Package ‘lfe’

March 11, 2015

Version 2.1-1640
Date 2015-03-11
Title Linear Group Fixed Effects
Author Simen Gaure, Ragnar Frisch Centre for Economic Research
Maintainer Simen Gaure <Simen.Gaure@frisch.uio.no>
Copyright 2011-2015, Simen Gaure
Depends R (>= 2.15.2), Matrix (>= 1.1-2)
Imports Formula, xtable, compiler
Suggests knitr, igraph
VignetteBuilder knitr
ByteCompile yes
Description Transforms away factors with many levels prior to doing an OLS.
Useful for estimating linear models with multiple group fixed effects, and for
estimating linear models which uses factors as pure control variables.
Includes support for instrumental variables, conditional F statistics for weak instruments,
robust and multi-way clustered standard errors, as well as limited mobility bias correction.
License Artistic-2.0
Classification/JEL C13, C23, C60
Classification/MSC 62J05, 65F10, 65F50
NeedsCompilation yes
Repository CRAN
Date/Publication 2015-03-11 11:05:24

R topics documented:

  lfe-package ................................................................. 2
  bccorr ................................................................. 4
  btrap ................................................................. 7
  compfactor ......................................................... 8
  confstat ......................................................... 9
Description

The package uses the Method of Alternating Projections to estimate linear models with multiple group fixed effects. A generalization of the within estimator. It supports IV-estimation with multiple endogenous variables via 2SLS, with conditional F statistics for detection of weak instruments. It is thread-parallelized and intended for large problems. A method for correcting limited mobility bias is also included.

Details

This package is intended for linear models with multiple group fixed effects, i.e. with 2 or more factors with a large number of levels. It performs similar functions as `lm`, but it uses a special method for projecting out multiple group fixed effects from the normal equations, hence it is faster. It is a generalization of the within estimator. This may be required if the groups have high cardinality (many levels), resulting in tens or hundreds of thousands of dummy-variables. It is also useful if one only wants to control for the group effects, without actually estimating them. The package may optionally compute standard errors for the group effects by bootstrapping, but this is a very time- and memory-consuming process compared to finding the point estimates. If you only have a single huge factor, the package `plm` is probably better suited.

As of version 1.6, projecting out interactions between continuous covariates and factors is supported. I.e. individual slopes, not only individual intercepts. As of version 2.0, multiple left hand sides are supported.

The estimation is done in two steps. First the other coefficients are estimated with the function `felm` by centering on all the group means, followed by an OLS (similar to `lm`). Then the group effects are extracted (if needed) with the function `getfe`. This method is described in Gaure (2013), but also appears in Guimaraes and Portugal (2010), disguised as the Gauss-Seidel algorithm.

There’s also a function `demeanlist` which just does the centering on an arbitrary matrix, and there’s a function `compfactor` which computes the connected components (which are used for interpreting the group effects when there are only two factors; see the Abowd et al references), they are also returned by `getfe`. 

Index

- demeanlist .................................................. 11
- efactory .................................................. 13
- felm .................................................. 15
- fevcov .................................................. 19
- getfe .................................................. 21
- is.estimable ........................................... 23
- kaczmarz ................................................. 24
- summary.felm ........................................... 26
- varvars .................................................. 27
- waldtest ................................................ 29
For those who study the correlation between the fixed effects, like in Abowd et al., there are functions `bccorr` and `fevcov` for computing limited mobility bias corrected correlations and variances as documented in Gaure (2014b).

Instrumented variable estimations are supported with 2SLS. Conditional F statistics for testing reduced rank weak instruments as in Sanderson and Windmeijer (2014) are available in `condfstat`.

The centering on the means is done with a tolerance which is set by options(lfe.eps=1e-8) (the default). This is a somewhat conservative tolerance, in many cases I’d guess 1e-6 may be sufficient. This may speed up the centering. In the other direction, setting options(lfe.eps=0) will provide maximum accuracy at the cost of computing time and warnings about convergence failure.

The package is threaded, that is, it may use more than one cpu. The number of threads is fetched upon loading the package from the environment variable `lfe_threads`, `omp_thread_limit`, `omp_num_threads` or `number_of_processors` (for Windows), and stored by options(lfe.threads=n). This option may be changed prior to calling `felm`, if so desired. Note that, typically, lfe is limited by memory-bandwidth, not cpu-speed, thus fast memory and large cache is more important than clock-frequency. It’s therefore also not always true that running on all available cores is much better than running on half of them.

Threading is only done for the centering: the extraction of the group effects is not threaded. The default method for extracting the group coefficients is the iterative Kaczmarz-method, its tolerance is also the lfe.eps option.

For some datasets the Kaczmarz-method is converging very slowly, in this case it may be replaced with a conjugate gradient method by setting the option options(lfe.usecg=TRUE).

The package has been tested on datasets with approx 20,000,000 observations with 15 covariates and approx 2,300,000 and 270,000 group levels (the `felm` took about 50 minutes on 8 cpus, the `getfe` takes 5 minutes). Though, beware that not only the size of the dataset matters, but also its structure.

The package will work with any positive number of grouping factors, but if more than two, their interpretation is in general not well understood, i.e. one should make sure that the coefficients are estimable.

In the exec-directory there is a perl-script `lfescript` which is used at the author’s site for creating R-scripts from a simple specification file. The format is documented in `doc/lfeguide.txt`.

`lfe` is similar in function, though not in method, to the Stata modules `a2reg` and `felsdvreg`.

References


Examples

oldopts <- options(lfe.threads=1)
x <- rnorm(1000)
x2 <- rnorm(length(x))
id <- factor(sample(10,length(x),replace=TRUE))
firm <- factor(sample(3,length(x),replace=TRUE,prob=c(2,1.5,1)))
year <- factor(sample(10,length(x),replace=TRUE,prob=c(2,1.5,rep(1,8))))
id.eff <- rnorm(nlevels(id))
firm.eff <- rnorm(nlevels(firm))
year.eff <- rnorm(nlevels(year))
y <- x + 0.25*x2 + id.eff[id] + firm.eff[firm] +
    year.eff[year] + rnorm(length(x))
est <- felm(y ~ x+x2 | id + firm + year)
summary(est)
getfe(est,se=TRUE)
# compare with an ordinary lm
summary(lm(y ~ x+x2+id+firm+year-1))
options(oldopts)

bccorr

Description

With a model like 'y = X beta + D theta + F psi + epsilon', where 'D' and 'F' are matrices with dummy encoded factors, one application of lfe is to study the correlation 'cor(D theta, F psi)'. However, if we use estimates for theta and psi, the resulting correlation is biased. The function bccorr computes a bias corrected correlation as described in Gaure (2014).
Usage

\[ \text{bccorr}(\text{est}, \alpha = \text{getfe}(\text{est}), \text{corrfactors}=1:2, \]
\[ \text{nocovar}=\text{is.null}(\text{est}\$X) \&\& \text{length}(\text{est}\$fe)==2, \]
\[ \text{tol}=0.01, \text{maxsamples}=\text{Inf}, \text{lhs}=\text{NULL}) \]

Arguments

- **est**: an object of class "felm", the result of a call to \text{felm}(\text{keepX}=\text{TRUE}).
- **alpha**: a data frame, the result of a call to \text{getfe}.
- **corrfactors**: integer or character vector of length 2. The factors to correlate. The default is fine if there are only two factors in the model.
- **nocovar**: logical. Assume no other covariates than the two factors are present, or that they are uncorrelated with them.
- **tol**: The absolute tolerance for the bias-corrected correlation.
- **maxsamples**: Maximum number of samples for the trace sample means estimates
- **lhs**: character. Name of left hand side if multiple left hand sides.

Details

The bias expressions from Andrews et al. are of the form \( \text{tr}(A B^{-1} C) \) where A, B, and C are matrices too large to be handled directly. \text{bccorr} estimates the trace by using the formula \( \text{tr}(M) = E(x^t M x) \) where x is a vector with coordinates drawn uniformly from the set \{-1,1\}. More specifically, the expectation is estimated by sample means, i.e. in each sample a vector x is drawn, the equation \( Bv = Cx \) is solved by a conjugate gradient method, and the real number \( x^t Av \) is computed.

There are three bias corrections, for the variances of D theta (vD) and F psi (vF), and their covariance (vDF). The correlation is computed as \( \rho \triangleq vDF / \sqrt{vD \cdot vF} \). The variances are estimated to a relative tolerance specified by the argument to1. The covariance bias is estimated to an absolute tolerance in the correlation \( \rho \) (conditional on the already bias corrected vD and vF) specified by to1. The CG algorithm does not need to be exceedingly precise, it is terminated when the solution reaches a precision which is sufficient for the chosen precision in vD, vF, vDF.

Value

\text{bccorr} returns a named integer vector with the following fields:

- **corr**: the bias corrected correlation.
- **v1**: the bias corrected variance for the first factor specified by corrfactors.
- **v2**: the bias corrected variance for the second factor.
- **cov**: the bias corrected covariance between the two factors.
- **d1**: the bias correction for the first factor.
- **d2**: the bias correction for the second factor.
- **d12**: the bias correction for covariance.

The bias corrections have been subtracted from the bias estimates. E.g. \( v2 = v2' - d2 \) where \( v2' \) is the biased variance.
**Note**

Note that if `est` is the result of a call to `felm` with `keepX=FALSE` (the default), the correlation will be computed as if the covariates X are independent of the two factors. This will be faster (typically by a factor of approx. 4), and possibly wronger.

Note also that the computations performed by this function are non-trivial, they may take quite some time. It would be wise to start out with quite liberal tolerances, e.g. `tol=0.1`, to get an idea of the time requirements.

The algorithm used is not very well suited for small datasets with only a few thousand levels in the factors.

Currently, only i.i.d. residuals are supported for this function.

**References**


**See Also**

`fevcov`

**Examples**

```r
x <- rnorm(500)
x2 <- rnorm(length(x))

## create individual and firm
id <- factor(sample(40,length(x),replace=TRUE))
firm <- factor(sample(30,length(x),replace=TRUE,prob=c(2,rep(1,29))))
foo <- factor(sample(20,length(x),replace=TRUE))

## effects
id.eff <- rnorm(nlevels(id))
firm.eff <- rnorm(nlevels(firm))
foo.eff <- rnorm(nlevels(foo))

## left hand side
y <- x + 0.25*x2 + id.eff[id] + firm.eff[firm] + foo.eff[foo] + rnorm(length(x))

# make a data frame
fr <- data.frame(y,x,x2,id,firm,foo)

## estimate and print result
est <- felm(y ~ x+x2|id+firm+foo, data=fr, keepX=TRUE)
# find bias corrections
bccorr(est)
```
btrap

Bootstrap standard errors for the group fixed effects

Description

Bootstrap standard errors for the group fixed effects which were swept out during an estimation with `felm`.

Usage

```r
btrap(alpha, obj, N=100, ef=NULL,
    eps=getOption('lfe.eps'), threads=getOption('lfe.threads'),
    robust=FALSE, cluster=NULL, lhs=NULL)
```

Arguments

- `alpha` data frame returned from `getfe`
- `obj` object of class "felm", usually, a result of a call to `felm`
- `N` integer. The number of bootstrap iterations
- `ef` function. An estimable function such as in `getfe`. The default is to use the one used on `alpha`
- `eps` double. Tolerance for centering, as in `getfe`
- `threads` integer. The number of threads to use
- `robust` logical. Should heteroskedastic standard errors be estimated?
- `cluster` logical or factor. Estimate clustered standard errors.
- `lhs` character vector. Specify which left hand side if `obj` has multiple `lhs`.

Details

The bootstrapping is done in parallel if `threads > 1`. `btrap` is run automatically from `getfe` if `se=TRUE` is specified. To save some overhead, the individual iterations are grouped together, the memory available for this grouping is fetched with `getOption('lfe.bootmem')`, which is initialized upon loading of `lfe` to `options(lfe.bootmem=500)` (MB).

If `robust=TRUE`, heteroskedastic robust standard errors are estimated. If `robust=FALSE` and `cluster=TRUE`, clustered standard errors with the cluster specified to `felm()` are estimated. If `cluster` is a factor, it is used for the cluster definition. `cluster` may also be a list of factors.

Value

A data-frame of the same size as `alpha` is returned, with standard errors filled in.
Examples

oldopts <- options(lfe.threads=2)
## create covariates
x <- rnorm(3000)
x2 <- rnorm(length(x))

## create individual and firm
id <- factor(sample(700,length(x),replace=TRUE))
firm <- factor(sample(300,length(x),replace=TRUE))

## effects
id.eff <- rlnorm(nlevels(id))
firm.eff <- rexp(nlevels(firm))

## left hand side
y <- x + 0.25*x2 + id.eff[id] + firm.eff[firm] + rnorm(length(x))

## estimate and print result
est <- felm(y ~ x+x2 | id + firm)
summary(est)
## extract the group effects
alpha <- getfe(est)
head(alpha)
## bootstrap standard errors
head(btrap(alpha,est))

## bootstrap some differences
ef <- function(v,addnames) {
    w <- c(v[2]-v[1],v[3]-v[2],v[3]-v[1])
    if(addnames) {
        names(w) <- c('id2-id1','id3-id2','id3-id1')
        attr(w,'extra') <- list(note=c('line1','line2','line3'))
    }
    w
}
# check that it's estimable
is.estimable(ef,est$fe)

head(btrap(alpha,est,ef=ef))
options(oldopts)

compfactor  Find the connected components

Description

'compfactor' computes the connected components of the dummy-part of the model.

Usage

compfactor(f1, WW=FALSE)
**condfstat**

Compute conditional F statistic for weak instruments in an IV-estimation with multiple endogenous variables.

**Description**

When using multiple instruments for multiple endogenous variables, the ordinary individual t-tests for the instruments in the first stage do not always reveal a weak set of instruments. Conditional F statistics can be used for such testing.

**Usage**

```
condfstat(object, type='default', quantiles=0.0, bN=100L)
```
condfstat

Arguments

- **object**: object of class "felm", a result of a call to `felm`.
- **type**: character. Error structure. Passed to `waldtest`. If NULL, both iid and robust Fs are returned.
- **quantiles**: numeric. Quantiles for bootstrap.
- **bn**: integer. Number of bootstrap samples.

Details

IV coefficient estimates are not normally distributed, in particular they do not have the right expectation. They follow a quite complicated distribution which is fairly close to normal if the instruments are good. The conditional F-statistic is a measure of how good the instruments are. If the F is large, the instruments are good, and any bias due to the instruments is small compared to the estimated standard errors, and also small relative to the bias in OLS. See Sanderson and Windmeijer (2014) and Stock and Yogo (2004). If F is small, the bias can be large compared to the standard error.

If any(quantiles > 0.0), a bootstrap with bn samples will be performed to estimate quantiles of the endogenous parameters which includes the variance both from the 1st and 2nd stage. The result is returned in an array attribute quantiles of the value returned by condfstat. The argument quantiles can be a vector to estimate more than one quantile at once. If quantiles=NULL, the bootstrapped estimates themselves are returned. The bootstrap is normally much faster than running `felm` over and over again. This is so because all exogenous variables are projected out of the equations before doing the bootstrap.

Value

A p x k matrix, where k is the number of endogenous variables. Each row are the conditional F statistics on a residual equation as described in Sanderson and Windmeijer (2014), for a certain error structure. The default is to use iid, or cluster if a cluster was specified to `felm`. The third choice is 'robust', for heteroskedastic errors. If type=NULL, iid and robust Fs are returned, and cluster, if that was specified to `felm`.

Note that for these F statistics it is not the p-value that matters, it is the F statistic itself which (coincidentally) pops up in the denominator for the asymptotic bias of the IV estimates, and thus a large F is beneficial.

References

Examples

```r
d1 <- rnorm(4000)
d2 <- rnorm(length(d1))
u <- rnorm(length(d1))
# make d1, d2 correlated with errors u

dx1 <- d1 + d2 + 0.2*u + rnorm(length(d1))
dx2 <- d1 + 0.94*d2 - 0.3*u + rnorm(length(d1))
y <- d1 + d2 + u
est <- felm(y ~ 1 | d1 | d2 ~ d1 + d2)
summary(est)
## Not run:
summary(est$stage1, lhs='d1')
summary(est$stage1, lhs='d2')

## End(Not run)
# everything above looks fine, t-tests for instruments,
# as well as F-tests for excluded instruments in the 1st stages.
# The standard errors are large, though
# The conditional F-test reveals that the instruments are weak
# (it's close to being only one instrument, d1+d2, for both d1 and d2)
condfstat(est, quantiles=c(0.05, 0.95))
```

demeanlist

Centre vectors on multiple groups

Description

Uses the method of alternating projections to centre a (model) matrix on multiple groups, as specified by a list of factors. This function is called by felm, but it has been made available as standalone in case it’s needed.

Usage
demeanlist(mtx, fl, icpt=0, eps=getOption('lfe.eps'),
threads=getOption('lfe.threads'),
progress=getOption('lfe.pint'),
accel=getOption('lfe.accel'),
randfact=TRUE,
means=FALSE)

Arguments

- `mtx`: matrix whose columns form vectors to be group-centred. mtx may also be a list of vectors or matrices.
- `fl`: list of factors defining the grouping structure
- `icpt`: the position of the intercept, this column is removed from the result matrix
- `eps`: a tolerance for the centering
threads an integer specifying the number of threads to use
progress integer. If positive, make progress reports (whenever a vector is centered, but not more often than every progress minutes)
accel integer. Set to 1 if Gearhart-Koshy acceleration should be done.
randfact logical. Should the order of the factors be randomized? This may improve convergence.
means logical. Should the means instead of the demeaned matrix be returned? Setting means=TRUE will return mtx - demeanlist(mtx,...), but without the extra copy.

Details

For each column y in mtx, the equivalent of the following centering is performed, with cy as the result.

cy <- y; oldy <- y-1
while(sqrt(sum((cy-oldy)**2)) >= eps) {
  oldy <- cy
  for(f in fl) cy <- cy - ave(cy,f)
}

Beginning with version 1.6, each factor in fl may contain an attribute 'x' which is a numeric vector of the same length as the factor. The centering is then not done on the means of each group, but on the projection onto the covariate in each group. That is, with a covariate x and a factor f, it is like projecting out the interaction x:f. The \(x\) can also be a matrix of column vectors, in this case it can be beneficial to orthogonalize the columns, either with a stabilized Gram-Schmidt method, or with the simple method \(x E^*E \text{solve(chol(crossprod(x)))}\).

In some applications it is known that a single centering iteration is sufficient. In particular, if length(fl)==1 and there is no interaction attribute x. In this case the centering algorithm is terminated after the first iteration. There may be other cases, e.g. if there is a single factor with an x with orthogonal columns. If you have such prior knowledge, it is possible to force termination after the first iteration by adding an attribute attr(fl, 'oneiter') <-TRUE. Convergence will be reached in the second iteration anyway, but you save one iteration, i.e. you double the speed.

Value

If mtx is a matrix, a matrix of the same shape, possibly with column icpt deleted. If mtx is a list of vectors and matrices, a list of the same length is returned, with the same vector and matrix-pattern, but the matrices have the column icpt deleted.

If mtx is a 'data.frame', a 'data.frame' with the same attributes are returned.

Note

In the case that the design-matrix is too large for R, i.e. with more than 2 billion entries, it is possible to create a list of column-vectors instead (provided the vector-length is smaller than 2 billion). demeanlist will be able to centre these vectors.

Examples

```r
oldopts <- options(lfe.threads=1)
## create a 15x3 matrix
mtx <- matrix(rnorm(45),15,3)

## a list of factors
fl <- list(g1=factor(sample(2,nrow(mtx),replace=TRUE)),
           g2=factor(sample(3,nrow(mtx),replace=TRUE)))

## centre on both means and print result
mtx0 <- demeanlist(mtx,fl)
cbind(mtx0,g1=fl[[1]],g2=fl[[2]],comp=compfactor(fl))

for(i in 1:ncol(mtx0))
  for(n in names(fl))
    cat('col','i','group','n','level sums:','tapply(mtx0[,i],fl[[n]],mean)','\n')

options(oldopts)
```

efactory

Create estimable function

Description

Creates an estimable function for a factor-structure.

Usage

```r
efactory(obj, opt='ref',...)
```

Arguments

- `obj` object of class "felm", usually, a result of a call to `felm`.
- `opt` character. Which type of estimable function.
- `...` various.

Details

There are several possibilities for the input parameter `opt`.

- "ref" yields an estimable function which is similar to the default one in `lm`, one reference is forced to 0 in each connected component.
- "zm" Similar to "ref", but the factor which does not contain a reference is made to have zero mean, and an intercept is added.
• "zm2" Similar to "zm", but both factors are made to have zero mean.
• "ln" Least norm function. This will yield the raw coefficients from the Kaczmarz-method, i.e. the solution with smallest norm. This function is not estimable.

Note that in the case with more than two factors, it is not known how to analyze the factors to find the structure of the rank-deficiencies, i.e. the estimable functions. In this case, the factors beyond the first two are assumed not to contribute to the rank-deficiency beyond a single dimension in each. Both "ref" and "zm" keep one such reference at zero in each of these factors. This is the common method when using dummies.

In the case that interactions are specified in the model, i.e. with x:f in the second part of the formula, these terms are not analyzed to create an estimable function. Only the pure f terms are used for this purpose. It is assumed that the x:f terms are all identified. Note that in this case, all the levels of f are included.

Value

A function of two parameters function(y, addnames). An estimable function (i.e. the result is the vector of some length N) of the input vector v. When addnames==TRUE the returned vector should have names, and optionally an attribute "extra" which is a list of vectors of length N which may be used to code additional information.

Note

The author is open to suggestions for other estimable functions, i.e. other useful normalizations of the solutions.

It is not strictly necessary that the obj argument is of class "felm", any list with entries "fe" and "cfactor" of the appropriate form will do. That is, list(fe=fl, cfactor=compfactor(fl)) where fl is the list of factors defining the component structure. I.e. if the model is y ~ ... | id + firm, we have fl=list(id=id, firm=fir).

Examples

oldopts <- options(lfe.threads=1)
id <- factor(sample(5000,50000,replace=TRUE))
fir <- factor(sample(3000,50000,replace=TRUE))
fl <- list(id=id, fir=fir)
obj <- list(fe=fl, cfactor=compfactor(fl))
## the trivial least-norm transformation, which by the way is non-estimable
print(ef <- efactory(obj, 'ln'))
is.estimable(ef, fl)
## then the default
print(ef <- efactory(obj, 'ref'))
is.estimable(ef, fl)
# get the names of the coefficients, i.e. the nm-variable in the function
head(evalq(nm, environment(ef)))
options(oldopts)
'felm' is used to fit linear models with multiple group fixed effects, similarly to `lm`. It uses the Method of Alternating projections to sweep out multiple group effects from the normal equations before estimating the remaining coefficients with OLS.

This function is intended for use with large datasets with multiple group effects of large cardinality. If dummy-encoding the group effects results in a manageable number of coefficients, you are probably better off by using `lm`.

**Usage**

`felm(formula, data, exactDOF=FALSE, subset, na.action, contrasts=NULL, ...)`

**Arguments**

- **formula**: an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. Similarly to 'lm'. See Details.
- **data**: a data frame containing the variables of the model.
- **exactDOF**: logical. If more than two factors, the degrees of freedom used to scale the covariance matrix (and the standard errors) is normally estimated. Setting `exactDOF=TRUE` causes felm to attempt to compute it, but this may fail if there are too many levels in the factors. `exactDOF="rM"` will use the exact method in `Matrix::rankMatrix()`, but this is slower. If neither of these methods works, it is possible to specify `exactDOF="mc"`, which utilizes a Monte-Carlo method to estimate the expectation $E(x' P x) = tr(P)$, the trace of a certain projection, a method which may be more accurate than the default guess.
  
  If the degrees of freedom for some reason are known, they can be specified like `exactDOF=342772`.
- **subset**: an optional vector specifying a subset of observations to be used in the fitting process.
- **na.action**: a function which indicates what should happen when the data contain NAs. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The 'factory-fresh' default is `na.omit`. Another possible value is `na.fail`, no action. `na.exclude` is currently not supported.
- **contrasts**: an optional list. See the `contrasts.arg` of `model.matrix.default`.
- **...**: other arguments.
  
  - `keepX` logical. to include a copy of the expanded data matrix in the return value, as needed by `bccorr` and `fevcov` for proper bias correction.
  
  - `nostats` logical. Don't include covariance matrices in the output, just the estimated coefficients and various descriptive information. For IV, `nostats` can be a logical vector of length 2, with the last value being used for the 1st stages.
• kclass character. For use with instrumental variables. Use a k-class estimator rather than 2SLS/IV. Currently, the values 'nagar', 'b2sls', 'mb2sls', 'liml' are accepted, where the names are from Kolesár et al (2014), as well as a numeric value for the 'k' in k-class. With kclass='liml', felm also accepts the argument fuller=<numeric>, for using a Füller adjustment of the liml-estimator.

• iv, clustervar deprecated. These arguments will be removed at a later time, but is still supported in this field. Users are STRONGLY encouraged to use multipart formulas instead. In particular, not all functionality is supported with the deprecated syntax; iv-estimations actually run a lot faster if multipart formulas are used, due to new algorithms which I didn’t bother to shoehorn in place for the deprecated syntax.

Details

The formula specification is a response variable followed by a four part formula. The first part consists of ordinary covariates, the second part consists of factors to be projected out. The third part is an IV-specification. The fourth part is a cluster specification for the standard errors. I.e. something like y ~ x1 + x2 | f1 + f2 | (Q|W ~ x3+x4) | clu1 + clu2 where y is the response, x1,x2 are ordinary covariates, f1,f2 are factors to be projected out, Q and W are covariates which are instrumented by x3 and x4, and clu1,clu2 are factors to be used for computing cluster robust standard errors. Parts that are not used should be specified as P, except if it’s at the end of the formula, where they can be omitted. The parentheses are needed in the third part since | has higher precedence than ~. As of lfe version 2.0, multiple left hand sides like y|w|x ~ xQ + xR |fQ+fR|NNN are allowed.

Interactions between a covariate x and a factor f can be projected out with the syntax x:f. The terms in the second and fourth parts are not treated as ordinary formulas, in particular it is not possible with things like y ~ x1 | x*f, rather one would specify y ~ x1 + x | x:f + f. Note that f:x also works, since R’s parser does not keep the order. This means that in interactions, the factor must be a factor, whereas a non-interacted factor will be coerced to a factor. I.e. in y ~ x1 | x:f1 + f2, the f1 must be a factor, whereas it will work as expected if f2 is an integer vector.

In older versions of lfe the syntax was felm(y ~ x1 + x2 + G(f1) + G(f2), iv=list(Q ~ x3+x4, W ~ x3+x4),clustervar). This syntax still works, but yields a warning. Users are strongly encouraged to change to the new multipart formula syntax. The old syntax will be removed at a later time.

The standard errors are adjusted for the reduced degrees of freedom coming from the dummies which are implicitly present. In the case of two factors, the exact number of implicit dummies is easy to compute. If there are more factors, the number of dummies is estimated by assuming there’s one reference-level for each factor, this may be a slight over-estimation, leading to slightly too large standard errors. Setting exactDOF='rM' computes the exact degrees of freedom with rankMatrix() in package Matrix. Note that version 1.1-0 of Matrix has a bug in rankMatrix() for sparse matrices which may cause it to return the wrong value. A fix is underway.

For the iv-part of the formula, it is only necessary to include the instruments on the right hand side. The other explanatory covariates, from the first and second part of formula, are added automatically in the first stage regression. See the examples.

The contrasts argument is similar to the one in lm(), it is used for factors in the first part of the formula. The factors in the second part are analyzed as part of a possible subsequent getfe() call.
The old syntax with a single part formula with the `G()` syntax for the factors to transform away is still supported, as well as the `clustervar` and `iv` arguments, but users are encouraged to move to the new multi part formulas as described here. The `clustervar` and `iv` arguments have been moved to the ... argument list. They will be removed in some future update.

Value

`felm` returns an object of class "`felm`". It is quite similar to an "`lm`" object, but not entirely compatible.

The generic `summary`-method will yield a summary which may be `print`ed. The object has some resemblance to an 'lm' object, and some postprocessing methods designed for `lm` may happen to work. It may however be necessary to coerce the object to succeed with this.

The "`felm`" object is a list containing the following fields:

- `coefficients` a numerical vector. The estimated coefficients.
- `N` an integer. The number of observations
- `p` an integer. The total number of coefficients, including those projected out.
- `response` a numerical vector. The response vector.
- `fitted.values` a numerical vector. The fitted values.
- `residuals` a numerical vector. The residuals of the full system, with dummies. For IV-estimations, this is the residuals when the original endogenous variables are used, not their predictions from the 1st stage.
- `r.residuals` a numerical vector. Reduced residuals, i.e. the residuals resulting from predicting without the dummies.
- `iv.residuals` numerical vector. When using instrumental variables, residuals from 2. stage, i.e. when predicting with the predicted endogenous variables from the 1st stage.
- `cfactor` factor of length `N`. The factor describing the connected components of the two first terms in the second part of the model formula.
- `vcv` a matrix. The variance-covariance matrix.
- `fe` list of factors. A list of the terms in the second part of the model formula.
- `stage1` The 'felm' objects for the IV 1st stage, if used. The 1st stage has multiple left hand sides if there are more than one instrumented variable.
- `ivfstat` list of numerical vectors. For IV 1st stage, F-value for excluded instruments, the number of parameters in restricted model and in the unrestricted model.
- `X` matrix. The expanded data matrix, i.e. from the first part of the formula. To save memory with large datasets, it is only included if `felm(keepX=TRUE)` is specified. Must be included if `bccorr` or `fevcov` is to be used for correcting limited mobility bias.

Note

For technical reasons, when running IV-estimations, the data frame supplied in the `data` argument to `felm`, should not contain variables with names ending in "(fit)". Variables with such names are used internally by `felm`, and may then accidentally be looked up in the data frame instead of the local environment where they are defined.
References


See Also
getfe summary.felm confstat waldtest

Examples

oldopts <- options(lfe.threads=1)
## create covariates
x <- rnorm(1000)
x2 <- rnorm(length(x))

## individual and firm
id <- factor(sample(20,length(x),replace=TRUE))
firm <- factor(sample(13,length(x),replace=TRUE))

## effects for them
id.eff <- rnorm(nlevels(id))
firm.eff <- rnorm(nlevels(firm))

## left hand side
u <- rnorm(length(x))
y <- x + 0.5*x2 + id.eff[id] + firm.eff[firm] + u

## estimate and print result
est <- felm(y ~ x+x2| id + firm)
summary(est)
## Not run:
## compare with lm
summary(lm(y ~ x + x2 + id + firm-1))
## End(Not run)

# make an example with 'reverse causation'
# Q and W are instrumented by x3 and the factor x4. Report robust s.e.
x3 <- rnorm(length(x))
x4 <- sample(12,length(x),replace=TRUE)

Q <- 0.3*x3 + x + 0.2*x2 + id.eff[id] + 0.3*log(x4) - 0.3*y + rnorm(length(x),sd=0.3)
W <- 0.7*x3 - 2*x + 0.1*x2 - 0.7*id.eff[id] + 0.8*cos(x4) - 0.2*y + rnorm(length(x),sd=0.6)

# add them to the outcome
y <- y + Q + W
```r
evest <- felm(y ~ x + x2 | id+firm | (Q|x3+factor(x4)))
summary(evest, robust=TRUE)
condfstat(evest)
## Not run:
# compare with the not instrumented fit:
summary(felm(y ~ x + x2 + Q + W | id+firm))
## End(Not run)
options(olddots)
```

text

**fevcov**

*Compute bias corrected covariance matrix between fixed effects*

**Description**

With a model like \( y = X \beta + D \theta + F \psi + \varepsilon \), where \( D \) and \( F \) are matrices with dummy encoded factors, one application of `felm` is to study the variances \( \text{var}(D \theta) \), \( \text{var}(F \psi) \) and covariances \( \text{cov}(D \theta, F \psi) \). However, if we use estimates for \( \theta \) and \( \psi \), the resulting variances are biased. The function `fevcov` computes a bias corrected covariance matrix as described in Gaure (2014).

**Usage**

```r
fevcov(est, alpha=getfe(est), tol=0.01,
        robust=!is.null(est$clustervar), maxsamples=Inf, lhs=NULL)
```

**Arguments**

- `est` an object of class `"felm"`, the result of a call to `felm(keepX=TRUE)`.
- `alpha` a data frame, the result of a call to `getfe`.
- `tol` numeric. The absolute tolerance for the bias-corrected correlation.
- `robust` logical. Should robust (heteroskedastic or cluster) residuals be used, rather than i.i.d.
- `maxsamples` integer. Maximum number of samples for expectation estimates.
- `lhs` character. Name of left hand side if multiple left hand sides.

**Details**

The `tol` argument specifies the tolerance. The tolerance is relative for the variances, i.e. the diagonal of the output. For the covariances, the tolerance is relative to the square root of the product of the variances, i.e. an absolute tolerance for the correlation. If a numeric of length 1, `tol` specifies the same tolerance for all variances/covariances. If it is of length 2, `tol[1]` specifies the variance tolerance, and `tol[2]` the covariance tolerance. `tol` can also be a square matrix of size `length(est$fe)`, in which case the tolerance for each variance and covariance is specified individually.
The function performs no checks for estimability. If the fixed effects are not estimable, the result of a call to `fevcov` is not usable. Moreover, there should be just a single connected component among the fixed effects.

alpha must contain a full set of coefficients, and contain columns 'fe' and 'effect' like the default estimable functions from `efactory`.

**Value**

`fevcov` returns a square matrix with the bias corrected covariances. An attribute 'bias' contains the biases. The bias corrections have been subtracted from the bias estimates. I.e. $vc = vc' - b$, where $vc'$ is the biased variance and $b$ is the bias.

**Note**

Note that if `est` is the result of a call to `felm` with `keepX=FALSE` (the default), the biases will be computed as if the covariates $X$ are independent of the factors. This will be faster (typically by a factor of approx. 4), and possibly wronger. Note also that the computations performed by this function are non-trivial, they may take quite some time. It would be wise to start out with quite liberal tolerances, e.g. `tol=0.1`, to get an idea of the time requirements.

**References**


**See Also**

`varvars`, `bccorr`

**Examples**

```r
x <- rnorm(5000)
x2 <- rnorm(length(x))

## create individual and firm
id <- factor(sample(40,length(x),replace=TRUE))
firm <- factor(sample(30,length(x),replace=TRUE,prob=c(2,rep(1,29))))
foo <- factor(sample(20,length(x),replace=TRUE))

## effects
id.eff <- rnorm(nlevels(id))
firm.eff <- runif(nlevels(firm))
foo.eff <- rchisq(nlevels(foo),df=1)

## left hand side
id.m <- id.eff[id]
firm.m <- firm.eff[firm]
foo.m <- foo.eff[foo]

## normalize them
id.m <- id.m/sd(id.m)
firm.m <- firm.m/sd(firm.m)
foo.m <- foo.m/sd(foo.m)

y <- x + 0.25*x2 + id.m + firm.m + foo.m + rnorm(length(x),sd=2)
```
getfe

Retrieve the group fixed effects

Description

Compute the group fixed effects, i.e. the dummy parameters, which were swept out during an estimation with `felm`.

Usage

```r
getfe(obj, references=NULL, se=FALSE,
      method='kaczmarz', ef='ref', bN=100, robust=FALSE,
      cluster=obj[['clustervar']], lhs=NULL)
```

Arguments

- `obj`: object of class "felm", usually, a result of a call to `felm`
- `references`: a vector of strings. If there are more than two factors and you have prior knowledge of what the reference levels should be like references="id.23". Not used with method='kaczmarz'
- `se`: logical. Set to TRUE if standard errors for the group effects are wanted. This is very time-consuming for large problems, so leave it as FALSE unless absolutely needed.
- `method`: character string. Either 'cholesky', or the default 'kaczmarz'. The latter is often very fast and consumes little memory, it requires an estimable function to be specified, see `efactory`. The 'cholesky' method is no longer maintained as the author sees no use for it.
- `ef`: function. A function of two variables, a vector of group fixed effects and a logical, i.e. function(v, addnames). This function should be estimable and is used to transform the raw-coefficients v from the kaczmarz-method. The second variable indicates whether the function must return a named vector (if this is FALSE, one may skip the names, saving memory allocations and time).
  - If a string is specified, it is fed to the `efactory`-function. The default function is one which picks one reference in each component.
  - Can be set to ef="ln" to yield the minimal-norm solution from the kaczmarz-method.
  - It can also be set to ef="zm" to get zero means (and intercept) in one of the factors, and a reference in the other.
getfe

bN integer. The number of bootstrap runs when standard errors are requested.
robust logical. Should heteroskedastic standard errors be estimated?
cluster logical or factor. Estimate clustered standard errors.
lhs character vector. Specify which left hand side if obj has multiple lhs.

Details

For the case with two factors (the terms in the second part of the formula supplied to felm), one reference in each connected component is adequate when interpreting the results.

For three or more factors, no such easy method is known; for the "cholesky" method- reference levels are found by analyzing the pivoted Cholesky-decomposition of a slightly perturbed system. The "kaczmarz" method provides no rank-deficiency analysis, it is assumed that the factors beyond the two first contribute nothing to the rank-deficiency, so one reference in each is used.

If there are more than two factors, only the first two will be used to report connected components. In this case, it is not known which graph theoretic concept may be used to analyze the rank-deficiency.

The standard errors returned by the Kaczmarz-method are bootstrapped, keeping the other coefficients (from felm) constant, i.e. they are from the variance when resampling the residuals. If robust=TRUE, heteroskedastic robust standard errors are estimated. If robust=FALSE and cluster=TRUE, clustered standard errors with the cluster specified to felm() are estimated. If cluster is a factor, it is used for the cluster definition.

Value

The function getfe computes and returns a data frame containing the group fixed effects. It has the columns c('effect','se','obs','comp','fe','idx')

- effect is the estimated effect.
- se is the standard error.
- obs is the number of observations of this level.
- comp is the graph-theoretic component number, useful for interpreting the effects.
- fe is the name of factor.
- idx is the level of the factor.

With the Kaczmarz-method it's possible to specify a different estimable function.

Examples

oldopts <- options(lfe.threads=2)
## create covariates
x <- rnorm(4000)
x2 <- rnorm(length(x))

## create individual and firm
id <- factor(sample(500,length(x),replace=TRUE))
fir <- factor(sample(300,length(x),replace=TRUE))

## effects
is.estimable <- rlnorm(nlevels(id))
firm.eff <- rexp(nlevels(firm))

## left hand side
y <- x + 0.25*x2 + id.eff[id] + firm.eff[firm] + rnorm(length(x))

## estimate and print result
est <- felm(y ~ x+x2 | id + firm)
summary(est)
## extract the group effects
alpha <- getfe(est,se=TRUE)

## find some estimable functions, with standard errors, we don't get
## names so we must precompute some numerical indices in ef
idx <- match(c('id.5','id.6','firm.11','firm.12'),rownames(alpha))
alpha[idx,]

ef <- function(v,addnames) {
  w <- c(v[idx[2]]-v[idx[1]],v[idx[4]]+v[idx[3]],
         v[idx[4]]-v[idx[3]])
  if(addnames) names(w) <- c('id6-id5','f12+id5','f12-f11')
  w
}
getfe(est,ef=ef,se=TRUE)
options(olddopts)
## Not run:
summary(lm(y ~ x+x2+id+firm-1))
## End(Not run)

---

**is.estimable**

*Verify estimability of function*

**Description**

Verify that a function you have written for `getfe` is indeed estimable.

**Usage**

```r
is.estimable(ef, fe, R=NULL, nowarn=FALSE, keepdiff=FALSE, threshold=1e-5)
```

**Arguments**

- **ef**: function. The function to be verified.
- **fe**: list of factors.
- **R**: numeric. Vector of residuals, if NULL, a random one is created.
- **nowarn**: logical. Set to `TRUE` if `is.estimable` should not throw a warning for non-estimable functions.
- **keepdiff**: logical. Return differences between two different runs of the Kaczmarz method.
- **threshold**: numeric. Threshold for determining estimability.
Details

When writing custom estimable functions for getfe, the function is_estimable can be used to test it for estimability. is_estimable() solves the sparse residual system with the Kaczmarz method, using two different initial values. Then ef() is applied to the two solutions. If the value of ef() differs by more than 1e-5 in any coordinate, FALSE is returned, otherwise TRUE is returned. If keepdiff=TRUE, the vector of differences is attached as an attribute 'diff' to the returned logical value. If you have problems with estimability, it is a fair guess that those entries with a difference in absolute values smaller than, say, 1e-5 are estimable, whereas the others are not.

Value

Returns a logical.

See Also

getfe

Examples

oldopts <- options(lfe.threads=1)
## create individual and firm
id <- factor(sample(5000,50000,replace=TRUE))
firm <- factor(sample(3000,50000,replace=TRUE))

## create some estimable functions. It's faster to
## use numerical indices in ef rather than strings, and the input v
## to ef has no names, we have to add them when requested
ef <- function(v,addnames) {
  w <- c(v[6]-v[5],v[7000]+v[5],v[7000]-v[6000])
  if(addnames) names(w) <-c('id6-id5','f2k+id5','f2k-f1k')
  w
}
is.estimable(ef,list(id=id,firm=firm))

## Then make an error; in the last coordinate, sum two firms
ef <- function(v,addnames) {
  w <- c(v[6]-v[5],v[7000]+v[5],v[7000]+v[6000])
  if(addnames) names(w) <-c('id6-id5','f2k+id5','f2k-f1k')
  w
}
is.estimable(ef, list(id=id,firm=firm), keepdiff=TRUE)
options(oldopts)

kaczmarz

Solve a linear system defined by factors

Description

Uses the Kaczmarz method to solve a system of the type Dx = R, where D is the matrix of dummies created from a list of factors.
Usage

kaczmarz(f1,R,eps=getOption('lfe.eps'),
    init=NULL,threads=getOption('lfe.threads'))

Arguments

- **f1**: A list of arbitrary factors of the same length
- **R**: numeric. A vector, matrix or list of such of the same length as the factors
- **eps**: a tolerance for the method
- **init**: numeric. A vector to use as initial value for the Kaczmarz iterations. The algorithm converges to the solution closest to this
- **threads**: integer. The number of threads to use when R is more than one vector

Value

A vector x of length equal to the sum of the number of levels of the factors in f1, which solves the system Dx=R. If the system is inconsistent, the algorithm may not converge, it will give a warning and return something which may or may not be close to a solution. By setting eps=0, maximum accuracy (with convergence warning) will be achieved.

Note

This function is used by getfe, it's quite specialized, but it might be useful for other purposes too.

In case of convergence problems, setting options(lfe.usecg=TRUE) will cause the kaczmarz() function to dispatch to the conjugate gradient method of Rcgmin. This may or may not be faster.

Examples

```r
oldopts <- options(lfe.threads=1)
## create factors
f1 <- factor(sample(24000,10000,replace=TRUE))
f2 <- factor(sample(20000,length(f1),replace=TRUE))
f3 <- factor(sample(10000,length(f1),replace=TRUE))
f4 <- factor(sample(8000,length(f1),replace=TRUE))
## the matrix of dummies
D <- t(do.call(rBind, lapply(list(f1,f2,f3,f4),as,'sparseMatrix')))
dim(D)
## an x
trux <- runif(ncol(D))
## and the right hand side
R <- as.vector(D %*% trux)
## solve it
sol <- kaczmarz(list(f1,f2,f3,f4),R)
## verify that the solution solves the system Dx = R
res <- D %*% sol - R
sqrt(sum(res^2))
## but the solution is not equal to the true x, because the system is underdetermined
sqrt(sum((sol - trux)^2))
```
```r
## moreover, the solution from kaczmarz has smaller norm
sqrt(sum(sol^2))
sqrt(sum(truex^2))
options(olddopts)
```

### summary.felm

**Summarize felm model fits**

**Description**

summary method for class "felm".

**Usage**

```r
## S3 method for class 'felm'
summary(object, ..., robust=!is.null(object$clustvar),
lhs=NULL)
```

**Arguments**

- `object` an object of class "felm", a result of a call to felm.
- `...` further arguments passed to or from other methods.
- `robust` logical. Use robust standard errors. See notes.
- `lhs` character. If multiple left hand sides, specify the name of the one to obtain a summary for.

**Value**

The function `summary.felm` returns an object of class "summary.felm". It is quite similar to an "summary.lm" object, but not entirely compatible.

The "summary.felm" object is a list containing the following fields:

- `residuals` a numerical vector. The residuals of the full system, with dummies.
- `p` an integer. The total number of coefficients, including those projected out.
- `Pp` an integer. The number of coefficients, excluding those projected out.
- `coefficients` a Pp x 4 matrix with columns for the estimated coefficients, their standard errors, t-statistic and corresponding (two-sided) p-value.
- `rse` residual standard error.
- `r2` $R^2$, the fraction of variance explained by the model.
- `r2adj` Adjusted $R^2$.
- `fstat` F-statistic.
- `pval` P-values.
- `P.fstat` Projected F-statistic. The result of a call to `waldtest`
list of factors. A list of the terms in the second part of the model.

character. If object is the result of an estimation with multiple left hand sides,
the actual argument lhs will be copied to this field.

F-statistic for excluded instruments in 1. step IV, see felm.

P-value for ivfstat.

The standard errors are adjusted for the reduced degrees of freedom coming from the dummies
which are implicitly present. They are also small-sample corrected.

If the robust parameter is FALSE, the returned object will contain ordinary standard errors. If the
robust parameter is TRUE, clustered standard errors are reported if a cluster was specified in the
call to felm; if not, heteroskedastic robust standard errors are reported.

Several F-statistics reported. The P.fstat is for the projected system. I.e. a joint test on whether
all the Pp coefficients in coefficients are zero. Then there are fstat and pval which is a test on
all the coefficients including the ones projected out, except for an intercept. This statistic assumes
i.i.d. errors and is not reliable for robust or clustered data.

For a 1st stage IV-regression, an F-statistic against the model with excluded instruments is also
computed.

See Also

waldtest

Compute the variance of the fixed effect variance estimate

With a model like 'y = X beta + D theta + F psi + epsilon', where 'D' and 'F' are matrices with
dummy encoded factors, one application of lfe is to study the variances 'var(D theta), var(F psi)'
and covariances 'cov(D theta, F psi)' The function fevcov computes bias corrected variances and
covariances. However, these variance estimates are still random variables for which fevcov only
estimate the expectation. The function varvars estimates the variance of these estimates.

Usage

varvars(est, alpha=getfe(est), tol=0.01, biascorrect=FALSE, lhs=NULL)

Arguments

est an object of class "felm", the result of a call to felm(keepX=TRUE).
alpha a data frame, the result of a call to getfe.
tol numeric. The absolute tolerance for the bias-corrected correlation.
biascorrect logical. Should the estimates be bias corrected?
lhs character. Name of left hand side if multiple left hand sides.
Details

This function returns valid results only for normally distributed residuals. Note that the estimates for the fixed effect variances from `fevcov` are not normally distributed, but a sum of chi-square distributions which depends on the eigenvalues of certain large matrices. We do not compute that distribution. The variances returned by `varvars` can therefore not be used directly to estimate confidence intervals, other than through coarse methods like the Chebyshev inequality. These estimates only serve as a rough guideline as to how wrong the variance estimates from `fevcov` might be.

Like the fixed effect variances themselves, their variances are also biased upwards. Correcting this bias can be costly, and is therefore by default switched off.

The variances tend to zero with increasing number of observations. Thus, for large datasets they will be quite small.

Value

`varvars` returns a vector with a variance estimate for each fixed effect variance. I.e. for the diagonal returned by `fevcov`.

Note

The `tol` argument specifies the tolerance as in `fevcov`. Note that if `est` is the result of a call to `felm` with `keepx=FALSE` (the default), the variances will be estimated as if the covariates $X$ are independent of the factors. There is currently no function available for estimating the variance of the covariance estimates from `fevcov`.

The cited paper does not contain the expressions for the variances computed by `varvars` (there’s a 10 page limit in that journal), though they can be derived in the same fashion as in the paper, with the formula for the variance of a quadratic form.

References


See Also

`bccorr` `fevcov`

Examples

```r
x <- rnorm(500)
x2 <- rnorm(length(x))

## create individual and firm
id <- factor(sample(40,length(x),replace=TRUE))
firm <- factor(sample(30,length(x),replace=TRUE,prob=c(2,rep(1,29))))
foo <- factor(sample(20,length(x),replace=TRUE))

## effects
id.eff <- rnorm(nlevels(id))
firm.eff <- rnorm(nlevels(firm))
foo.eff <- rnorm(nlevels(foo))
```
waldtest

## left hand side

id.m <- id.eff[id]
firm.m <- 2*firm.eff[firm]/foo.m <- 3*foo.eff[foo]
y <- x + 0.25*x2 + id.m + firm.m + foo.m + rnorm(length(x))

# make a data frame
fr <- data.frame(y, x, x2, id, firm, foo)

## estimate and print result

est <- felm(y ~ x + x2 | id + firm + foo, data=fr, keepX=TRUE)
alpha <- getfe(est)

# estimate the covariance matrix of the fixed effects
fevcov(est, alpha)

# estimate variances of the diagonal
varvars(est, alpha)

---

**waldtest**

*Compute Wald test for linear restriction or joint significance.*

### Description

Compute a Wald test for a linear hypothesis on some coefficients.

### Usage

```r
waldtest(object, R, r, type=c('default','iid','robust','cluster'), 
lhs=NULL, df1, df2)
```

### Arguments

- `object` object of class "felm", a result of a call to `felm`.
- `R` matrix, character, formula, integer or logical. Specification of which exclusions to test.
- `r` numerical vector.
- `type` character. Error structure type.
- `lhs` character. Name of left hand side if multiple left hand sides.
- `df1` integer. If you know better than the default df, specify it here.
- `df2` integer. If you know better than the default df, specify it here.

### Details

The function `waldtest` computes a Wald test for the H0: R beta = r, where beta is the estimated vector `coef(object)`.

If `R` is a character, integer, or logical vector it is assumed to specify a matrix which merely picks out a subset of the coefficients for joint testing. If `r` is not specified, it is assumed to be a zero vector of the appropriate length.
R can also be a formula which is linear in the estimated coefficients, e.g. of the type \(~Q=2|x-2*z\) which will test the joint hypothesis \(Q=2\) and \(x=2*z\).

In case of an IV-estimation, the names for the endogenous variables in `coef(object)` are of the type "\(Q(\text{fit})\)" which is a bit dull to type; if all the endogenous variables are to be tested they can be specified as "endovars". It is also possible to specify an endogenous variable simply as "Q", and `waldtest` will add the other syntactic sugar to obtain "\(Q(\text{fit})\)".

The `type` argument works as follows. If `type`='default' it is assumed that the residuals are i.i.d., unless a cluster structure was specified to `felm`. If `type`='robust', a heteroscedastic structure is assumed, even if a cluster structure was specified in `felm`.

**Value**

The function `waldtest` computes and returns a named numeric vector containing the following elements.

- \(p\) is the p-value for the Chi\(^2\)-test
- \(\chi^2\) is the Chi\(^2\)-distributed statistic.
- \(df1\) is the degrees of freedom for the Chi\(^2\) statistic.
- \(p.F\) is the p-value for the F statistics
- \(F\) is the F-distributed statistic.
- \(df2\) is the additional degrees of freedom for the F statistic.

The return value has an attribute `formula` which encodes the restrictions.

**Examples**

```r
x <- rnorm(10000)
x2 <- rnorm(length(x))
y <- x - 0.2*x2 + rnorm(length(x))
# Also works for lm
summary(est <- lm(y ~ x + x2))
# We do not reject the true values
waldtest(est, ~ x-1|x2+0.2|`(Intercept)`)```
Index

*Topic models
  getfe, 21
  lfe-package, 2
*Topic regression
  getfe, 21
  lfe-package, 2

bccorr, 3, 4, 15, 17, 20, 28
btrap, 7, 7
compfactor, 2, 8
condfstat, 3, 9, 18
demeanlist, 2, 11
efactory, 13, 20, 21
felm, 2, 3, 5–7, 10, 11, 13, 15, 19–22, 27–30
fevcov, 3, 6, 15, 17, 19, 27, 28
getfe, 2, 3, 5, 7, 18, 19, 21, 23–25, 27
is.estimable, 23
kaczmarz, 24
lfe (lfe-package), 2
lfe-package, 2
lm, 2, 13, 15
summary.felm, 18, 26
varvars, 20, 27
waldtest, 10, 18, 26, 27, 29