lm(RealInv ~ RealGNP + RealInt, data = Investment)
summary(fm)

## visualize residuals, p. 412, Figure 15.1
plot(ts(residuals(fm), start = 1964),
     type = "b", pch = 19, ylim = c(-35, 35), ylab = "Residuals")
sigma <- sqrt(sum(residuals(fm)^2)/fm$df.residual) # maybe used df = 26 instead of 16 ??
abline(h = c(-2, 0, 2) * sigma, lty = 2)

if(require(lmtest)) {
  ## Newey-West covariances, Example 15.3
```
isoacf

Isotonic Autocorrelation Function

Description

Autocorrelation function (forced to be decreasing by isotonic regression).

Usage

isoacf(x, lagmax = NULL, weave = FALSE)

Arguments

x

numeric vector.

lagmax

numeric. The maximal lag of the autocorrelations.

weave

logical. If set to TRUE isoacf uses the acf.R and pava.blocks function from the original weave package, otherwise R's own acf and isoreg functions are used.

Details

isoacf computes the autocorrelation function (ACF) of x enforcing the ACF to be decreasing by isotonic regression. See also Robertson et al. (1988).

Value

isoacf returns a numeric vector containing the ACF.
Kernel Weights

Description


Usage

```r
kweights(x, kernel = c("Truncated", "Bartlett", "Parzen", "Tukey-Hanning", "Quadratic Spectral"), normalize = FALSE)
```

Arguments

- `x`: numeric.
- `kernel`: a character specifying the kernel used. All kernels used are described in Andrews (1991).
- `normalize`: logical. If set to TRUE the kernels are normalized as described in Andrews (1991).

Value

Value of the kernel function at x.

References

lrvar

Long-Run Variance of the Mean

Description
Convenience function for computing the long-run variance (matrix) of a (possibly multivariate) series of observations.

Usage
lrvar(x, type = c("Andrews", "Newey-West"), prewhite = TRUE, adjust = TRUE, ...)

Arguments
x numeric vector, matrix, or time series.
type character specifying the type of estimator, i.e., whether kernHAC for the Andrews quadratic spectral kernel HAC estimator is used or NeweyWest for the Newey-West Bartlett HAC estimator.
prewhite logical or integer. Should the series be prewhitened? Passed to kernHAC or NeweyWest.
adjust logical. Should a finite sample adjustment be made? Passed to kernHAC or NeweyWest.
...
further arguments passed on to kernHAC or NeweyWest.

Details
lrvar is a simple wrapper function for computing the long-run variance (matrix) of a (possibly multivariate) series x. First, this simply fits a linear regression model x ~ 1 by lm. Second, the corresponding variance of the mean(s) is estimated either by kernHAC (Andrews quadratic spectral kernel HAC estimator) or by NeweyWest (Newey-West Bartlett HAC estimator).
**Value**

For a univariate series \( x \) a scalar variance is computed. For a multivariate series \( x \) the covariance matrix is computed.

**See Also**

`kernHAC`, `NeweyWest`, `vcovHAC`

**Examples**

```r
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
## iid series (with variance of mean 1/n)
## and Andrews kernel HAC (with prewhitening)
x <- rnorm(100)
lrvar(x)

## analogous multivariate case with Newey-West estimator (without prewhitening)
y <- matrix(rnorm(200), ncol = 2)
lrvar(y, type = "Newey-West", prewhite = FALSE)

## AR(1) series with autocorrelation 0.9
z <- filter(rnorm(100), 0.9, method = "recursive")
lrvar(z)
```

---

**meat**

*A Simple Meat Matrix Estimator*

**Description**

Estimating the variance of the estimating functions of a regression model by cross products of the empirical estimating functions.

**Usage**

```r
meat(x, adjust = FALSE, ...)
```

**Arguments**

- `x`  
  a fitted model object.
- `adjust`  
  logical. Should a finite sample adjustment be made? This amounts to multiplication with \( n/(n - k) \) where \( n \) is the number of observations and \( k \) the number of estimated parameters.
- `...`  
  arguments passed to the `estfun` function.

**Details**

For some theoretical background along with implementation details see Zeileis (2006).
Value

A \( k \times k \) matrix corresponding containing the scaled cross products of the empirical estimating functions.

References


See Also

sandwich, bread, estfun

Examples

```r
x <- sin(1:10)
y <- rnorm(10)
fm <- lm(y ~ x)

meat(fm)
meatHC(fm, type = "HC")
meatHAC(fm)
```

Description


Usage

```r
NeweyWest(x, lag = NULL, order.by = NULL, prewhite = TRUE, adjust = FALSE,
          diagnostics = FALSE, sandwich = TRUE, ar.method = "ols", data = list(),
          verbose = FALSE)

bwNeweyWest(x, order.by = NULL, kernel = c("Bartlett", "Parzen",
       "Quadratic Spectral", "Truncated", "Tukey-Hanning"), weights = NULL,
       prewhite = 1, ar.method = "ols", data = list(), ...)
```

Arguments

- **x**: a fitted model object.
- **lag**: integer specifying the maximum lag with positive weight for the Newey-West estimator. If set to NULL \( \text{floor}(\text{bwNeweyWest}(x, \ldots)) \) is used.
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).

prewhite
logical or integer. Should the estimating functions be prewhitened? If TRUE or greater than 0 a VAR model of order as.integer(prewhte) is fitted via ar with method "ols" and demean = FALSE. The default is to use VAR(1) prewhitening.

kernel
a character specifying the kernel used. All kernels used are described in Andrews (1991). bwNeweyWest can only compute bandwidths for "Bartlett", "Parzen" and "Quadratic Spectral".

adjust
logical. Should a finite sample adjustment be made? This amounts to multiplication with \( n/(n-k) \) where \( n \) is the number of observations and \( k \) the number of estimated parameters.

diagnostics
logical. Should additional model diagnostics be returned? See vcovHAC for details.

sandwich
logical. Should the sandwich estimator be computed? If set to FALSE only the middle matrix is returned.

ar.method
character. The method argument passed to ar for prewhitening (only, not for bandwidth selection).

data
an optional data frame containing the variables in the order.by model. By default the variables are taken from the environment which the function is called from.

verbose
logical. Should the lag truncation parameter used be printed?

weights
numeric. A vector of weights used for weighting the estimated coefficients of the approximation model (as specified by approx). By default all weights are 1 except that for the intercept term (if there is more than one variable).

Details
NeweyWest is a convenience interface to vcovHAC using Bartlett kernel weights as described in Newey & West (1987, 1994). The automatic bandwidth selection procedure described in Newey & West (1994) is used as the default and can also be supplied to kernHAC for the Parzen and quadratic spectral kernel. It is implemented in bwNeweyWest which does not truncate its results - if the results for the Parzen and Bartlett kernels should be truncated, this has to be applied afterwards. For Bartlett weights this is implemented in NeweyWest.

To obtain the estimator described in Newey & West (1987), prewhitening has to be suppressed.

Value
NeweyWest returns the same type of object as vcovHAC which is typically just the covariance matrix.
bwNeweyWest returns the selected bandwidth parameter.
References


See Also

vcovHAC, weightsAndrews, kernHAC

Examples

```r
## fit investment equation
data(Investment)
fm <- lm(RealInv ~ RealGNP + RealInt, data = Investment)

## Newey & West (1994) compute this type of estimator
NeweyWest(fm)

## The Newey & West (1987) estimator requires specification
## of the lag and suppression of prewhitening
NeweyWest(fm, lag = 4, prewhite = FALSE)

## bwNeweyWest() can also be passed to kernHAC(), e.g.
## for the quadratic spectral kernel
kernHAC(fm, bw = bwNeweyWest)
```

PetersenCL

*Petersen’s Simulated Data for Assessing Clustered Standard Errors*

Description

Artificial balanced panel data set from Petersen (2009) for illustrating and benchmarking clustered standard errors.

Usage

```r
data("PetersenCL")
```
PublicSchools

Format
A data frame containing 5000 observations on 4 variables.

firm  integer. Firm identifier (500 firms).
year  integer. Time variable (10 years per firm).
x     numeric. Independent regressor variable.
y     numeric. Dependent response variable.

Details
This simulated data set was created to illustrate and benchmark clustered standard errors. The residual and the regressor variable both contain a firm effect, but no year effect. Thus, standard errors clustered by firm are different from the OLS standard errors and similarly double-clustered standard errors (by firm and year) are different from the standard errors clustered by year.

Source
http://www.kellogg.northwestern.edu/faculty/petersen/htm/papers/se/test_data.htm

References

PublicSchools

Description
Per capita expenditure on public schools and per capita income by state in 1979.

Usage
data(PublicSchools)

Format
A data frame containing 51 observations of 2 variables.

Expenditure per capita expenditure on public schools,
Income  per capita income.

Source
Table 14.1 in Greene (1993)
PublicSchools

References


Examples

```r
## Willam H. Greene, Econometric Analysis, 2nd Ed.
## Chapter 14
## load data set, p. 385, Table 14.1
data(PublicSchools)

## omit NA in Wisconsin and scale income
ps <- na.omit(PublicSchools)
ps$Income <- ps$Income * 0.0001

## fit quadratic regression, p. 385, Table 14.2
fmq <- lm(Expenditure ~ Income + I(Income^2), data = ps)
summary(fmq)

## compare standard and HC0 standard errors
## p. 391, Table 14.3
library(sandwich)
coef(fmq)
sqrt(diag(vcovHC(fmq, type = "const")))
sqrt(diag(vcovHC(fmq, type = "HC0")))

if(require(lmtest)) {
## compare t ratio
coeftest(fmq, vcov = vcovHC(fmq, type = "HC0"))

## White test, p. 393, Example 14.5
wt <- lm(residuals(fmq)^2 ~ poly(Income, 4), data = ps)
wt.stat <- summary(wt)$r.squared * nrow(ps)
c(wt.stat, pchisq(wt.stat, df = 3, lower = FALSE))

## Bresch-Pagan test, p. 395, Example 14.7
bptest(fmq, studentize = FALSE)

## Francisco Cribari-Neto, Asymptotic Inference, CSDA 45
## quasi z-tests, p. 229, Table 8
## with Alaska
coeftest(fmq, df = Inf)[3,4]
coeftest(fmq, df = Inf, vcov = vcovHC(fmq, type = "HC0"))[3,4]
coeftest(fmq, df = Inf, vcov = vcovHC(fmq, type = "HC3"))[3,4]
coeftest(fmq, df = Inf, vcov = vcovHC(fmq, type = "HC4"))[3,4]
```
sandwich

## Description

Constructing sandwich covariance matrix estimators by multiplying bread and meat matrices.

## Usage

```
sandwich(x, bread. = bread, meat. = meat, ...)
```

## Arguments

- `x`: a fitted model object.
- `bread.`: either a bread matrix or a function for computing this via `bread.1(x)`.  
- `meat.`: either a bread matrix or a function for computing this via `meat.1(x, ...)`.  
- `...`: arguments passed to the `meat` function.

## Details

`sandwich` is a simple convenience function that takes a bread matrix (i.e., estimator of the expectation of the negative derivative of the estimating functions) and a meat matrix (i.e., estimator of the variance of the estimating functions) and multiplies them to a sandwich with meat between two slices of bread. By default `bread` and `meat` are called.

Some theoretical background along with implementation details is given in Zeileis (2006).

## Value

A matrix containing the sandwich covariance matrix estimate. Typically, this should be an $k \times k$ matrix corresponding to $k$ parameters.
References


See Also

*bread, meat, meatHC, meatHAC*

Examples

```r
x <- sin(1:10)
y <- rnorm(10)
fm <- lm(y ~ x)

sandwich(fm)
vcovHC(fm, type = "HC")
```

---

**vcovBS** *(Clustered) Bootstrap Covariance Matrix Estimation*

**Description**

Object-oriented estimation of basic bootstrap covariances, using simple (clustered) case-based re-sampling, plus more refined methods for `lm` and `glm` models.

**Usage**

```r
vcovBS(x, ...)
```

## Default S3 method:

```r
vcovBS(x, cluster = NULL, R = 250, start = FALSE, ..., fix = FALSE, use = "pairwise.complete.obs", applyfun = NULL, cores = NULL)
```

## S3 method for class 'lm'

```r
vcovBS(x, cluster = NULL, R = 250, type = "xy", ..., fix = FALSE, use = "pairwise.complete.obs", applyfun = NULL, cores = NULL)
```

## S3 method for class 'glm'

```r
vcovBS(x, cluster = NULL, R = 250, start = FALSE, ..., fix = FALSE, use = "pairwise.complete.obs", applyfun = NULL, cores = NULL)
```

**Arguments**

- `x` a fitted model object.
- `cluster` a variable indicating the clustering of observations, a list (or `data.frame`) thereof, or a formula specifying which variables from the fitted model should be used (see examples). By default (`cluster = NULL`), `attr(x, "cluster")` is used (if any) or otherwise every observation is assumed to be its own cluster.


vcovBS

integer. Number of bootstrap replications.

start logical. Should coef(x) be passed as start to the update(x, subset = ...) call? In case the model x is computed by some numeric iteration, this may speed up the bootstrapping.

type character or function. The character string specifies the type of bootstrap to use: One of "xy", "residual", "wild" (or equivalently: "wild- rademacher" or "rademacher"), "mammen" (or "wild-mammen"), "norm" (or "wild-norm"), "webb" (or "wild-webb"). Alternatively, type can be a function(n) for drawing wild bootstrap factors.

... arguments passed to methods. For the default method, this is passed to update, and for the lm method to lm.fit.

fix logical. Should the covariance matrix be fixed to be positive semi-definite in case it is not?

use character. Specification passed to cov for handling missing coefficients/parameters.

applyfun an optional lapply-style function with arguments function(X, FUN, ...). It is used for refitting the model to the bootstrap samples. The default is to use the basic lapply function unless the cores argument is specified (see below).

cores numeric. If set to an integer the applyfun is set to mclapply with the desired number of cores, except on Windows where parLapply with makeCluster(cores) is used.

Details

Clustered sandwich estimators are used to adjust inference when errors are correlated within (but not between) clusters. See the documentation for vcovCL for specifics about covariance clustering. This function allows for clustering in arbitrary many cluster dimensions (e.g., firm, time, industry), given all dimensions have enough clusters (for more details, see Cameron et al. 2011). Unlike vcovCL, vcovBS uses a bootstrap rather than an asymptotic solution.

Basic (clustered) bootstrap covariance matrix estimation is provided by the default vcovBS method. It samples clusters (where each observation is its own cluster by default), i.e., using case-based resampling. For obtaining a covariance matrix estimate it is assumed that an update of the model with the resampled subset can be obtained, the coef extracted, and finally the covariance computed with cov.

The update model is evaluated in the environment(terms(x)) (if available). To speed up computations two further arguments can be leveraged.

1. Instead of lapply a parallelized function such as parLapply or mclapply can be specified to iterate over the bootstrap replications. For the latter, specifying cores = ... is a convenience shortcut.

2. When specifying start = TRUE, the coef(x) are passed to update as start = coef(x).

This may not be supported by all model fitting functions and is hence not turned on by default.

The “xy” or “pairs” bootstrap is consistent for heteroskedasticity and clustered errors, and converges to the asymptotic solution used in vcovCL as R, n, and g become large (n and g are the number of observations and the number of clusters, respectively; see Efron 1979, or Mammen 1992, for a discussion of bootstrap asymptotics). For small g—particularly under 30 groups—the bootstrap will
converge to a slightly different value than the asymptotic method, due to the limited number of distinct bootstrap replications possible (see Webb 2014 for a discussion of this phenomenon). The bootstrap will not necessarily converge to an asymptotic estimate that has been corrected for small samples.

The $xy$ approach to bootstrapping is generally only of interest to the practitioner when the asymptotic solution is unavailable (this can happen when using estimators that have no $\text{estfun}$ function, for example). The residual bootstrap, by contrast, is rarely of practical interest, because while it provides consistent inference for for clustered standard errors, it is not robust to heteroskedasticity. More generally, bootstrapping is useful when the bootstrap makes different assumptions than the asymptotic estimator, in particular when the number of clusters is small and large $n$ or $g$ assumptions are unreasonable. Bootstrapping is also often effective for nonlinear models, particularly in smaller samples, where asymptotic approaches often perform relatively poorly. See Cameron and Miller (2015) for further discussion of bootstrap techniques in practical applications, and Berger et al. (2017) show simulations comparing $\text{vcovBS}$ to $\text{vcovCL}$ in several settings.

The $\text{glm}$ method works essentially like the default method but call $\text{glm.fit}$ instead of codeupdate.

The $\text{lm}$ method provides additional bootstrapping types and computes the bootstrapped coefficient estimates somewhat more efficiently using $\text{lm.fit}$ (for case-based resampling) or $\text{qr.coef}$ rather than update. The default type is case-based resampling (type = "xy") as in the default method. Alternative type specifications are:

- "residual". The residual cluster bootstrap resamples the residuals (as above, by cluster) which are subsequently added to the fitted values to obtain the bootstrapped response variable: $y^* = \hat{y} + e^*$. Coefficients can then be estimated using $\text{qr.coef}$(), reusing the QR decomposition from the original fit. As Cameron et al. (2008) point out, the residual cluster bootstrap is not well-defined when the clusters are unbalanced as residuals from one cluster cannot be easily assigned to another cluster with different size. Hence a warning is issued in that case.

- "wild" (or equivalently "wild-rademacher" or "rademacher"). The wild cluster bootstrap does not actually resample the residuals but instead reforms the dependent variable by multiplying the residual by a randomly drawn value and adding the result to the fitted value: $y^* = \hat{y} + e \cdot w$ (see Cameron et al. 2008). By default, the factors are drawn from the Rademacher distribution: $\text{function}\(\text{n}\) \text{sample}(\text{c(-1, 1)}, \text{n}, \text{replace = TRUE})$.

- "mammen" (or "wild-mammen"). This draws the wild bootstrap factors as suggested by Mammen (1993): $\text{sample}(\text{c(-1, 1)} \ast (\text{sqrt}(5) \pm (\text{c(-1, 1)})/2, \text{n}, \text{replace = TRUE, prob = (sqrt(5) + c(1, -1)})/2}$.

- "webb" (or "wild-webb"). This implements the six-point distribution suggested by Webb (2014), which may improve inference when the number of clusters is small: $\text{sample}(\text{c(-sqrt((3:1)/2), sqrt((1:3)}))$.

- "norm" (or "wild-norm"). The standard normal/Gaussian distribution is used for drawing the wild bootstrap factors: $\text{function}\(\text{n}\) \text{rnorm}\(\text{n}\)$.

- User-defined function. This needs of the form as above, i.e., a $\text{function}\(\text{n}\)$ returning a vector of random wild bootstrap factors of corresponding length.

Value

A matrix containing the covariance matrix estimate.
References


See Also

vcovCL

Examples

```r
## Petersen's data
data("PetersenCL", package = "sandwich")
m <- lm(y ~ x, data = PetersenCL)

## comparison of different standard errors
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
cbind(
  "classical" = sqrt(diag(vcov(m))),
  "HC-cluster" = sqrt(diag(vcovCL(m, cluster = ~ firm)));
  "BS-cluster" = sqrt(diag(vcovBS(m, cluster = ~ firm)))
)

## two-way wild cluster bootstrap with Mammen distribution
vcovBS(m, cluster = ~ firm + year, type = "wild-mammen")
```
Description

Estimation of one-way and multi-way clustered covariance matrices using an object-oriented approach.

Usage

vcovCL(x, cluster = NULL, type = NULL, sandwich = TRUE, fix = FALSE, ...)
meatCL(x, cluster = NULL, type = NULL, cadjust = TRUE, multi0 = FALSE, ...)

Arguments

x

a fitted model object.

cluster

a variable indicating the clustering of observations, a list (or data.frame) thereof, or a formula specifying which variables from the fitted model should be used (see examples). By default (cluster = NULL), either attr(x, "cluster") is used (if any) or otherwise every observation is assumed to be its own cluster.

type

a character string specifying the estimation type (HC0–HC3). The default is to use "HC1" for lm objects and "HC0" otherwise.

sandwich

logical. Should the sandwich estimator be computed? If set to FALSE only the meat matrix is returned.

fix

logical. Should the covariance matrix be fixed to be positive semi-definite in case it is not?

cadjust

logical. Should a cluster adjustment be applied?

multi0

logical. Should the HC0 estimate be used for the final adjustment in multi-way clustered covariances?

... arguments passed to meatCL.

Details

Clustered sandwich estimators are used to adjust inference when errors are correlated within (but not between) clusters. vcovCL allows for clustering in arbitrary many cluster dimensions (e.g., firm, time, industry), given all dimensions have enough clusters (for more details, see Cameron et al. 2011). If each observation is its own cluster, the clustered sandwich collapses to the basic sandwich covariance.

The function meatCL is the work horse for estimating the meat of clustered sandwich estimators. vcovCL is a wrapper calling sandwich and bread (Zeileis 2006). vcovCL is applicable beyond lm or glm class objects.

bread and meat matrices are multiplied to construct clustered sandwich estimators. The meat of a clustered sandwich estimator is the cross product of the clusterwise summed estimating functions. Instead of summing over all individuals, first sum over cluster.
A two-way clustered sandwich estimator \( M \) (e.g., for cluster dimensions "firm" and "industry" or "id" and "time") is a linear combination of one-way clustered sandwich estimators for both dimensions \( M_{id}, M_{time} \) minus the clustered sandwich estimator, with clusters formed out of the intersection of both dimensions \( M_{id \cap time} \):

\[
M = M_{id} + M_{time} - M_{id \cap time}
\]

Additionally, each of the three terms can be weighted by the corresponding cluster bias adjustment factor (see below and Equation 20 in Berger et al. 2017). Instead of subtracting \( M_{id \cap time} \) as the last subtracted matrix, Ma (2014) suggests to subtract the basic HC0 covariance matrix when only a single observation is in each intersection of \( id \) and \( time \). Set `multi0 = TRUE` to subtract the basic HC0 covariance matrix as the last subtracted matrix in multi-way clustering. For details, see also Petersen (2009) and Thompson (2011).

With the `type` argument, HC0 to HC3 types of bias adjustment can be employed, following the terminology used by MacKinnon and White (1985) for heteroskedasticity corrections. HC0 applies no small sample bias adjustment. HC1 applies a degrees of freedom-based correction, \((n-1)/(n-k)\) where \( n \) is the number of observations and \( k \) is the number of explanatory or predictor variables in the model. HC1 is the most commonly used approach, and is the default, though it is less effective than HC2 and HC3 when the number of clusters is relatively small (Cameron et al. 2008). HC2 and HC3 types of bias adjustment are geared towards the linear model, but they are also applicable for GLMs (see Bell and McCaffrey 2002, and Kauermann and Carroll 2001, for details). A precondition for HC2 and HC3 types of bias adjustment is the availability of a hat matrix (or a weighted version thereof for GLMs) and hence these two types are currently only implemented for `lm` and `glm` objects.

The `cadjjust` argument allows to switch the cluster bias adjustment factor \( G/(G-1) \) on and off (where \( G \) is the number of clusters in a cluster dimension \( g \)). See Cameron et al. (2011) for more details about small-sample modifications.

The cluster specification can be made in a number of ways: The cluster can be a single variable or a list/data.frame of multiple clustering variables. If `expand.model.frame` works for the model object \( x \), the cluster can also be a formula. By default \( \text{cluster} = \text{NULL} \), `attr(x, "cluster")` is checked and used if available. If not, every observation is assumed to be its own cluster. If the number of observations in the model \( x \) is smaller than in the original data due to NA processing, then the same NA processing can be applied to `cluster` if necessary (and `x$na.action being available).

Cameron et al. (2011) observe that sometimes the covariance matrix is not positive-semidefinite and recommend to employ the eigendecomposition of the estimated covariance matrix, setting any negative eigenvalue(s) to zero. This fix is applied, if necessary, when `fix = TRUE` is specified.

Value

A matrix containing the covariance matrix estimate.

References


See Also
vcovHC

Examples

```r
## Petersen's data
data("PetersenCL", package = "sandwich")
m <- lm(y ~ x, data = PetersenCL)

## clustered covariances
## one-way
vcovCL(m, cluster = ~ firm)
vcovCL(m, cluster = PetersenCL$firm) ## same

## one-way with HC2
vcovCL(m, cluster = ~ firm, type = "HC2")

## two-way
vcovCL(m, cluster = ~ firm + year)
vcovCL(m, cluster = PetersenCL[, c("firm", "year")]) ## same

## comparison with cross-section sandwiches
## HC0
all.equal(sandwich(m), vcovCL(m, type = "HC0", cadj = FALSE))
## HC2
```


all.equal(vcovHC(m, type = "HC2"), vcovCL(m, type = "HC2"))
## HC3
all.equal(vcovHC(m, type = "HC3"), vcovCL(m, type = "HC3"))
## Innovation data
data("InstInnovation", package = "sandwich")

## replication of one-way clustered standard errors for model 3, Table I
## and model 1, Table II in Berger et al. (2016)

## count regression formula
f1 <- cites ~ institutions + log(capital/employment) + log(sales) + industry + year

## model 3, Table I: Poisson model
## one-way clustered standard errors
tab_1_3_pois <- glm(f1, data = InstInnovation, family = poisson)
vcov_pois <- vcovCL(tab_1_3_pois, InstInnovation$company)
sqrt(diag(vcov_pois))[2:4]

## coefficient tables
if(require("lmtest")) {
  coeftest(tab_1_3_pois, vcov = vcov_pois)[2:4]
}

## End(Not run)

## model 1, Table II: negative binomial hurdle model
## (requires "pscl" or alternatively "countreg" from R-Forge)
library("pscl")
library("lmtest")
tab_1_3_hurdle <- hurdle(f1, data = InstInnovation, dist = "negbin")
# dist = "negbin", zero.dist = "negbin", separate = FALSE)
vcov_hurdle <- vcovCL(tab_1_3_hurdle, InstInnovation$company)
sqrt(diag(vcov_hurdle))[c(2:4, 149:151)]
coeftest(tab_1_3_hurdle, vcov = vcov_hurdle)[c(2:4, 149:151)]

## End(Not run)

---

**vcovHAC**  
*Heteroskedasticity and Autocorrelation Consistent (HAC) Covariance Matrix Estimation*

**Description**  
Heteroskedasticity and autocorrelation consistent (HAC) estimation of the covariance matrix of the coefficient estimates in a (generalized) linear regression model.

**Usage**  
vcovHAC(x, ...)
Arguments

- **x**: a fitted model object.

- **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).

- **prewhite**: logical or integer. Should the estimating functions be prewhitened? If TRUE or greater than 0 a VAR model of order \( \text{as.integer}(\text{prewhite}) \) is fitted via \text{ar} with method "ols" and \text{demean} = FALSE.

- **weights**: Either a vector of weights for the autocovariances or a function to compute these weights based on \( x \), \text{order.by}, \text{prewhite}, \text{ar.method} and \text{data}. If \text{weights} is a function it has to take these arguments. See also details.

- **adjust**: logical. Should a finite sample adjustment be made? This amounts to multiplication with \( n/(n-k) \) where \( n \) is the number of observations and \( k \) the number of estimated parameters.

- **diagnostics**: logical. Should additional model diagnostics be returned? See below for details.

- **sandwich**: logical. Should the sandwich estimator be computed? If set to FALSE only the meat matrix is returned.

- **ar.method**: character. The method argument passed to \text{ar} for prewhitening.

- **data**: an optional data frame containing the variables in the \text{order.by} model. By default the variables are taken from the environment which \text{vcovHAC} is called from.

- **...**: arguments passed to \text{sandwich} (in \text{vcovHAC}) and \text{estfun} (in \text{meathAC}), respectively.

Details

The function \text{meathAC} is the real work horse for estimating the meat of HAC sandwich estimators – the default \text{vcovHAC} method is a wrapper calling \text{sandwich} and \text{bread}. See Zeileis (2006) for more implementation details. The theoretical background, exemplified for the linear regression model, is described in Zeileis (2004).

Both functions construct weighted information sandwich variance estimators for parametric models fitted to time series data. These are basically constructed from weighted sums of autocovariances of the estimating functions (as extracted by \text{estfun}). The crucial step is the specification of weights: the user can either supply \text{vcovHAC} with some vector of weights or with a function that computes these weights adaptively (based on the arguments \( x \), \text{order.by}, \text{prewhite} and \text{data}). Two functions for adaptively choosing weights are implemented in \text{weightsAndrews} implementing the results of...
Andrews (1991) and in weightsLumley implementing the results of Lumley (1999). The functions kernHAC and weave respectively are to more convenient interfaces for vcovHAC with these functions.

Prewhtening based on VAR approximations is described as suggested in Andrews & Monahan (1992).

The covariance matrix estimators have been improved by the addition of a bias correction and an approximate denominator degrees of freedom for test and confidence interval construction. See Lumley & Heagerty (1999) for details.

Value

A matrix containing the covariance matrix estimate. If diagnostics was set to TRUE this has an attribute "diagnostics" which is a list with

bias.correction

multiplicative bias correction

df

Approximate denominator degrees of freedom

References


See Also

weightsLumley, weightsAndrews, weave, kernHAC

Examples

```r
x <- sin(1:100)
y <- 1 + x + rnorm(100)
fm <- lm(y ~ x)
vcovHAC(fm)
vcov(fm)
```
Description

Heteroskedasticity-consistent estimation of the covariance matrix of the coefficient estimates in regression models.

Usage

vcovHC(x, ...)  
## Default S3 method:
vcovHC(x,
  type = c("HC3", "const", "HC", "HC0", "HC1", "HC2", "HC4", "HC4m", "HC5"),
  omega = NULL, sandwich = TRUE, ...)

meathc(x, type = , omega = NULL, ...)

Arguments

x  
a fitted model object.

type  
a character string specifying the estimation type. For details see below.

omega  
a vector or a function depending on the arguments residuals (the working residuals of the model), diaghat (the diagonal of the corresponding hat matrix) and df (the residual degrees of freedom). For details see below.

sandwich  
logical. Should the sandwich estimator be computed? If set to FALSE only the meat matrix is returned.

...  
arguments passed to sandwich (in vcovHC) and estfun (in meathc), respectively.

Details

The function meathc is the real work horse for estimating the meat of HC sandwich estimators – the default vcovHC method is a wrapper calling sandwich and bread. See Zeileis (2006) for more implementation details. The theoretical background, exemplified for the linear regression model, is described below and in Zeileis (2004). Analogous formulas are employed for other types of models.

When type = "const" constant variances are assumed and and vcovHC gives the usual estimate of the covariance matrix of the coefficient estimates:

\[ \hat{\sigma}^2(X^T X)^{-1} \]

All other methods do not assume constant variances and are suitable in case of heteroskedasticity. "HC" (or equivalently "HC0") gives White’s estimator, the other estimators are refinements of this. They are all of form
and differ in the choice of Omega. This is in all cases a diagonal matrix whose elements can be either supplied as a vector \( \text{omega} \) or as a function \( \text{omega} \) of the residuals, the diagonal elements of the hat matrix and the residual degrees of freedom. For White’s estimator

\[
\text{omega} <- \text{function(residuals, diaghat, df)} \text{ residuals}^2
\]

Instead of specifying the diagonal \( \text{omega} \) or a function for estimating it, the \text{type} argument can be used to specify the HC0 to HC5 estimators. If \( \text{omega} \) is used, \text{type} is ignored.

Long & Ervin (2000) conduct a simulation study of HC estimators (HC0 to HC3) in the linear regression model, recommending to use HC3 which is thus the default in vcovHC. Cribari-Neto (2004), Cribari-Neto, Souza, & Vasconcellos (2007), and Cribari-Neto & Da Silva (2011), respectively, suggest the HC4, HC5, and modified HC4m type estimators. All of them are tailored to take into account the effect of leverage points in the design matrix. For more details see the references.

**Value**

A matrix containing the covariance matrix estimate.

**References**


**See Also**

*lm, hccm, bptest, ncv.test*
Examples

```r
## generate linear regression relationship
## with homoskedastic variances
x <- sin(1:100)
y <- 1 + x + rnorm(100)
## model fit and HC3 covariance
fm <- lm(y ~ x)
vcovHC(fm)
## usual covariance matrix
vcov(fm)

sigma2 <- sum(residuals(lm(y ~ x))^2)/98
sigma2 * solve(crossprod(cbind(1, x)))
```

vcovOPG

### Outer-Product-of-Gradients Covariance Matrix Estimation

Description

Outer product of gradients estimation for the covariance matrix of the coefficient estimates in regression models.

Usage

vcovOPG(x, adjust = FALSE, ...)

Arguments

- `x`: a fitted model object.
- `adjust`: logical. Should a finite sample adjustment be made? This amounts to multiplication with \( n/(n-k) \) where \( n \) is the number of observations and \( k \) the number of estimated parameters.
- `...`: arguments passed to the `estfun` function.

Details

In correctly specified models, the “meat” matrix (cross product of estimating functions, see `meat`) and the inverse of the “bread” matrix (inverse of the derivative of the estimating functions, see `bread`) are equal and correspond to the Fisher information matrix. Typically, an empirical version of the bread is used for estimation of the information but alternatively it is also possible to use the meat. This method is also known as the outer product of gradients (OPG) estimator (Cameron & Trivedi 2005).

Using the `sandwich` infrastructure, the OPG estimator could easily be computed via `solve(meat(obj))` (modulo scaling). To employ numerically more stable implementation of the inversion, this simple convenience function can be used: `vcovOPG(obj)`.

Note that this only works if the `estfun()` method computes the maximum likelihood scores (and not a scaled version such as least squares scores for “lm” objects).
vcovPC

Value

A matrix containing the covariance matrix estimate.

References


See Also

meat, bread, sandwich

Examples

```r
## generate poisson regression relationship
x <- sin(1:100)
y <- rpois(100, exp(1 + x))
## compute usual covariance matrix of coefficient estimates
fm <- glm(y ~ x, family = poisson)
vcov(fm)
vcovOPG(fm)
```

---

vcovPC

*Panel-Corrected Covariance Matrix Estimation*

Description


Usage

```
vcovPC(x, cluster = NULL, order.by = NULL,
        pairwise = FALSE, sandwich = TRUE, fix = FALSE, ...)
```

```
meatPC(x, cluster = NULL, order.by = NULL,
       pairwise = FALSE, kronecker = TRUE, ...)
```

Arguments

- `x` a fitted model object.
- `cluster` a variable indicating the clustering of observations, a list (or `data.frame`) thereof, or a formula specifying which variables from the fitted model should be used (see examples). By default (`cluster = NULL`), either `attr(x, "cluster")` is used (if any) or otherwise every observation is assumed to be its own cluster.
vcovPC

order.by a variable, list/data.frame, or formula indicating the aggregation within time periods. By default attr(x, "order.by") is used (if any) or otherwise observations within clusters are assumed to be ordered.
	pairwise logical. For unbalanced panels. Indicating whether the meat should be estimated pair- or casewise.

sandwich logical. Should the sandwich estimator be computed? If set to FALSE only the meat matrix is returned.

fix logical. Should the covariance matrix be fixed to be positive semi-definite in case it is not?

kronecker logical. Calculate the meat via the Kronecker-product, shortening the computation time for small matrices. For large matrices, set kronecker = FALSE.

... arguments passed to the meatPC or estfun function, respectively.

Details


Following Bailey and Katz (2011), there are two alternatives to estimate the meat for unbalanced panels. For pairwise = FALSE, a balanced subset of the panel is used, whereas for pairwise = TRUE, a pairwise balanced sample is employed.

The cluster/order.by specification can be made in a number of ways: Either both can be a single variable or cluster can be a list/data.frame of two variables. If expand.model.frame works for the model object x, the cluster (and potentially additionally order.by) can also be a formula. By default (cluster = NULL, order.by = NULL), attr(x, "cluster") and attr(x, "order.by") are checked and used if available. If not, every observation is assumed to be its own cluster, and observations within clusters are assumed to be ordered accordingly. If the number of observations in the model x is smaller than in the original data due to NA processing, then the same NA processing can be applied to cluster if necessary (and x$na.action being available).

Value

A matrix containing the covariance matrix estimate.

References


vcovPL


See Also

vcovCL, vcovPL

Examples

```r
## Petersen's data
data("PetersenCL", package = "sandwich")
m <- lm(y ~ x, data = PetersenCL)

## balanced panel
sqrt(diag(vcovPC(m, cluster = ~ firm + year)))

## unbalanced panel
PU <- subset(PetersenCL, !(firm == 1 & year == 10))
pu_lm <- lm(y ~ x, data = PU)
sqrt(diag(vcovPC(pu_lm, cluster = ~ firm + year, pairwise = TRUE)))
sqrt(diag(vcovPC(pu_lm, cluster = ~ firm + year, pairwise = FALSE)))

## the following specifications of cluster/order.by are equivalent
vcovPC(m, cluster = ~ firm + year)
vcovPC(m, cluster = PetersenCL[, c("firm", "year")])
vcovPC(m, cluster = ~ firm, order.by = ~ year)
vcovPC(m, cluster = PetersenCL$firm, order.by = PetersenCL$year)

## these are also the same when observations within each
## cluster are already ordered
vcovPC(m, cluster = ~ firm)
vcovPC(m, cluster = PetersenCL$firm)
```

---

vcovPL  
*Clustered Covariance Matrix Estimation for Panel Data*

Description


Usage

```r
vcovPL(x, cluster = NULL, order.by = NULL,  
   kernel = "Bartlett", sandwich = TRUE, fix = FALSE, ...)
```
meatPL(x, cluster = NULL, order.by = NULL, kernel = "Bartlett", lag = "NW1987", bw = NULL, adjust = TRUE, ...)

Arguments

x a fitted model object.

cluster a variable indicating the clustering of observations, a list (or data.frame) thereof, or a formula specifying which variables from the fitted model should be used (see examples). By default (cluster = NULL), either attr(x, "cluster") is used (if any) or otherwise every observation is assumed to be its own cluster.

order.by a variable, list/data.frame, or formula indicating the aggregation within time periods. By default attr(x, "order.by") is used (if any) or otherwise observations within clusters are assumed to be ordered.

kernel a character specifying the kernel used. All kernels described in Andrews (1991) are supported, see kweights.

lag character or numeric, indicating the lag length used. Three rules of thumb ("max" or equivalently "P2009", "NW1987", or "NW1994") can be specified, or a numeric number of lags can be specified directly. By default, "NW1987" is used.

bw numeric. The bandwidth of the kernel which by default corresponds to lag + 1. Only one of lag and bw should be used.

sandwich logical. Should the sandwich estimator be computed? If set to FALSE only the meat matrix is returned.

fix logical. Should the covariance matrix be fixed to be positive semi-definite in case it is not?

adjust logical. Should a finite sample adjustment be made? This amounts to multiplication with n/(n - k) where n is the number of observations and k is the number of estimated parameters.

... arguments passed to the metaPL or estfun function, respectively.

Details


The function meatPL is the work horse for estimating the meat of Newey-West (1978) and Driscoll and Kraay (1998) covariance matrix estimators. vcovPL is a wrapper calling sandwich and bread (Zeileis 2006).

Default lag length is the "NW1987". For lag = "NW1987", the lag length is chosen from the heuristic floor(T^{1/4}). More details on lag length selection in Hoechle (2007). For lag = "NW1994", the lag length is taken from the first step of Newey and West’s (1994) plug-in procedure.
The cluster/order.by specification can be made in a number of ways: Either both can be a single variable or cluster can be a list/data.frame of two variables. If expand.model.frame works for the model object x, the cluster (and potentially additionally order.by) can also be a formula. By default (cluster = NULL, order.by = NULL), attr(x, "cluster") and attr(x, "order.by") are checked and used if available. If not, every observation is assumed to be its own cluster, and observations within clusters are assumed to be ordered accordingly. If the number of observations in the model x is smaller than in the original data due to NA processing, then the same NA processing can be applied to cluster if necessary (and x$na.action being available).

Value
A matrix containing the covariance matrix estimate.

References

See Also
vcovCL, vcovPC

Examples
```r
# Petersen's data
data("PetersenCL", package = "sandwich")
m <- lm(y ~ x, data = PetersenCL)
# Driscoll and Kraay standard errors
```
weightsAndrews

Kernel-based HAC Covariance Matrix Estimation

Description


Usage

kernHAC(x, order.by = NULL, prewhite = 1, bw = bwAndrews,
kernel = c("Quadratic Spectral", "Truncated", "Bartlett", "Parzen", "Tukey-Hanning"),
approx = c("AR(1)", "ARMA(1,1)"), adjust = TRUE, diagnostics = FALSE,
sandwich = TRUE, ar.method = "ols", tol = 1e-7, data = list(), verbose = FALSE, ...)

weightsAndrews(x, order.by = NULL, bw = bwAndrews,
kernel = c("Quadratic Spectral", "Truncated", "Bartlett", "Parzen", "Tukey-Hanning"),
prewhite = 1, ar.method = "ols", tol = 1e-7, data = list(), verbose = FALSE, ...)

bwAndrews(x, order.by = NULL, kernel = c("Quadratic Spectral", "Truncated",
"Bartlett", "Parzen", "Tukey-Hanning"), approx = c("AR(1)", "ARMA(1,1)"),
weights = NULL, prewhite = 1, ar.method = "ols", data = list(), ...)
weightsAndrews

Arguments

- **x**: a fitted model object.
- **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).
- **prewhite**: logical or integer. Should the estimating functions be prewhitened? If TRUE or greater than 0 a VAR model of order `as.integer(prewhite)` is fitted via `ar` with method "ols" and `demean = FALSE`. The default is to use VAR(1) prewhitening.
- **bw**: numeric or a function. The bandwidth of the kernel (corresponds to the truncation lag). If set to to a function (the default is `bwAndrews`) it is adaptively chosen.
- **kernel**: a character specifying the kernel used. All kernels used are described in Andrews (1991).
- **approx**: a character specifying the approximation method if the bandwidth `bw` has to be chosen by `bwAndrews`.
- **adjust**: logical. Should a finite sample adjustment be made? This amounts to multiplication with `n/(n - k)` where `n` is the number of observations and `k` the number of estimated parameters.
- **diagnostics**: logical. Should additional model diagnostics be returned? See `vcovHAC` for details.
- **sandwich**: logical. Should the sandwich estimator be computed? If set to FALSE only the middle matrix is returned.
- **ar.method**: character. The method argument passed to `ar` for prewhitening (only, not for bandwidth selection).
- **tol**: numeric. Weights that exceed `tol` are used for computing the covariance matrix, all other weights are treated as 0.
- **data**: an optional data frame containing the variables in the `order.by` model. By default the variables are taken from the environment which the function is called from.
- **verbose**: logical. Should the bandwidth parameter used be printed?
- **...**: further arguments passed to `bwAndrews`.
- **weights**: numeric. A vector of weights used for weighting the estimated coefficients of the approximation model (as specified by `approx`). By default all weights are 1 except that for the intercept term (if there is more than one variable).

Details

`kernHAC` is a convenience interface to `vcovHAC` using `weightsAndrews`: first a weights function is defined and then `vcovHAC` is called.

The kernel weights underlying `weightsAndrews` are directly accessible via the function `kweights` and require the specification of the bandwidth parameter `bw`. If this is not specified it can be chosen adaptively by the function `bwAndrews` (except for the "Truncated" kernel). The automatic
bandwidth selection is based on an approximation of the estimating functions by either AR(1) or ARMA(1,1) processes. To aggregate the estimated parameters from these approximations a weighted sum is used. The weights in this aggregation are by default all equal to 1 except that corresponding to the intercept term which is set to 0 (unless there is no other variable in the model) making the covariance matrix scale invariant.

Further details can be found in Andrews (1991).

The estimator of Newey & West (1987) is a special case of the class of estimators introduced by Andrews (1991). It can be obtained using the "Bartlett" kernel and setting bw to lag + 1. A convenience interface is provided in NeweyWest.

Value

kernHAC returns the same type of object as vcovHAC which is typically just the covariance matrix.
weightsAndrews returns a vector of weights.
bwAndrews returns the selected bandwidth parameter.

References


See Also

vcovHAC, NeweyWest, weightsLumley, weave

Examples

curve(kweights(x, kernel = "Quadratic", normalize = TRUE),
    from = 0, to = 3.2, xlab = "x", ylab = "k(x)"
) curve(kweights(x, kernel = "Bartlett", normalize = TRUE),
    from = 0, to = 3.2, col = 2, add = TRUE)
curve(kweights(x, kernel = "Parzen", normalize = TRUE),
    from = 0, to = 3.2, col = 3, add = TRUE)
curve(kweights(x, kernel = "Tukey", normalize = TRUE),
    from = 0, to = 3.2, col = 4, add = TRUE)
curve(kweights(x, kernel = "Truncated", normalize = TRUE),
    from = 0, to = 3.2, col = 5, add = TRUE)

## fit investment equation
data(Investment)
fm <- lm(RealInv ~ RealGnp + RealInt, data = Investment)

## compute quadratic spectral kernel HAC estimator
kernHAC(fm)
kernHAC(fm, verbose = TRUE)

## use Parzen kernel instead, VAR(2) prewhitening, no finite sample
## 

The document contains code snippets for implementing kernel-based heteroskedasticity and autocorrelation consistent (HAC) covariance matrix estimators as introduced by Andrews (1991). The code includes functions for adjusting and comparing estimates under the assumption of spheric errors. The functions are part of the `weightsLumley` package, which is used for weighted empirical adaptive variance estimation.

### Description


### Usage

```r
# adjustment and Newey & West (1994) bandwidth selection
kernHAC(fm, kernel = "Parzen", prewhite = 2, adjust = FALSE,
bw = bwNeweyWest, verbose = TRUE)

# compare with estimate under assumption of spheric errors
vcov(fm)
```

### Arguments

- **x**: a fitted model object.
- **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).
- **prewhite**: logical or integer. Should the estimating functions be prewhitened? If TRUE or greater than 0 a VAR model of order as.integer(prewhite) is fitted via `ar` with method "ols" and `demean = FALSE`.
- **C**: numeric. The cutoff constant `C` is by default 4 for method "truncate" and 1 for method "smooth".
- **method**: a character specifying the method used, see details.
- **acf**: a function that computes the autocorrelation function of a vector, by default `isoacf` is used.
- **adjust**: logical. Should a finite sample adjustment be made? This amounts to multiplication with `n/(n - k)` where `n` is the number of observations and `k` the number of estimated parameters.
- **diagnostics**: logical. Should additional model diagnostics be returned? See `vcovHAC` for details.
- **sandwich**: logical. Should the sandwich estimator be computed? If set to FALSE only the middle matrix is returned.
weightsLumley

**tol** numeric. Weights that exceed `tol` are used for computing the covariance matrix, all other weights are treated as 0.

**data** an optional data frame containing the variables in the order by model. By default the variables are taken from the environment which the function is called from.

... currently not used.

**Details**

`weave` is a convenience interface to `vcovHAC` using `weightsLumley`: first a weights function is defined and then `vcovHAC` is called.

Both weighting methods are based on some estimate of the autocorrelation function $\rho$ (as computed by `acf`) of the residuals of the model $x$. The weights for the "truncate" method are

$$I\{n\rho^2 > C\}$$

and the weights for the "smooth" method are

$$\min\{1, Cn\rho^2\}$$

where $n$ is the number of observations in the model an $C$ is the truncation constant $C$. Further details can be found in Lumley & Heagerty (1999).

**Value**

`weave` returns the same type of object as `vcovHAC` which is typically just the covariance matrix. `weightsLumley` returns a vector of weights.

**References**


**See Also**

`vcovHAC`, `weightsAndrews`, `kernHAC`

**Examples**

```r
x <- sin(1:100)
y <- 1 + x + rnorm(100)
fm <- lm(y ~ x)
weave(fm)
vcov(fm)
```
Index

*Topic bootstrap
  vcovBS, 18
*Topic datasets
  InstInnovation, 4
  Investment, 6
  PetersenCL, 14
  PublicSchools, 15
*Topic regression
  bread, 2
  estfun, 3
  isoacf, 8
  kweights, 9
  lrvar, 10
  meat, 11
  NeweyWest, 12
  sandwich, 17
  vcovBS, 18
  vcovCL, 22
  vcovHC, 25
  vcovOPG, 30
  vcovPC, 31
  vcovPL, 33
  weightsAndrews, 36
  weightsLumley, 39
*Topic ts
  isoacf, 8
  kweights, 9
  lrvar, 10
  NeweyWest, 12
  vcovHAC, 25
  vcovHC, 28
  vcovOPG, 30
  weightsAndrews, 36
  weightsLumley, 39
  .vcovBSenv (vcovBS), 18

ar, 13, 26, 37
bp.test, 29

bread, 2, 12, 17, 18, 22, 26, 28, 30–32, 34
bwAndrews (weightsAndrews), 36
bwNeweyWest (NeweyWest), 12

cdf, 2, 3, 19
cov, 19

estfun, 3, 11, 12, 26, 28, 30
expand.model.frame, 23, 32, 35

glm, 2, 3, 18, 20, 23
glm.fit, 20

hccm, 29
InstInnovation, 4
Investment, 6
isoacf, 8, 39

kernHAC, 10, 11, 14, 27, 40
kernHAC (weightsAndrews), 36
kweights, 9, 34, 37

lapply, 19
lm, 2, 3, 10, 18, 20, 23, 29
lm.fit, 20
lrvar, 10

mclapply, 19
meat, 11, 17, 18, 22, 30, 31
meatCL (vcovCL), 22
meatHAC, 18
meatHAC (vcovHAC), 25
meatHC, 18
meatHC (vcovHC), 28
meatPC (vcovPC), 31
meatPL (vcovPL), 33

ncv.test, 29
NeweyWest, 10, 11, 12, 38
nobs, 2
parLapply, 19
pava.blocks (isoacf), 8
PetersenCL, 14
PublicSchools, 15
qr.coef, 20
sandwich, 12, 17, 22, 26, 28, 31, 32, 34
terms, 2, 3
update, 19
vcov, 2
vcovBS, 18
vcovCL, 19, 21, 22, 33, 35
vcovHAC, 11, 13, 14, 25, 37–40
vcovHC, 24, 28
vcovOPG, 30
vcovPC, 31, 35
vcovPL, 33, 33
weave, 9, 27, 38
weave (weightsLumley), 39
weightsAndrews, 10, 14, 26, 27, 36, 40
weightsLumley, 9, 27, 38, 39