Package: lfe (via r-universe)

October 28, 2026

Version 3.1.1

Title Linear Group Fixed Effects

Depends R (>= 2.15.2), Matrix (>= 1.1-2)

Imports Formula, xtable, compiler, utils, methods, sandwich, parallel

Suggests knitr, digest, igraph, plm, cubature (>= 2.0.3), numDeriv, data.table, alpaca

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 with many levels as pure control variables. See Gaure (2013) <doi:10.1016/j.csda.2013.03.024> 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 (Gaure 2014 <doi:10.1002/sta4.68>). Since version 3.0, it provides dedicated functions to estimate Poisson models.

License Apache License (>= 2)

Classification/JEL C13, C23, C60

Classification/MSC 62J05, 65F10, 65F50

URL https://github.com/r-econometrics/lfe

BugReports https://github.com/r-econometrics/lfe/issues

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation yes

Author Simen Gaure [aut] (<https://orcid.org/0000-0001-7251-8747>), Grant McDermott [ctb], Mauricio Vargas Sepulveda [ctb, cre] (<https://orcid.org/0000-0003-1017-7574>), Karl Dunkle Werner [ctb], Matthieu Stigler [ctb] (<https://orcid.org/0000-0002-6802-4290>), Daniel Lüdecke [ctb]

Maintainer Mauricio Vargas Sepulveda <m.sepulveda@mail.utoronto.ca>

Repository https://cran.r-universe.dev

Date/Publication 2025-02-11 07:30:01 UTC

RemoteUrl https://github.com/cran/lfe

RemoteRef HEAD

RemoteSha 367ab39d53fb3c68ba8a1b746d95728e8116b106

Contents

lfe-package . . . 2
..oldfelm . . . 6
bccorr . . . 8
btrap . . . 10
cgsolve . . . 12
chainsubset . . . 13
compfactor . . . 14
condfstat . . . 15
demeanlist . . . 17
efactory . . . 20
felm . . . 22
fepois . . . 28
fevcov . . . 29
getfe . . . 31
is.estimable . . . 34
kaczmarz . . . 36
makeDmatrix . . . 37
mctrace . . . 38
nlexpect . . . 39
sargan . . . 42
summary.felm . . . 43
varvars . . . 44
waldtest . . . 46

Index **49**

`lfe-package` *Overview. Linear Group Fixed Effects*

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 `stats::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. If your factors don't have thousands of levels, `stats::lm()` or other packages are probably better suited. **lfe** is designed to produce the same results as `stats::lm()` will do if run with the full set of dummies.

Projecting out interactions between continuous covariates and factors is supported. I.e. individual slopes, not only individual intercepts. Multiple left hand sides are supported.

The package does not support non-linear models. For GLMs with many dummies there is a package **alpaca** which uses similar methods to project them out.

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 by *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 or data frame, 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()`.

For those who study the correlation between the fixed effects, like in *Abowd et al. (1999)*, there are functions `bccorr()` and `fevcov()` for computing limited mobility bias corrected correlations and variances with the method described in *Gaure (2014b)*.

Instrumental variable estimations are supported with 2SLS. Conditional F statistics for testing reduced rank weak instruments as in *Sanderson and Windmeijer (2015)* are available in `condfstat()`. Joint significance testing of coefficients is available in `waldtest()`.

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 can 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 is 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)`. Various time-consuming parts of **lfe** may print progress reports, the minimum interval in seconds is `options(lfe.pint=1800)`.

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, as demonstrated by *Gaure (2014a)*.

The package will work with any 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 group coefficients are estimable. A discussion of estimability, the algorithm used, and convergence rate are available in vignettes, as well as in the published papers in the citation list (`citation('lfe')`).

In the exec-directory there is a perl-script `lfescript` which is used at the author's site for automated creation of 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`. The method is very similar to the one in the Stata module `reghdfe`.

Author(s)

Maintainer: Mauricio Vargas Sepulveda <m.sepulveda@mail.utoronto.ca> (ORCID) [contributor]

Authors:

- Simen Gaure (ORCID)

Other contributors:

- Grant McDermott [contributor]
- Karl Dunkle Werner [contributor]
- Matthieu Stigler (ORCID) [contributor]
- Daniel Lüdecke [contributor]

References

Abowd, J.M., F. Kramarz and D.N. Margolis (1999) *High Wage Workers and High Wage Firms*, Econometrica 67 (1999), no. 2, 251–333. doi:10.1111/14680262.00020

Abowd, J.M., R. Creecy and F. Kramarz (2002) *Computing Person and Firm Effects Using Linked Longitudinal Employer-Employee Data.* Technical Report TP-2002-06, U.S. Census Bureau. https://www2.census.gov/ces/tp/tp-2002-06.pdf

Andrews, M., L. Gill, T. Schank and R. Upward (2008) *High wage workers and low wage firms: negative assortative matching or limited mobility bias?* J.R. Stat. Soc.(A) 171(3), 673–697. doi:10.1111/j.1467985X.2007.00533.x

Cornelissen, T. (2008) *The stata command felsdvreg to fit a linear model with two high-dimensional fixed effects.* Stata Journal, 8(2):170–189, 2008. https://econpapers.repec.org/RePEc:tsj:stataj:v:8:y:2008:i:2:p:170-189

Correia, S. (2014) *REGHDFE: Stata module to perform linear or instrumental-variable regression absorbing any number of high-dimensional fixed effects*, Statistical Software Components, Boston College Department of Economics. https://econpapers.repec.org/RePEc:boc:bocode:s457874

Croissant, Y. and G. Millo (2008) *Panel Data Econometrics in R: The plm Package*, Journal of Statistical Software, 27(2). https://www.jstatsoft.org/v27/i02/

Gaure, S. (2013) *OLS with Multiple High Dimensional Category Variables.* Computational Statistics and Data Analysis, 66:8–18, 2013 doi:10.1016/j.csda.2013.03.024

Gaure, S. (2014a) *lfe: Linear Group Fixed Effects.* The R Journal, 5(2):104-117, Dec 2013. https://journal.r-project.org/archive/2013/RJ-2013-031/RJ-2013-031.pdf

Gaure, S. (2014b), *Correlation bias correction in two-way fixed-effects linear regression*, Stat 3(1):379-390, 2014. doi:10.1002/sta4.68

Guimaraes, P. and Portugal, P. (2010) *A simple feasible procedure to fit models with high-dimensional fixed effects.* The Stata Journal, 10(4):629–649, 2010. https://www.stata-journal.com/article.html?article=st0212

Ouazad, A. (2008) *A2REG: Stata module to estimate models with two fixed effects.* Statistical Software Components S456942, Boston College Department of Economics. https://ideas.repec.org/c/boc/bocode/s456942.html

Sanderson, E. and F. Windmeijer (2014) *A weak instrument F-test in linear IV models with multiple endogenous variables*, Journal of Econometrics, 2015. doi:10.1016/j.jeconom.2015.06.004

See Also

Useful links:

- https://github.com/r-econometrics/lfe
- Report bugs at https://github.com/r-econometrics/lfe/issues

Examples

```
oldopts <- options("lfe.threads")
options(lfe.threads = 2)
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)
```

..oldfelm *Fit a linear model with multiple group fixed effects (old interface)*

Description

Fit a linear model with multiple group fixed effects (old interface)

Usage

```
..oldfelm(
  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 `NA`s. 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 `NULL`, no action. `na.exclude` is currently not supported.
contrasts	an optional list. See the `contrasts.arg` of `model.matrix.default`.
...	other arguments. • `cmethod` character. Which clustering method to use. Known arguments are `'cgm'` (the default), `'cgm2'` (or `'reghdfe'`, its alias). These alternate methods will generally yield equivalent results, except in the case of multiway clustering with few clusters along at least one dimension.

- keepX logical. To include a copy of the expanded data matrix in the return value, as needed by bccorr() and fevcov() for proper limited mobility bias correction.
- keepCX logical. Keep a copy of the centred expanded data matrix in the return value. As list elements cX for the explanatory variables, and cY for the outcome.
- keepModel logical. Keep a copy of the model frame.
- 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.
- psdef logical. In case of multiway clustering, the method of Cameron, Gelbach and Miller may yield a non-definite variance matrix. Ordinarily this is forced to be semidefinite by setting negative eigenvalues to zero. Setting psdef=FALSE will switch off this adjustment. Since the variance estimator is asymptotically correct, this should only have an effect when the clustering factors have very few levels.
- 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 *Kolesar 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 Fuller adjustment of the liml-estimator.
- Nboot, bootexpr, bootcluster Since felm has quite a bit of overhead in the creation of the model matrix, if one wants confidence intervals for some function of the estimated parameters, it is possible to bootstrap internally in felm. That is, the model matrix is resampled Nboot times and estimated, and the bootexpr is evaluated inside an sapply. The estimated coefficients and the left hand side(s) are available by name. Any right hand side variable x is available by the name var.x. The "felm"-object for each estimation is available as est. If a bootcluster is specified as a factor, entire levels are resampled. bootcluster can also be a function with no arguments, it should return a vector of integers, the rows to use in the sample. It can also be the string 'model', in which case the cluster is taken from the model. bootexpr should be an expression, e.g. like quote(x/x2 * abs(x3)/mean(y)). It could be wise to specify nostats=TRUE when bootstrapping, unless the covariance matrices are needed in the bootstrap. If you need the covariance matrices in the full estimate, but not in the bootstrap, you can specify it in an attribute "boot" as nostats=structure(FALSE, boot=TRUE).
- iv, clustervar deprecated. These arguments will be removed at a later time, but are 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.

bccorr *Compute limited mobility bias corrected correlation between fixed effects*

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 θ and ψ, the resulting correlation is biased. The function `bccorr` computes a bias corrected correlation as described in *Gaure (2014)*.

Usage

```
bccorr(
  est,
  alpha = getfe(est),
  corrfactors = 1L:2L,
  nocovar = (length(est$X) == 0) && length(est$fe) == 2,
  tol = 0.01,
  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()`.
`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 $tr(AB^{-1}C)$ where A, B, and C are matrices too large to be handled directly. `bccorr` estimates the trace by using the formula $tr(M) = E(x^t Mx)$ 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 <- vDF/sqrt(vD*vF)`. The variances are estimated to a

relative tolerance specified by the argument `tol`. The covariance bias is estimated to an absolute tolerance in the correlation `rho` (conditional on the already bias corrected `vD` and `vF`) specified by `tol`. 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`.

If `est` is the result of a weighted `felm()` estimation, the variances and correlations are weighted too.

Value

`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

Bias correction for IV-estimates are not supported as of now.

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.

References

Gaure, S. (2014), *Correlation bias correction in two-way fixed-effects linear regression*, Stat 3(1):379:390, 2014.

See Also

`fevcov()`

Examples

```
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

```
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")
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)
```

`cgsolve` *Solve a symmetric linear system with the conjugate gradient method*

Description

`cgsolve` uses a conjugate gradient algorithm to solve the linear system $Ax = b$ where A is a symmetric matrix. `cgsolve` is used internally in **lfe** in the routines `fevcov()` and `bccorr()`, but has been made public because it might be useful for other purposes as well.

Usage

```
cgsolve(A, b, eps = 0.001, init = NULL, symmtest = FALSE, name = "")
```

Arguments

`A`	matrix, Matrix or function.
`b`	vector or matrix of columns vectors.
`eps`	numeric. Tolerance.
`init`	numeric. Initial guess.
`symmtest`	logical. Should the matrix be tested for symmetry?
`name`	character. Arbitrary name used in progress reports.

Details

The argument `A` can be a symmetric matrix or a symmetric sparse matrix inheriting from `"Matrix"` of the package **Matrix**. It can also be a function which performs the matrix-vector product. If so, the function must be able to take as input a matrix of column vectors.

If the matrix `A` is rank deficient, some solution is returned. If there is no solution, a vector is returned which may or may not be close to a solution. If `symmtest` is `FALSE`, no check is performed that `A` is symmetric. If not symmetric, `cgsolve` is likely to raise an error about divergence.

The tolerance eps is a relative tolerance, i.e. $||x - x_0|| < \epsilon||x_0||$ where x_0 is the true solution and x is the solution returned by cgsolve. Use a negative eps for absolute tolerance. The termination criterion for cgsolve is the one from *Kaasschieter (1988)*, Algorithm 3.

Preconditioning is currently not supported.

If A is a function, the test for symmetry is performed by drawing two random vectors x,y, and testing whether $|(Ax, y) - (x, Ay)| < 10^{-6} sqrt((||Ax||^2 + ||Ay||^2)/N)$, where N is the vector length. Thus, the test is neither deterministic nor perfect.

Value

A solution x of the linear system $Ax = b$ is returned.

References

Kaasschieter, E. (1988) *A practical termination criterion for the conjugate gradient method*, BIT Numerical Mathematics, 28(2):308-322. https://link.springer.com/article/10.1007/BF01934094

See Also

kaczmarz()

Examples

```
N <- 100000
# create some factors
f1 <- factor(sample(34000, N, replace = TRUE))
f2 <- factor(sample(25000, N, replace = TRUE))
# a matrix of dummies, which probably is rank deficient
B <- makeDmatrix(list(f1, f2))
dim(B)
# create a right hand side
b <- as.matrix(B %*% rnorm(ncol(B)))
# solve B' B x = B' b
sol <- cgsolve(crossprod(B), crossprod(B, b), eps = -1e-2)
# verify solution
sqrt(sum((B %*% sol - b)^2))
```

---

chainsubset *Chain subset conditions*

Description

Chain subset conditions

Usage

```
chainsubset(..., out.vars)
```

Arguments

...	Logical conditions to be chained.
out.vars	character. Variables not in data.frame, only needed if you use variables which are not in the frame. If out.vars is not specified, it is assumed to match all variables starting with a dot ('.').

Details

A set of logical conditions are chained, not and'ed. That is, each argument to chainsubset is used as a filter to create a smaller dataset. Each subsequent argument filters further. For independent conditions this will be the same as and'ing them. I.e. chainsubset(x < 0 , y < 0) will yield the same subset as (x < 0) & (y < 0). However, for aggregate filters like chainsubset(x < mean(y), x > mean(y)) we first find all the observations with x < mean(y), then among these we find the ones with x > mean(y). The last mean(y) is now conditional on x < mean(y).

Value

Expression that can be eval'ed to yield a logical subset mask.

Note

Some trickery is done to make this work directly in the subset argument of functions like felm() and lm(). It might possibly fail with an error message in some situations. If this happens, it should be done in two steps: ss <- eval(chainsubset(...),data); lm(...,data=data, subset=ss). In particular, the arguments are taken literally, constructions like function(...) {chainsubset(...)} or a <- quote(x < y); chainsubset(a) do not work, but do.call(chainsubset,list(a)) does.

Examples

```
set.seed(48)
N <- 10000
dat <- data.frame(y = rnorm(N), x = rnorm(N))
# It's not the same as and'ing the conditions:
felm(y ~ x, data = dat, subset = chainsubset(x < mean(y), y < 2 * mean(x)))
felm(y ~ x, data = dat, subset = chainsubset(y < 2 * mean(x), x < mean(y)))
felm(y ~ x, data = dat, subset = (x < mean(y)) & (y < 2 * mean(x)))
lm(y ~ x, data = dat, subset = chainsubset(x < mean(y), x > mean(y)))
```

---

compfactor *Find the connected components*

Description

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

Usage

```
compfactor(fl, WW = FALSE)
```

Arguments

fl	a list of factors defining the dummies
WW	logical. Use Weeks and Williams components

Details

If there are more than two factors and WW=FALSE, only the first two will be used.

If WW=TRUE and length(fl) > 2, the component structure will be as in "A Note on the Determination of Connectedness in an N-Way Cross Classification" by D.L. Weeks and D.R. Williams, Technometrics, vol 6 no 3, August 1964. I.e. in each component, the coefficients within each factor are comparable, that is, their difference is estimable even if there are more than two factors. That is, one may use one reference in each factor in each component, and interpret the coefficients within a component as usual. This is not an exhaustion of all the estimable functions. There is somewhat more about this in one of the vignettes.

Value

A factor of the same length as the factors in the input argument. It defines the connected components. E.g. nlevels(compfactor(fl)) will yield the number of connected components.

Examples

```
## create two factors
f1 <- factor(sample(300, 400, replace = TRUE))
f2 <- factor(sample(300, 400, replace = TRUE))

## find the components
cf <- compfactor(list(f1 = f1, f2 = f2))

## show the third largest component
fr <- data.frame(f1, f2, cf)
fr[cf == 3, ]
```

---

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, bN = 100L)
```

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.

Note

Please note that `condfstat` does not work with the old syntax for IV in `[felm](...,iv=)`. The new multipart syntax must be used.

References

Sanderson, E. and F. Windmeijer (2014) *A weak instrument F-test in linear IV models with multiple endogenous variables*, Journal of Econometrics, 2015. doi:10.1016/j.jeconom.2015.06.004

Stock, J.H. and M. Yogo (2004) *Testing for weak instruments in linear IV regression*, <doi:10.3386/t0284 > in *Identification and inference for econometric models: Essays in honor of Thomas Rothenberg*, 2005.

Examples

```
z1 <- rnorm(4000)
z2 <- rnorm(length(z1))
u <- rnorm(length(z1))
# make x1, x2 correlated with errors u

x1 <- z1 + z2 + 0.2 * u + rnorm(length(z1))
x2 <- z1 + 0.94 * z2 - 0.3 * u + rnorm(length(z1))
y <- x1 + x2 + u
est <- felm(y ~ 1 | 0 | (x1 | x2 ~ z1 + z2))
summary(est)
## Not run:
summary(est$stage1, lhs = "x1")
summary(est$stage1, lhs = "x2")

## End(Not run)

# the joint significance of the instruments in both the first stages are ok:
t(sapply(est$stage1$lhs, function(lh) waldtest(est$stage1, ~ z1 | z2, lhs = lh)))
# everything above looks fine, t-tests for instruments,
# as well as F-tests for excluded instruments in the 1st stages.
# The conditional F-test reveals that the instruments are jointly weak
# (it's close to being only one instrument, z1+z2, for both x1 and x2)
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. In particular, if one does not need transformations provided by R-formulas but have the covariates present as a matrix or a data.frame, a substantial amount of time can be saved in the centering.

Usage

```
demeanlist(
  mtx,
  fl,
  icpt = 0L,
  eps = getOption("lfe.eps"),
```

```
  threads = getOption("lfe.threads"),
  progress = getOption("lfe.pint"),
  accel = getOption("lfe.accel"),
  randfact = TRUE,
  means = FALSE,
  weights = NULL,
  scale = TRUE,
  na.rm = FALSE,
  attrs = NULL
)
```

Arguments

mtx	matrix whose columns form vectors to be group-centred. mtx can also be a list of vectors or matrices, such as a data frame.
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.
weights	numeric. For weighted demeaning.
scale	logical. Specify scaling for weighted demeaning.
na.rm	logical which indicates what should happen when the data contain NAs. If TRUE, rows in the input mtx are removed prior to centering. If FALSE, they are kept, leading to entire groups becoming NA in the output.
attrs	list. List of attributes which should be attached to the output. Used internally.

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)
}
```

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'` attribute 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 \%*\% solve(chol(crossprod(x)))`.

The `weights` argument is used if a weighted projection is computed. If W is the diagonal matrix with `weights` on the diagonal, `demeanlist` computes $W^{-1}M_{WD}Wx$ where x is the input vector, D is the matrix of dummies from `fl` and M_{WD} is the projection on the orthogonal complement of the range of the matrix WD. It is possible to implement the weighted projection with the `'x'` attribute mentioned above, but it is a separate argument for convenience. If `scale=FALSE`, `demeanlist` computes $M_{WD}x$ without any W scaling. If `length(scale) > 1`, then `scale[1]` specifies whether the input should be scaled by W, and `scale[2]` specifies whether the output should be scaled by W^{-1}. This is just a convenience to save some memory copies in other functions in the package.

Note that for certain large datasets the overhead in `felm()` is large compared to the time spent in `demeanlist`. If the data are present directly without having to use the formula-interface to `felm` for transformations etc, it is possible to run `demeanlist` directly on a matrix or `"data.frame"` and do the OLS "manually", e.g. with something like `cx <- demeanlist(x,...); beta <- solve(crossprod(cx), crossprod(c`

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 a matrix `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 names is returned; the `icpt` argument is ignored.

If `na.rm` is specified, the return value has an attribute `'na.rm'` with a vector of row numbers which has been removed. In case the input is a matrix or list, the same rows are removed from all of them. Note that removing NAs incurs a copy of the input, so if memory usage is an issue and many runs are done, one might consider removing NAs from the data set entirely.

Note

The `accel` argument enables Gearhart-Koshy acceleration as described in Theorem 3.16 by Bauschke, Deutsch, Hundal and Park in "Accelerating the convergence of the method of alternating projections", Trans. Amer. Math. Soc. 355 pp 3433-3461 (2003).

`demeanlist` will use an in place transform to save memory, provided the `mtx` argument is unnamed. Thus, as always in R, you shouldn't use temporary variables like `tmp <- fun(x[v,]); bar <- demeanlist(tmp,...); rm(t` it will be much better to do `bar <- demeanlist(fun(x[v,]),...)`. However, demeanlist allows a

construction like `bar <- demeanlist(unnamed(tmp),...)` which will use an in place transformation, i.e. tmp will be modified, quite contrary to the usual semantics of R.

Examples

```
oldopts <- options("lfe.threads")
options(lfe.threads = 2)
## create a matrix
mtx <- data.frame(matrix(rnorm(999), ncol = 3))
# a list of factors
rgb <- c("red", "green", "blue")
fl <- replicate(4, factor(sample(rgb, nrow(mtx), replace = TRUE)), simplify = FALSE)
names(fl) <- paste("g", seq_along(fl), sep = "")
# centre on all means
mtx0 <- demeanlist(mtx, fl)
head(data.frame(mtx0, fl))
# verify that the group means for the columns are zero
lapply(fl, function(f) apply(mtx0, 2, tapply, f, mean))
options(oldopts)
```

`efactory` *Create estimable function*

Description

Creates an estimable function for a factor-structure.

Usage

```
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(v,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=firm)`.

Examples

```
oldopts <- options("lfe.threads")
options(lfe.threads = 2)
id <- factor(sample(5000, 50000, replace = TRUE))
firm <- factor(sample(3000, 50000, replace = TRUE))
fl <- list(id = id, firm = firm)
obj <- list(fe = fl, cfactor = compfactor(fl))
## the trivial least-norm  transformtion, 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` *Fit a linear model with multiple group fixed effects*

Description

'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.

Usage

```
felm(
  formula,
  data,
  exactDOF = FALSE,
  subset,
  na.action,
  contrasts = NULL,
  weights = 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 `NA`s. 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 `NULL`, no action. `na.exclude` is currently not supported.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.

`weights` an optional vector of weights to be used in the fitting process. Should be 'NULL' or a numeric vector. If non-NULL, weighted least squares is used with weights `weights` (that is, minimizing `sum(w*e^2)`); otherwise ordinary least squares is used.

`...` other arguments.

- `cmethod` character. Which clustering method to use. Known arguments are `'cgm'` (the default), `'cgm2'` (or `'reghdfe'`, its alias). These alternate methods will generally yield equivalent results, except in the case of multiway clustering with few clusters along at least one dimension.
- `keepX` logical. To include a copy of the expanded data matrix in the return value, as needed by `bccorr()` and `fevcov()` for proper limited mobility bias correction.
- `keepCX` logical. Keep a copy of the centred expanded data matrix in the return value. As list elements `cX` for the explanatory variables, and `cY` for the outcome.
- `keepModel` logical. Keep a copy of the model frame.
- `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.
- `psdef` logical. In case of multiway clustering, the method of Cameron, Gelbach and Miller may yield a non-definite variance matrix. Ordinarily this is forced to be semidefinite by setting negative eigenvalues to zero. Setting `psdef=FALSE` will switch off this adjustment. Since the variance estimator is asymptotically correct, this should only have an effect when the clustering factors have very few levels.
- `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 *Kolesar 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 Fuller adjustment of the liml-estimator.
- `Nboot, bootexpr, bootcluster` Since `felm` has quite a bit of overhead in the creation of the model matrix, if one wants confidence intervals for some function of the estimated parameters, it is possible to bootstrap internally in `felm`. That is, the model matrix is resampled `Nboot` times and estimated, and the `bootexpr` is evaluated inside an `sapply`. The estimated coefficients and the left hand side(s) are available by name. Any right hand side variable `x` is available by the name `var.x`. The `"felm"`-object for each estimation is available as `est`. If a `bootcluster` is specified as a factor, entire levels are resampled. `bootcluster` can also be a function with no arguments, it should return a vector of integers, the rows to use in the sample. It can also be the string 'model', in which case the cluster is taken from the model. `bootexpr` should be an expression, e.g. like `quote(x/x2 * abs(x3)/mean(y))`. It could be wise to specify `nostats=TRUE` when bootstrapping, unless the covariance matrices are needed in the bootstrap. If you need the covariance matrices in the full estimate, but not in the bootstrap,

you can specify it in an attribute "boot" as nostats=structure(FALSE, boot=TRUE).

- iv, clustervar deprecated. These arguments will be removed at a later time, but are 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

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().

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 0, 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 ~. Multiple left hand sides like y|w|x ~ x1 + x2 |f1+f2|... 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=c('clu1','clu2'))'. 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. (An exception occurs in the case of clustered standard errors and, specifically, where clusters are nested within fixed effects; see here.) 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**.

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 cmethod argument may affect the clustered covariance matrix (and thus regressor standard errors), either directly or via adjustments to a degrees of freedom scaling factor. In particular, Cameron, Gelbach and Miller (CGM2011, sec. 2.3) describe two possible small cluster corrections that are relevant in the case of multiway clustering.

- The first approach adjusts each component of the cluster-robust variance estimator (CRVE) by its own c_i adjustment factor. For example, the first component (with G clusters) is adjusted by $c_1 = \frac{G}{G-1}\frac{N-1}{N-K}$, the second component (with H clusters) is adjusted by $c_2 = \frac{H}{H-1}\frac{N-1}{N-K}$, etc.
- The second approach applies the same adjustment to all CRVE components: $c = \frac{J}{J-1}\frac{N-1}{N-K}$, where $J = \min(G, H)$ in the case of two-way clustering, for example.

Any differences resulting from these two approaches are likely to be minor, and they will obviously yield exactly the same results when there is only one cluster dimension. Still, CGM2011 adopt the former approach in their own paper and simulations. This is also the default method that felm uses (i.e. cmethod = 'cgm'). However, the latter approach has since been adopted by several other packages that allow for robust inference with multiway clustering. This includes the popular Stata package reghdfe, as well as the FixedEffectModels.jl implementation in Julia. To match results from these packages exactly, use cmethod = 'cgm2' (or its alias, cmethod = 'reghdfe'). It is possible that some residual differences may still remain; see discussion here.

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.
weights	numeric. The square root of the argument weights.

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.
iv1fstat	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.
cX, cY	matrix. The centred expanded data matrix. Only included if felm(keepCX=TRUE).
boot	The result of a replicate applied to the bootexpr (if used).

Note

Side effect: If data is an object of class "pdata.frame" (from the **plm** package), the **plm** namespace is loaded if available, and data is coerced to a "data.frame" with as.data.frame which dispatches to a **plm** method. This ensures that transformations like diff and lag from **plm** works as expected, but it also incurs an additional copy of the data, and the **plm** namespace remains loaded after felm returns. When working with "pdata.frame"s, this is what is usually wanted anyway.

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

Cameron, A.C., J.B. Gelbach and D.L. Miller (2011) *Robust inference with multiway clustering*, Journal of Business & Economic Statistics 29 (2011), no. 2, 238–249. doi:10.1198/jbes.2010.07136

Kolesar, M., R. Chetty, J. Friedman, E. Glaeser, and G.W. Imbens (2014) *Identification and Inference with Many Invalid Instruments*, Journal of Business & Economic Statistics (to appear). doi:10.1080/07350015.2014.978175

See Also

getfe() summary.felm() condfstat() waldtest()

Examples

```
## Default is to use all cores. We'll limit it to 2 for this example.
oldopts <- options("lfe.threads")
options(lfe.threads = 2)

## Simulate data
```

```
set.seed(42)
n <- 1e3

d <- data.frame(
  # Covariates
  x1 = rnorm(n),
  x2 = rnorm(n),
  # Individuals and firms
  id = factor(sample(20, n, replace = TRUE)),
  firm = factor(sample(13, n, replace = TRUE)),
  # Noise
  u = rnorm(n)
)

# Effects for individuals and firms
id.eff <- rnorm(nlevels(d$id))
firm.eff <- rnorm(nlevels(d$firm))

# Left hand side
d$y <- d$x1 + 0.5 * d$x2 + id.eff[d$id] + firm.eff[d$firm] + d$u

## Estimate the model and print the results
est <- felm(y ~ x1 + x2 | id + firm, data = d)
summary(est)
# Compare with lm
summary(lm(y ~ x1 + x2 + id + firm - 1, data = d))

## Example with 'reverse causation' (IV regression)

# Q and W are instrumented by x3 and the factor x4.
d$x3 <- rnorm(n)
d$x4 <- sample(12, n, replace = TRUE)
d$Q <- 0.3 * d$x3 + d$x1 + 0.2 * d$x2 + id.eff[d$id] + 0.3 * log(d$x4) - 0.3 * d$y +
  rnorm(n, sd = 0.3)
d$W <- 0.7 * d$x3 - 2 * d$x1 + 0.1 * d$x2 - 0.7 * id.eff[d$id] + 0.8 * cos(d$x4) -
  0.2 * d$y + rnorm(n, sd = 0.6)

# Add them to the outcome variable
d$y <- d$y + d$Q + d$W

## Estimate the IV model and report robust SEs
ivest <- felm(y ~ x1 + x2 | id + firm | (Q | W ~ x3 + factor(x4)), data = d)
summary(ivest, robust = TRUE)
condfstat(ivest)
# Compare with the not instrumented fit:
summary(felm(y ~ x1 + x2 + Q + W | id + firm, data = d))

## Example with multiway clustering

# Create a large cluster group (500 clusters) and a small one (20 clusters)
d$cl1 <- factor(sample(rep(1:500, length.out = n)))
d$cl2 <- factor(sample(rep(1:20, length.out = n)))
# Function for adding clustered noise to our outcome variable
```

```
cl_noise <- function(cl) {
  obs_per_cluster <- n / nlevels(cl)
  unlist(replicate(nlevels(cl),
    rnorm(obs_per_cluster, mean = rnorm(1), sd = runif(1)),
    simplify = FALSE
  ))
}

# New outcome variable
d$y_cl <- d$x1 + 0.5 * d$x2 + id.eff[d$id] + firm.eff[d$firm] +
  cl_noise(d$cl1) + cl_noise(d$cl2)

## Estimate and print the model with cluster-robust SEs (default)
est_cl <- felm(y_cl ~ x1 + x2 | id + firm | 0 | cl1 + cl2, data = d)
summary(est_cl)

# Print ordinary standard errors:
summary(est_cl, robust = FALSE)
# Match cluster-robust SEs from Stata's reghdfe package:
summary(felm(y_cl ~ x1 + x2 | id + firm | 0 | cl1 + cl2,
  data = d,
  cmethod = "reghdfe"
))

## Restore default options
options(oldopts)
```

`fepois` *Fit a Poisson model with multiple group fixed effects*

Description

Fit a Poisson model with multiple group fixed effects

Usage

```
fepois(
  formula,
  data,
  offset = NULL,
  subset = NULL,
  robust = TRUE,
  cluster = NULL,
  pseudo_rsq = FALSE,
  tol = 1e-10
)
```

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.
offset	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector or matrix of extents matching those of the response. One or more offset terms can be included in the formula instead or as well, and if more than one are specified their sum is used. See model.offset.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
robust	logical value to return a robust standard error computation.
cluster	optional variable to group by and compute sandwich-type robust standard errors. Should be a formula of the form ~x_j or an object that be coerced to a formula.
pseudo_rsq	logical value to return a a pseudo-R2 based on Kendall's correlation.
tol	tolerance value for GLM convergence criteria.

See Also

felm

---

fevcov *Compute limited mobility bias corrected covariance matrix between fixed effects*

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 variances $var(D\theta)$, $var(F\psi)$ and covariances $cov(D\theta, F\psi)$. However, if we use estimates for θ and ψ, the resulting variances are biased. The function fevcov computes a bias corrected covariance matrix as described in *Gaure (2014)*.

Usage

```
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 useable. 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().

In the case that the felm()-estimation has weights, it is the weighted variances and covariance which are bias corrected.

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

Bias correction for IV-estimates are not supported as of now.

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.

If there are only two fixed effects, fevcov returns the same information as bccorr(), though in a slightly different format.

References

Gaure, S. (2014), *Correlation bias correction in two-way fixed-effects linear regression*, Stat 3(1):379-390, 2014. doi:10.1002/sta4.68

See Also

varvars() bccorr()

Examples

```
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)
z <- x + 0.5 * x2 + 0.7 * id.m + 0.5 * firm.m + 0.3 * foo.m + rnorm(length(x), sd = 2)
# make a data frame
fr <- data.frame(y, z, x, x2, id, firm, foo)
## estimate and print result
est <- felm(y | z ~ x + x2 | id + firm + foo, data = fr, keepX = TRUE)
# find bias corrections, there's little bias in this example
print(yv <- fevcov(est, lhs = "y"))
## Here's how to compute the unbiased correlation matrix:
cm <- cov2cor(yv)
structure(cm, bias = NULL)
```

---

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

```
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', 'cg', 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.
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")
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))
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, 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[[1]]],
    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(oldopts)
## 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

```
is.estimable(
  ef,
  fe,
  R = NULL,
  nowarn = FALSE,
  keepdiff = FALSE,
  threshold = 500 * getOption("lfe.eps")
)
```

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")
options(lfe.threads = 2)
## 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(
  fl,
  R,
  eps = getOption("lfe.eps"),
  init = NULL,
  threads = getOption("lfe.threads")
)
```

Arguments

fl	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 fl, 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 more general conjugate gradient method of cgsolve(). This may or may not be faster.

See Also

cgsolve()

Examples

```
## create factors
f1 <- factor(sample(24000, 100000, 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 <- makeDmatrix(list(f1, f2, f3, f4))
dim(D)
## an x
truex <- runif(ncol(D))
## and the right hand side
R <- as.vector(D %*% truex)
## solve it
sol <- kaczmarz(list(f1, f2, f3, f4), R)
## verify that the solution solves the system Dx = R
sqrt(sum((D %*% sol - R)^2))
## but the solution is not equal to the true x, because the system is
## underdetermined
sqrt(sum((sol - truex)^2))
## moreover, the solution from kaczmarz has smaller norm
sqrt(sum(sol^2)) < sqrt(sum(truex^2))
```

---

makeDmatrix *Make sparse matrix of dummies from factor list*

Description

Given a list of factors, return the matrix of dummies as a sparse matrix.

Usage

```
makeDmatrix(fl, weights = NULL)
```

Arguments

fl	list of factors.
weights	numeric vector. Multiplied into the rows.

Details

The function returns the model matrix for a list of factors. This matrix is not used internally by the package, but it's used in some of the documentation for illustrative purposes.

Value

Returns a sparse matrix.

Examples

```
fl <- lapply(1:3, function(i) factor(sample(3, 10, replace = TRUE)))
fl
makeDmatrix(fl, weights = seq(0.1, 1, 0.1))
```

`mctrace` *Compute trace of a large matrix by sample means*

Description

Some matrices are too large to be represented as a matrix, even as a sparse matrix. Nevertheless, it can be possible to compute the matrix vector product fairly easy, and this is utilized to estimate the trace of the matrix.

Usage

```
mctrace(mat, N, tol = 0.001, maxsamples = Inf, trname = "", init)
```

Arguments

`mat`	square matrix, Matrix, function or list of factors.
`N`	integer. if `mat` is a function, the size of the matrix is specified here.
`tol`	numeric. Tolerance.
`maxsamples`	numeric. Maximum number of samples in the expectation estimation.
`trname`	character. Arbitrary name used in progress reports.
`init`	numeric. Initial guess for the trace.

Details

`mctrace` is used internally by `fevcov()` and `bccorr()`, but has been made public since it might be useful for other tasks as well.

For any matrix A, the trace equals the sum of the diagonal elements, or the sum of the eigenvalues. However, if the size of the matrix is very large, we may not have a matrix representation, so the diagonal is not immediately available. In that case we can use the formula $tr(A) = E(x^t Ax)$ where x is a random vector with zero expectation and $Var(x) = I$. We estimate the expectation with sample means. `mctrace` draws x in $\{-1, 1\}^N$, and evaluates `mat` on these vectors.

If `mat` is a function, it must be able to take a matrix of column vectors as input. Since $x^t Ax = (Ax, x)$ is evaluated, where $(\cdot, \cdot)$ is the Euclidean inner product, the function `mat` can perform this inner product itself. In that case the function should have an attribute `attr(mat,'IP') <- TRUE` to signal this.

If `mat` is a list of factors, the matrix for which to estimate the trace, is the projection matrix which projects out the factors. I.e. how many dimensions are left when the factors have been projected out. Thus, it is possible to estimate the degrees of freedom in an OLS where factors are projected out.

The tolerance `tol` is a relative tolerance. The iteration terminates when the normalized standard deviation of the sample mean (s.d. divided by absolute value of the current sample mean) goes below `tol`. Specify a negative `tol` to use the absolute standard deviation. The tolerance can also change during the iterations; you can specify `tol=function(curest) {...}` and return a tolerance based on the current estimate of the trace (i.e. the current sample mean).

Value

An estimate of the trace of the matrix represented by `mat` is returned.

Examples

```
options(lfe.threads = 1, digits = 5, warn = 1)
f1 <- factor(sample(1500, 3000, replace = TRUE))
f2 <- factor(sample(1500, 3000, replace = TRUE))
fl <- list(f1, f2)
mctrace(fl, tol = -5)
# exact:
# length(f1) - nlevels(f1) - nlevels(f2) + nlevels(compfactor(fl))
```

---

nlexpect *Compute expectation of a function of the coefficients.*

Description

Use integration of the joint distribution of the coefficients to compute the expectation of some function of the coefficients. Can be used for non-linear inference tests.

Usage

```
nlexpect(
  est,
  fun,
  coefs,
  ...,
  tol = getOption("lfe.etol"),
  lhs = NULL,
  cv,
  istats = FALSE,
  flags = list(verbose = 0),
  max.eval = 200000L,
  method = c("hcubature", "pcubature", "cuhre", "suave", "vegas", "divonne"),
  vectorize = FALSE
)
```

Arguments

est	object of class "felm" or "lm", a result of a call to felm() or lm.
fun	function of coefficients to be integrated. Can also be a quoted expression.
coefs	character. Names of coefficients to test. Only needed if fun is a function.
...	other arguments passed to fun or the integration routine.
tol	numeric. Tolerance for the computed integral.
lhs	character. Name of the left hand side, if est has more than one.
cv	Covariance matrix to use in place of vcov(est)
istats	logical. Should convergence information from the integration routine be included as attributes?
flags	list. Additional flags for the underlying integration routine. Not used after the package **R2Cuba** disappeared.
max.eval	integer. Maximum number of integral evaluations.
method	character. A method specification usable by cubature::cubintegrate. The documentation there says that "pcubature" is good for smooth integrands of low dimensions.
vectorize	logical or numeric. Use vectorized function evaluation from package **cubature**. This can speed up integration significantly. If method is from the Cuba library (i.e. not pcubature or hcubature), vectorize should be specified as a numeric, a vectorization factor. The default is 128.

Details

The function nlexpect integrates the function fun(x) over the multivariate normal distribution specified by the point estimates and the covariance matrix vcov(est). This is the expectation of fun(beta) if we were to bootstrap the data (e.g. by drawing the residuals anew) and do repeated estimations.

The list of coefficients used by fun must be specified in coefs.

If the function is simple, it can be specified as a quoted expression like quote(a*b+log(abs(d))). In this case, if coefs is not specified, it will be set to the list of all the variables occurring in the expression which are also names of coefficients.

fun may return a vector of values, in which case a vector of expectations is computed, like quote(c(a*b, a^3-b)). However, if the expressions contain different variables, like quote(c(a*b, d*e)), a quite compute intensive 4-dimensional integral will be computed, compared to two cheap 2-dimensional integrals if you do them separately. There is nothing to gain from using vector-valued functions compared to multiple calls to nlexpect().

You may of course also integrate inequalities like quote(abs(x1-0.2) > 0.2) to simulate the probability from t-tests or Wald-tests. See the examples.

The function you provide will get an argument ... if it does not have one already. It will also be passed an argument .z which contains the actual coefficients in normalized coordinates, i.e. if ch is the Cholesky decomposition of the covariance matrix, and pt are the point estimates, the coefficients will be pt + ch \%*\% .z. The first argument is a vector with names corresponding to the coefficients.

If you specify `vectorized=TRUE`, your function will be passed a list with vectors in its first argument. The function must be able to handle a list, and must return a vector of the same length as the vectors in the list. If you pass an expression like `x < y`, each variable will be a vector. If your function is vector valued, it must return a matrix where each column is the values.

The `tol` argument specifies both the relative tolerance and the absolute tolerance. If these should not be the same, specify `tol` as a vector of length 2. The first value is the relative tolerance, the second is the absolute tolerance. Termination occurs when at least one of the tolerances is met.

The `...` can be used for passing other arguments to the integration routine `cubature::cubintegrate` and the function to be integrated.

Value

The function `nlexpect` computes and returns the expectation of the function `fun(beta)`, with `beta` a vector of coefficients. I.e., if the coefficients `beta` are bootstrapped a large number of times, `nlexpect(est, fun)` should be equal to `mean(fun(beta))`.

Note

An alternative to this method is to use the `bootexpr` argument with `felm()`, to do a Monte Carlo integration.

See Also

`waldtest()`

Examples

```
N <- 100
x1 <- rnorm(N)
# make some correlation
x2 <- 0.1 * rnorm(N) + 0.1 * x1
y <- 0.1 * x1 + x2 + rnorm(N)
summary(est <- felm(y ~ x1 + x2))
pt1 <- coef(est)["x1"]
pt2 <- coef(est)["x2"]
# expected values of coefficients, should match the summary
# and variance, i.e. square of standard errors in the summary
nlexpect(est, quote(c(x1 = x1, x2 = x2, var = c((x1 - pt1)^2, (x2 - pt2)^2))))

# the covariance matrix:
nlexpect(est, tcrossprod(as.matrix(c(x1 - pt1, x2 - pt2))))

# Wald test of single variable
waldtest(est, ~x1)["p.F"]
# the same with nlexpect, i.e. probability for observing abs(x1)>abs(pt1) conditional
# on E(x1) = 0.
nlexpect(est, (x1 - pt1)^2 > pt1^2, tol = 1e-7, vectorize = TRUE)
# which is the same as
2 * nlexpect(est, x1 * sign(pt1) < 0)

# Here's a multivalued, vectorized example
```

```
nlexpect(est, rbind(a = x1 * x2 < pt1, b = x1 * x2 > 0), vectorize = TRUE, method = "hcubature")


# Non-linear test:

# A simple one, what's the probability that product x1*x2 is between 0 and |E(x1)|?
nlexpect(est, x1 * x2 > 0 & x1 * x2 < abs(pt1), vectorize = TRUE, method = "hcubature")
# Then a more complicated one with the expected value of a polynomal in the coefficients
f <- function(x) c(poly = x[["x1"]] * (6 * x[["x1"]] - x[["x2"]]^2))
# This is the linearized test:
waldtest(est, f)["p.F"]
# In general, for a function f, the non-linear Wald test is something like
# the following:
# expected value of function
Ef <- nlexpect(est, f, coefs = c("x1", "x2"))
# point value of function
Pf <- f(c(pt1, pt2))
# similar to a Wald test, but non-linear:
nlexpect(est, function(x) (f(x) - Ef)^2 > Pf^2, c("x1", "x2"), vectorize = TRUE)
# one-sided
nlexpect(est, function(x) f(x) - Ef > abs(Pf), c("x1", "x2"), vectorize = TRUE)
# other sided
nlexpect(est, function(x) f(x) - Ef < -abs(Pf), c("x1", "x2"), vectorize = TRUE)
```

---

sargan *Compute Sargan's S*

Description

Compute Sargan's S

Usage

```
sargan(object, ..., lhs = object$lhs[1])
```

Arguments

object	and object type '"felm"', the return value from `felm()`.
...	Not used at the moment.
lhs	in case of multiple left hand sides, specify the name of the left hand side for which you want to compute Sargan's S.

Value

sargan returns a numeric, the Sargan's S. The Basmann statistic is returned in the '"basmann"' attribute.

summary.felm *Summarize felm model fits*

Description

summary method for class "felm".

Usage

```
## S3 method for class 'felm'
summary(
  object,
  ...,
  robust = !is.null(object$clustervar) || getOption("lfe.robust"),
  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 en "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()
fe	list of factors. A list of the terms in the second part of the model.

lhs.	character. If object is the result of an estimation with multiple left hand sides, the actual argument lhs will be copied to this field.
iv1fstat	F-statistic for excluded instruments in 1. step IV, see felm().
iv1pval	P-value for iv1fstat.

Note

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()

---

varvars *Compute the variance of the fixed effect variance estimate*

Description

Compute the variance of the fixed effect variance estimate

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

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.

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

Gaure, S. (2014), *Correlation bias correction in two-way fixed-effects linear regression*, Stat 3(1):379-390, 2014.

See Also

`bccorr()` `fevcov()`

Examples

```
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
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 joint restrictions on coefficients*

Description

Compute a Wald test for a linear hypothesis on the coefficients. Also supports Delta-approximation for non-linear hypotheses.

Usage

```
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, function, 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.

If R is a function (of the coefficients), an approximate Wald test against H0: R(beta) == 0, using the Delta-method, is computed.

In case of an IV-estimation, the names for the endogenous variables in coef(object) are of the type "Q(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(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
- chi2 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.

See Also

nlexpect()

Examples

```
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)`)
# The Delta-method coincides when the function is linear:
waldtest(est, function(x) x - c(0, 1, -0.2))
```

Index

∗ **Limited Mobility Bias**
    bccorr, 8
∗ **models**
    getfe, 31
    lfe-package, 2
∗ **regression**
    getfe, 31
    lfe-package, 2
..oldfelm, 6

bccorr, 8
bccorr(), *3*, *7*, *12*, *23*, *26*, *30, 31*, *38*, *45*
btrap, 10
btrap(), *11*

cgsolve, 12
cgsolve(), *36*
chainsubset, 13
compfactor, 14
compfactor(), *3*
condfstat, 15
condfstat(), *3*, *26*

demeanlist, 17
demeanlist(), *3*

efactory, 20
efactory(), *30*, *32*

felm, 22
felm(), *3, 4*, *9, 10*, *16, 17*, *19, 20*, *30–33*, *40–42*, *44–47*
fepois, 28
fevcov, 29
fevcov(), *3*, *7*, *9*, *12*, *23*, *26*, *38*, *45*

getfe, 31
getfe(), *3, 4*, *8*, *10, 11*, *26*, *30*, *34–36*, *44*

is.estimable, 34

kaczmarz, 36
kaczmarz(), *13*

lfe *(*lfe-package*)*, 2
lfe-package, 2
lm(), *20*, *24*

makeDmatrix, 37
mctrace, 38
model.offset, *29*

nlexpect, 39
nlexpect(), *47*

offset, *29*

sargan, 42
stats::lm(), *3*
summary.felm, 43
summary.felm(), *26*

varvars, 44
varvars(), *31*

waldtest, 46
waldtest(), *3*, *16*, *26*, *41*, *43, 44*