Package 'fixest'

November 23, 2019

Type Package

Title Fast Fixed-Effects Estimations

Version 0.2.1

Imports stats, graphics, utils, Formula, MASS, numDeriv, nlme, Rcpp

Suggests knitr, rmarkdown, data.table

LinkingTo Rcpp

Depends R(>= 3.1.0)

Description Fast and user-friendly estimation of econometric models with multiple fixed-effects. Includes ordinary least squares (OLS), generalized linear models (GLM) and the negative binomial. The core of the package is based on optimized parallel C++ code, scaling especially well for large data sets. The method to obtain the fixed-effects coefficients is based on Berge (2018) <https://wwwen.uni.lu/content/download/110162/1299525/file/2018_13>. Further provides tools to export and view the results of several estimations with intuitive design to cluster the standard-errors.

License GPL-3

BugReports https://github.com/lrberge/fixest/issues

SystemRequirements C++11

VignetteBuilder knitr

LazyData true

RoxygenNote 6.1.1

Encoding UTF-8

NeedsCompilation yes

Author Laurent Berge [aut, cre]

Maintainer Laurent Berge <laurent.berge@uni.lu>

Repository CRAN

Date/Publication 2019-11-23 09:40:02 UTC

R topics documented:

`fixest-package` *Fast and User-Friendly Fixed-Effects Estimations*

Description

The package **fixest** provides a family of functions to perform estimations with multiple fixed-effects. Standard-errors can be easily and intuitively clustered. It also includes tools to seamlessly export the results of various estimations.

Details

This package efficiently estimates models with multiple fixed-effect (i.e. multiple large factor variables).

The core functions are: `feols`, `feglm` and `femlm` to estimate, respectively, linear models, generalized linear models and maximum likelihood models with multiple fixed-effects. The function `feNmlm` allows the inclusion of non-linear in parameters right hand sides. Finally `fepois` and `fenegbin` are shorthands to estimate Poisson and Negative Binomial models.

Note that the functions `feglm` and `femlm` provide the same results when using the same families but differ in that the latter is a direct maximum likelihood optimization (so the two can really have different convergence rate).

Several features are also included such as the possibility to easily compute different types of standard-errors (including multi-way clustering).

It is possible to compare the results of several estimations by using the function `esttable`, and to export them to Latex using `esttex`.

Author(s)

Maintainer: Laurent Berge <laurent.berge@uni.lu>

References

Berg\'e, Laurent, 2018, "Efficient estimation of maximum likelihood models with multiple fixed-effects: the R package FENmlm." CREA Discussion Papers, 13 (https://wwwen.uni.lu/content/download/110162/1299525/file/2018_13).

See Also

Useful links:

- Report bugs at https://github.com/lrberge/fixest/issues

`AIC.fixest` *Aikake's an information criterion*

Description

This function computes the AIC (Aikake's, an information criterion) from a `fixest` estimation.

Usage

```
## S3 method for class 'fixest'
AIC(object, ..., k = 2)
```

Arguments

`object`	A `fixest` object. Obtained using the functions `femlm`, `feols` or `feglm`.
`...`	Optionally, more fitted objects.
`k`	A numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC (i.e. `AIC=-2*LL+k*nparams`).

Details

The AIC is computed as:

$$AIC = -2 \times LogLikelihood + k \times nbParams$$

with k the penalty parameter.

You can have more information on this criterion on `AIC`.

Value

It return a numeric vector, with length the same as the number of objects taken as arguments.

Author(s)

Laurent Berge

See Also

See also the main estimation functions `femlm`, `feols` or `feglm`. Other statictics methods: `BIC.fixest`, `logLik.fixest`, `nobs.fixest`.

Examples

```
# two fitted models with different expl. variables:
res1 = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
             Petal.Width | Species, iris)
res2 = femlm(Sepal.Length ~ Petal.Width | Species, iris)

AIC(res1, res2)
BIC(res1, res2)
```

`base_did` *Sample data for difference in difference*

Description

This data has been generated to illustrate the use of difference in difference functions in package **fixest**. This is a balanced panel of 104 individuals and 10 periods. About half the individuals are treated, the treatment having a positive effect on the dependent variable `y` after the 5th period. The effect of the treatment on `y` is gradual.

Usage

```
data(base_did)
```

Format

`base_did` is a data frame with 1,040 observations and 6 variables named `y`, `x1`, `id`, `period`, `post` and `treat`.

- y: The dependent variable affected by the treatment.
- x1: An explanatory variable.
- id: Identifier of the individual.
- period: From 1 to 10
- post: Indicator taking value 1 if the period is strictly greater than 5, 0 otherwise.
- treat: Indicator taking value 1 if the individual is treated, 0 otherwise.

Source

This data has been generated from **R**.

See Also

The DiD functions of the package **fixest** are `did_estimate_yearly_effects` and `did_plot_yearly_effects`.

`BIC.fixest` *Bayesian information criterion*

Description

This function computes the BIC (Bayesian information criterion) from a `fixest` estimation.

Usage

```
## S3 method for class 'fixest'
BIC(object, ...)
```

Arguments

`object`	A `fixest` object. Obtained using the functions `femlm`, `feols` or `feglm`.
`...`	Optionally, more fitted objects.

Details

The BIC is computed as follows:

$$BIC = -2 \times LogLikelihood + \log(nobs) \times nbParams$$

with k the penalty parameter.

You can have more information on this criterion on `AIC`.

Value

It return a numeric vector, with length the same as the number of objects taken as arguments.

Author(s)

Laurent Berge

See Also

See also the main estimation functions `femlm`, `feols` or `feglm`. Other statistics functions: `AIC.fixest`, `logLik.fixest`.

Examples

```
# two fitted models with different expl. variables:
res1 = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
             Petal.Width | Species, iris)
res2 = femlm(Sepal.Length ~ Petal.Width | Species, iris)

AIC(res1, res2)
BIC(res1, res2)
```

`coef.fixest` *Extracts the coefficients from a* `fixest` *estimation*

Description

This function extracts the coefficients obtained from a model estimated with `femlm`, `feols` or `feglm`.

Usage

```
## S3 method for class 'fixest'
coef(object, ...)

## S3 method for class 'fixest'
coefficients(object, ...)
```

Arguments

`object`	A `fixest` object. Obtained using the functions `femlm`, `feols` or `feglm`.
`...`	Not currently used.

Details

The coefficients are the ones that have been found to maximize the log-likelihood of the specified model. More information can be found on the models from the estimations help pages: `femlm`, `feols` or `feglm`.

Note that if the model has been estimated with clusters, to obtain the cluster coefficients, you need to use the function `fixef.fixest`.

Value

This function returns a named numeric vector.

Author(s)

Laurent Berge

See Also

See also the main estimation functions `femlm`, `feols` or `feglm`. `summary.fixest`, `confint.fixest`, `vcov.fixest`, `esttable`, `esttex`, `fixef.fixest`.

Examples

```
# simple estimation on iris data, clustering by "Species"
res = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
            Petal.Width | Species, iris)

# the coefficients of the variables:
coef(res)

# the cluster coefficients:
fixef(res)
```

`collinearity` *Collinearity diagnostics for* `fixest` *objects*

Description

In some occasions, the optimization algorithm of `femlm` may fail to converge, or the variance-covariance matrix may not be available. The most common reason of why this happens is collinearity among variables. This function helps to find out which set of variables is problematic.

Usage

```
collinearity(x, verbose)
```

Arguments

`x`	A `fixest` object obtained from, e.g. functions `femlm`, `feols` or `feglm`.
`verbose`	An integer. If higher than or equal to 1, then a note is prompted at each step of the algorithm. By default `verbose = 0` for small problems and to 1 for large problems.

Details

This function tests: 1) collinearity with the cluster variables, 2) perfect multi-collinearity between the variables, 4) perfect multi-collinearity between several variables and the clusters, and 4) identification issues when there are non-linear in parameters parts.

Value

It returns a text message with the identified diagnostics.

Author(s)

Laurent Berge

Examples

```
# Creating an example data base:
cluster_1 = sample(3, 100, TRUE)
cluster_2 = sample(20, 100, TRUE)
x = rnorm(100, cluster_1)**2
y = rnorm(100, cluster_2)**2
z = rnorm(100, 3)**2
dep = rpois(100, x*y*z)
base = data.frame(cluster_1, cluster_2, x, y, z, dep)

# creating collinearity problems:
base$v1 = base$v2 = base$v3 = base$v4 = 0
base$v1[base$cluster_1 == 1] = 1
base$v2[base$cluster_1 == 2] = 1
base$v3[base$cluster_1 == 3] = 1
base$v4[base$cluster_2 == 1] = 1

# Estimations:

# Collinearity with the cluster variables:
res_1 = femlm(dep ~ log(x) + v1 + v2 + v4 | cluster_1 + cluster_2, base)
collinearity(res_1)
# => collinearity with cluster identified, we drop v1 and v2
res_1bis = femlm(dep ~ log(x) + v4 | cluster_1 + cluster_2, base)
collinearity(res_1bis)

# Multi-Collinearity:
res_2 =  femlm(dep ~ log(x) + v1 + v2 + v3 + v4, base)
collinearity(res_2)

# In non-linear part:
res_3 = feNmlm(dep ~ log(z), base, NL.fml = ~log(a*x + b*y),
               NL.start = list(a=1, b=1), lower = list(a=0, b=0))
collinearity(res_3)
```

`confint.fixest` *Confidence interval for parameters estimated with* `fixest`

Description

This function computes the confidence interval of parameter estimates obtained from a model estimated with `femlm`, `feols` or `feglm`.

Usage

```
## S3 method for class 'fixest'
confint(object, parm, level = 0.95, se, cluster,
  dof = TRUE, ...)
```

Arguments

object	A fixest object. Obtained using the functions femlm, feols or feglm.
parm	The parameters for which to compute the confidence interval (either an integer vector OR a character vector with the parameter name). If missing, all parameters are used.
level	The confidence level. Default is 0.95.
se	Character scalar. Which kind of standard error should be computed: "standard", "White", "cluster", "twoway", "threeway" or "fourway"? By default if there are clusters in the estimation: se = "cluster", otherwise se = "standard". Note that this argument can be implicitly deduced from the argument cluster.
cluster	Tells how to cluster the standard-errors (if clustering is requested). Can be either a list of vectors, a character vector of variable names, a formula or an integer vector. Assume we want to perform 2-way clustering over var1 and var2 contained in the data.frame base used for the estimation. All the following cluster arguments are valid and do the same thing: cluster = base[,c("var1","var2")]},\code{cluster = c("var1","var2"), cluster = ~var1+var2. If the two variables were used as clusters in the estimation, you could further use cluster = 1:2 or leave it blank with se = "twoway" (assuming var1 [resp. var2] was the 1st [res. 2nd] cluster).
dof	Logical, default is TRUE. Should there be a degree of freedom correction to the standard errors of the coefficients?
...	Not currently used.

Value

Returns a data.frame with two columns giving respectively the lower and upper bound of the confidence interval. There is as many rows as parameters.

Author(s)

Laurent Berge

Examples

```
# Load trade data
data(trade)

# We estimate the effect of distance on trade (with 3 cluster effects)
est_pois = femlm(Euros ~ log(dist_km) + log(Year) | Origin + Destination +
                 Product, trade)

# confidence interval with "normal" VCOV
```

```
confint(est_pois)

# confidence interval with "clustered" VCOV (w.r.t. the Origin factor)
confint(est_pois, se = "cluster")
```

`did_estimate_yearly_effects` *Estimates yearly treatment effects*

Description

This facility helps to estimate yearly treatment effects in a difference-in-difference setup without having to manually make the estimation. It is made as general as possible such that non-`fixest` estimation functions can also be used.

Usage

```
did_estimate_yearly_effects(fml, data, treat_time, reference,
  returnData = FALSE, ..., estfun = feols)
```

Arguments

`fml`	A formula containing the variables WITHOUT the yearly treatment effects (which will be added by this function).
`data`	A `data.frame` containing all the variables.
`treat_time`	Either a character vector of length two containing the name of the treatment variable and the name of the time variable (e.g. `c("treat","year")`). Either a one-sided formula containing the treatment and the time (e.g. `~treat~year`).
`reference`	The time period of reference. It should be a numeric scalar. The treatment will not be included for this time period so that it serves as reference.
`returnData`	Logical, default is `FALSE`. If `TRUE`, then the original database with the yearly treatment variables is returned.
`...`	Other arguments to be passed to `estfun`, the estimation function.
`estfun`	The estimation function. Defaults to `feols`.

Value

It returns an estimation object. In case of `fixest` estimations, it will return a `fixest` object.

Author(s)

Laurent Berge

See Also

did_plot_yearly_effects, errbar.

Examples

```
# Sample data illustrating the DiD
data(base_did)

# Estimation of yearly effect (they are automatically added)
est = did_estimate_yearly_effects(y ~ x1 + treat + post, base_did,
                                  treat_time = ~treat+period, reference = 5)

# Now we plot the results
did_plot_yearly_effects(est)

# Now with fixed-effects:
est_fe = did_estimate_yearly_effects(y ~ x1 | id + period, base_did,
                                     treat_time = ~treat+period, reference = 5)
did_plot_yearly_effects(est_fe)

# you can change the type of SE to be plotted:
did_plot_yearly_effects(est_fe, se = "cluster") # default
did_plot_yearly_effects(est_fe, se = "standard")
```

---

did_means *Treated and control sample descriptives*

Description

This function shows the means and standard-deviations of several variables conditional on whether they are from the treated or the control group. The groups can further be split according to a pre/post variable. Results can be seamlessly be exported to Latex.

Usage

```
did_means(fml, base, treat_var, post_var, tex = FALSE, treat_dict,
  dict = getFixest_dict(), file, replace = FALSE, title, label,
  raw = FALSE, indiv, treat_first, prepostnames = c("Before", "After"),
  diff.inv = FALSE)
```

Arguments

fml	Either a formula of the type var1 + ... + var[N] ~ treat or var1 + ... + var[N] ~ treat | post. Either a data.frame/matrix containing all the variables for which the means are to be computed (they must be numeric of course). Both the treatment and the post variables must contain only exactly two values. You can use a point to select all the variables of the data set: . ~ treat.

base	A data base containing all the variables in the formula fml.
treat_var	Only if argument fml is *not* a formula. The vector identifying the treated and the control observations (the vector can be of any type but must contain only two possible values). Must be of the same length as the data.
post_var	Only if argument fml is *not* a formula. The vector identifying the periods (pre/post) of the observations (the vector can be of any type but must contain only two possible values). The first value (in the sorted sense) of the vector is taken as the pre period. Must be of the same length as the data.
tex	Should the result be displayed in Latex? Default is FALSE. Automatically set to TRUE if the table is to be saved in a file using the argument file.
treat_dict	A character vector of length two. What are the names of the treated and the control? This should be a dictionnary: e.g. c("1"="Treated","0" = "Control").
dict	A named character vector. A dictionnary between the variables names and an alias. For instance dict=c("x"="Inflation Rate") would replace the variable name x by "Inflation Rate".
file	A file path. If given, the table is written in Latex into this file.
replace	Default is TRUE, which means that when the table is exported, the existing file is not erased.
title	Character string giving the Latex title of the table. (Only if exported.)
label	Character string giving the Latex label of the table. (Only if exported.)
raw	Logical, default is FALSE. If TRUE, it returns the information without formatting.
indiv	Either the variable name of individual identifiers, a one sided formula, or a vector. If the data is that of a panel, this can be used to track the number of individuals per group.
treat_first	Which value of the 'treatment' vector should appear on the left? By default the max value appears first (e.g. if the treatment variable is a 0/1 vector, 1 appears first).
prepostnames	Only if there is a 'post' variable. The names of the pre and post periods to be displayed in Latex. Default is c("Before","After").
diff.inv	Logical, default to FALSE. Whether to inverse the difference.

Details

By default, when the user tries to apply this function to nun-numeric variables, an error is raised. The exception is when the all variables are selected with the dot (like in . ~ treat. In this case, non-numeric variables are automatically omitted (with a message).

NAs are removed automatically: if the data contains NAs an information message will be prompted. First all observations containing NAs relating to the treatment or post variables are removed. Then if there are still NAs for the variables, they are excluded separately for each variable, and a new message detailing the NA breakup is prompted.

Value

It returns a data.frame or a Latex table with the conditional means and statistical differences between the groups.

Examples

```
# Playing around with the DiD data
data(base_did)

# means of treat/control
did_means(y+x1+period~treat, base_did)

# same but inverting the difference
did_means(y+x1+period~treat, base_did, diff.inv = TRUE)

# now treat/control, before/after
did_means(y+x1+period~treat|post, base_did)

# same but with a new line giving the number of unique "indiv" for each case
did_means(y+x1+period~treat|post, base_did, indiv = "id")

# same but with the treat case "0" coming first
did_means(y+x1+period~treat|post, base_did, indiv = ~id, treat_first = 0)

# Selecting all the variables with "."
did_means(.~treat|post, base_did, indiv = "id")
```

---

did_plot_yearly_effects

Plots the results of yearly treatment effects estimation

Description

This function plots the results of a `did_estimate_yearly_effects` estimation.

Usage

```
did_plot_yearly_effects(object, x.shift = 0, w = 0.1,
  ci_level = 0.95, style = c("bar", "interval", "tube"), add = FALSE,
  col = 1, bar.col = col, bar.lwd = par("lwd"), bar.lty,
  grid = TRUE, grid.par = list(lty = 1), bar.join = TRUE, ...)
```

Arguments

`object`	An object returned by the function `did_estimate_yearly_effects`.
`x.shift`	Shifts the confidence intervals bars to the left or right, depending on the value of `x.shift`. Default is 0.
`w`	The width of the confidence intervals.
`ci_level`	Scalar between 0 and 1: the level of the CI. By default it is equal to 0.95.

style	One of "bar" (default), "interval" or "tube". The style of the plot.
add	Default is FALSE, if the intervals are to be added to an existing graph. Note that if it is the case, then the argument x MUST be numeric.
col	Color of the point estimate and of the line joining them (if style = "interval").
bar.col	Color of the bars of the confidence interval. Defaults to col.
bar.lwd	Line width of the confidence intervals, defaults to 1.
bar.lty	Line type of the confidence intervals, defaults to 1 for style = "bar" and to 2 for style = "interval".
grid	Whether to add an horizontal grid. Default is TRUE.
grid.par	Graphical parameters used when plotting the grid in the background. Default is list(lty=1).
bar.join	Logical, default is FALSE. Whether to join the dots when style = "bar".
...	Any other argument to be passed to summary or to plot.

Author(s)

Laurent Berge

See Also

did_estimate_yearly_effects, errbar.

Examples

```
# Sample data illustrating the DiD
data(base_did)

# Estimation of yearly effect (they are automatically added)
est = did_estimate_yearly_effects(y ~ x1 + treat + post, base_did,
                                  treat_time = ~treat+period, reference = 5)

# Now we plot the results
did_plot_yearly_effects(est)

# Now with fixed-effects:
est_fe = did_estimate_yearly_effects(y ~ x1 | id + period, base_did,
                                     treat_time = ~treat+period, reference = 5)
did_plot_yearly_effects(est_fe)

# you can change the type of SE to be plotted:
did_plot_yearly_effects(est_fe, se = "cluster") # default
did_plot_yearly_effects(est_fe, se = "standard")
```

errbar *Plots confidence intervals*

Description

This function draws confidence intervals in a graph.

Usage

```
errbar(estimate, sd, ci_low, ci_top, x, x.shift = 0, w = 0.1,
  ci_level = 0.95, style = c("bar", "interval", "tube"), add = FALSE,
  col = 1, bar.col = col, bar.lwd = par("lwd"), bar.lty,
  grid = TRUE, grid.par = list(lty = 1), bar.join = FALSE,
  only.params = FALSE, ...)
```

Arguments

estimate	Numeric vector. The point estimates.
sd	The standard errors of the estimates. It may be missing.
ci_low	If sd is not provided, the lower bound of the confidence interval. For each estimate.
ci_top	If sd is not provided, the upper bound of the confidence interval. For each estimate.
x	The value of the x-axis. If missing, the names of the argument estimate is used.
x.shift	Shifts the confidence intervals bars to the left or right, depending on the value of x.shift. Default is 0.
w	The width of the confidence intervals.
ci_level	Scalar between 0 and 1: the level of the CI. By default it is equal to 0.95.
style	If "interval": it plots a confidence interval. If "bar", it plots simply error bars. If "tube": as interval, but with a grayed area.
add	Default is FALSE, if the intervals are to be added to an existing graph. Note that if it is the case, then the argument x MUST be numeric.
col	Color of the point estimate and of the line joining them (if style = "interval").
bar.col	Color of the bars of the confidence interval. Defaults to col.
bar.lwd	Line width of the confidence intervals, defaults to 1.
bar.lty	Line type of the confidence intervals, defaults to 1 for style = "bar" and to 2 for style = "interval".
grid	Whether to add an horizontal grid. Default is TRUE.
grid.par	Graphical parameters used when plotting the grid in the background. Default is list(lty=1).
bar.join	Logical, default is FALSE. Whether to join the dots when style = "bar".
only.params	Logical, default is FALSE. If TRUE: no graph is plotted, only the ylim is returned. Useful to stack estimates from different estimations in the same graph.
...	Other arguments to be passed to the function plot or lines (if add = TRUE).

Author(s)

Laurent Berge

See Also

did_estimate_yearly_effects, did_plot_yearly_effects.

Examples

```
a = rnorm(100)
b = 0.5*a + rnorm(100)
c = -0.5*b + rnorm(100)

est = summary(lm(a ~ c + b))

errbar(est$coefficients, x.shift = -.2)

errbar(est$coefficients, , x.shift = .2, add = TRUE, col = 2, bar.lty = 2, pch=15)
```

---

esttable *Facility to display the results of multiple* fixest *estimations.*

Description

This function aggregates the results of multiple estimations and display them in the form of only one table whose row names are the variables and the columns contain the coefficients and standard-errors.

Usage

```
esttable(..., se = c("standard", "white", "cluster", "twoway",
  "threeway", "fourway"), dof = TRUE, cluster, depvar, drop, order,
  digits = 4, fitstat, convergence, signifCode = c(`***` = 0.001, `**`
  = 0.01, `*` = 0.05, . = 0.1), titles, keepFactors = FALSE, family)
```

Arguments

...	Used to capture different fixest objects (obtained with femlm, feols or feglm). Note that any other type of element is discarded. Note that you can give a list of fixest objects.
se	Character scalar. Which kind of standard error should be computed: "standard", "White", "cluster", "twoway", "threeway" or "fourway"? By default if there are clusters in the estimation: se = "cluster", otherwise se = "standard". Note that this argument can be implicitly deduced from the argument cluster.
dof	Logical, default is TRUE. Should there be a degree of freedom correction to the standard errors of the coefficients?

`cluster` Tells how to cluster the standard-errors (if clustering is requested). Can be either a list of vectors, a character vector of variable names, a formula or an integer vector. Assume we want to perform 2-way clustering over `var1` and `var2` contained in the data.frame `base` used for the estimation. All the following `cluster` arguments are valid and do the same thing: `cluster = base[,c("var1","var2")]`,\code{cluster = c("var1","var2")}, `cluster = ~var1+var2`. If the two variables were used as clusters in the estimation, you could further use `cluster = 1:2` or leave it blank with `se = "twoway"` (assuming `var1` [resp. `var2`] was the 1st [res. 2nd] cluster).

`depvar` Logical, default is missing. Whether a first line containing the dependent variables should be shown. By default, the dependent variables are shown only if they differ across models.

`drop` Character vector. This element is used if some variables are not to be displayed. This should be a regular expression (see `regex` help for more info). There can be more than one regular expression. Each variable satisfying the regular expression will be discarded.

`order` Character vector. This element is used if the user wants the variables to be ordered in a certain way. This should be a regular expression (see `regex` help for more info). There can be more than one regular expression. The variables satisfying the first regular expression will be placed first, then the order follows the sequence of regular expressions.

`digits` Integer, default is 4. The number of digits to be displayed.

`fitstat` A character vector or a one sided formula. A vector listing which fit statistics to display. The valid types are 'll', 'aic', 'bic' and r2 types like 'r2', 'pr2', 'war2', etc (see all valid types in `r2`). The default value depends on the models to display. Example of use: `fitstat=c('sq.cor','ar2','war2')`, or `fitstat=~sq.cor+ar2+war2` using a formula.

`convergence` Logical, default is missing. Should the convergence state of the algorithm be displayed? By default, convergence information is displayed if at least one model did not converge.

`signifCode` Named numeric vector, used to provide the significance codes with respect to the p-value of the coefficients. Default is `c("***"=0.001,"**"=0.01,"*"=0.05,"."=0.10)`.

`titles` A character vector. The length must match the number of models.

`keepFactors` Logical, default is `TRUE`. If `FALSE`, then factor variables are displayed as fixed-effects and no coefficient is shown.

`family` A logical, default is missing. Whether to display the families of the models. By default this line is displayed when at least two models are from different families.

Value

Returns a data.frame containing the formatted results.

Author(s)

Laurent Berge

See Also

See also the main estimation functions femlm, feols or feglm. Use summary.fixest to see the results with the appropriate standard-errors, fixef.fixest to extract the cluster coefficients, and the functions esttable and esttex to visualize the results of multiple estimations.

Examples

```
# two fitted models with different expl. variables:
res1 = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
            Petal.Width | Species, iris)
# estimation without clusters
res2 = update(res1, . ~ Sepal.Width | 0)

# We export the two results in one Latex table:
esttable(res1, res2)

# With clustered standard-errors + showing the dependent variable
esttable(res1, res2, se = "cluster", cluster = iris$Species, depvar = TRUE)

# Changing the model names + the order of the variables
# + dropping the intercept.
esttable(model_1 = res1, res2,
          order = c("Width", "Petal"), drop = "Int",
          signifCode = c("**" = 0, "*" = 0.2, "n.s."=1))
```

---

esttex *Facility to export the results of multiple* fixest *estimations in a Latex table.*

Description

This function aggregates the results of multiple estimations and display them in the form of one Latex table whose row names are the variables and the columns contain the coefficients and standard-errors.

Usage

```
esttex(..., se = c("standard", "white", "cluster", "twoway", "threeway",
  "fourway"), dof = TRUE, cluster, digits = 4, fitstat, title,
  sdBelow = TRUE, drop, order, dict = getFixest_dict(), file,
  replace = FALSE, convergence, signifCode = c(`***` = 0.01, `**` =
  0.05, `*` = 0.1), label, subtitles, fixef_sizes = FALSE,
  yesNoFixef = c("Yes", "No"), keepFactors = TRUE, family,
  powerBelow = -5)
```

Arguments

...	Used to capture different fixest objects (obtained with femlm, feols or feglm). Note that any other type of element is discarded. Note that you can give a list of fixest objects.
se	Character scalar. Which kind of standard error should be computed: "standard", "White", "cluster", "twoway", "threeway" or "fourway"? By default if there are clusters in the estimation: se = "cluster", otherwise se = "standard". Note that this argument can be implicitly deduced from the argument cluster.
dof	Logical, default is TRUE. Should there be a degree of freedom correction to the standard errors of the coefficients?
cluster	Tells how to cluster the standard-errors (if clustering is requested). Can be either a list of vectors, a character vector of variable names, a formula or an integer vector. Assume we want to perform 2-way clustering over var1 and var2 contained in the data.frame base used for the estimation. All the following cluster arguments are valid and do the same thing: cluster = base[,c("var1","var2")]},\code{cluster = c("var1","var2"), cluster = ~var1+var2. If the two variables were used as clusters in the estimation, you could further use cluster = 1:2 or leave it blank with se = "twoway" (assuming var1 [resp. var2] was the 1st [res. 2nd] cluster).
digits	Integer, default is 4. The number of digits to be displayed.
fitstat	A character vector or a one sided formula. A vector listing which fit statistics to display. The valid types are 'll', 'aic', 'bic' and r2 types like 'r2', 'pr2', 'war2', etc (see all valid types in r2). The default value depends on the models to display. Example of use: fitstat=c('sq.cor','ar2','war2'), or fitstat=~sq.cor+ar2+war2 using a formula.
title	Character scalar. The title of the Latex table.
sdBelow	Logical, default is TRUE. Should the standard-errors be displayed below the coefficients?
drop	Character vector. This element is used if some variables are not to be displayed. This should be a regular expression (see regex help for more info). There can be more than one regular expression. Each variable satisfying the regular expression will be discarded.
order	Character vector. This element is used if the user wants the variables to be ordered in a certain way. This should be a regular expression (see regex help for more info). There can be more than one regular expression. The variables satisfying the first regular expression will be placed first, then the order follows the sequence of regular expressions.
dict	A named character vector. It changes the original variable names to the ones contained in the dict. E.g. to change the variables named a and b3 to (resp.) "$log(a)$" and to "$bonus^3$", use dict=c(a="$log(a)$",b3="$bonus^3$"). By default it is equal to getFixest_dict(), a default dictionary can be set with setFixest_dict.
file	A character scalar. If provided, the Latex table will be saved in a file whose path is file.
replace	Logical, default is FALSE. Only used if option file is used. Should the Latex table be written in a new file that replaces any existing file?

convergence	Logical, default is missing. Should the convergence state of the algorithm be displayed? By default, convergence information is displayed if at least one model did not converge.
signifCode	Named numeric vector, used to provide the significance codes with respect to the p-value of the coefficients. Default is `c("***"=0.01,"**"=0.05,"*"=0.10)`.
label	Character scalar. The label of the Latex table.
subtitles	Character vector of the same length as the number of models to be displayed. If provided, subtitles are added underneath the dependent variable name.
fixef_sizes	Logical, default is `FALSE`. If `TRUE` and fixed-effects were used in the models, then the number "individuals" per fixed-effect dimension is also displayed.
yesNoFixef	A character vector of length 2. Default is `c("Yes","No")`. This is the message displayed when a given cluster is (or is not) included in a regression.
keepFactors	Logical, default is `TRUE`. If `FALSE`, then factor variables are displayed as fixed-effects and no coefficient is shown.
family	A logical, default is missing. Whether to display the families of the models. By default this line is displayed when at least two models are from different families.
powerBelow	Integer, default is -5. A coefficient whose value is below `10**(powerBelow+1)` is written with a power in Latex. For example `0.0000456` would be written `4.56$\times 10^{-5}$` by default. Setting `powerBelow = -6` would lead to `0.00004` in Latex.

Value

There is nothing returned, the result is only displayed on the console or saved in a file.

Author(s)

Laurent Berge

See Also

See also the main estimation functions `femlm`, `feols` or `feglm`. Use `summary.fixest` to see the results with the appropriate standard-errors, `fixef.fixest` to extract the cluster coefficients, and the functions `esttable` and `esttex` to visualize the results of multiple estimations.

Examples

```
# two fitted models with different expl. variables:
res1 = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
             Petal.Width | Species, iris)
res2 = femlm(Sepal.Length ~ Petal.Width | Species, iris)

# We export the three results in one Latex table,
# with clustered standard-errors:
esttex(res1, res2, se = "cluster")
```

```
# Changing the names & significance codes
esttex(res1, res2, dict = c(Sepal.Length = "The sepal length", Sepal.Width = "SW"),
        signifCode = c("**" = 0.1, "*" = 0.2, "n.s."=1))
```

---

feglm *Fixed-effects GLM estimations*

Description

Estimates GLM models with any number of fixed-effects.

Usage

```
feglm(fml, data, family = "poisson", offset, weights, start = NULL,
  etastart = NULL, mustart = NULL, fixef, fixef.tol = 1e-06,
  fixef.iter = 1000, glm.iter = 25, glm.tol = 1e-08,
  na_inf.rm = getFixest_na_inf.rm(), nthreads = getFixest_nthreads(),
  warn = TRUE, notes = getFixest_notes(), verbose = 0, combine.quick,
  ...)

feglm.fit(y, X, fixef_mat, family = "poisson", offset, weights,
  start = NULL, etastart = NULL, mustart = NULL, fixef.tol = 1e-06,
  fixef.iter = 1000, glm.iter = 25, glm.tol = 1e-08,
  na_inf.rm = getFixest_na_inf.rm(), nthreads = getFixest_nthreads(),
  warn = TRUE, notes = getFixest_notes(), verbose = 0, ...)

fepois(fml, data, offset, weights, start = NULL, etastart = NULL,
  mustart = NULL, fixef, fixef.tol = 1e-06, fixef.iter = 1000,
  glm.iter = 25, glm.tol = 1e-08, na_inf.rm = getFixest_na_inf.rm(),
  nthreads = getFixest_nthreads(), warn = TRUE,
  notes = getFixest_notes(), verbose = 0, combine.quick, ...)
```

Arguments

`fml`	A formula representing the relation to be estimated. For example: `fml = z~x+y`. To include fixed-effects, insert them in this formula using a pipe: e.g. `fml = z~x+y | fe_1+fe_2`. You can combine two clusters with `^`: e.g. `fml = z~x+y|fe_1^fe_2`, see details. You can also use variables with varying slopes using square brackets: e.g. in `fml = z~y|fe_1[x] + fe_2` the variable `x` will have one coefficient for each value of `fe_1` – if you use varying slopes, please have a look at the details section (can't describe it all here).
`data`	A data.frame containing the necessary variables to run the model. The variables of the non-linear right hand side of the formula are identified with this `data.frame` names. Can also be a matrix.
`family`	Family to be used for the estimation. Defaults to `poisson()`. See `family` for details of family functions.

`offset` A formula or a numeric vector. An offset can be added to the estimation. If equal to a formula, it should be of the form (for example) `~0.5*x**2`. This offset is linearly added to the elements of the main formula 'fml'.

`weights` A formula or a numeric vector. Each observation can be weighted, the weights must be greater than 0. If equal to a formula, it should be of one-sided: for example `~ var_weight`.

`start` Starting values for the coefficients. Can be: i) a numeric of length 1 (e.g. `start = 0`), ii) a numeric vector of the exact same length as the number of variables, or iii) a named vector of any length (the names will be used to initialize the appropriate coefficients). Default is missing.

`etastart` Numeric vector of the same length as the data. Starting values for the linear predictor. Default is missing.

`mustart` Numeric vector of the same length as the data. Starting values for the vector of means. Default is missing.

`fixef` Character vector. The name/s of a/some variable/s within the dataset to be used as fixed-effects. These variables should contain the identifier of each observation (e.g., think of it as a panel identifier).

`fixef.tol` Precision used to obtain the fixed-effects (ie cluster coefficients). Defaults to `1e-5`. It corresponds to the maximum absolute difference allowed between two coefficients of successive iterations. Argument `fixef.tol` cannot be lower than `10000*.Machine$double.eps`. Note that this parameter is dynamically controlled by the algorithm.

`fixef.iter` Maximum number of iterations in the step obtaining the fixed-effects (only in use for 2+ clusters). Default is 10000.

`glm.iter` Number of iterations of the glm algorithm. Default is 25.

`glm.tol` Tolerance level for the glm algorithm. Default is `1e-8`.

`na_inf.rm` Logical, default is `TRUE`. If the variables necessary for the estimation contain NA/Infs and `na_inf.rm = TRUE`, then all observations containing NA are removed prior to estimation and a note is displayed detailing the number of observations removed. Otherwise, an error is raised.

`nthreads` Integer: Number of nthreads to be used (accelerates the algorithm via the use of openMP routines). The default is to use the total number of nthreads available minus two. You can set permanently the number of nthreads used within this package using the function `setFixest_nthreads`.

`warn` Logical, default is `TRUE`. Whether warnings should be displayed (concerns warnings relating to: convergence state, collinearity issues and observation removal due to only 0/1 outcomes or presence of NA values).

`notes` Logical. By default, two notes are displayed: when NAs are removed (to show additional information) and when some observations are removed because of only 0 (or 0/1) outcomes in a fixed-effect (in Poisson/Neg. Bin./Logit models). To avoid displaying these messages, you can set `notes = FALSE`. You can remove these messages permanently by using `setFixest_notes(FALSE)`.

`verbose` Integer, default is 0. It represents the level of information that should be reported during the optimisation process. If `verbose=0`: nothing is reported. If

	verbose=1: the value of the coefficients and the likelihood are reported. If verbose=2: 1 + information on the computing time of the null model, the cluster coefficients and the hessian are reported.
combine.quick	Logical. When you combine different variables to transform them into a single fixed-effects you can do e.g. y ~ x | paste(var1,var2). The algorithm provides a shorthand to do the same operation: y ~ x | var1^var2. Because pasting variables is a costly operation, the internal algorithm may use a numerical trick to hasten the process. The cost of doing so is that you lose the labels. If you are interested in getting the value of the fixed-effects coefficients after the estimation, you should use combine.quick = FALSE. By default it is equal to FALSE if the number of observations is lower than 50,000, and to TRUE otherwise.
...	Not currently used.
y	Numeric vector of the dependent variable.
X	Numeric matrix of the regressors.
fixef_mat	Matrix/data.frame of the fixed-effects.

Details

The core of the GLM are the weighted OLS estimations. These estimations are performed with feols. The method used to demean each variable along the fixed-effects is based on Berge (2018), since this is the same problem to solve as for the Gaussian case in a ML setup.

Functions

- feglm.fit: Matrix method for fixed-effects GLM estimation
- fepois: Fixed-effects Poisson estimation

Combining the fixed-effects

You can combine two variables to make it a new fixed-effect using ^. The syntax is as follows: fe_1^fe_2. Here you created a new variable which is the combination of the two variables fe_1 and fe_2. This is identical to doing paste0(fe_1,"_",fe_2) but more convenient.

Note that pasting is a costly operation, especially for large data sets. Thus, the internal algorithm uses a numerical trick which is fast, but the drawback is that the identity of each observation is lost (i.e. they are now equal to a meaningless number instead of being equal to paste0(fe_1,"_",fe_2)). These "identities" are useful only if you're interested in the value of the fixed-effects (that you can extract with fixef.fixest). If you're only interested in coefficients of the variables, it doesn't matter. Anyway, you can use combine.quick = FALSE to tell the internal algorithm to use paste instead of the numerical trick. By default, the numerical trick is performed only for large data sets.

Varying slopes

You can add variables with varying slopes in the fixed-effect part of the formula. The syntax is as follows: cluster_var[var1, var2]. Here the variables var1 and var2 will be with varying slopes (one slope per value in cluster_var) and the fixed-effect cluster_var will also be added.

To add only the variables with varying slopes and not the fixed-effect, use double square brackets: cluster_var[[var1, var2]].

In other words:

- cluster_var[var1, var2] is equivalent to cluster_var + cluster_var[[var1]] + cluster_var[[var2]]
- cluster_var[[var1, var2]] is equivalent to cluster_var[[var1]] + cluster_var[[var2]]

@seealso See also summary.fixest to see the results with the appropriate standard-errors, fixef.fixest to extract the cluster coefficients, and the functions esttable and esttex to visualize the results of multiple estimations. And other estimation methods: femlm, feglm, fepois, fenegbin, feNmlm.

Author(s)

Laurent Berge

References

Berge, Laurent, 2018, "Efficient estimation of maximum likelihood models with multiple fixed-effects: the R package FENmlm." CREA Discussion Papers, 13 (https://wwwen.uni.lu/content/download/110162/1299525/file/2018_13).

For models with multiple fixed-effects:

Gaure, Simen, 2013, "OLS with multiple high dimensional category variables", Computational Statistics & Data Analysis 66 pp. 8–18

See Also

See also summary.fixest to see the results with the appropriate standard-errors, fixef.fixest to extract the cluster coefficients, and the functions esttable and esttex to visualize the results of multiple estimations. And other estimation methods: feols, femlm, fenegbin, feNmlm.

Examples

```
# default is a poisson model
res = feglm(Sepal.Length ~ Sepal.Width + Petal.Length | Species, iris)

# with the fit method:
res_bis = feglm.fit(iris$Sepal.Length, iris[, 2:3], iris$Species)
```

---

femlm *Fixed-effects maximum likelihood model*

Description

This function estimates maximum likelihood models with any number of fixed-effects.

Usage

```
femlm(fml, data, family = c("poisson", "negbin", "logit", "gaussian"),
  start = 0, fixef, offset, na_inf.rm = getFixest_na_inf.rm(),
  fixef.tol = 1e-05, fixef.iter = 1000,
  nthreads = getFixest_nthreads(), verbose = 0, warn = TRUE,
  notes = getFixest_notes(), theta.init, combine.quick, ...)

fenegbin(fml, data, theta.init, start = 0, fixef, offset,
  na_inf.rm = getFixest_na_inf.rm(), fixef.tol = 1e-05,
  fixef.iter = 1000, nthreads = getFixest_nthreads(), verbose = 0,
  warn = TRUE, notes = getFixest_notes(), combine.quick, ...)
```

Arguments

fml	A formula representing the relation to be estimated. For example: fml = z~x+y. To include fixed-effects, you can 1) either insert them in this formula using a pipe (e.g. fml = z~x+y|cluster1+cluster2), or 2) either use the argument fixef.
data	A data.frame containing the necessary variables to run the model. The variables of the non-linear right hand side of the formula are identified with this data.frame names. Can also be a matrix.
family	Character scalar. It should provide the family. The possible values are "poisson" (Poisson model with log-link, the default), "negbin" (Negative Binomial model with log-link), "logit" (LOGIT model with log-link), "gaussian" (Gaussian model).
start	Starting values for the coefficients. Can be: i) a numeric of length 1 (e.g. start = 0, the default), ii) a numeric vector of the exact same length as the number of variables, or iii) a named vector of any length (the names will be used to initialize the appropriate coefficients).
fixef	Character vector. The name/s of a/some variable/s within the dataset to be used as fixed-effects. These variables should contain the identifier of each observation (e.g., think of it as a panel identifier).
offset	A formula or a numeric vector. An offset can be added to the estimation. If equal to a formula, it should be of the form (for example) ~0.5*x**2. This offset is linearly added to the elements of the main formula 'fml'.
na_inf.rm	Logical, default is TRUE. If the variables necessary for the estimation contain NA/Infs and na_inf.rm = TRUE, then all observations containing NA are removed prior to estimation and a note is displayed detailing the number of observations removed. Otherwise, an error is raised.
fixef.tol	Precision used to obtain the fixed-effects (ie cluster coefficients). Defaults to 1e-5. It corresponds to the maximum absolute difference allowed between two coefficients of successive iterations. Argument fixef.tol cannot be lower than 10000*.Machine$double.eps. Note that this parameter is dynamically controlled by the algorithm.
fixef.iter	Maximum number of iterations in the step obtaining the fixed-effects (only in use for 2+ clusters). Default is 10000.

`nthreads` Integer: Number of nthreads to be used (accelerates the algorithm via the use of openMP routines). The default is to use the total number of nthreads available minus two. You can set permanently the number of nthreads used within this package using the function `setFixest_nthreads`.

`verbose` Integer, default is 0. It represents the level of information that should be reported during the optimisation process. If `verbose=0`: nothing is reported. If `verbose=1`: the value of the coefficients and the likelihood are reported. If `verbose=2`: 1 + information on the computing time of the null model, the cluster coefficients and the hessian are reported.

`warn` Logical, default is `TRUE`. Whether warnings should be displayed (concerns warnings relating to: convergence state, collinearity issues and observation removal due to only 0/1 outcomes or presence of NA values).

`notes` Logical. By default, two notes are displayed: when NAs are removed (to show additional information) and when some observations are removed because of only 0 (or 0/1) outcomes in a fixed-effect (in Poisson/Neg. Bin./Logit models). To avoid displaying these messages, you can set `notes = FALSE`. You can remove these messages permanently by using `setFixest_notes(FALSE)`.

`theta.init` Positive numeric scalar. The starting value of the dispersion parameter if `family="negbin"`. By default, the algorithm uses as a starting value the theta obtained from the model with only the intercept.

`combine.quick` Logical. When you combine different variables to transform them into a single fixed-effects you can do e.g. `y ~ x | paste(var1,var2)`. The algorithm provides a shorthand to do the same operation: `y ~ x | var1^var2`. Because pasting variables is a costly operation, the internal algorithm may use a numerical trick to hasten the process. The cost of doing so is that you lose the labels. If you are interested in getting the value of the fixed-effects coefficients after the estimation, you should use `combine.quick = FALSE`. By default it is equal to `FALSE` if the number of observations is lower than 50,000, and to `TRUE` otherwise.

`...` Not currently used.

Details

This function estimates maximum likelihood models where the conditional expectations are as follows:

Gaussian likelihood:

$$E(Y|X) = X\beta$$

Poisson and Negative Binomial likelihoods:

$$E(Y|X) = \exp(X\beta)$$

where in the Negative Binomial there is the parameter θ used to model the variance as $\mu + \mu^2/\theta$, with μ the conditional expectation. Logit likelihood:

$$E(Y|X) = \frac{\exp(X\beta)}{1 + \exp(X\beta)}$$

When there are one or more clusters, the conditional expectation can be written as:

$$E(Y|X) = h(X\beta + \sum_k \sum_m \gamma_m^k \times C_{im}^k),$$

where $h(.)$ is the function corresponding to the likelihood function as shown before. C^k is the matrix associated to cluster k such that C_{im}^k is equal to 1 if observation i is of category m in cluster k and 0 otherwise.

When there are non linear in parameters functions, we can schematically split the set of regressors in two:

$$f(X, \beta) = X^1\beta^1 + g(X^2, \beta^2)$$

with first a linear term and then a non linear part expressed by the function g. That is, we add a non-linear term to the linear terms (which are $X * beta$ and the cluster coefficients). It is always better (more efficient) to put into the argument `NL.fml` only the non-linear in parameter terms, and add all linear terms in the `fml` argument.

To estimate only a non-linear formula without even the intercept, you must exclude the intercept from the linear formula by using, e.g., `fml = z~0`.

The over-dispersion parameter of the Negative Binomial family, theta, is capped at 10,000. If theta reaches this high value, it means that there is no overdispersion.

Value

An `femlm` object.

`coefficients`	The named vector of coefficients.
`coeftable`	The table of the coefficients with their standard errors, z-values and p-values.
`loglik`	The loglikelihood.
`iterations`	Number of iterations of the algorithm.
`n`	The number of observations.
`nparams`	The number of parameters of the model.
`call`	The call.
`fml`	The linear formula of the call.
`ll_null`	Log-likelihood of the null model (i.e. with the intercept only).
`pseudo_r2`	The adjusted pseudo R2.
`message`	The convergence message from the optimization procedures.
`sq.cor`	Squared correlation between the dependent variable and the expected predictor (i.e. fitted.values) obtained by the estimation.
`hessian`	The Hessian of the parameters.
`fitted.values`	The fitted values are the expected value of the dependent variable for the fitted model: that is $E(Y\|X)$.
`cov.unscaled`	The variance-covariance matrix of the parameters.
`se`	The standard-error of the parameters.
`scores`	The matrix of the scores (first derivative for each observation).

family	The ML family that was used for the estimation.
residuals	The difference between the dependent variable and the expected predictor.
sumFE	The sum of the fixed-effects for each observation.
offset	The offset formula.
NL.fml	The nonlinear formula of the call.
bounds	Whether the coefficients were upper or lower bounded. – This can only be the case when a non-linear formula is included and the arguments 'lower' or 'upper' are provided.
isBounded	The logical vector that gives for each coefficient whether it was bounded or not. This can only be the case when a non-linear formula is included and the arguments 'lower' or 'upper' are provided.
fixef_vars	The names of each cluster.
fixef_id	The list (of length the number of clusters) of the cluster identifiers for each observation.
fixef_sizes	The size of each cluster.
obsRemoved	In the case there were clusters and some observations were removed because of only 0/1 outcome within a cluster, it gives the row numbers of the observations that were removed.
fixef_removed	In the case there were clusters and some observations were removed because of only 0/1 outcome within a cluster, it gives the list (for each cluster) of the cluster identifiers that were removed.
theta	In the case of a negative binomial estimation: the overdispersion parameter.

@seealso See also `summary.fixest` to see the results with the appropriate standard-errors, `fixef.fixest` to extract the cluster coefficients, and the functions `esttable` and `esttex` to visualize the results of multiple estimations.

And other estimation methods: `feols`, `femlm`, `feglm`, `fepois`, `fenegbin`.

Functions

- `fenegbin`: Fixed-effects negative binomial estimation

Combining the fixed-effects

You can combine two variables to make it a new fixed-effect using `^`. The syntax is as follows: `fe_1^fe_2`. Here you created a new variable which is the combination of the two variables fe_1 and fe_2. This is identical to doing `paste0(fe_1,"_",fe_2)` but more convenient.

Note that pasting is a costly operation, especially for large data sets. Thus, the internal algorithm uses a numerical trick which is fast, but the drawback is that the identity of each observation is lost (i.e. they are now equal to a meaningless number instead of being equal to `paste0(fe_1,"_",fe_2)`). These "identities" are useful only if you're interested in the value of the fixed-effects (that you can extract with `fixef.fixest`). If you're only interested in coefficients of the variables, it doesn't matter. Anyway, you can use `combine.quick = FALSE` to tell the internal algorithm to use `paste` instead of the numerical trick. By default, the numerical trick is performed only for large data sets.

Author(s)

Laurent Berge

References

Berge, Laurent, 2018, "Efficient estimation of maximum likelihood models with multiple fixed-effects: the R package FENmlm." CREA Discussion Papers, 13 (https://wwwen.uni.lu/content/download/110162/1299525/file/2018_13).

For models with multiple fixed-effects:

Gaure, Simen, 2013, "OLS with multiple high dimensional category variables", Computational Statistics & Data Analysis 66 pp. 8–18

On the unconditionnal Negative Binomial model:

Allison, Paul D and Waterman, Richard P, 2002, "Fixed-Effects Negative Binomial Regression Models", Sociological Methodology 32(1) pp. 247–265

See Also

See also summary.fixest to see the results with the appropriate standard-errors, fixef.fixest to extract the cluster coefficients, and the functions esttable and esttex to visualize the results of multiple estimations. And other estimation methods: feols, feglm, fepois, feNmlm.

Examples

```
#
# Linear examples
#

# Load trade data
data(trade)

# We estimate the effect of distance on trade => we account for 3 fixed-effects
# 1) Poisson estimation
est_pois = femlm(Euros ~ log(dist_km)|Origin+Destination+Product, trade)

# 2) Log-Log Gaussian estimation (with same FEs)
est_gaus = update(est_pois, log(Euros+1) ~ ., family="gaussian")

# Comparison of the results using the function esttable
esttable(est_pois, est_gaus)
# Now using two way clustered standard-errors
esttable(est_pois, est_gaus, se = "twoway")

# Comparing different types of standard errors
sum_white    = summary(est_pois, se = "white")
sum_oneway   = summary(est_pois, se = "cluster")
sum_twoway   = summary(est_pois, se = "twoway")
sum_threeway = summary(est_pois, se = "threeway")

esttable(sum_white, sum_oneway, sum_twoway, sum_threeway)
```

feNmlm *Fixed effects nonlinear maximum likelihood models*

Description

This function estimates maximum likelihood models (e.g., Poisson or Logit) with non-linear in parameters right-hand-sides and is efficient to handle any number of fixed effects. If you do not use non-linear in parameters right-hand-side, use femlm or feglm instead (design is simpler).

Usage

```
feNmlm(fml, data, family = c("poisson", "negbin", "logit", "gaussian"),
  NL.fml, fixef, na_inf.rm = getFixest_na_inf.rm(), NL.start, lower,
  upper, NL.start.init, offset, start = 0, jacobian.method = "simple",
  useHessian = TRUE, hessian.args = NULL, opt.control = list(),
  nthreads = getFixest_nthreads(), verbose = 0, theta.init,
  fixef.tol = 1e-05, fixef.iter = 1000, deriv.tol = 1e-04,
  deriv.iter = 1000, warn = TRUE, notes = getFixest_notes(),
  combine.quick, ...)
```

Arguments

fml	A formula. This formula gives the linear formula to be estimated (it is similar to a lm formula), for example: fml = z~x+y. To include cluster variables, you can 1) either insert them in this formula using a pipe (e.g. fml = z~x+y|cluster1+cluster2), or 2) either use the argument cluster. To include a non-linear in parameters element, you must use the argment NL.fml.
data	A data.frame containing the necessary variables to run the model. The variables of the non-linear right hand side of the formula are identified with this data.frame names. Can also be a matrix.
family	Character scalar. It should provide the family. The possible values are "poisson" (Poisson model with log-link, the default), "negbin" (Negative Binomial model with log-link), "logit" (LOGIT model with log-link), "gaussian" (Gaussian model).
NL.fml	A formula. If provided, this formula represents the non-linear part of the right hand side (RHS). Note that contrary to the fml argument, the coefficients must explicitly appear in this formula. For instance, it can be ~a*log(b*x + c*x^3), where a, b, and c are the coefficients to be estimated. Note that only the RHS of the formula is to be provided, and NOT the left hand side.
fixef	Character vector. The name/s of a/some variable/s within the dataset to be used as fixed-effects. These variables should contain the identifier of each observation (e.g., think of it as a panel identifier).

`na_inf.rm` Logical, default is `TRUE`. If the variables necessary for the estimation contain NA/Infs and `na_inf.rm = TRUE`, then all observations containing NA are removed prior to estimation and a note is displayed detailing the number of observations removed. Otherwise, an error is raised.

`NL.start` (For NL models only) A list of starting values for the non-linear parameters. ALL the parameters are to be named and given a staring value. Example: `NL.start=list(a=1,b=5,c=0)`. Though, there is an exception: if all parameters are to be given the same starting value, you can use the argument `NL.start.init`.

`lower` (For NL models only) A list. The lower bound for each of the non-linear parameters that requires one. Example: `lower=list(b=0,c=0)`. Beware, if the estimated parameter is at his lower bound, then asymptotic theory cannot be applied and the standard-error of the parameter cannot be estimated because the gradient will not be null. In other words, when at its upper/lower bound, the parameter is considered as 'fixed'.

`upper` (For NL models only) A list. The upper bound for each of the non-linear parameters that requires one. Example: `upper=list(a=10,c=50)`. Beware, if the estimated parameter is at his upper bound, then asymptotic theory cannot be applied and the standard-error of the parameter cannot be estimated because the gradient will not be null. In other words, when at its upper/lower bound, the parameter is considered as 'fixed'.

`NL.start.init` (For NL models only) Numeric scalar. If the argument `NL.start` is not provided, or only partially filled (i.e. there remain non-linear parameters with no starting value), then the starting value of all remaining non-linear parameters is set to `NL.start.init`.

`offset` A formula or a numeric vector. An offset can be added to the estimation. If equal to a formula, it should be of the form (for example) `~0.5*x**2`. This offset is linearly added to the elements of the main formula 'fml'.

`start` Starting values for the coefficients in the linear part (for the non-linear part, use NL.start). Can be: i) a numeric of length 1 (e.g. `start = 0`, the default), ii) a numeric vector of the exact same length as the number of variables, or iii) a named vector of any length (the names will be used to initialize the appropriate coefficients).

`jacobian.method` (For NL models only) Character scalar. Provides the method used to numerically compute the Jacobian of the non-linear part. Can be either `"simple"` or `"Richardson"`. Default is `"simple"`. See the help of `jacobian` for more information.

`useHessian` Logical. Should the Hessian be computed in the optimization stage? Default is `TRUE`.

`hessian.args` List of arguments to be passed to function `genD`. Defaults is missing. Only used with the presence of `NL.fml`.

`opt.control` List of elements to be passed to the optimization method `nlminb`. See the help page of `nlminb` for more information.

`nthreads` Integer: Number of nthreads to be used (accelerates the algorithm via the use of openMP routines). The default is to use the total number of nthreads available

minus two. You can set permanently the number of nthreads used within this package using the function setFixest_nthreads.

verbose — Integer, default is 0. It represents the level of information that should be reported during the optimisation process. If verbose=0: nothing is reported. If verbose=1: the value of the coefficients and the likelihood are reported. If verbose=2: 1 + information on the computing time of the null model, the cluster coefficients and the hessian are reported.

theta.init — Positive numeric scalar. The starting value of the dispersion parameter if family="negbin". By default, the algorithm uses as a starting value the theta obtained from the model with only the intercept.

fixef.tol — Precision used to obtain the fixed-effects (ie cluster coefficients). Defaults to 1e-5. It corresponds to the maximum absolute difference allowed between two coefficients of successive iterations. Argument fixef.tol cannot be lower than 10000*.Machine$double.eps. Note that this parameter is dynamically controlled by the algorithm.

fixef.iter — Maximum number of iterations in the step obtaining the fixed-effects (only in use for 2+ clusters). Default is 10000.

deriv.tol — Precision used to obtain the fixed-effects derivatives. Defaults to 1e-4. It corresponds to the maximum absolute difference allowed between two coefficients of successive iterations. Argument deriv.tol cannot be lower than 10000*.Machine$double.eps.

deriv.iter — Maximum number of iterations in the step obtaining the derivative of the fixed-effects (only in use for 2+ clusters). Default is 1000.

warn — Logical, default is TRUE. Whether warnings should be displayed (concerns warnings relating to: convergence state, collinearity issues and observation removal due to only 0/1 outcomes or presence of NA values).

notes — Logical. By default, two notes are displayed: when NAs are removed (to show additional information) and when some observations are removed because of only 0 (or 0/1) outcomes in a fixed-effect (in Poisson/Neg. Bin./Logit models). To avoid displaying these messages, you can set notes = FALSE. You can remove these messages permanently by using setFixest_notes(FALSE).

combine.quick — Logical. When you combine different variables to transform them into a single fixed-effects you can do e.g. y ~ x | paste(var1,var2). The algorithm provides a shorthand to do the same operation: y ~ x | var1^var2. Because pasting variables is a costly operation, the internal algorithm may use a numerical trick to hasten the process. The cost of doing so is that you lose the labels. If you are interested in getting the value of the fixed-effects coefficients after the estimation, you should use combine.quick = FALSE. By default it is equal to FALSE if the number of observations is lower than 50,000, and to TRUE otherwise.

... — Not currently used.

Details

This function estimates maximum likelihood models where the conditional expectations are as follows:

Gaussian likelihood:

$$E(Y|X) = X\beta$$

Poisson and Negative Binomial likelihoods:

$$E(Y|X) = \exp(X\beta)$$

where in the Negative Binomial there is the parameter θ used to model the variance as $\mu + \mu^2/\theta$, with μ the conditional expectation. Logit likelihood:

$$E(Y|X) = \frac{\exp(X\beta)}{1 + \exp(X\beta)}$$

When there are one or more clusters, the conditional expectation can be written as:

$$E(Y|X) = h(X\beta + \sum_k \sum_m \gamma_m^k \times C_{im}^k),$$

where $h(.)$ is the function corresponding to the likelihood function as shown before. C^k is the matrix associated to cluster k such that C_{im}^k is equal to 1 if observation i is of category m in cluster k and 0 otherwise.

When there are non linear in parameters functions, we can schematically split the set of regressors in two:

$$f(X, \beta) = X^1\beta^1 + g(X^2, \beta^2)$$

with first a linear term and then a non linear part expressed by the function g. That is, we add a non-linear term to the linear terms (which are $X * beta$ and the cluster coefficients). It is always better (more efficient) to put into the argument `NL.fml` only the non-linear in parameter terms, and add all linear terms in the `fml` argument.

To estimate only a non-linear formula without even the intercept, you must exclude the intercept from the linear formula by using, e.g., `fml = z~0`.

The over-dispersion parameter of the Negative Binomial family, theta, is capped at 10,000. If theta reaches this high value, it means that there is no overdispersion.

Value

An `femlm` object.

`coefficients`	The named vector of coefficients.
`coeftable`	The table of the coefficients with their standard errors, z-values and p-values.
`loglik`	The loglikelihood.
`iterations`	Number of iterations of the algorithm.
`n`	The number of observations.
`nparams`	The number of parameters of the model.
`call`	The call.
`fml`	The linear formula of the call.
`ll_null`	Log-likelihood of the null model (i.e. with the intercept only).
`pseudo_r2`	The adjusted pseudo R2.

message	The convergence message from the optimization procedures.
sq.cor	Squared correlation between the dependent variable and the expected predictor (i.e. fitted.values) obtained by the estimation.
hessian	The Hessian of the parameters.
fitted.values	The fitted values are the expected value of the dependent variable for the fitted model: that is $E(Y\|X)$.
cov.unscaled	The variance-covariance matrix of the parameters.
se	The standard-error of the parameters.
scores	The matrix of the scores (first derivative for each observation).
family	The ML family that was used for the estimation.
residuals	The difference between the dependent variable and the expected predictor.
sumFE	The sum of the fixed-effects for each observation.
offset	The offset formula.
NL.fml	The nonlinear formula of the call.
bounds	Whether the coefficients were upper or lower bounded. – This can only be the case when a non-linear formula is included and the arguments 'lower' or 'upper' are provided.
isBounded	The logical vector that gives for each coefficient whether it was bounded or not. This can only be the case when a non-linear formula is included and the arguments 'lower' or 'upper' are provided.
fixef_vars	The names of each cluster.
fixef_id	The list (of length the number of clusters) of the cluster identifiers for each observation.
fixef_sizes	The size of each cluster.
obsRemoved	In the case there were clusters and some observations were removed because of only 0/1 outcome within a cluster, it gives the row numbers of the observations that were removed.
fixef_removed	In the case there were clusters and some observations were removed because of only 0/1 outcome within a cluster, it gives the list (for each cluster) of the cluster identifiers that were removed.
theta	In the case of a negative binomial estimation: the overdispersion parameter.

@seealso See also summary.fixest to see the results with the appropriate standard-errors, fixef.fixest to extract the cluster coefficients, and the functions esttable and esttex to visualize the results of multiple estimations.

And other estimation methods: feols, femlm, feglm, fepois, fenegbin.

Author(s)

Laurent Berge

References

Berge, Laurent, 2018, "Efficient estimation of maximum likelihood models with multiple fixed-effects: the R package FENmlm." CREA Discussion Papers, 13 (https://wwwen.uni.lu/content/download/110162/1299525/file/2018_13).

For models with multiple fixed-effects:

Gaure, Simen, 2013, "OLS with multiple high dimensional category variables", Computational Statistics & Data Analysis 66 pp. 8–18

On the unconditionnal Negative Binomial model:

Allison, Paul D and Waterman, Richard P, 2002, "Fixed-Effects Negative Binomial Regression Models", Sociological Methodology 32(1) pp. 247–265

Examples

```
# This section covers only non-linear in parameters examples
# For linear relationships: use femlm instead

# Generating data for a simple example
n = 100
x = rnorm(n, 1, 5)**2
y = rnorm(n, -1, 5)**2
z1 = rpois(n, x*y) + rpois(n, 2)
base = data.frame(x, y, z1)

# Estimating a 'linear' relation:
est1_L = femlm(z1 ~ log(x) + log(y), base)
# Estimating the same 'linear' relation using a 'non-linear' call
est1_NL = feNmlm(z1 ~ 1, base, NL.fml = ~a*log(x)+b*log(y), NL.start = list(a=0, b=0))
# we compare the estimates with the function esttable (they are identical)
esttable(est1_L, est1_NL)

# Now generating a non-linear relation (E(z2) = x + y + 1):
z2 = rpois(n, x + y) + rpois(n, 1)
base$z2 = z2

# Estimation using this non-linear form
est2_NL = feNmlm(z2~0, base, NL.fml = ~log(a*x + b*y),
                 NL.start = list(a=1, b=2), lower = list(a=0, b=0))
# we can't estimate this relation linearily
# => closest we can do:
est2_L = femlm(z2~log(x)+log(y), base)

# Difference between the two models:
esttable(est2_L, est2_NL)

# Plotting the fits:
plot(x, z2, pch = 18)
points(x, fitted(est2_L), col = 2, pch = 1)
points(x, fitted(est2_NL), col = 4, pch = 2)
```

feols *Fixed-effects OLS estimation*

Description

Estimates OLS with any number of fixed-effects.

Usage

```
feols(fml, data, weights, offset, fixef, fixef.tol = 1e-07,
  fixef.iter = 2000, na_inf.rm = getFixest_na_inf.rm(),
  nthreads = getFixest_nthreads(), verbose = 0, warn = TRUE,
  notes = getFixest_notes(), combine.quick, ...)
```

Arguments

fml	A formula representing the relation to be estimated. For example: `fml = z~x+y`. To include fixed-effects, insert them in this formula using a pipe: e.g. `fml = z~x+y | fe_1+fe_2`. You can combine two clusters with `^`: e.g. `fml = z~x+y|fe_1^fe_2`, see details. You can also use variables with varying slopes using square brackets: e.g. in `fml = z~y|fe_1[x] + fe_2` the variable `x` will have one coefficient for each value of `fe_1` – if you use varying slopes, please have a look at the details section (can't describe it all here).
data	A data.frame containing the necessary variables to run the model. The variables of the non-linear right hand side of the formula are identified with this `data.frame` names. Can also be a matrix.
weights	A formula or a numeric vector. Each observation can be weighted, the weights must be greater than 0. If equal to a formula, it should be of one-sided: for example `~ var_weight`.
offset	A formula or a numeric vector. An offset can be added to the estimation. If equal to a formula, it should be of the form (for example) `~0.5*x**2`. This offset is linearly added to the elements of the main formula 'fml'.
fixef	Character vector. The name/s of a/some variable/s within the dataset to be used as fixed-effects. These variables should contain the identifier of each observation (e.g., think of it as a panel identifier).
fixef.tol	Precision used to obtain the fixed-effects (ie cluster coefficients). Defaults to `1e-5`. It corresponds to the maximum absolute difference allowed between two coefficients of successive iterations. Argument `fixef.tol` cannot be lower than `10000*.Machine$double.eps`. Note that this parameter is dynamically controlled by the algorithm.
fixef.iter	Maximum number of iterations in the step obtaining the fixed-effects (only in use for 2+ clusters). Default is 10000.

`na_inf.rm` Logical, default is `TRUE`. If the variables necessary for the estimation contain NA/Infs and `na_inf.rm = TRUE`, then all observations containing NA are removed prior to estimation and a note is displayed detailing the number of observations removed. Otherwise, an error is raised.

`nthreads` Integer: Number of nthreads to be used (accelerates the algorithm via the use of openMP routines). The default is to use the total number of nthreads available minus two. You can set permanently the number of nthreads used within this package using the function `setFixest_nthreads`.

`verbose` Integer, default is 0. It represents the level of information that should be reported during the optimisation process. If `verbose=0`: nothing is reported. If `verbose=1`: the value of the coefficients and the likelihood are reported. If `verbose=2`: 1 + information on the computing time of the null model, the cluster coefficients and the hessian are reported.

`warn` Logical, default is `TRUE`. Whether warnings should be displayed (concerns warnings relating to: convergence state, collinearity issues and observation removal due to only 0/1 outcomes or presence of NA values).

`notes` Logical. By default, two notes are displayed: when NAs are removed (to show additional information) and when some observations are removed because of only 0 (or 0/1) outcomes in a fixed-effect (in Poisson/Neg. Bin./Logit models). To avoid displaying these messages, you can set `notes = FALSE`. You can remove these messages permanently by using `setFixest_notes(FALSE)`.

`combine.quick` Logical. When you combine different variables to transform them into a single fixed-effects you can do e.g. `y ~ x | paste(var1,var2)`. The algorithm provides a shorthand to do the same operation: `y ~ x | var1^var2`. Because pasting variables is a costly operation, the internal algorithm may use a numerical trick to hasten the process. The cost of doing so is that you lose the labels. If you are interested in getting the value of the fixed-effects coefficients after the estimation, you should use `combine.quick = FALSE`. By default it is equal to `FALSE` if the number of observations is lower than 50,000, and to `TRUE` otherwise.

`...` Not currently used.

Details

The method used to demean each variable along the fixed-effects is based on Berge (2018), since this is the same problem to solve as for the Gaussian case in a ML setup.

Combining the fixed-effects

You can combine two variables to make it a new fixed-effect using `^`. The syntax is as follows: `fe_1^fe_2`. Here you created a new variable which is the combination of the two variables fe_1 and fe_2. This is identical to doing `paste0(fe_1,"_",fe_2)` but more convenient.

Note that pasting is a costly operation, especially for large data sets. Thus, the internal algorithm uses a numerical trick which is fast, but the drawback is that the identity of each observation is lost (i.e. they are now equal to a meaningless number instead of being equal to `paste0(fe_1,"_",fe_2)`). These "identities" are useful only if you're interested in the value of the fixed-effects (that you can extract with `fixef.fixest`). If you're only interested in coefficients of the variables, it doesn't matter. Anyway, you can use `combine.quick = FALSE` to tell the internal algorithm to use `paste` instead of the numerical trick. By default, the numerical trick is performed only for large data sets.

Varying slopes

You can add variables with varying slopes in the fixed-effect part of the formula. The syntax is as follows: cluster_var[var1, var2]. Here the variables var1 and var2 will be with varying slopes (one slope per value in cluster_var) and the fixed-effect cluster_var will also be added.

To add only the variables with varying slopes and not the fixed-effect, use double square brackets: cluster_var[[var1, var2]].

In other words:

- cluster_var[var1, var2] is equivalent to cluster_var + cluster_var[[var1]] + cluster_var[[var2]]
- cluster_var[[var1, var2]] is equivalent to cluster_var[[var1]] + cluster_var[[var2]]

@seealso See also `summary.fixest` to see the results with the appropriate standard-errors, `fixef.fixest` to extract the cluster coefficients, and the functions `esttable` and `esttex` to visualize the results of multiple estimations. And other estimation methods: `femlm`, `feglm`, `fepois`, `fenegbin`, `feNmlm`.

Author(s)

Laurent Berge

References

Berge, Laurent, 2018, "Efficient estimation of maximum likelihood models with multiple fixed-effects: the R package FENmlm." CREA Discussion Papers, 13 (https://wwwen.uni.lu/content/download/110162/1299525/file/2018_13).

For models with multiple fixed-effects:

Gaure, Simen, 2013, "OLS with multiple high dimensional category variables", Computational Statistics & Data Analysis 66 pp. 8–18

Examples

```
# just one set of fixed-effects:
res = feols(Sepal.Length ~ Sepal.Width + Petal.Length | Species, iris)
summary(res)

# now with varying slopes:
res = feols(Sepal.Length ~ Petal.Length | Species[Sepal.Width], iris)
summary(res)

# combining the FEs
base = iris
base$fe_2 = rep(1:10, 15)
res_comb = feols(Sepal.Length ~ Petal.Length | Species^fe_2, base)
summary(res_comb)
fixef(res_comb)[[1]]
```

`fitted.fixest` *Extracts fitted values from a* `fixest` *fit*

Description

This function extracts the fitted values from a model estimated with `femlm`, `feols` or `feglm`. The fitted values that are returned are the *expected predictor*.

Usage

```
## S3 method for class 'fixest'
fitted(object, type = c("response", "link"), ...)

## S3 method for class 'values.fixest'
fitted(object, type = c("response", "link"), ...)
```

Arguments

`object`	A `fixest` object. Obtained using the functions `femlm`, `feols` or `feglm`.
`type`	Character either equal to `"response"` (default) or `"link"`. If `type="response"`, then the output is at the level of the response variable, i.e. it is the expected predictor $E(Y\|X)$. If `"link"`, then the output is at the level of the explanatory variables, i.e. the linear predictor $X \cdot \beta$.
`...`	Not currently used.

Details

This function returns the *expected predictor* of a `fixest` fit. The likelihood functions are detailed in `femlm` help page.

Value

It returns a numeric vector of length the number of observations used to estimate the model.

If `type = "response"`, the value returned is the expected predictor, i.e. the expected value of the dependent variable for the fitted model: $E(Y|X)$. If `type = "link"`, the value returned is the linear predictor of the fitted model, that is $X \cdot \beta$ (remind that $E(Y|X) = f(X \cdot \beta)$).

Author(s)

Laurent Berge

See Also

See also the main estimation functions `femlm`, `feols` or `feglm`. `resid.fixest`, `predict.fixest`, `summary.fixest`, `vcov.fixest`, `fixef.fixest`.

Examples

```
# simple estimation on iris data, clustering by "Species"
res_poisson = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
                    Petal.Width | Species, iris)

# we extract the fitted values
y_fitted_poisson = fitted(res_poisson)

# Same estimation but in OLS (Gaussian family)
res_gaussian = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
                    Petal.Width | Species, iris, family = "gaussian")

y_fitted_gaussian = fitted(res_gaussian)

# comparison of the fit for the two families
plot(iris$Sepal.Length, y_fitted_poisson)
points(iris$Sepal.Length, y_fitted_gaussian, col = 2, pch = 2)
```

---

fixef.fixest *Extract the Fixed-Effects from a* fixest *estimation.*

Description

This function retrieves the fixed effects from a fixest estimation. It is useful only when there are one or more clusters.

Usage

```
## S3 method for class 'fixest'
fixef(object, notes = getFixest_notes(), ...)
```

Arguments

object	A fixest estimation (e.g. obtained using feols or feglm).
notes	Logical. Whether to display a note when the fixed-effects coefficients are not regular.
...	Not currently used.

Details

If the fixed-effect coefficients not regular, then several reference points need to be set, leading to the coefficients to be NOT interpretable. If this is the case, then a warning is raised.

Value

A list containing the vectors of the fixed effects.

If there is more than 1 fixed-effect, then the attribute "references" is created. This is a vector of length the number of fixed-effects, each element contains the number of coefficients set as references. By construction, the elements of the first fixed-effect dimension are never set as references. In the presence of regular fixed-effects, there should be Q-1 references (with Q the number of fixed-effects).

Author(s)

Laurent Berge

See Also

`plot.fixest.fixef`. See also the main estimation functions `femlm`, `feols` or `feglm`. Use `summary.fixest` to see the results with the appropriate standard-errors, `fixef.fixest` to extract the cluster coefficients, and the functions `esttable` and `esttex` to visualize the results of multiple estimations.

Examples

```
data(trade)

# We estimate the effect of distance on trade => we account for 3 fixed-effects
est_pois = femlm(Euros ~ log(dist_km)|Origin+Destination+Product, trade)

# Obtaining the fixed-effects coefficients:
fe_trade = fixef(est_pois)

# The fixed-effects of the first cluster:
head(fe_trade$Origin)

# Summary information:
summary(fe_trade)

# Plotting them:
plot(fe_trade)
```

`formula.fixest` *Extract the formula of a* `fixest` *fit*

Description

This function extracts the formula from a `fixest` estimation (obtained with `femlm`, `feols` or `feglm`). If the estimation was done with fixed-effects, they are added in the formula after a pipe ("|"). If the estimation was done with a non linear in parameters part, then this will be added in the formula in between `I()`.

Usage

```
## S3 method for class 'fixest'
formula(x, type = c("full", "linear", "NL"), ...)
```

Arguments

x	An object of class fixest. Typically the result of a femlm, feols or feglm estimation.
type	A character scalar. Default is type = "full" which gives back a formula containing the linear part of the model along with the clusters (if any) and the non-linear in parameters part (if any). If type = "linear" then only the linear formula is returned. If type = "NL" then only the non linear in parameters part is returned.
...	Not currently used.

Value

It returns a formula.

Author(s)

Laurent Berge

See Also

See also the main estimation functions femlm, feols or feglm. model.matrix.fixest, update.fixest, summary.fixest, vcov.fixest.

Examples

```
# simple estimation on iris data, clustering by "Species"
res = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
            Petal.Width | Species, iris)

# formula with the cluster variable
formula(res)
# linear part without the cluster variable
formula(res, "linear")
```

`lag.formula` *Lags a variable using a formula*

Description

Lags a variable using panel id+time identifiers in a formula.

Usage

```
## S3 method for class 'formula'
lag(x, k, data, time.step = "unitary", fill = NA,
  duplicate.method = c("none", "first"), ...)
```

Arguments

`x`	A formula of the type `var ~ id + time` where `var` is the variable to be lagged, `id` is a variable representing the panel id, and `time` is the time variable of the panel.
`k`	An integer giving the number of lags. For leads, just use a negative number.
`data`	Optional, the data.frame in which to evaluate the formula.
`time.step`	The method to compute the lags. Can be equal to: `"unitary"` (default), `"consecutive"` or to a number. If `"unitary"`, then the largest common divisor between consecutive time periods is used (typically if the time variable represents years, it will be 1). This method can apply only to integer (or convertible to integer) variables. If `"consecutive"`, then the time variable can be of any type: two successive time periods represent a lag of 1. Finally, if the time variable is numeric, you can provide your own numeric time step.
`fill`	How to fill the observations without defined lead/lag values. Default is `NA`.
`duplicate.method`	If several observations have the same id and time values, then the notion of lag is not defined for them. If `duplicate.method = "none"` (default) and duplicate values are found, this leads to an error. You can use `duplicate.method = "first"` so that the first occurrence of identical id/time observations will be used as lag.
`...`	Not currently used.

Value

It returns a vector of the same type and length as the variable to be lagged in the formula.

Author(s)

Laurent Berge

Examples

```
# simple example with an unbalanced panel
base = data.frame(id = rep(1:2, each = 4),
                  time = c(1, 2, 3, 4, 1, 4, 6, 9), x = 1:8)

lag(x~id+time,  1, base) # lag 1
lag(x~id+time, -1, base) # lead 1

lag(x~id+time, 2, base, fill = 0)

# with time.step = "consecutive"
lag(x~id+time, 1, base, time.step = "cons")
# => works for indiv. 2 because 9 (resp. 6) is consecutive to 6 (resp. 4)
# mostly useful when the time variable is not a number:
# e.g. c("1991q1", "1991q2", "1991q3") etc

# with duplicates
base_dup = data.frame(id = rep(1:2, each = 4),
                      time = c(1, 1, 1, 2, 1, 2, 2, 3), x = 1:8)

# by default: error

lag(x~id+time, 1, base_dup)

# with duplicate.method = "first"
lag(x~id+time, 1, base_dup, duplicate.method = "first")


# You can create variables without specifying the data within data.table:
library(data.table)
base = data.table(id = rep(1:2, each = 3), year = 1990 + rep(1:3, 2), x = 1:6)
base[, x.l1 := lag(x~id+year, 1)]
```

---

logLik.fixest *Extracts the log-likelihood*

Description

This function extracts the log-likelihood from a `fixest` estimation.

Usage

```
## S3 method for class 'fixest'
logLik(object, ...)
```

Arguments

`object`	A `fixest` object. Obtained using the functions `femlm`, `feols` or `feglm`.
`...`	Not currently used.

Details

This function extracts the log-likelihood based on the model fit. You can have more information on the likelihoods in the details of the function `femlm`.

Value

It returns a numeric scalar.

Author(s)

Laurent Berge

See Also

See also the main estimation functions `femlm`, `feols` or `feglm`. Other statistics functions: `AIC.fixest`, `BIC.fixest`.

Examples

```
# simple estimation on iris data with "Species" fixed-effects
res = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
            Petal.Width | Species, iris)

nobs(res)
logLik(res)
```

`model.matrix.fixest` *Design matrix of a* `femlm` *model*

Description

This function creates a design matrix of the linear part of a `femlm`, `feols` or `feglm` estimation. Note that it is only the linear part and the cluster variables (which can be considered as factors) are excluded from the matrix.

Usage

```
## S3 method for class 'fixest'
model.matrix(object, data, ...)
```

Arguments

object	A fixest object. Obtained using the functions femlm, feols or feglm.
data	If missing (default) then the original data is obtained by evaluating the call. Otherwise, it should be a data.frame.
...	Not currently used.

Value

It returns a design matrix.

Author(s)

Laurent Berge

See Also

See also the main estimation functions femlm, feols or feglm. formula.fixest, update.fixest, summary.fixest, vcov.fixest.

Examples

```
# simple estimation on iris data, clustering by "Species"
res = femlm(Sepal.Length ~ Sepal.Width*Petal.Length +
            Petal.Width | Species, iris)

head(model.matrix(res))
```

---

nobs.fixest *Extracts the number of observations form a* fixest *object*

Description

This function simply extracts the number of observations form a fixest object, obtained using the functions femlm, feols or feglm.

Usage

```
## S3 method for class 'fixest'
nobs(object, ...)
```

Arguments

object	A fixest object. Obtained using the functions femlm, feols or feglm.
...	Not currently used.

Value

It returns an interger.

Author(s)

Laurent Berge

See Also

See also the main estimation functions `femlm`, `feols` or `feglm`. Use `summary.fixest` to see the results with the appropriate standard-errors, `fixef.fixest` to extract the cluster coefficients, and the functions `esttable` and `esttex` to visualize the results of multiple estimations.

Examples

```
# simple estimation on iris data with "Species" fixed-effects
res = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
            Petal.Width | Species, iris)

nobs(res)
logLik(res)
```

---

obs2remove *Finds observations to be removed from ML estimation with factors/clusters*

Description

For Poisson, Negative Binomial or Logit estimations with fixed-effects, when the dependent variable is only equal to 0 (or 1 for Logit) for one cluster value this leads to a perfect fit for that cluster value by setting its associated cluster coefficient to `-Inf`. Thus these observations need to be removed before estimation. This function gives the observations to be removed. Note that by default the function `femlm` or `feglm` drops them before performing the estimation.

Usage

```
obs2remove(fml, data, family = c("poisson", "negbin", "logit"))
```

Arguments

`fml`	A formula containing the dependent variable and the clusters. It can be of the type: `y ~ cluster_1 + cluster_2` or `y ~ x1	cluster_1 + cluster_1` (in which case variables before the pipe are ignored).
`data`	A data.frame containing the variables in the formula.
`family`	Character scalar: either "poisson" (default), "negbin" or "logit".

Value

It returns an integer vector of observations to be removed. If no observations are to be removed, an empty integer vector is returned. In both cases, it is of class `fixest.obs2remove`. The vector has an attribute `cluster` which is a list giving the IDs of the clusters that have been removed, for each cluster dimension.

Examples

```
base = iris
# v6: Petal.Length with only 0 values for 'setosa'
base$v6 = base$Petal.Length
base$v6[base$Species == "setosa"] = 0

(x = obs2remove(v6 ~ Species, base))
attr(x, "cluster")

# The two results are identical:
res_1 = femlm(v6 ~ Petal.Width | Species, base)
# => note + obsRemoved is created

res_2 = femlm(v6 ~ Petal.Width | Species, base[-x, ])
# => no note because observations are removed before

esttable(res_1, res_2)

all(res_1$obsRemoved == x)
```

---

plot.fixest.fixef *Displaying the most notable fixed-effects*

Description

This function plots the 5 fixed-effects with the highest and lowest values, for each of the clusters. It takes as an argument the fixed-effects obtained from the function `fixef.fixest` after an estimation using `femlm`, `feols` or `feglm`.

Usage

```
## S3 method for class 'fixest.fixef'
plot(x, n = 5, ...)
```

Arguments

x	An object obtained from the function `fixef.fixest`.
n	The number of fixed-effects to be drawn. Defaults to 5.

`...` Not currently used.

Note that the fixed-effect coefficients might NOT be interpretable. This function is useful only for fully regular panels.

If the data are not regular in the cluster coefficients, this means that several 'reference points' are set to obtain the fixed-effects, thereby impeding their interpretation. In this case a warning is raised.

Author(s)

Laurent Berge

See Also

`fixef.fixest` to extract clouster coefficients. See also the main estimation function `femlm`, `feols` or `feglm`. Use `summary.fixest` to see the results with the appropriate standard-errors, the functions `esttable` and `esttex` to visualize the results of multiple estimations.

Examples

```
data(trade)

# We estimate the effect of distance on trade
# => we account for 3 fixed-effects
est_pois = femlm(Euros ~ log(dist_km)|Origin+Destination+Product, trade)

# obtaining the fixed-effects coefficients
fe_trade = fixef(est_pois)

# plotting them
plot(fe_trade)
```

---

predict.fixest *Predict method for* `fixest` *fits*

Description

This function obtains prediction from a fitted model estimated with `femlm`, `feols` or `feglm`.

Usage

```
## S3 method for class 'fixest'
predict(object, newdata, type = c("response", "link"),
  ...)
```

Arguments

object	A fixest object. Obtained using the functions femlm, feols or feglm.
newdata	A data.frame containing the variables used to make the prediction. If not provided, the fitted expected (or linear if type = "link") predictors are returned.
type	Character either equal to "response" (default) or "link". If type="response", then the output is at the level of the response variable, i.e. it is the expected predictor $E(Y\|X)$. If "link", then the output is at the level of the explanatory variables, i.e. the linear predictor $X \cdot \beta$.
...	Not currently used.

Value

It returns a numeric vector of length equal to the number of observations in argument newdata.

Author(s)

Laurent Berge

See Also

See also the main estimation functions femlm, feols or feglm. update.fixest, summary.fixest, vcov.fixest, fixef.fixest.

Examples

```
# Estimation on iris data
res = femlm(Sepal.Length ~ Petal.Length | Species, iris)

# what would be the prediction if the data was all setosa?
newdata = data.frame(Petal.Length = iris$Petal.Length, Species = "setosa")
pred_setosa = predict(res, newdata = newdata)

# Let's look at it graphically
plot(c(1, 7), c(3, 11), type = "n", xlab = "Petal.Length",
     ylab = "Sepal.Length")

newdata = iris[order(iris$Petal.Length), ]
newdata$Species = "setosa"
lines(newdata$Petal.Length, predict(res, newdata))

# versicolor
newdata$Species = "versicolor"
lines(newdata$Petal.Length, predict(res, newdata), col=2)

# virginica
newdata$Species = "virginica"
lines(newdata$Petal.Length, predict(res, newdata), col=3)

# The original data
```

```
points(iris$Petal.Length, iris$Sepal.Length, col = iris$Species, pch = 18)
legend("topleft", lty = 1, col = 1:3, legend = levels(iris$Species))
```

`print.fixest` *A print facility for* `fixest` *objects.*

Description

This function is very similar to usual `summary` functions as it provides the table of coefficients along with other information on the fit of the estimation. The type of output is customizable by the user (using function `setFixest_print.type`).

Usage

```
## S3 method for class 'fixest'
print(x, n, type = getFixest_print.type(), ...)
```

Arguments

`x`	A `fixest` object. Obtained using the methods `femlm`, `feols` or `feglm`.
`n`	Integer, number of coefficients to display. By default, only the first 8 coefficients are displayed if `x` does not come from `summary.fixest`.
`type`	Either `"table"` (default) to display the coefficients table or `"coef"` to display only the coefficients. By default the value is `getFixest_print.type()` which can be permanently set with `setFixest_print.type`).
`...`	Other arguments to be passed to `vcov.fixest`.

Author(s)

Laurent Berge

See Also

See also the main estimation functions `femlm`, `feols` or `feglm`. Use `summary.fixest` to see the results with the appropriate standard-errors, `fixef.fixest` to extract the cluster coefficients, and the functions `esttable` and `esttex` to visualize the results of multiple estimations.

Examples

```
# Load trade data
data(trade)

# We estimate the effect of distance on trade
#   => we account for 3 fixed-effects (FEs)
est_pois = femlm(Euros ~ log(dist_km)|Origin+Destination+Product, trade)
```

```
# displaying the results
#  (by default SEs are clustered if FEs are used)
print(est_pois)

# By default the coefficient table is displayed.
#  If the user wished to display only the coefficents, use option type:
print(est_pois, type = "coef")

# To permanently display coef. only, use setFixest_print.type:
setFixest_print.type("coef")
est_pois
# back to default:
setFixest_print.type("table")
```

`r2` *R2s of* `fixest` *models*

Description

Reports different R2s for `fixest` estimations (e.g. `feglm` or `feols`).

Usage

```
r2(x, type = "all")
```

Arguments

`x`	A `fixest` object, e.g. obtained with function `feglm` or `feols`.
`type`	A character vector representing the R2 to compute. The R2 codes are of the form: "wapr2" with letters "w" (within), "a" (adjusted) and "p" (pseudo) possibly missing. E.g. to get the regular R2: use `type = "r2"`, the within adjusted R2: use `type = "war2"`, the pseudo R2: use `type = "pr2"`, etc. Use `"sq.cor"` for the squared correlation. By default, all R2s are computed.

Details

For R2s with no theoretical justification, like e.g. regular R2s for maximum likelihood models – or within R2s for models without fixed-effects, NA is returned. The single measure to possibly compare all kinds of models is the squared correlation between the dependent variable and the expected predictor.

Value

Returns a named vector.

Author(s)

Laurent Berge

Examples

```
# Load trade data
data(trade)

# We estimate the effect of distance on trade (with 3 fixed-effects)
est_pois = femlm(Euros ~ log(dist_km)|Origin+Destination+Product, trade)

# Squared correlation:
r2(est_pois, "sq.cor")

# "regular" r2:
r2(est_pois, "r2")

# pseudo r2
r2(est_pois, "pr2")

# within adjusted r2
r2(est_pois, "war2")

# all four at once
r2(est_pois, c("sq.cor", "r2", "pr2", "war2"))
```

---

resid.fixest *Extracts residuals from a* `fixest` *object*

Description

This function extracts residuals from a fitted model estimated with `femlm`, `feols` or `feglm`.

Usage

```
## S3 method for class 'fixest'
resid(object, ...)

## S3 method for class 'fixest'
residuals(object, ...)
```

Arguments

`object`	A `fixest` object. Obtained using the functions `femlm`, `feols` or `feglm`.
`...`	Not currently used.

Details

The residuals returned are the difference between the dependent variable and the expected predictor.

Value

It returns a numeric vector of the length the number of observations used for the estimation.

Author(s)

Laurent Berge

See Also

See also the main estimation functions `femlm`, `feols` or `feglm`. `fitted.fixest`, `predict.fixest`, `summary.fixest`, `vcov.fixest`, `fixef.fixest`.

Examples

```
# simple estimation on iris data, clustering by "Species"
res_poisson = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
                    Petal.Width | Species, iris)

# we plot the residuals
plot(resid(res_poisson))
```

---

setFixest_dict *Sets/gets the dictionary used in* esttex

Description

Sets/gets the default dictionary used in the function `esttex`. The dictionaries are used to relabel variables (usually towards a fancier, more explicit formatting) when exporting them into a Latex table. By setting the dictionary with `setFixest_dict`, you can avoid providing the argument `dict` in function `esttex`.

Usage

```
setFixest_dict(dict)

getFixest_dict()
```

Arguments

`dict`	A named character vector. E.g. to change my variable named "a" and "b" to (resp.) "$log(a)$" and "$bonus^3$", then use `dict = c(a="$log(a)$",b3="$bonus^3$")`.

Author(s)

Laurent Berge

Examples

```
data(trade)
est = feols(log(Euros) ~ log(dist_km)|Origin+Destination+Product, trade)
# we export the result & rename some variables
esttex(est, dict = c("log(Euros)"="Euros (ln)", Origin="Country of Origin"))
# If you export many tables, it can be more convenient to use setFixest_dict:
setFixest_dict(c("log(Euros)"="Euros (ln)", Origin="Country of Origin"))
esttex(est) # variables are properly relabeled
```

---

setFixest_na_inf.rm *Sets/gets whether to remove NA/Inf values from* fixest *estimations*

Description

Sets/gets the default policy of NA/Inf behavior in `fixest` estimations. By default, NA/Inf values are removed (and a note is displayed). If you prefer a no NA policy, just set `setFixest_na_inf.rm(FALSE)`.

Usage

```
setFixest_na_inf.rm(x)

getFixest_na_inf.rm()
```

Arguments

x	A Logical.

Author(s)

Laurent Berge

Examples

```
base = iris
base[1, 1] = NA
# default: NAs removed
res = feols(Sepal.Length ~ Sepal.Width, base)
# no tolerance: estimation fails
res = feols(Sepal.Length ~ Sepal.Width, base, na_inf.rm = FALSE)
```

```
# to set no tolerance as default:
setFixest_na_inf.rm(FALSE)
```

`setFixest_notes` *Sets/gets whether to display notes in* `fixest` *estimation functions*

Description

Sets/gets the default values of whether notes (informing for NA and observations removed) should be displayed in `fixest` estimation functions.

Usage

```
setFixest_notes(x)

getFixest_notes()
```

Arguments

`x`	A logical. If `FALSE`, then notes are permanently removed.

Author(s)

Laurent Berge

Examples

```
# Default is TRUE
getFixest_notes()
# Change default with
setFixest_notes(FALSE)
```

setFixest_nthreads *Sets/gets the number of threads to use in* fixest *functions*

Description

Sets/gets the default number of threads to used in fixest estimation functions. The default is the maximum number of threads minus two.

Usage

```
setFixest_nthreads(nthreads)

getFixest_nthreads()
```

Arguments

nthreads	An integer strictly greater than one and lower than the maximum number of threads (if OpenMP is available). If missing, the default is the maximum number of threads minus two.

Author(s)

Laurent Berge

Examples

```
# Gets the current number of threads
getFixest_nthreads()
# To set multi-threading off:
setFixest_nthreads(1)
# To set it back to default:
setFixest_nthreads()
```

setFixest_print.type *Sets/gets what* print *does to* fixest *estimations*

Description

Sets/gets the default behavior of the print method for non-summary fixest estimations. Default is to display the coefficients table but it can be changed to displaying only the coefficients.

Usage

```
setFixest_print.type(x)

getFixest_print.type()
```

Arguments

x	Either "table" or "coef".

Author(s)

Laurent Berge

Examples

```
res = feols(Sepal.Length ~ Sepal.Width + Petal.Length, base)
# default is coef. table:
res
# can be changed to only the coefficients:
print(res, type = "coef")
setFixest_print.type("coef")
res # only the coefs
```

---

summary.fixest *Summary of a* fixest *object. Computes different types of standard errors.*

Description

This function is similar to print.fixest. It provides the table of coefficients along with other information on the fit of the estimation. It can compute different types of standard errors. The new variance covariance matrix is an object returned.

Usage

```
## S3 method for class 'fixest'
summary(object, se, cluster, dof = TRUE,
  exact_dof = FALSE, forceCovariance = FALSE, keepBounded = FALSE,
  ...)
```

Arguments

object	A fixest object. Obtained using the functions femlm, feols or feglm.
se	Character scalar. Which kind of standard error should be computed: "standard", "White", "cluster", "twoway", "threeway" or "fourway"? By default if there are clusters in the estimation: se = "cluster", otherwise se = "standard". Note that this argument can be implicitly deduced from the argument cluster.
cluster	Tells how to cluster the standard-errors (if clustering is requested). Can be either a list of vectors, a character vector of variable names, a formula or an integer vector. Assume we want to perform 2-way clustering over var1 and var2 contained in the data.frame base used for the estimation. All the following cluster arguments are valid and do the same thing: cluster = base[,c("var1","var2")]},\code{cluster = c("var1","var2"), cluster = ~var1+var2. If the two variables were used as clusters in the estimation, you could further use cluster = 1:2 or leave it blank with se = "twoway" (assuming var1 [resp. var2] was the 1st [res. 2nd] cluster).
dof	Logical, default is TRUE. Should there be a degree of freedom correction to the standard errors of the coefficients?
exact_dof	Logical, default is FALSE. In case there were 2+ clusters in the estimation, it computes the exact number of degrees of freedom (this is not needed in case of balanced panels).
forceCovariance	(Advanced users.) Logical, default is FALSE. In the peculiar case where the obtained Hessian is not invertible (usually because of collinearity of some variables), use this option to force the covariance matrix, by using a generalized inverse of the Hessian. This can be useful to spot where possible problems come from.
keepBounded	(Advanced users – feNmlm with non-linear part and bounded coefficients only.) Logical, default is FALSE. If TRUE, then the bounded coefficients (if any) are treated as unrestricted coefficients and their S.E. is computed (otherwise it is not).
...	Not currently used.

Value

It returns a fixest object with:

cov.scaled	The new variance-covariance matrix (computed according to the argument se).
se	The new standard-errors (computed according to the argument se).
coeftable	The table of coefficients with the new standard errors.

Author(s)

Laurent Berge

See Also

See also the main estimation functions femlm, feols or feglm. Use fixef.fixest to extract the cluster coefficients, and the functions esttable and esttex to visualize the results of multiple estimations.

Examples

```
# Load trade data
data(trade)

# We estimate the effect of distance on trade (with 3 cluster effects)
est_pois = femlm(Euros ~ log(dist_km)|Origin+Destination+Product, trade)

# Comparing different types of standard errors
sum_white    = summary(est_pois, se = "white")
sum_oneway   = summary(est_pois, se = "cluster")
sum_twoway   = summary(est_pois, se = "twoway")
sum_threeway = summary(est_pois, se = "threeway")

esttable(sum_white, sum_oneway, sum_twoway, sum_threeway)

# Alternative ways to cluster the SE:

# two-way clustering: Destination and Product
# (Note that arg. se = "twoway" is implicitly deduced from the argument cluster)
summary(est_pois, cluster = c("Destination", "Product"))
summary(est_pois, cluster = trade[, c("Destination", "Product")])
summary(est_pois, cluster = list(trade$Destination, trade$Product))
summary(est_pois, cluster = ~Destination+Product)
# Since Destination and Product are used as fixed-effects, you can also use:
summary(est_pois, cluster = 2:3)
```

`summary.fixest.fixef` *Summary method for cluster coefficients*

Description

This function summarizes the main characteristics of the cluster coefficients. It shows the number of fixed-effects that have been set as references and the first elements of the fixed-effects.

Usage

```
## S3 method for class 'fixest.fixef'
summary(object, n = 5, ...)
```

Arguments

`object`	An object returned by the function `fixef.fixest`.
`n`	Positive integer, defaults to 5. The `n` first fixed-effects for each cluster are reported.
`...`	Not currently used.

Value

It prints the number of fixed-effect coefficients per cluster, as well as the number of fixed-effects used as references for each cluster, and the mean and variance of the cluster coefficients. Finally it reports the first 5 elements of each cluster.

Author(s)

Laurent Berge

See Also

femlm, fixef.fixest, plot.fixest.fixef.

Examples

```
data(trade)

# We estimate the effect of distance on trade
# => we account for 3 fixed-effects effects
est_pois = femlm(Euros ~ log(dist_km)|Origin+Destination+Product, trade)

# obtaining the fixed-effects coefficients
fe_trade = fixef(est_pois)

# printing some summary information on the cluster coefficients:
summary(fe_trade)
```

---

summary.fixest.obs2remove

Summary method for fixest.obs2remove objects

Description

This function synthesizes the information of function obs2remove. It reports the number of observations to be removed as well as the number of clusters removed per cluster dimension.

Usage

```
## S3 method for class 'fixest.obs2remove'
summary(object, ...)
```

Arguments

object	A fixest.obs2remove object obtained from function obs2remove.
...	Not currently used.

Examples

```
base = iris
# v6: Petal.Length with only 0 values for 'setosa'
base$v6 = base$Petal.Length
base$v6[base$Species == "setosa"] = 0

x = obs2remove(v6 ~ Species, base)
summary(x)
```

`trade` *Trade data sample*

Description

This data reports trade information between countries of the European Union (EU15).

Usage

```
data(trade)
```

Format

`trade` is a data frame with 38,325 observations and 6 variables named `Destination`, `Origin`, `Product`, `Year`, `dist_km` and `Euros`.

- Origin: 2-digits codes of the countries of origin of the trade flow.
- Destination: 2-digits codes of the countries of destination of the trade flow.
- Products: Number representing the product categories (from 1 to 20).
- Year: Years from 2007 to 2016
- dist_km: Geographic distance in km between the centers of the countries of origin and destination.
- Euros: The total amount in euros of the trade flow for the specific year/product category/origin-destination country pair.

Source

This data has been extrated from Eurostat on October 2017.

`update.fixest` *Updates a* `fixest` *estimation*

Description

Updates and re-estimates a `fixest` model (estimated with `femlm`, `feols` or `feglm`). This function updates the formulas and use previous starting values to estimate a new `fixest` model. The data is obtained from the original `call`.

Usage

```
## S3 method for class 'fixest'
update(object, fml.update, nframes = 1, ...)
```

Arguments

`object`	A `fixest` object. Obtained using the functions `femlm`, `feols` or `feglm`.
`fml.update`	Changes to be made to the original argument `fml`. See more information on `update.formula`. You can add/withdraw both variables and clusters. E.g. `. ~ . + x2 | . + z2` would add the variable `x2` and the cluster `z2` to the former estimation.
`nframes`	(Advanced users.) Defaults to 1. Number of frames up the stack where to perform the evaluation of the updated call. By default, this is the parent frame.
`...`	Other arguments to be passed to the functions `femlm`, `feols` or `feglm`.

Value

It returns a `fixest` object (see details in `femlm`, `feols` or `feglm`).

Author(s)

Laurent Berge

See Also

See also the main estimation functions `femlm`, `feols` or `feglm`. `predict.fixest`, `summary.fixest`, `vcov.fixest`, `fixef.fixest`.

Examples

```
# Example using trade data
data(trade)

# main estimation
est_pois <- femlm(Euros ~ log(dist_km) | Origin + Destination, trade)
```

```
# we add the variable log(Year)
est_2 <- update(est_pois, . ~ . + log(Year))

# we add another cluster: "Product"
est_3 <- update(est_2, . ~ . | . + Product)

# we remove the cluster "Origin" and the variable log(dist_km)
est_4 <- update(est_3, . ~ . - log(dist_km) | . - Origin)

# Quick look at the 4 estimations
esttable(est_pois, est_2, est_3, est_4)
```

`vcov.fixest` *Extracts the variance/covariance of a* `femlm` *fit*

Description

This function extracts the variance-covariance of estimated parameters from a model estimated with `femlm`, `feols` or `feglm`.

Usage

```
## S3 method for class 'fixest'
vcov(object, se, cluster, dof = TRUE,
  exact_dof = FALSE, forceCovariance = FALSE, keepBounded = FALSE,
  ...)
```

Arguments

`object`	A `fixest` object. Obtained using the functions `femlm`, `feols` or `feglm`.
`se`	Character scalar. Which kind of standard error should be computed: "standard", "White", "cluster", "twoway", "threeway" or "fourway"? By default if there are clusters in the estimation: `se = "cluster"`, otherwise `se = "standard"`. Note that this argument can be implicitly deduced from the argument `cluster`.
`cluster`	Tells how to cluster the standard-errors (if clustering is requested). Can be either a list of vectors, a character vector of variable names, a formula or an integer vector. Assume we want to perform 2-way clustering over `var1` and `var2` contained in the data.frame `base` used for the estimation. All the following `cluster` arguments are valid and do the same thing: `cluster = base[,c("var1","var2")]`,\code{cluster = c("var1","var2")}, `cluster = ~var1+var2`. If the two variables were used as clusters in the estimation, you could further use `cluster = 1:2` or leave it blank with `se = "twoway"` (assuming `var1` [resp. `var2`] was the 1st [res. 2nd] cluster).
`dof`	Logical, default is `TRUE`. Should there be a degree of freedom correction to the standard errors of the coefficients?
`exact_dof`	Logical, default is `FALSE`. In case there were 2+ clusters in the estimation, it computes the exact number of degrees of freedom (this is not needed in case of balanced panels).

forceCovariance
: (Advanced users.) Logical, default is FALSE. In the peculiar case where the obtained Hessian is not invertible (usually because of collinearity of some variables), use this option to force the covariance matrix, by using a generalized inverse of the Hessian. This can be useful to spot where possible problems come from.

keepBounded
: (Advanced users – feNmlm with non-linear part and bounded coefficients only.) Logical, default is FALSE. If TRUE, then the bounded coefficients (if any) are treated as unrestricted coefficients and their S.E. is computed (otherwise it is not).

...
: Other arguments to be passed to summary.fixest.

The computation of the VCOV matrix is first done in summary.fixest.

Value

It returns a $N \times N$ square matrix where N is the number of variables of the fitted model. This matrix has an attribute "type" specifying how this variance/covariance matrix has been computed (i.e. was it created using White correction, or was it clustered along a specific factor, etc).

Author(s)

Laurent Berge

See Also

See also the main estimation functions femlm, feols or feglm. summary.fixest, confint.fixest, resid.fixest, predict.fixest, fixef.fixest.

Examples

```
# Load trade data
data(trade)

# We estimate the effect of distance on trade (with 3 fixed-effects)
est_pois = femlm(Euros ~ log(dist_km) + log(Year) | Origin + Destination +
                 Product, trade)

# By default, in the presence of FEs
# the VCOV is clustered along the first FE
vcov(est_pois)

# "white" VCOV
vcov(est_pois, se = "white")

# "clustered" VCOV (with respect to the Product factor)
vcov(est_pois, se = "cluster", cluster = trade$Product)
# another way to make the same request:
# note that previously arg. se was optional since deduced from arg. cluster
vcov(est_pois, cluster = "Product")
# yet another way:
```

```
vcov(est_pois, cluster = ~Product)

# Another estimation without cluster:
est_pois_simple = femlm(Euros ~ log(dist_km) + log(Year), trade)

# We can still get the clustered VCOV,
# but we need to give the argument cluster:
vcov(est_pois_simple, cluster = ~Product)
```

Index