Package 'fixest'

July 13, 2020

Type Package

Title Fast Fixed-Effects Estimations

Version 0.6.0

Imports stats, graphics, grDevices, utils, Formula, numDeriv, nlme,
sandwich, Rcpp, dreamerr(>= 1.2.0)

Suggests knitr, rmarkdown, data.table, plm, lfe, MASS

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 7.1.0

Encoding UTF-8

NeedsCompilation yes

Author Laurent Berge [aut, cre]

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

Repository CRAN

Date/Publication 2020-07-13 11:50:03 UTC

R **topics documented:**

Index **116**

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)
```

---

bread.fixest   *Extracts the bread matrix from fixest objects*

Description

Extracts the bread matrix from fixest objects to be used to compute sanwich variance-covariance matrices.

Usage

```
## S3 method for class 'fixest'
bread(x, ...)
```

Arguments

x   A `fixest` object, obtained for instance from [feols](#).

...   Not currently used.

Value

Returns a matrix of the same dimension as the number of variables used in the estimation.

Examples

```
est = feols(Petal.Length ~ Petal.Width + Sepal.Width, iris)
bread(est)
```

---

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, ...)

coefficients.fixest
```

Arguments

object   A fixest object. Obtained using the functions femlm, feols or feglm.

...   Not currently used.

Format

An object of class function of length 1.

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 fixed-effects, to obtain the fixed-effect 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, etable, fixef.fixest.

Examples

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

# the coefficients of the variables:
coef(res)

# the fixed-effects coefficients:
fixef(res)
```

---

coefplot   *Plots confidence intervals and point estimates*

Description

This function plots the results of estimations (coefficients and confidence intervals). It is flexible and handles interactions in a special way.

Usage

```
coefplot(
  object,
  ...,
  style,
  sd,
  ci_low,
  ci_high,
  x,
  x.shift = 0,
  horiz = FALSE,
  dict = getFixest_dict(),
  keep,
  drop,
  order,
  ci.width = "1%",
  ci_level = 0.95,
  add = FALSE,
  pt.pch = 20,
  pt.bg = NULL,
  cex = 1,
  pt.cex = cex,
  col = 1:8,
  pt.col = col,
```

```
        ci.col = col,
        lwd = 1,
        pt.lwd = lwd,
        ci.lwd = lwd,
        ci.lty = 1,
        grid = TRUE,
        grid.par = list(lty = 3, col = "gray"),
        zero = TRUE,
        zero.par = list(col = "black", lwd = 1),
        pt.join = FALSE,
        pt.join.par = list(col = pt.col, lwd = lwd),
        ci.join = FALSE,
        ci.join.par = list(lwd = lwd, col = col, lty = 2),
        ci.fill = FALSE,
        ci.fill.par = list(col = "lightgray", alpha = 0.5),
        ref = "auto",
        ref.line = "auto",
        ref.line.par = list(col = "black", lty = 2),
        lab.cex,
        lab.min.cex = 0.85,
        lab.max.mar = 0.25,
        lab.fit = "auto",
        xlim.add,
        ylim.add,
        only.params = FALSE,
        only.inter = TRUE,
        sep,
        as.multiple = FALSE,
        bg,
        group = "auto",
        group.par = list(lwd = 2, line = 3, tcl = 0.75),
        main = "Effect on __depvar__",
        value.lab = "Estimate and __ci__ Conf. Int.",
        ylab = NULL,
        xlab = NULL,
        sub = NULL
    )
```

Arguments

object
: Can be either: i) an estimation object (obtained for example from [feols](#), ii) a list of estimation objects (several results will be plotted at once), iii) a matrix of coefficients table, iv) a numeric vector of the point estimates – the latter requiring the extra arguments sd or ci_low and ci_high.

...
: Other arguments to be passed to summary, if object is an estimation, and/or to the function plot or lines (if add = TRUE).

style
: A character scalar giving the style of the plot to be used. You can set styles with the function [setFixest_coefplot](#), setting all the default values of the func-

	tion. If missing, then it switches to either "default", "interaction" or "multiple", depending on the data given in input.
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_high	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 are used.
x.shift	Shifts the confidence intervals bars to the left or right, depending on the value of x.shift. Default is 0.
horiz	A logical scalar, default is FALSE. Whether to display the confidence intervals horizontally instead of vertically.
dict	A named character vector or a logical scalar. It changes the original variable names to the ones contained in the dictionary. 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, if Tex output is requested or if argument file is not missing, it is equal to getFixest_dict(), a default dictionary which can be set with [setFixest_dict](). The default is not to change names if a data.frame is requested (i.e. tex = FALSE); if so, you can use dict = TRUE to use the dictionary you've set globally with setFixest_dict().
keep	Character vector. This element is used to display only a subset of variables. This should be a vector of regular expressions (see [regex] help for more info). Each variable satisfying any of the regular expressions will be kept. This argument is applied post aliasing (see argument dict). Example: you have the variable x1 to x55 and want to display only x1 to x9, then you could use keep = "x[[:digit:]]$". If the first character is an exclamation mark, the effect is reversed (e.g. keep = "!Intercept" means: every variable that does not contain "Intercept" is kept). See details.
drop	Character vector. This element is used if some variables are not to be displayed. This should be a vector of regular expressions (see [regex] help for more info). Each variable satisfying any of the regular expressions will be discarded. This argument is applied post aliasing (see argument dict). Example: you have the variable x1 to x55 and want to display only x1 to x9, then you could use drop = "x[[:digit:]]{2}". If the first character is an exclamation mark, the effect is reversed (e.g. drop = "!Intercept" means: every variable that does not contain "Intercept" is dropped). See details.
order	Character vector. This element is used if the user wants the variables to be ordered in a certain way. This should be a vector of regular expressions (see [regex] help for more info). The variables satisfying the first regular expression will be placed first, then the order follows the sequence of regular expressions. This argument is applied post aliasing (see argument dict). Example: you have the following variables: month1 to month6, then x1 to x5, then year1 to year6. If you want to display first the x's, then the years, then the months you could use: order = c("x","year"). If the first character is an exclamation mark, the

	effect is reversed (e.g. order = "!Intercept" means: every variable that does not contain "Intercept" goes first). See details.
ci.width	The width of the extremities of the confidence intervals. Default is 0.1.
ci_level	Scalar between 0 and 1: the level of the CI. By default it is equal to 0.95.
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.
pt.pch	The patch of the coefficient estimates. Default is 1 (circle).
pt.bg	The background color of the point estimate (when the pt.pch is in 21 to 25). Defaults to NULL.
cex	Numeric, default is 1. Expansion factor for the points
pt.cex	The size of the coefficient estimates. Default is the other argument cex.
col	The color of the points and the confidence intervals. Default is 1 ("black"). Note that you can set the colors separately for each of them with pt.col and ci.col.
pt.col	The color of the coefficient estimates. Default is equal to the other argument col.
ci.col	The color of the confidence intervals. Default is equal to the other argument col.
lwd	General liwe with. Default is 1.
pt.lwd	The line width of the coefficient estimates. Default is equal to the other argument lwd.
ci.lwd	The line width of the confidence intervals. Default is equal to the other argument lwd.
ci.lty	The line type of the confidence intervals. Default is 1.
grid	Logical, default is TRUE. Whether a grid should be displayed. You can set the display of the grid with the argument grid.par.
grid.par	List. Parameters of the grid. The default values are: lty = 3 and col = "gray". You can add any graphical parameter that will be passed to abline. You also have two additional arguments: use horiz = FALSE to disable the horizontal lines, and use vert = FALSE to disable the vertical lines. Eg: grid.par = list(vert = FALSE, col = "red", lwd = 2).
zero	Logical, default is TRUE. Whether the 0-line should be emphasized. You can set the parameters of that line with the argument zero.par.
zero.par	List. Parameters of the zero-line. The default values are col = "black" and lwd = 1. You can add any graphical parameter that will be passed to abline. Example: zero.par = list(col = "darkblue", lwd = 3).
pt.join	Logical, default depends on the situation. If TRUE, then the coefficient estimates are joined with a line. By default, it is equal to TRUE only if: i) interactions are plotted, ii) the x values are numeric and iii) a reference is found.
pt.join.par	List. Parameters of the line joining the cofficients. The default values are: col = pt.col and lwd = lwd. You can add any graphical parameter that will be passed to lines. Eg: pt.join.par = list(lty = 2).

ci.join	Logical default to FALSE. Whether to join the extremities of the confidence intervals. If TRUE, then you can set the graphical parameters with the argument ci.join.par.
ci.join.par	A list of parameters to be passed to [lines](). Only used if ci.join=TRUE. By default it is equal to list(lwd = lwd,col = col,lty = 2).
ci.fill	Logical default to FALSE. Whether to fille the confidence intervals with a color. If TRUE, then you can set the graphical parameters with the argument ci.fill.par.
ci.fill.par	A list of parameters to be passed to [polygon](). Only used if ci.fill=TRUE. By default it is equal to list(col = "lightgray",alpha = 0.5). Note that alpha is a special parameter that adds transparency to the color (ranges from 0 to 1).
ref	Only used in interactions. Either: i) "auto" (default), ii) a character vector of length 1, iii) a list of length 1, or iv) a named integer vector of length 1. It gives the value that has been set as a reference in the estimation of the interactions. By default, if the estimation has been done with fixest, the reference is automatically found. If ii), ie a character scalar, then that coefficient equal to zero is added as the first coefficient. If a list or a named integer vector of length 1, then the integer gives the position of the reference among the coefficients and the name gives the coefficient name.
ref.line	Logical, default is "auto", the behavior depending on the situation. It is TRUE only if: i) interactions are plotted, ii) the x values are numeric and iii) a reference is found. If TRUE, then a vertical line is drawn at the level of the reference value. You can set the parameters of this line with the argument ref.line.par.
ref.line.par	List. Parameters of the vertical line on the reference. The default values are: col = "black" and lty = 2. You can add any graphical parameter that will be passed to [abline](). Eg: ref.line.par = list(lty = 1,lwd = 3).
lab.cex	The size of the labels of the coefficients. Default is missing. It is automatically set by an internal algorithm which can go as low as lab.min.cex (another argument).
lab.min.cex	The minimum size of the coefficients labels, as set by the internal algorithm. Default is 0.85.
lab.max.mar	The maximum size the left margin can take when trying to fit the coefficient labels into it (only when horiz = TRUE). This is used in the internal algorithm fitting the coefficient labels. Default is 0.25.
lab.fit	The method to fit the coefficient labels into the plotting region (only when horiz = FALSE). Can be "auto" (the default), "simple", "multi" or "tilted". If "simple", then the classic axis is drawn. If "multi", then the coefficient labels are fit horizontally across several lines, such that they don't collide. If "tilted", then the labels are tilted. If "auto", an automatic choice between the three is made.
xlim.add	A numeric vector of length 1 or 2. It represents an extension factor of xlim, in percentage. Eg: xlim.add = c(0,0.5) extends xlim of 50% on the right. If of lentgh 1, positive values represent the right, and negative values the left (Eg: xlim.add = -0.5 is equivalent to xlim.add = c(0.5,0)).
ylim.add	A numeric vector of length 1 or 2. It represents an extension factor of ylim, in percentage. Eg: ylim.add = c(0,0.5) extends ylim of 50% on the top. If of

lentgh 1, positive values represent the top, and negative values the bottom (Eg: ylim.add = -0.5 is equivalent to ylim.add = c(0.5,0)).

only.params   Logical, default is FALSE. If TRUE no graphic is displayed, only the values of x and y used in the plot are returned.

only.inter   Logical, default is TRUE. If an interaction of the type of var::fe (see [feols](#) help for details) is found, then only these interactions are plotted. If FALSE, then interactions are treated as regular coefficients.

sep   The distance between two estimates – only when argument object is a list of estimation results.

as.multiple   Logical: default is FALSE. Only when object is a single estimation result: whether each coefficient should have a different color, line type, etc. By default they all get the same style.

bg   Background color for the plot. By default it is white.

group   A list, default is missing. Each element of the list reports the coefficients to be grouped while the name of the element is the group name. Each element of the list can be either: i) a character vector of length 1, ii) of length 2, or ii) a numeric vector. If equal to: i) then it is interpreted as a pattern: all element fitting the regular expression will be grouped, if ii) it corrsponds to the first and last elements to be grouped, if iii) it corresponds to the coefficients numbers to be grouped. If equal to a character vector, you can use a percentage to tell the algorithm to look at the coefficients before aliasing (e.g. "%varname"). Example of valid uses: group=list(group_name=\"pattern\"), group=list(group_name=c(\"var_start\",\"var_end\" group=list(group_name=1:2)). See details.

group.par   A list of parameters controlling the display of the group. The parameters controlling the line are: lwd, tcl (length of the tick), line.adj (adjustment of the position, default is 0), tick (whether to add the ticks), lwd.ticks, col.ticks. Then the parameters controlling the text: text.adj (adjustment of the position, default is 0), text.cex, text.font, text.col.

main   The title of the plot. Default is "Effect on __depvar__". You can use the special variable __depvar__ to set the title (useful when you set the plot default with [setFixest_coefplot](#)).

value.lab   The label to appear on the side of the coefficient values. If horiz = FALSE, the label appears in the y-axis. If horiz = TRUE, then it appears on the x-axis. The default is equal to "Estimate and __ci__ Conf. Int.", with __ci__ a special variable giving the value of the confidence interval.

ylab   The label of the y-axis, default is NULL. Note that if horiz = FALSE, it overrides the value of the argument value.lab.

xlab   The label of the x-axis, default is NULL. Note that if horiz = TRUE, it overrides the value of the argument value.lab.

sub   A subtitle, default is NULL.

Setting custom default values

The function coefplot dispose of many arguments to parametrize the plots. Most of these arguments can be set once an for all using the function [setFixest_coefplot](#). See Example 3 below for a demonstration.

Arguments keep, drop and order

The arguments `keep`, `drop` and `order` use regular expressions. If you are not aware of regular expressions, I urge you to learn it, since it is an extremely powerful way to manipulate character strings (and it exists across most programming languages).

For example drop = "Wind" would drop any variable whose name contains "Wind". Note that variables such as "Temp:Wind" or "StrongWind" do contain "Wind", so would be dropped. To drop only the variable named "Wind", you need to use drop = `"^Wind$"` (with "^" meaning beginning, resp. "$" meaning end, of the string => this is the language of regular expressions).

Although you can combine several regular expressions in a single character string using pipes, `drop` also accepts a vector of regular expressions.

You can use the special character "!" (exclamation mark) to reverse the effect of the regular expression (this feature is specific to this fonction). For example drop = `"!Wind"` would drop any variable that does not contain "Wind".

You can use the special character "

The argument `order` takes in a vector of regular expressions, the order will follow the elments of this vector. The vector gives a list of priorities, on the left the elements with highest priority. For example, order = c("Wind", "!Inter", "!Temp") would give highest priorities to the variables containing "Wind" (which would then appear first), second highest priority is the variables not containing "Inter", last, with lowest priority, the variables not containing "Temp". If you had the following variables: (Intercept), Temp:Wind, Wind, Temp you would end up with the following order: Wind, Temp:Wind, Temp, (Intercept).

Author(s)

Laurent Berge

See Also

See `setFixest_coefplot` to set the default values of `coefplot`, and the estimation functions: e.g. `feols`, `fepois`, `feglm`, `fenegbin`.

Examples

```
#
# Example 1: Stacking two sets of results on the same graph
#

# Estimation on Iris data with one fixed-effect (Species)
est = feols(Petal.Length ~ Petal.Width + Sepal.Length +
            Sepal.Width | Species, iris)

# Estimation results with clustered standard-errors
# (the default when fixed-effects are present)
est_clu = summary(est)
# Now with "regular" standard-errors
est_std = summary(est, se = "standard")
```

```
# You can plot the two results at once
coefplot(list(est_clu, est_std))


# Alternatively, you can use the argument x.shift
# to do it sequentially:

# First graph with clustered standard-errors
coefplot(est, x.shift = -.2)

# 'x.shift' was used to shift the coefficients on the left.

# Second set of results: this time with
#  standard-errors that are not clustered.
coefplot(est, se = "standard", x.shift = .2,
         add = TRUE, col = 2, ci.lty = 2, pch=15)

 # Note that we used 'se', an argument that will
 #  be passed to summary.fixest

legend("topright", col = 1:2, pch = 20, lwd = 1, lty = 1:2,
       legend = c("Clustered", "Standard"), title = "Standard-Errors")


#
# Example 2: Interactions
#


# Now we estimate and plot the "yearly" treatment effects

data(base_did)
base_inter = base_did

# We interact the variable 'period' with the variable 'treat'
est_did = feols(y ~ x1 + i(treat, period, 5) | id+period, base_inter)

# You could have written the following formula instead:
# y ~ x1 + treat::period(5) | id+period

# In the estimation, the variable treat is interacted
#  with each value of period but 5, set as a reference

# When estimations contain interactions, as before,
#  the default behavior of coefplot changes,
#  it now only plots interactions:
coefplot(est_did)

# We can see that the graph is different from before:
#  - only interactions are shown,
#  - the reference is present,
#  - the estimates are joined.
# => this is fully flexible
```

```
coefplot(est_did, ref.line = FALSE, pt.join = FALSE)

# Now to display all coefficients, use 'only.inter'
coefplot(est_did, only.inter = FALSE)

#
# What if the interacted variable is not numeric?

# Let's create a "month" variable
all_months = c("aug", "sept", "oct", "nov", "dec", "jan",
               "feb", "mar", "apr", "may", "jun", "jul")
base_inter$period_month = all_months[base_inter$period]

# The new estimation
est = feols(y ~ x1 + i(treat, period_month, "oct") | id+period, base_inter)
# Since 'period_month' of type character, coefplot sorts it
coefplot(est)

# To respect a plotting order, use a factor
base_inter$month_factor = factor(base_inter$period_month, levels = all_months)
est = feols(y ~ x1 + i(treat, month_factor, "oct") | id+period, base_inter)
coefplot(est)


#
# Example 3: Setting defaults
#

# coefplot has many arguments, which makes it highly flexible.
# If you don't like the default style of coefplot. No worries,
# you can set *your* default by using the function
# setFixest_coefplot()

dict = c("Petal.Length"="Length (Petal)", "Petal.Width"="Width (Petal)",
         "Sepal.Length"="Length (Sepal)", "Sepal.Width"="Width (Sepal)")

setFixest_coefplot(ci.col = 2, pt.col = "darkblue", ci.lwd = 3,
                   pt.cex = 2, pt.pch = 15, ci.width = 0, dict = dict)

est = feols(Petal.Length ~ Petal.Width + Sepal.Length +
                Sepal.Width | Species, iris)

# Tadaaa! (Although the colors could be better)
coefplot(est)

# To reset to the default settings:
setFixest_coefplot(reset = TRUE)
coefplot(est)
```

---

coeftable *Obtain various statistics from an estimation*

Description

Set of functions to directly extract some commonly used statistics, like the p-value or the table of coefficients, from estimations. This was first implemented for fixest estimations, but has some support for other models.

Usage

```
coeftable(object, se, cluster, ...)

ctable

pvalue(object, se, cluster, ...)

tstat(object, se, cluster, ...)

se(object, se, cluster, ...)
```

Arguments

object	An estimation. For example obtained from [feols](#).
se	[Fixest specific.] 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	[Fixest specific.] 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).
...	Other arguments to be passed to summary.

Format

An object of class function of length 1.

Details

This set of functions is primarily constructed for `fixest` estimations. Although it can work for non-`fixest` estimations, support for exotic estimation procedures that do not report standardized coefficient tables is highly limited.

Value

Returns a table of coefficients, with in rows the variables and four columns: the estimate, the standard-error, the t-statistic and the p-value.

Functions

- `pvalue`: Extracts the p-value of an estimation
- `tstat`: Extracts the t-statistics of an estimation
- `se`: Extracts the standard-error of an estimation

Examples

```
# Some data and estimation
data(trade)
est = fepois(Euros ~ log(dist_km) | Origin^Product + Year, trade)


#
# Coeftable/se/tstat/pvalue
#

# Default is clustering along Origin^Product
coeftable(est)
se(est)
tstat(est)
pvalue(est)

# Now with two-way clustered standard-errors
#  and using ctable(), the alias to coeftable()

ctable(est, cluster = ~Origin + Product)
se(est, cluster = ~Origin + Product)
pvalue(est, cluster = ~Origin + Product)
tstat(est, cluster = ~Origin + Product)

# Or you can cluster only once:
est_sum = summary(est, cluster = ~Origin + Product)
ctable(est_sum)
se(est_sum)
tstat(est_sum)
pvalue(est_sum)
```

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 fixed-effect variables, 2) perfect multi-collinearity between the variables, 4) perfect multi-collinearity between several variables and the fixed-effects, 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:
set.seed(1)
fe_1 = sample(3, 100, TRUE)
fe_2 = sample(20, 100, TRUE)
x = rnorm(100, fe_1)**2
y = rnorm(100, fe_2)**2
z = rnorm(100, 3)**2
dep = rpois(100, x*y*z)
base = data.frame(fe_1, fe_2, x, y, z, dep)

# creating collinearity problems:
```

```
base$v1 = base$v2 = base$v3 = base$v4 = 0
base$v1[base$fe_1 == 1] = 1
base$v2[base$fe_1 == 2] = 1
base$v3[base$fe_1 == 3] = 1
base$v4[base$fe_2 == 1] = 1

# Estimations:

# Collinearity with the fixed-effects:
res_1 = femlm(dep ~ log(x) + v1 + v2 + v4 | fe_1 + fe_2, base)
collinearity(res_1)

# => collinearity with the first fixed-effect identified, we drop v1 and v2
res_1bis = femlm(dep ~ log(x) + v4 | fe_1 + fe_2, base)
collinearity(res_1bis)

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

---

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 = getFixest_dof(), ...)
```

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 parame-
                ters 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　　　　　　　An object of class dof.type obtained with the function [dof]. Represents how the degree of freedom correction should be done.You must use the function [dof] for this argument. The arguments and defaults of the function [dof] are: adj = TRUE, fixef.K="nested", cluster.adj = TRUE, cluster.df = "conventional", t.df = "conventional", fixef.force_exact=FALSE). See the help of the function [dof] for details.

...　　　　　　　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 fixed-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")
```

---

demean　　　　　　　　　*Centers a set of variables around a set of factors*

Description

Centers a set of variables around a set of factors

Usage

```
demean(
  X,
  fe,
  weights,
  nthreads = getFixest_nthreads(),
  notes = getFixest_notes(),
  iter = 2000,
  tol = 1e-06,
  im_confident = FALSE
)
```

Arguments

X	A matrix, vector or a list. The vectors to be centered. There must be the same number of observations as in the factors used for centering (argument `fe`).
fe	A matrix, vector or list. The factors used to center the variables in argument `X`. (Note: fe stands for fixed-effects.)
weights	Vector, can be missing or NULL. If present, it must contain the same number of observations as in `X`.
nthreads	Number of threads to be used. By default it is equal to `getFixest_nthreads()`.
notes	Logical, whether to display a message when NA values are removed. By default it is equal to `getFixest_notes()`.
iter	Number of iterations, default is 2000.
tol	Stopping criterion of the algorithm. Default is `1e-6`. The algorithm stops when the maximum absolute increase in the coefficients values is lower than `tol`.
im_confident	Logical, default is `FALSE`. FOR EXPERT USERS ONLY! This argument allows to skip some of the preprocessing of the arguments given in input. If `TRUE`, then `X` MUST be a numeric matrix (not integer, numeric), `fe` MUST be a list and `weights`, if given, MUST be numeric (not integer!). Further the three MUST NOT contain any NA values and MUST have the same number of observations. Non compliance to these rules may simply lead your R session to break.

Value

It returns a matrix of the same number of columns as `X` in input. The number of rows is equal to the number of rows of `X` minus the number of NA values (contained in `X`, `fe` or `weights`).

Examples

```
# Illustration of the FWL theorem
data(trade)

base = trade
base$ln_dist = log(base$dist_km)
base$ln_euros = log(base$Euros)
```

```
# We center the two variables ln_dist and ln_euros
#  on the factors Origin and Destination
X_demean = demean(X = base[, c("ln_dist", "ln_euros")],
                  fe = base[, c("Origin", "Destination")])
base[, c("ln_dist_dm", "ln_euros_dm")] = X_demean

est = feols(ln_euros_dm ~ ln_dist_dm, base)
est_fe = feols(ln_euros ~ ln_dist | Origin + Destination, base)

# The results are the same as if we used the two factors
# as fixed-effects
etable(est, est_fe, se = "st")
```

| deviance.fixest | *Extracts the deviance of a fixest estimation* |

Description

Returns the deviance from a `fixest` estimation.

Usage

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

Arguments

| object | A `fixest` object. |
| ... | Not currently used. |

Value

Returns a numeric scalar equal to the deviance.

See Also

[feols](#), [fepois](#), [feglm](#), [fenegbin](#), [feNmlm](#).

Examples

```
est = feols(Petal.Length ~ Petal.Width, iris)
deviance(est)

est_pois = fepois(Petal.Length ~ Petal.Width, iris)
deviance(est_pois)
```

---

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 treat-
               ment 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")
```

---

dof   *Type of degree of freedom in fixest summary*

Description

Provides how the degrees of freedom should be calculated in `vcov.fixest`/`summary.fixest`.

Usage

```
dof(
  adj = TRUE,
  fixef.K = "nested",
  cluster.adj = TRUE,
  cluster.df = "conventional",
  t.df = "conventional",
  fixef.force_exact = FALSE
)

setFixest_dof(dof.type = dof())

getFixest_dof
```

Arguments

adj	Logical scalar, defaults to TRUE. Whether to apply a small sample adjustment of the form (n -1) / (n -K), with K the number of estimated parameters. If FALSE, then no adjustment is made.
fixef.K	Character scalar equal to "nested" (default), "none" or "full". In the small sample adjustment, how to account for the fixed-effects parameters. If "none", the fixed-effects parameters are discarded, meaning the number of parameters (K) is only equal to the number of variables. If "full", then the number of parameters is equal to the number of variables plus the number of fixed-effects. Finally, if "nested", then the number of parameters is equal to the number of variables plus the number of fixed-effects that *are not* nested in the clusters used to cluster the standard-errors.
cluster.adj	Logical scalar, default is TRUE. How to make the small sample correction when clustering the standard-errors? If TRUE a G/(G-1) correction is performed with G the number of cluster values.
cluster.df	Either "conventional" (default) or "min". Only relevant when the variance-covariance matrix is two-way clustered (or higher). It governs how the small sample adjustment for the clusters is to be performed. [Sorry for the jargon that follows.] By default the i-th "sandwich" matrix is adjusted with $G_i/(G_i\text{-}1)$ with G_i the number of unique clusters. If cluster.df="min", a unique adjustment is made, of the form $G\_min/(G\_min\text{-}1)$ with G_min the smallest G_i.
t.df	Either "conventional" (default) or "min". Only relevant when the variance-covariance matrix is clustered. It governs how the p-values should be computed. By default, the degrees of freedom of the Student t distribution is equal to the number of observations minus the number of estimated variables. If t.df="min", then the degrees of freedom of the Student t distribution is equal to the minimum size of the clusters with which the VCOV has been clustered.
fixef.force_exact	Logical, default is FALSE. If there are 2 or more fixed-effects, these fixed-effects they can be irregular, meaning they can provide the same information. If so, the "real" number of parameters should be lower than the total number of fixed-effects. If fixef.force_exact = TRUE, then fixef.fixest is first run to determine the exact number of parameters among the fixed-effects. Mostly, panels of the type individual-firm require fixef.force_exact = TRUE (but it adds computational costs).
dof.type	An object of class dof.type obtained with the function dof.

Format

An object of class function of length 1.

Details

The following vignette: On standard-errors, describes in details how the standard-errors are computed in fixest and how you can replicate standard-errors from other software.

Value

It returns a dof.type object.

Author(s)

Laurent Berge

See Also

[summary.fixest](#), [vcov.fixest](#)

Examples

```
#
# Equivalence with lm/glm standard-errors
#

# LM
# In the absence of fixed-effects,
# by default, the standard-errors are computed in the same way

res = feols(Petal.Length ~ Petal.Width + Species, iris)
res_lm = lm(Petal.Length ~ Petal.Width + Species, iris)
vcov(res) / vcov(res_lm)

# GLM
# By default, there is no small sample adjustment in glm, as opposed to feglm.
# To get the same SEs, we need to use dof(adj = FALSE)

res_pois = fepois(round(Petal.Length) ~ Petal.Width + Species, iris)
res_glm = glm(round(Petal.Length) ~ Petal.Width + Species, iris, family = poisson())
vcov(res_pois, dof = dof(adj = FALSE)) / vcov(res_glm)

# Same example with the Gamma
res_gamma = feglm(round(Petal.Length) ~ Petal.Width + Species, iris, family = Gamma())
res_glm_gamma = glm(round(Petal.Length) ~ Petal.Width + Species, iris, family = Gamma())
vcov(res_gamma, dof = dof(adj = FALSE)) / vcov(res_glm_gamma)

#
# Fixed-effects corrections
#

# We create "irregular" FEs
base = data.frame(x = rnorm(10))
base$y = base$x + rnorm(10)
base$fe1 = rep(1:3, c(4, 3, 3))
base$fe2 = rep(1:5, each = 2)

est = feols(y ~ x | fe1 + fe2, base)

# fe1: 3 FEs
```

```
# fe2: 5 FEs

#
# Clustered standard-errors: by fe1
#

# Default: fixef.K = "nested"
#  => adjustment K = 1 + 5 (i.e. x + fe2)
summary(est)
attributes(vcov(est))[c("dof.type", "dof.K")]


# fixef.K = FALSE
#  => adjustment K = 1 (i.e. only x)
summary(est, dof = dof(fixef.K = "none"))
attr(vcov(est, dof = dof(fixef.K = "none")), "dof.K")


# fixef.K = TRUE
#  => adjustment K = 1 + 3 + 5 - 1 (i.e. x + fe1 + fe2 - 1 restriction)
summary(est, dof = dof(fixef.K = "full"))
attr(vcov(est, dof = dof(fixef.K = "full")), "dof.K")


# fixef.K = TRUE & fixef.force_exact = TRUE
#  => adjustment K = 1 + 3 + 5 - 2 (i.e. x + fe1 + fe2 - 2 restrictions)
summary(est, dof = dof(fixef.K = "full", fixef.force_exact = TRUE))
attr(vcov(est, dof = dof(fixef.K = "full", fixef.force_exact = TRUE)), "dof.K")

# There are two restrictions:
attr(fixef(est), "references")

#
# To permanently set the default dof:
#

# eg to set it a la Stata's reghdfe:
setFixest_dof(dof(cluster.df = "min", t.df = "min"))

# To reset it
setFixest_dof()
```

---

estfun.fixest   *Extracts the scores from a fixest estimation*

Description

Extracts the scores from a fixest estimation.

Usage

```
## S3 method for class 'fixest'
estfun(x, ...)
```

Arguments

x	A fixest object, obtained for instance from [feols](#).
...	Not currently used.

Value

Returns a matrix of the same number of rows as the number of observations used for the estimation, and the same number of columns as there were variables.

Examples

```
est = feols(Petal.Length ~ Petal.Width + Sepal.Width, iris)
head(estfun(est))
```

---

etable	*Estimations table (export the results of multiples estimations to a DF or to Latex)*

Description

Aggregates the results of multiple estimations and displays them in the form of either a Latex table or a data.frame.

Usage

```
etable(
  ...,
  se = c("standard", "white", "cluster", "twoway", "threeway", "fourway"),
  dof = getFixest_dof(),
  cluster,
  digits = 4,
  tex,
  fitstat,
  title,
  coefstat = c("se", "tstat", "confint"),
  ci = 0.95,
  sdBelow = TRUE,
  keep,
  drop,
  order,
```

```
      dict,
      file,
      replace = FALSE,
      convergence,
      signifCode,
      label,
      float,
      subtitles,
      fixef_sizes = FALSE,
      fixef_sizes.simplify = TRUE,
      yesNo = "Yes",
      keepFactors = TRUE,
      family,
      powerBelow = -5,
      interaction.combine = " $\\times $ ",
      depvar,
      style = list(),
      notes = NULL,
      group = NULL,
      extraline = NULL,
      tablefoot = TRUE
    )

    esttex(
      ...,
      se = c("standard", "white", "cluster", "twoway", "threeway", "fourway"),
      dof = getFixest_dof(),
      cluster,
      digits = 4,
      fitstat,
      coefstat = c("se", "tstat", "confint"),
      ci = 0.95,
      title,
      float = float,
      sdBelow = TRUE,
      keep,
      drop,
      order,
      dict,
      file,
      replace = FALSE,
      convergence,
      signifCode = c(`***` = 0.01, `**` = 0.05, `*` = 0.1),
      label,
      subtitles,
      fixef_sizes = FALSE,
      fixef_sizes.simplify = TRUE,
      yesNo = "Yes",
```

```
    keepFactors = TRUE,
    family,
    powerBelow = -5,
    interaction.combine = " $\\times $ ",
    style = list(),
    notes = NULL,
    group = NULL,
    tablefoot = TRUE,
    extraline = NULL
)

esttable(
    ...,
    se = c("standard", "white", "cluster", "twoway", "threeway", "fourway"),
    dof = getFixest_dof(),
    cluster,
    coefstat = c("se", "tstat", "confint"),
    ci = 0.95,
    depvar,
    keep,
    drop,
    dict,
    order,
    digits = 4,
    fitstat,
    convergence,
    signifCode = c(`***` = 0.001, `**` = 0.01, `*` = 0.05, . = 0.1),
    subtitles,
    keepFactors = FALSE,
    family,
    group = NULL,
    extraline = NULL
)

setFixest_etable(
    digits = 4,
    fitstat,
    coefstat = c("se", "tstat", "confint"),
    ci = 0.95,
    sdBelow = TRUE,
    keep,
    drop,
    order,
    dict,
    signifCode,
    float,
    fixef_sizes = FALSE,
    fixef_sizes.simplify = TRUE,
```

```
    yesNo = c("Yes", "No"),
    family,
    powerBelow = -5,
    interaction.combine = " $\\times $ ",
    depvar,
    style = list(),
    notes = NULL,
    group = NULL,
    extraline = NULL,
    tablefoot = TRUE,
    reset = FALSE
)

getFixest_etable()
```

Arguments

`...`	Used to capture different `fixest` estimation 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`	An object of class `dof.type` obtained with the function `dof`. Represents how the degree of freedom correction should be done.You must use the function `dof` for this argument. The arguments and defaults of the function `dof` are: `adj = TRUE`, `fixef.K="nested"`, `cluster.adj = TRUE`, `cluster.df = "conventional"`, `t.df = "conventional"`, `fixef.force_exact=FALSE`). See the help of the function `dof` for details.
`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.
`tex`	Logical: whether the results should be a data.frame or a Latex table. By default, this argument is `TRUE` if the argument `file` (used for exportation) is not missing; it is equal to `FALSE` otherwise.
`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	(Tex only.) Character scalar. The title of the Latex table.
coefstat	One of "se" (default), "tstat" or "confint". The statistic to report for each coefficient: the standard-error, the t-statistics or the confidence interval. You can adjust the confidence interval with the argument ci.
ci	Level of the confidence interval, defaults to 0.95. Only used if coefstat = confint.
sdBelow	(Tex only.) Logical, default is TRUE. Should the standard-errors be displayed below the coefficients?
keep	Character vector. This element is used to display only a subset of variables. This should be a vector of regular expressions (see [regex](#) help for more info). Each variable satisfying any of the regular expressions will be kept. This argument is applied post aliasing (see argument dict). Example: you have the variable x1 to x55 and want to display only x1 to x9, then you could use keep = "x[[:digit:]]$". If the first character is an exclamation mark, the effect is reversed (e.g. keep = "!Intercept" means: every variable that does not contain "Intercept" is kept). See details.
drop	Character vector. This element is used if some variables are not to be displayed. This should be a vector of regular expressions (see [regex](#) help for more info). Each variable satisfying any of the regular expressions will be discarded. This argument is applied post aliasing (see argument dict). Example: you have the variable x1 to x55 and want to display only x1 to x9, then you could use drop = "x[[:digit:]]{2}". If the first character is an exclamation mark, the effect is reversed (e.g. drop = "!Intercept" means: every variable that does not contain "Intercept" is dropped). See details.
order	Character vector. This element is used if the user wants the variables to be ordered in a certain way. This should be a vector of regular expressions (see [regex](#) help for more info). The variables satisfying the first regular expression will be placed first, then the order follows the sequence of regular expressions. This argument is applied post aliasing (see argument dict). Example: you have the following variables: month1 to month6, then x1 to x5, then year1 to year6. If you want to display first the x's, then the years, then the months you could use: order = c("x","year"). If the first character is an exclamation mark, the effect is reversed (e.g. order = "!Intercept" means: every variable that does not contain "Intercept" goes first). See details.
dict	A named character vector or a logical scalar. It changes the original variable names to the ones contained in the dictionary. E.g. to change the variables named a and b3 to (resp.) "$log(a)$" and to "$bonus^3$", use dict=c(a="$log(a)$",b3="$bonus^3$"). By default, if Tex output is requested or if argument file is not missing, it is equal to getFixest_dict(), a default dictionary which can be set with [setFixest_dict](#). The default is not to change names if a data.frame is requested (i.e. tex = FALSE); if so, you can use dict = TRUE to use the dictionary you've set globally with setFixest_dict().
file	A character scalar. If provided, the Latex (or data frame) table will be saved in a file whose path is file. If you provide this argument, then a Latex table will be exported, to export a regular data.frame, use argument tex = FALSE.
replace	Logical, default is FALSE. Only used if option file is used. Should the exported 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) for a Latex table and c("***"=0.001,"**"=0.01,"*"=0.05,"."=0.10) for a data.frame (to conform with R's default). To supress the significance codes, use signifCode=NA or signifCode=NULL. Can also be equal to "letters", then the default becomes c("a"=0.01,"b"=0.05,"c"=0.10).
label	(Tex only.) Character scalar. The label of the Latex table.
float	(Tex only.) Logical. By default, if the argument title or label is provided, it is set to TRUE. Otherwise, it is set to FALSE.
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	(Tex only.) Logical, default is FALSE. If TRUE and fixed-effects were used in the models, then the number of "individuals" per fixed-effect dimension is also displayed.
fixef_sizes.simplify	Logical, default is TRUE. Only used if fixef_sizes = TRUE. If TRUE, the fixed-effects sizes will be displayed in parentheses instead of in a separate line if there is no ambiguity (i.e. if the size is constant across models).
yesNo	(Tex only.) A character vector of length 1 or 2. Default is c("Yes","No"). This is the message displayed when a given fixed-effect is (or is not) included in a regression. If yesNo is of length 1, then the second element is the empty string.
keepFactors	Logical, default is TRUE. If FALSE, then factor variables are displayed as fixed-effects and no coefficient is shown.
family	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	(Tex only.) 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.
interaction.combine	(Tex only.) Character scalar, defaults to " $\times$ ". When the estimation contains interactions, then the variables names (after aliasing) are combined with this argument. For example: if dict = c(x1="Wind",x2="Rain") and you have the following interaction x1:x2, then it will be renamed (by default) Wind $\times$ Rain – using interaction.combine = "*" would lead to Wind*Rain.
depvar	(Data frame only.) 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 or if the argumen file is not missing.
style	A list. You can change the general style of the table with this argument. It should be of the form style = list(keyword="key1:value1;key2:value2") etc. The available keywords are lines (to manage the type of lines appearing

in the table), and depvar, model, var, fixef, slopes, fixef.sizes, stats and notes. Most of these keywords accept the key title: which affects the title appearing just before the section. Eg to drop the *Variables* header, just use style=list("title:"). Note that if you use style=list("title: ") (note the space after ":"), then an empty line will still be there. The keywords fixef, slopes and fixef.sizes also accept the keys prefix and suffix. E.g. if style=list(fixef="suffix: FE"), then there will be no header showing but the text " FE" will be appended to the ficed-effects variable names. The keys accepted in lines are top, bottom, foot and sep.

notes   Character vector. If provided, a "notes" section will be added at the end right after the end of the table, containing the text of this argument. Note that if it is a vector, it will be collapsed with new lines.

group   A list. The list elements should be vectors of regular expressions. For each elements of this list: A new line in the table is created, all variables that are matched by the regular expressions are discarded (same effect as the argument drop) and TRUE or FALSE will appear in the model cell, depending on whether some of the previous variables were found in the model. Example: group=list("Controls: personal traits"=c("gender","height","weight")) will create an new line with "Controls: personal traits" in the leftmost cell, all three variables gender, height and weight are discarded, TRUE appearing in each model containing at least one of the three variables (the style of TRUE/FALSE is governed by the argument yesNo). You can control the style with the title and where keywords in curly brackets. For example group=list("{title:Controls; where:stats}Personal traits"=c("gender","height","weight")) will add an extra line right before with "Control" written in it, and the group information will appear after the statistics. The keyword where can be equal to either var (default), fixef or stats.

extraline   A list. The list elements should be either a single logical or a vector of the same length as the number of models. For each elements of this list: A new line in the table is created, the list name being the row name and the vector being the content of the cells. Example: extraline=list("Sub-sample"=c("<20 yo","all",">50 yo")) will create an new line with "Sub-sample" in the leftmost cell, the vector filling the content of the cells for the three models. You can control the style with the title and where keywords in curly brackets. For example group=list("{title:Sub-sample; where:stats}By age"=c("<20 yo","all",">50 yo")) will add an extra line right before with "Sub-sample" written in it, and the extraline information will appear after the statistics section. The keyword where can be equal to either var (default), fixef or stats.

tablefoot   Logical, default is TRUE. Whether to display the table footer containing the information on the way the standard-errors where computed and the meaning of the significance codes.

reset   (setFixest_etable only.) Logical, default is FALSE. If TRUE, this will reset all the default values that were already set by the user in previous calls.

Details

The function esttex is equivalent to the function etable with argument tex = TRUE.

The function esttable is equivalent to the function etable with argument tex = FALSE.

You can permanently change the way your table looks in Latex by using setFixest_etable. The following vignette gives and example as well as illustrates how to use the argument style: Exporting estimation tables.

Value

If tex = TRUE, the lines composing the Latex table are returned invisibly while the table is directly prompted on the console.

If tex = FALSE, the data.frame is directly returned. If the argument file is not missing, the data.frame is returned invisibly.

Functions

- esttex: Exports the results of multiple fixest estimations in a Latex table.
- esttable: Facility to display the results of multiple fixest estimations.

Arguments keep, drop and order

The arguments keep, drop and order use regular expressions. If you are not aware of regular expressions, I urge you to learn it, since it is an extremely powerful way to manipulate character strings (and it exists across most programming languages).

For example drop = "Wind" would drop any variable whose name contains "Wind". Note that variables such as "Temp:Wind" or "StrongWind" do contain "Wind", so would be dropped. To drop only the variable named "Wind", you need to use drop = "^Wind$" (with "^" meaning beginning, resp. "$" meaning end, of the string => this is the language of regular expressions).

Although you can combine several regular expressions in a single character string using pipes, drop also accepts a vector of regular expressions.

You can use the special character "!" (exclamation mark) to reverse the effect of the regular expression (this feature is specific to this fonction). For example drop = "!Wind" would drop any variable that does not contain "Wind".

You can use the special character "

The argument order takes in a vector of regular expressions, the order will follow the elments of this vector. The vector gives a list of priorities, on the left the elements with highest priority. For example, order = c("Wind", "!Inter", "!Temp") would give highest priorities to the variables containing "Wind" (which would then appear first), second highest priority is the variables not containing "Inter", last, with lowest priority, the variables not containing "Temp". If you had the following variables: (Intercept), Temp:Wind, Wind, Temp you would end up with the following order: Wind, Temp:Wind, Temp, (Intercept).

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 fixed-effects coefficients.

Examples

```
aq = airquality
aq$Month = factor(aq$Month)

est1 = feols(Ozone ~ Month / Wind + Temp, data = aq)
est2 = feols(Ozone ~ Wind + Temp | Month, data = aq)

# Displaying the two results in a single table
etable(est1, est2)

# keep/drop: keeping only interactions
etable(est1, est2, keep = ":")
# or using drop  (see regexp help):
etable(est1, est2, drop = "^[[:alnum:]]+$")

# keep/drop: dropping interactions
etable(est1, est2, drop = ":")
# or using keep ("!" reverses the effect):
etable(est1, est2, keep = "!:")

# order: Wind variable first, intercept last
etable(est1, est2, order = c("Wind", "Month"))
etable(est1, est2, order = c("^Wind", "Wind", "Month"))
# Interactions, then Intercept, last ("!" reverses the effect)
etable(est1, est2, order = c("!Int", "!:"))

# dict + keep/drop/order: using "%" to match the original names
dict = c("Month5"="May", "Month6"="Jun", "Month7"="Jul",
         "Month8"="Aug", "Month9"="Sep")
etable(est1, est2, tex = TRUE, dict = dict)
# keeping only June and July
etable(est1, est2, tex = TRUE, dict = dict, keep = c("%Month6", "Jul"))
# All months variabes first
etable(est1, est2, tex = TRUE, dict = dict, order = c("%Month"))

# signifCode
etable(est1, est2, signifCode = c(" A"=0.01, " B"=0.05, " C"=0.1,
                                  " D"=0.15, " F"=1))

# fitstat
etable(est1, est2, fitstat = ~r2+ar2+apr2+war2)

# Adding a dictionnary (Tex only)
dict = c(Month5="May", Month6="Jun", Month7="Jul", Month8="Aug", Month9="Sep")
etable(est1, est2, dict = dict, tex = TRUE)

#
# Using the argument style to customize Latex exports
#

# If you don't like the default layout of the table, no worries!
```

```
# You can modify many parameters with the argument style

# To drop the headers before each section, use:
style_noHeaders = list(var="", fixef="suffix: FE", stats = "")
etable(est1, est2, dict = dict, tex = TRUE, style = style_noHeaders)

# To change the lines of the table
style_lines = list(lines = "top:\\toprule;bottom:\\bottomrule;sep:\\midrule;foot:\\midrule")
etable(est1, est2, dict = dict, tex = TRUE, style = style_lines)


#
# Group and extraline
#

# Sometimes it's useful to group control variables into a single line
# You can achieve that with the group argument

setFixest_fml(..ctrl = ~ poly(Wind, 2) + poly(Temp, 2))
est_c0 = feols(Ozone ~ Solar.R, data = aq)
est_c1 = feols(Ozone ~ Solar.R + ..ctrl, data = aq)
est_c2 = feols(Ozone ~ Solar.R + I(Solar.R**2) + ..ctrl, data = aq)

etable(est_c0, est_c1, est_c2, group = list(Controls = "poly"))

# 'group' here does the same as drop = "poly", but adds an extra line
# with TRUE/FALSE where the variables were found

# 'extraline' adds an extra line, where you can add the value for each model
est_all  = feols(Ozone ~ Solar.R + Temp + Wind, data = aq)
est_sub1 = feols(Ozone ~ Solar.R + Temp + Wind, data = aq[aq$Month %in% 5:6, ])
est_sub2 = feols(Ozone ~ Solar.R + Temp + Wind, data = aq[aq$Month %in% 7:8, ])
est_sub3 = feols(Ozone ~ Solar.R + Temp + Wind, data = aq[aq$Month == 9, ])

etable(est_all, est_sub1, est_sub2, est_sub3,
       extraline = list("Sub-sample" = c("All", "May-June", "Jul.-Aug.", "Sept.")))

# When exporting to Latex, you can add meta arguments to 'group' and 'extraline'
# Two keywords are allowed: 'title' and 'where'
# 'title' adds a line just before with the content of 'title' in the leftmost cell
# 'where' governs the location of the line. It can be equal to 'var', 'stats' or 'fixef'.
# The syntax is: {title:Controls; where:stats}Group name.
# Note that starting with curly braces is mandatory.

# Examples
etable(est_c0, est_c1, est_c2, tex = TRUE, group = list("{where:stats}Controls" = "poly"))
etable(est_all, est_sub1, est_sub2, est_sub3, tex = TRUE,
       extraline = list("{title:\\midrule}Sub-sample" =
                           c("All", "May-June", "Jul.-Aug.", "Sept.")))
```

f	*Lags a variable in a* `fixest` *estimation*

Description

Produce lags or leads in the formulas of `fixest` estimations or when creating variables in a `data.table`. The data must be set as a panel beforehand (either with the function `panel` or with the argument `panel.id` in the estimation).

Usage

```
f(x, lead = 1, fill = NA)

l(x, lag = 1, fill = NA)
```

Arguments

x	The variable.
lead	A vector of integers giving the number of leads. Negative values lead to lags. This argument can be a vector when using it in fixest estimations. When creating variables in a `data.table`, it **must** be of length one.
fill	A scalar, default is NA. How to fill the missing values due to the lag/lead? Note that in a `fixest` estimation, 'fill' must be numeric (not required when creating new variables).
lag	A vector of integers giving the number of lags. Negative values lead to leads. This argument can be a vector when using it in fixest estimations. When creating variables in a `data.table`, it **must** be of length one.

Value

These functions can only be used i) in a formula of a `fixest` estimation, or ii) when creating variables within a fixest_panel object (obtained with function `panel`) which is alaos a `data.table`.

Functions

- `f`: Forwards a variable (inverse of lagging) in a `fixest` estimation

See Also

The function `panel` changes data.frames into a panel from which the functions l and f can be called. Otherwise you can set the panel 'live' during the estimation using the argument `panel.id` (see for example in the function `feols`).

Examples

```
data(base_did)

# Setting a data set as a panel...
```

```
pdat = panel(base_did, ~id+period)

# ...then using the functions l and f
est1 = feols(y~l(x1, 0:1), pdat)
est2 = feols(f(y)~l(x1, -1:1), pdat)
est3 = feols(l(y)~l(x1, 0:3), pdat)
etable(est1, est2, est3, order = c("f", "^x"), drop="Int")

# or using the argument panel.id
feols(f(y)~l(x1, -1:1), base_did, panel.id = ~id+period)

# l() and f() can also be used within a data.table:
if(require("data.table")){
  pdat_dt = panel(as.data.table(base_did), ~id+period)
  # Now since pdat_dt is also a data.table
  #   you can create lags/leads directly
  pdat_dt[, x1_l1 := l(x1)]
  pdat_dt[, c("x1_l1_fill0", "y_f2") := .(l(x1, fill = 0), f(y, 2))]
}
```

---

feglm   *Fixed-effects GLM estimations*

Description

Estimates GLM models with any number of fixed-effects.

Usage

```
feglm(
  fml,
  data,
  family = "poisson",
  offset,
  weights,
  panel.id,
  start = NULL,
  etastart = NULL,
  mustart = NULL,
  fixef,
  fixef.tol = 1e-06,
  fixef.iter = 10000,
  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 = 10000,
    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,
    panel.id,
    start = NULL,
    etastart = NULL,
    mustart = NULL,
    fixef,
    fixef.tol = 1e-06,
    fixef.iter = 10000,
    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 fixed-effects 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.
panel.id	The panel identifiers. Can either be: i) a one sided formula (e.g. panel.id = ~id+time), ii) a character vector of length 2 (e.g. panel.id=c('id','time'), or iii) a character scalar of two variables separated by a comma (e.g. panel.id='id,time'). Note that you can combine variables with ^ only inside formulas (see the dedicated section in [feols](#)).
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 names of variables to be used as fixed-effects. These variables should contain the identifier of each observation (e.g., think of it as a panel identifier). Note that the recommended way to include fixed-effects is to insert them directly in the formula.
fixef.tol	Precision used to obtain the fixed-effects. Defaults to 1e-6. It corresponds to the maximum absolute difference allowed between two coefficients of successive iterations.
fixef.iter	Maximum number of iterations in fixed-effects algorithm (only in use for 2+ fixed-effects). 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 threads 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).
notes	Logical. By default, three notes are displayed: when NAs are removed, when some fixed-effects are removed because of only 0 (or 0/1) outcomes, or when a variable is dropped because of collinearity. To avoid displaying these messages, you can set notes = FALSE. You can remove these messages permanently by using setFixest_notes(FALSE).
verbose	Integer. Higher values give more information. In particular, it can detail the number of iterations in the demeaning algoritmh (the first number is the left-hand-side, the other numbers are the right-hand-side variables). It can also detail the step-halving algorithm.
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.

Value

A fixest object.

nobs	The number of observations.
fml	The linear formula of the call.

`call`	The call of the function.
`method`	The method used to estimate the model.
`family`	The family used to estimate the model.
`fml_full`	[where relevant] The "full" formula containing the linear part and the fixed-effects.
`nparams`	The number of parameters of the model.
`fixef_vars`	The names of each fixed-effect dimension.
`fixef_id`	The list (of length the number of fixed-effects) of the fixed-effects identifiers for each observation.
`fixef_sizes`	The size of each fixed-effect (i.e. the number of unique identifierfor each fixed-effect dimension).
`y`	[where relevant] The dependent variable (used to compute the within-R2 when fixed-effects are present).
`convStatus`	Logical, convergence status of the IRWLS algorithm.
`irls_weights`	The weights of the last iteration of the IRWLS algorithm.
`obsRemoved`	[where relevant] In the case there were fixed-effects and some observations were removed because of only 0/1 outcome within a fixed-effect, it gives the row numbers of the observations that were removed. Also reports the NA observations that were removed.
`fixef_removed`	[where relevant] In the case there were fixed-effects and some observations were removed because of only 0/1 outcome within a fixed-effect, it gives the list (for each fixed-effect dimension) of the fixed-effect identifiers that were removed.
`coefficients`	The named vector of estimated coefficients.
`coeftable`	The table of the coefficients with their standard errors, z-values and p-values.
`loglik`	The loglikelihood.
`deviance`	Deviance of the fitted model.
`iterations`	Number of iterations of the algorithm.
`ll_null`	Log-likelihood of the null model (i.e. with the intercept only).
`ssr_null`	Sum of the squared residuals of the null model (containing only with the intercept).
`pseudo_r2`	The adjusted pseudo R2.
`fitted.values`	The fitted values are the expected value of the dependent variable for the fitted model: that is $E(Y|X)$.
`linear.predictors`	
	The linear predictors.
`residuals`	The residuals (y minus the fitted values).
`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.
`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).
residuals	The difference between the dependent variable and the expected predictor.
sumFE	The sum of the fixed-effects coefficients for each observation.
offset	[where relevant] The offset formula.
weights	[where relevant] The weights formula.
collin.var	[where relevant] Vector containing the variables removed because of collinearity.
collin.coef	[where relevant] Vector of coefficients, where the values of the variables removed because of collinearity are NA.

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: fixef_var[var1, var2]. Here the variables var1 and var2 will be with varying slopes (one slope per value in fixef_var) and the fixed-effect fixef_var will also be added.

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

In other words:

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

Lagging variables

To use leads/lags of variables in the estimation, you can: i) either provide the argument panel.id, ii) either set your data set as a panel with the function `panel`. Doing either of the two will give you acceess to the lagging functions `l` and `f`.

You can provide several leads/lags at once: e.g. if your formula is equal to f(y) ~ l(x,-1:1), it means that the dependent variable is equal to the lead of y, and you will have as explanatory variables the lead of x1, x1 and the lag of x1. See the examples in function `l` for more details.

Interactions

You can interact a numeric variable with a "factor-like" variable by using interact(var,fe,ref), where fe is the variable to be interacted with and the argument ref is a value of fe taken as a reference (optional). Instead of using the function interact, you can use the alias i(var,fe,ref) or even the highly specific syntax var::fe(ref).

It is important to note that *if you do not care about the standard-errors of the interactions*, then you can add interactions in the fixed-effects part of the formula (using the syntax fe[[var]], as explained in the section "Varying slopes").

Using this specific way to create interactions leads to a different display of the interacted values in etable and offers a special representation of the interacted coefficients in the function coefplot. See examples.

The function interact has in fact more arguments, please see details in its associated help page.

On standard-errors

Standard-errors can be computed in different ways, you can use the arguments se and dof in summary.fixest to define how to compute them. By default, in the presence of fixed-effects, standard-errors are automatically clustered.

The following vignette: On standard-errors describes in details how the standard-errors are computed in fixest and how you can replicate standard-errors from other software.

You can use the functions setFixest_se and setFixest_dof to permanently set the way the standard-errors are computed.

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 fixed-effects coefficients, and the function etable 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)
```

```
# You could also use fepois
res_pois = fepois(Sepal.Length ~ Sepal.Width + Petal.Length | Species, iris)

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

# All results are identical:
etable(res, res_pois, res_fit)
```

---

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,
  panel.id,
  na_inf.rm = getFixest_na_inf.rm(),
  fixef.tol = 1e-05,
  fixef.iter = 10000,
  nthreads = getFixest_nthreads(),
  verbose = 0,
  warn = TRUE,
  notes = getFixest_notes(),
  theta.init,
  combine.quick,
  ...
)

fenegbin(
  fml,
  data,
  theta.init,
  start = 0,
  fixef,
  offset,
```

```
    panel.id,
    na_inf.rm = getFixest_na_inf.rm(),
    fixef.tol = 1e-05,
    fixef.iter = 10000,
    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|fixef_1+fixef_2), or 2) either use the argument
               fixef.

data   A data.frame containing the necessary variables to run the model. The vari-
               ables 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 "pois-
               son" (Poisson model with log-link, the default), "negbin" (Negative Binomial
               model with log-link), "logit" (LOGIT model with log-link), "gaussian" (Gaus-
               sian 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 names of variables to be used as fixed-effects. These
               variables should contain the identifier of each observation (e.g., think of it as a
               panel identifier). Note that the recommended way to include fixed-effects is to
               insert them directly in the formula.

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'.

panel.id   The panel identifiers. Can either be: i) a one sided formula (e.g. panel.id =
               ~id+time), ii) a character vector of length 2 (e.g. panel.id=c('id','time'),
               or iii) a character scalar of two variables separated by a comma (e.g. panel.id='id,time').
               Note that you can combine variables with ^ only inside formulas (see the dedi-
               cated section in [feols](#)).

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 re-
               moved prior to estimation and a note is displayed detailing the number of obser-
               vations removed. Otherwise, an error is raised.

fixef.tol   Precision used to obtain the fixed-effects. Defaults to 1e-5. It corresponds to the
                 maximum absolute difference allowed between two coefficients of successive it-
                 erations. 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 fixed-effects algorithm (only in use for 2+
                 fixed-effects). 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 threads used within this
                 package using the function [setFixest_nthreads](#).

verbose   Integer, default is 0. It represents the level of information that should be re-
                 ported 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 fixed-
                 effects coefficients and the hessian are reported.

warn   Logical, default is TRUE. Whether warnings should be displayed (concerns warn-
                 ings relating to convergence state).

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 setup (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 pro-
                 vides 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 estima-
                 tion, 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

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 rates).

Value

A fixest object.

nobs   The number of observations.
fml   The linear formula of the call.

`call`	The call of the function.
`method`	The method used to estimate the model.
`family`	The family used to estimate the model.
`fml_full`	[where relevant] The "full" formula containing the linear part and the fixed-effects.
`nparams`	The number of parameters of the model.
`fixef_vars`	The names of each fixed-effect dimension.
`fixef_id`	The list (of length the number of fixed-effects) of the fixed-effects identifiers for each observation.
`fixef_sizes`	The size of each fixed-effect (i.e. the number of unique identifierfor each fixed-effect dimension).
`convStatus`	Logical, convergence status.
`message`	The convergence message from the optimization procedures.
`obsRemoved`	[where relevant] In the case there were fixed-effects and some observations were removed because of only 0/1 outcome within a fixed-effect, it gives the row numbers of the observations that were removed. Also reports the NA observations that were removed.
`fixef_removed`	[where relevant] In the case there were fixed-effects and some observations were removed because of only 0/1 outcome within a fixed-effect, it gives the list (for each fixed-effect dimension) of the fixed-effect identifiers that were removed.
`coefficients`	The named vector of estimated coefficients.
`coeftable`	The table of the coefficients with their standard errors, z-values and p-values.
`loglik`	The log-likelihood.
`iterations`	Number of iterations of the algorithm.
`ll_null`	Log-likelihood of the null model (i.e. with the intercept only).
`ll_fe_only`	Log-likelihood of the model with only the fixed-effects.
`ssr_null`	Sum of the squared residuals of the null model (containing only with the intercept).
`pseudo_r2`	The adjusted pseudo R2.
`fitted.values`	The fitted values are the expected value of the dependent variable for the fitted model: that is $E(Y|X)$.
`residuals`	The residuals (y minus the fitted values).
`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.
`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).
`residuals`	The difference between the dependent variable and the expected predictor.
`sumFE`	The sum of the fixed-effects coefficients for each observation.
`offset`	[where relevant] The offset formula.
`weights`	[where relevant] The weights formula.

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.

Lagging variables

To use leads/lags of variables in the estimation, you can: i) either provide the argument `panel.id`, ii) either set your data set as a panel with the function `panel`. Doing either of the two will give you acceess to the lagging functions `l` and `f`.

You can provide several leads/lags at once: e.g. if your formula is equal to `f(y) ~ l(x,-1:1)`, it means that the dependent variable is equal to the lead of y, and you will have as explanatory variables the lead of x1, x1 and the lag of x1. See the examples in function `l` for more details.

Interactions

You can interact a numeric variable with a "factor-like" variable by using `interact(var,fe,ref)`, where `fe` is the variable to be interacted with and the argument `ref` is a value of `fe` taken as a reference (optional). Instead of using the function `interact`, you can use the alias `i(var,fe,ref)` or even the highly specific syntax `var::fe(ref)`.

It is important to note that *if you do not care about the standard-errors of the interactions*, then you can add interactions in the fixed-effects part of the formula (using the syntax fe[[var]], as explained in the section "Varying slopes").

Using this specific way to create interactions leads to a different display of the interacted values in `etable` and offers a special representation of the interacted coefficients in the function `coefplot`. See examples.

The function `interact` has in fact more arguments, please see details in its associated help page.

On standard-errors

Standard-errors can be computed in different ways, you can use the arguments `se` and `dof` in `summary.fixest` to define how to compute them. By default, in the presence of fixed-effects, standard-errors are automatically clustered.

The following vignette: On standard-errors describes in details how the standard-errors are computed in `fixest` and how you can replicate standard-errors from other software.

You can use the functions `setFixest_se` and `setFixest_dof` to permanently set the way the standard-errors are computed.

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 fixed-effects coefficients, and the function `etable` 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,
  panel.id,
  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 = 10000,
  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 fixed-effects variables, you can 1) either insert them in this formula using a pipe (e.g. `fml = z~x+y|fixef_1+fixef_2`), or 2) either use the argument `fixef`. 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 names of variables to be used as fixed-effects. These variables should contain the identifier of each observation (e.g., think of it as a panel identifier). Note that the recommended way to include fixed-effects is to insert them directly in the formula.
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'.
panel.id	The panel identifiers. Can either be: i) a one sided formula (e.g. `panel.id = ~id+time`), ii) a character vector of length 2 (e.g. `panel.id=c('id','time')`), or iii) a character scalar of two variables separated by a comma (e.g. `panel.id='id,time'`). Note that you can combine variables with ^ only inside formulas (see the dedicated section in [feols](#)).
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 threads 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 fixed-effects 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. 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$double.eps. Note that this parameter is dynamically controlled by the algorithm.
fixef.iter	Maximum number of iterations in fixed-effects algorithm (only in use for 2+ fixed-effects). Default is 10000.
deriv.tol	Precision used to obtain the fixed-effects derivatives. Defaults to 1e-4. It corresponds to the maximum absolute difference allowed between two coefficients of successive iterations. Argument `deriv.tol` cannot be lower than 10000*.Machine$double.eps.

deriv.iter	Maximum number of iterations in the algorithm to obtain the derivative of the fixed-effects (only in use for 2+ fixed-effects). Default is 1000.
warn	Logical, default is TRUE. Whether warnings should be displayed (concerns warnings relating to convergence state).
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 setup (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 fixed-effects, 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 fixed-effect dimension k such that C_{im}^k is equal to 1 if observation i is of category m in the fixed-effect dimension 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 fixed-effects 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

A `fixest` 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.
`nobs`	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 fixed-effect dimension.
fixef_id	The list (of length the number of fixed-effects) of the fixed-effects identifiers for each observation.
fixef_sizes	The size of each fixed-effect (i.e. the number of unique identifierfor each fixed-effect dimension).
obsRemoved	In the case there were fixed-effects and some observations were removed because of only 0/1 outcome within a fixed-effect, it gives the row numbers of the observations that were removed. Also reports the NA observations that were removed.
fixef_removed	In the case there were fixed-effects and some observations were removed because of only 0/1 outcome within a fixed-effect, it gives the list (for each fixed-effect dimension) of the fixed-effect 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 fixed-effects coefficients, and the function `etable` to visualize the results of multiple estimations.

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

Lagging variables

To use leads/lags of variables in the estimation, you can: i) either provide the argument `panel.id`, ii) either set your data set as a panel with the function `panel`. Doing either of the two will give you acceess to the lagging functions `l` and `f`.

You can provide several leads/lags at once: e.g. if your formula is equal to `f(y) ~ l(x,-1:1)`, it means that the dependent variable is equal to the lead of y, and you will have as explanatory variables the lead of x1, x1 and the lag of x1. See the examples in function `l` for more details.

Interactions

You can interact a numeric variable with a "factor-like" variable by using `interact(var,fe,ref)`, where fe is the variable to be interacted with and the argument ref is a value of fe taken as a reference (optional). Instead of using the function `interact`, you can use the alias `i(var,fe,ref)` or even the highly specific syntax `var::fe(ref)`.

It is important to note that *if you do not care about the standard-errors of the interactions*, then you can add interactions in the fixed-effects part of the formula (using the syntax fe[[var]], as explained in the section "Varying slopes").

Using this specific way to create interactions leads to a different display of the interacted values in `etable` and offers a special representation of the interacted coefficients in the function `coefplot`. See examples.

The function `interact` has in fact more arguments, please see details in its associated help page.

On standard-errors

Standard-errors can be computed in different ways, you can use the arguments se and dof in `summary.fixest` to define how to compute them. By default, in the presence of fixed-effects, standard-errors are automatically clustered.

The following vignette: On standard-errors describes in details how the standard-errors are computed in `fixest` and how you can replicate standard-errors from other software.

You can use the functions `setFixest_se` and `setFixest_dof` to permanently set the way the standard-errors are computed.

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 or feglm instead

# Generating data for a simple example
set.seed(1)
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,
  panel.id,
  fixef,
  fixef.tol = 1e-06,
  fixef.iter = 10000,
  na_inf.rm = getFixest_na_inf.rm(),
  nthreads = getFixest_nthreads(),
  verbose = 0,
  warn = TRUE,
  notes = getFixest_notes(),
  combine.quick,
  demeaned = FALSE,
  ...
)
```

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 fixed-effects 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'.
panel.id	The panel identifiers. Can either be: i) a one sided formula (e.g. panel.id = ~id+time), ii) a character vector of length 2 (e.g. panel.id=c('id','time'), or iii) a character scalar of two variables separated by a comma (e.g. panel.id='id,time'). Note that you can combine variables with ^ only inside formulas (see the dedicated section in [feols](#)).
fixef	Character vector. The names of variables to be used as fixed-effects. These variables should contain the identifier of each observation (e.g., think of it as a panel identifier). Note that the recommended way to include fixed-effects is to insert them directly in the formula.
fixef.tol	Precision used to obtain the fixed-effects. 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 fixed-effects algorithm (only in use for 2+ fixed-effects). 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 threads used within this package using the function [setFixest_nthreads](#).
verbose	Integer. Higher values give more information. In particular, it can detail the number of iterations in the demeaning algoritmh (the first number is the left-hand-side, the other numbers are the right-hand-side variables).
warn	Logical, default is TRUE. Whether warnings should be displayed (concerns warnings relating to convergence state).
notes	Logical. By default, two notes are displayed: when NAs are removed (to show additional information) and when some observations are removed because of collinearity. 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.
demeaned	Logical, default is `FALSE`. Only used in the presence of fixed-effects: should the centered variables be returned? If `TRUE`, it creates the items `y_demeaned` and `X_demeaned`.
...	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.

Value

A `fixest` object.

nobs	The number of observations.
fml	The linear formula of the call.
call	The call of the function.
method	The method used to estimate the model.
family	The family used to estimate the model.
fml_full	[where relevant] The "full" formula containing the linear part and the fixed-effects.
nparams	The number of parameters of the model.
fixef_vars	The names of each fixed-effect dimension.
fixef_id	The list (of length the number of fixed-effects) of the fixed-effects identifiers for each observation.
fixef_sizes	The size of each fixed-effect (i.e. the number of unique identifierfor each fixed-effect dimension).
coefficients	The named vector of estimated coefficients.
multicol	Logical, if multicollinearity was found.
coeftable	The table of the coefficients with their standard errors, z-values and p-values.
loglik	The loglikelihood.
ssr_null	Sum of the squared residuals of the null model (containing only with the intercept).
ssr_fe_only	Sum of the squared residuals of the model estimated with fixed-effects only.
ll_null	The log-likelihood of the null model (containing only with the intercept).

`ll_fe_only`	The log-likelihood of the model estimated with fixed-effects only.
`pseudo_r2`	The adjusted pseudo R2.
`fitted.values`	The fitted values.
`linear.predictors`	The linear predictors.
`residuals`	The residuals (y minus the fitted values).
`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.
`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).
`residuals`	The difference between the dependent variable and the expected predictor.
`sumFE`	The sum of the fixed-effects coefficients for each observation.
`offset`	[where relevant] The offset formula.
`weights`	[where relevant] The weights formula.
`collin.var`	[where relevant] Vector containing the variables removed because of collinearity.
`collin.coef`	[where relevant] Vector of coefficients, where the values of the variables removed because of collinearity are NA.
`y_demeaned`	Only when demeaned = TRUE: the centered dependent variable.
`X_demeaned`	Only when demeaned = TRUE: the centered explanatory variable.

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`](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: fixef_var[var1, var2]. Here the variables var1 and var2 will be with varying slopes (one slope per value in fixef_var) and the fixed-effect fixef_var will also be added.

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

In other words:

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

Lagging variables

To use leads/lags of variables in the estimation, you can: i) either provide the argument panel.id, ii) either set your data set as a panel with the function panel. Doing either of the two will give you acceess to the lagging functions l and f.

You can provide several leads/lags at once: e.g. if your formula is equal to f(y) ~ l(x,-1:1), it means that the dependent variable is equal to the lead of y, and you will have as explanatory variables the lead of x1, x1 and the lag of x1. See the examples in function l for more details.

Interactions

You can interact a numeric variable with a "factor-like" variable by using interact(var,fe,ref), where fe is the variable to be interacted with and the argument ref is a value of fe taken as a reference (optional). Instead of using the function interact, you can use the alias i(var,fe,ref) or even the highly specific syntax var::fe(ref).

It is important to note that *if you do not care about the standard-errors of the interactions*, then you can add interactions in the fixed-effects part of the formula (using the syntax fe[[var]], as explained in the section "Varying slopes").

Using this specific way to create interactions leads to a different display of the interacted values in etable and offers a special representation of the interacted coefficients in the function coefplot. See examples.

The function interact has in fact more arguments, please see details in its associated help page.

On standard-errors

Standard-errors can be computed in different ways, you can use the arguments se and dof in summary.fixest to define how to compute them. By default, in the presence of fixed-effects, standard-errors are automatically clustered.

The following vignette: On standard-errors describes in details how the standard-errors are computed in fixest and how you can replicate standard-errors from other software.

You can use the functions setFixest_se and setFixest_dof to permanently set the way the standard-errors are computed.

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 fixed-effects coefficients, and the function `etable` to visualize the results of multiple estimations. For plotting coefficients: see `coefplot`.

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

Examples

```
#
# Just one set of fixed-effects:
#

res = feols(Sepal.Length ~ Sepal.Width + Petal.Length | Species, iris)
summary(res)

#
# 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]]

#
# Using leads/lags:
#

data(base_did)
# We need to set up the panel with the arg. panel.id
est1 = feols(y~l(x1, 0:1), base_did, panel.id = ~id+period)
est2 = feols(f(y)~l(x1, -1:1), base_did, panel.id = ~id+period)
etable(est1, est2, order = "f", drop="Int")

#
# Using interactions:
#

# NOTA: in fixest estimations, i(var, fe, ref) is equivalent to var::fe(ref)

data(base_did)
# We interact the variable 'period' with the variable 'treat'
```

```
est_did = feols(y ~ x1 + i(treat, period, 5) | id+period, base_did)

# You could have used the following formula instead:
# y ~ x1 + treat::period(5) | id+period

# Now we can plot the result of the interaction with coefplot
coefplot(est_did)
# You have many more example in coefplot help
```

---

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"), ...)

fitted.values.fixest
```

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 pre-
            dictor $E(Y|X)$. If "link", then the output is at the level of the explanatory
            variables, i.e. the linear predictor $X \cdot \beta$.

...   Not currently used.

Format

An object of class function of length 1.

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, using "Species" fixed-effects
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 fixed-effect dimensions.

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 fixed-effect coefficients, and the function [etable] 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 fixed-effect dimension:
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 fixed-effects (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, using "Species" fixed-effects
res = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
            Petal.Width | Species, iris)

# formula with the fixed-effect variable
formula(res)
```

```
# linear part without the fixed-effects
formula(res, "linear")
```

---

hatvalues.fixest   *Hat values for* fixest *objects*

Description

Computes the hat values for [feols](#) or [feglm](#) estimations. Only works when there are no fixed-effects.

Usage

```
## S3 method for class 'fixest'
hatvalues(model, ...)
```

Arguments

model	A fixest object. For instance from feols or feglm.
...	Not currently used.

Details

Hat values are not available for [fenegbin](#), [femlm](#) and [feNmlm](#) estimations.

When there are fixed-effects, the hat values of the reduced form are different from the hat values of the full model. And we cannot get costlessly the hat values of the full model from the reduced form. It would require to reestimate the model with the fixed-effects as regular variables.

Value

Returns a vector of the same length as the number of observations used in the estimation.

Examples

```
est = feols(Petal.Length ~ Petal.Width + Sepal.Width, iris)
head(hatvalues(est))
```

---

i *Interact variables with factors*

Description

Interacts a variable with another treated as a factor, and sets a reference

Usage

```
i(var, fe, ref, drop, keep)

interact
```

Arguments

var	A vector.
fe	A vector (of any type). Must be of the same length as var.
ref	A single value that belongs to the interacted variable (fe). Can be missing.
drop	A vector of values that belongs to the interacted variable (fe). If provided, all values from fe that match drop will be removed.
keep	A vector of values that belongs to the interacted variable (fe). If provided, only the values from fe that match keep will be kept.

Format

An object of class function of length 1.

Value

It returns a matrix with number of rows the length of var. The number of columns is equal to the number of cases contained in fe minus the reference.

Shorthand in fixest estimations

In fixest estimations, instead of using i(var,fe,ref), you can instead use the following writing var::fe(ref).

Author(s)

Laurent Berge

See Also

coefplot to plot interactions, feols for OLS estimation with multiple fixed-effects.

Examples

```
#
# Simple illustration
#

x = rnorm(10)
y = rep(1:4, 3)[1:10]

cbind(x, y, i(x, y, 1))

#
# In fixest estimations
#

# NOTA: in fixest estimations, i(var, fe, ref) is equivalent to var::fe(ref)

data(base_did)
# We interact the variable 'period' with the variable 'treat'
est_did = feols(y ~ x1 + i(treat, period, 5) | id+period, base_did)

# You could have used the following formula instead:
# y ~ x1 + treat::period(5) | id+period
```

---

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 = 1,
  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. Default is 1. For leads, just use a negative number.
data	Optional, the data.frame in which to evaluate the formula. If not provided, variables will be fetched in the current environment.
time.step	The method to compute the lags. Can be equal to: `"unitary"` (default), `"consecutive"`, `"within.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. If `"witihn.consecutive"` then **within a given id**, 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	Scalar. 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

See Also

Alternatively, the function [panel](#) changes a data.frame into a panel from which the functions l and f (creating leads and lags) can be called. Otherwise you can set the panel 'live' during the estimation using the argument panel.id (see for example in the function [feols](#)).

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)

base$lag1 = lag(x~id+time,  1, base) # lag 1
base$lead1 = lag(x~id+time, -1, base) # lead 1
base$lag2_fill0 = lag(x~id+time, 2, base, fill = 0)
# with time.step = "consecutive"
base$lag1_consecutive = lag(x~id+time, 1, base, time.step = "consecutive")
#   => works for indiv. 2 because 9 (resp. 6) is consecutive to 6 (resp. 4)
```

```
base$lag1_within.consecutive = lag(x~id+time, 1, base, time.step = "within")
#   => now two consecutive years within each indiv is one lag

print(base)

# Argument time.step = "consecutive" is
# 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)

# Error because of duplicate values for (id, time)
try(lag(x~id+time, 1, base_dup))


# Error is bypassed, lag corresponds to first occurence of (id, time)
lag(x~id+time, 1, base_dup, duplicate.method = "first")


# Playing with time steps
base = data.frame(id = rep(1:2, each = 4),
                  time = c(1, 2, 3, 4, 1, 4, 6, 9), x = 1:8)

# time step: 0.5 (here equivalent to lag of 1)
lag(x~id+time, 2, base, time.step = 0.5)

# Error: wrong time step
try(lag(x~id+time, 2, base, time.step = 7))

# Adding NAs + unsorted IDs
base = data.frame(id = rep(1:2, each = 4),
                  time = c(4, NA, 3, 1, 2, NA, 1, 3), x = 1:8)

base$lag1 = lag(x~id+time, 1, base)
base$lag1_within = lag(x~id+time, 1, base, time.step = "w")
base_bis = base[order(base$id, base$time),]

print(base_bis)

# You can create variables without specifying the data within data.table:
if(require("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. The fixed-effects variables (which can be considered as factors) are excluded from the matrix.

Usage

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

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.
na.rm	Default is TRUE. Should observations with NAs be removed from the matrix?
...	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, using "Species" fixed-effects
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 fixed-effects coefficients, and the function etable 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 fixed-effects*

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 fixed-effect value this leads to a perfect fit for that fixed-effect value by setting its associated fixed-effect 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 fixed-effects. It can be of the type: y ~ fixef_1 + fixef_2 or y ~ x1 | fixef_1 + fixef_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 fixef which is a list giving the IDs of the fixed-effects that have been removed, for each fixed-effect 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, "fixef")

# 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)
```

---

panel   *Constructs a* fixest *panel data base*

Description

Constructs a fixest panel data base out of a data.frame which allows to use leads and lags in
fixest estimations and to create new variables from leads and lags if the data.frame was also a
[data.table](#).

Usage

```
panel(
  data,
  panel.id,
  time.step = "unitary",
  duplicate.method = c("none", "first")
)
```

Arguments

data   A data.frame.

panel.id   The panel identifiers. Can either be: i) a one sided formula (e.g. panel.id =
                ~id+time), ii) a character vector of length 2 (e.g. panel.id=c('id','time'),
                or iii) a character scalar of two variables separated by a comma (e.g. panel.id='id,time').
                Note that you can combine variables with ^ only inside formulas (see the dedi-
                cated section in [feols](#)).

time.step   The method to compute the lags. Can be equal to: "unitary" (default), "consecutive",
                "within.consecutive", or to a number. If "unitary", then the largest com-
                mon divisor between consecutive time periods is used (typically if the time
                variable represents years, it will be 1). This method can apply only to inte-
                ger (or convertible to integer) variables. If "consecutive", then the time vari-
                able can be of any type: two successive time periods represent a lag of 1. If
                "witihn.consecutive" then **within a given id**, two successive time peri-
                ods represent a lag of 1. Finally, if the time variable is numeric, you can provide
                your own numeric time step.

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 dupli-
                cate 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.
```

## Details

This function allows you to use leads and lags in a `fixest` estimation without having to provide the argument `panel.id`. It also offers more options on how to set the panel (with the additional arguments 'time.step' and 'duplicate.method').

When the initial data set was also a `data.table`, not all operations are supported and some may dissolve the `fixest_panel`. This is the case when creating subselections of the initial data with additional attributes (e.g. pdt[x>0, .(x, y, z)] would dissolve the `fixest_panel`, meaning only a data.table would be the result of the call).

If the initial data set was also a `data.table`, then you can create new variables from lags and leads using the functions l() and f(). See the example.

## Value

It returns a data base identical to the one given in input, but with an additional attribute: "panel_info". This attribute contains vectors used to efficiently create lags/leads of the data. When the data is subselected, some bookeeping is performed on the attribute "panel_info".

## Author(s)

Laurent Berge

## See Also

The estimation methods feols, fepois and feglm.

The functions l and f to create lags and leads within fixest_panel objects.

## Examples

```
data(base_did)

# Setting a data set as a panel...
pdat = panel(base_did, ~id+period)

# ...then using the functions l and f
est1 = feols(y~l(x1, 0:1), pdat)
est2 = feols(f(y)~l(x1, -1:1), pdat)
est3 = feols(l(y)~l(x1, 0:3), pdat)
etable(est1, est2, est3, order = c("f", "^x"), drop="Int")

# or using the argument panel.id
feols(f(y)~l(x1, -1:1), base_did, panel.id = ~id+period)

# You can use panel.id in various ways:
pdat = panel(base_did, ~id+period)
# is identical to:
pdat = panel(base_did, c("id", "period"))
# and also to:
pdat = panel(base_did, "id,period")
```

```
# l() and f() can also be used within a data.table:
if(require("data.table")){
  pdat_dt = panel(as.data.table(base_did), ~id+period)
  # Now since pdat_dt is also a data.table
  #   you can create lags/leads directly
  pdat_dt[, x1_l1 := l(x1)]
  pdat_dt[, c("x1_l1_fill0", "y_f2") := .(l(x1, fill = 0), f(y, 2))]
}
```

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 fixed-effect
dimension. 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 fixed-effect 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 function
`etable` 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 fixed-effects coefficients, and the function `etable` 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", full_names = FALSE)
```

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) possi-
              bly 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.

full_names   Logical scalar, default is FALSE. If TRUE then names of the vector in output will
              have full names instead of keywords (e.g. Squared Correlation instead of
              sq.cor, etc).

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.

The pseudo-R2 is also returned in the OLS case, it corresponds to the pseudo-R2 of the equivalent
GLM model with a Gaussian family.

For the adjusted within-R2s, the adjustment factor is (n -nb_fe) / (n -nb_fe -K) with n the num-
ber of observations, nb_fe the number of fixed-effects and K the number of variables.

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 = feols(log(Euros) ~ log(dist_km)|Origin+Destination+Product, trade)

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

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

# pseudo r2 (equivalent to GLM with Gaussian family)
r2(est, "pr2")

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

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

# same with full names instead of codes
r2(est, c("sq.cor", "r2", "pr2", "war2"), full_names = TRUE)
```

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, type = c("response", "deviance", "pearson", "working"), ...)

residuals.fixest
```

Arguments

object	A fixest object. Obtained using the functions [femlm](#), [feols](#) or [feglm](#).
type	A character scalar, either "response" (default), "deviance", "pearson", or "working". Note that the "working" corresponds to the residuals from the weighted least square and only applies to [feglm](#) models.
...	Not currently used.

Format

An object of class function of length 1.

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, using "Species" fixed-effects
res_poisson = femlm(Sepal.Length ~ Sepal.Width + Petal.Length +
                    Petal.Width | Species, iris)

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

---

setFixest_coefplot *Sets the defaults of coefplot*

Description

You can set the default values of most arguments of coefplot with this function.

Usage

```
setFixest_coefplot(
  style,
  horiz = FALSE,
  dict = getFixest_dict(),
  keep,
  ci.width = "1%",
  ci_level = 0.95,
  pt.pch = 20,
```

```
    pt.bg = NULL,
    cex = 1,
    pt.cex = cex,
    col = 1:8,
    pt.col = col,
    ci.col = col,
    lwd = 1,
    pt.lwd = lwd,
    ci.lwd = lwd,
    ci.lty = 1,
    grid = TRUE,
    grid.par = list(lty = 3, col = "gray"),
    zero = TRUE,
    zero.par = list(col = "black", lwd = 1),
    pt.join = FALSE,
    pt.join.par = list(col = pt.col, lwd = lwd),
    ci.join = FALSE,
    ci.join.par = list(lwd = lwd, col = col, lty = 2),
    ci.fill = FALSE,
    ci.fill.par = list(col = "lightgray", alpha = 0.5),
    ref.line = "auto",
    ref.line.par = list(col = "black", lty = 2),
    lab.cex,
    lab.min.cex = 0.85,
    lab.max.mar = 0.25,
    lab.fit = "auto",
    xlim.add,
    ylim.add,
    sep,
    bg,
    group = "auto",
    group.par = list(lwd = 2, line = 3, tcl = 0.75),
    main = "Effect on __depvar__",
    value.lab = "Estimate and __ci__ Conf. Int.",
    ylab = NULL,
    xlab = NULL,
    sub = NULL,
    reset = FALSE
)
```

Arguments

style   A character scalar giving the style of the plot to be used. You can set styles with
               the function [setFixest_coefplot](), setting all the default values of the func-
               tion. If missing, then it switches to either "default", "interaction" or "multiple",
               depending on the data given in input.

horiz   A logical scalar, default is FALSE. Whether to display the confidence intervals
               horizontally instead of vertically.

dict   A named character vector or a logical scalar. It changes the original variable
                names to the ones contained in the dictionary. 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, if Tex output is requested or if argument file is not missing, it is
                equal to getFixest_dict(), a default dictionary which can be set with [setFixest_dict]().
                The default is not to change names if a data.frame is requested (i.e. tex =
                FALSE); if so, you can use dict = TRUE to use the dictionary you've set globally
                with setFixest_dict().

keep   Character vector. This element is used to display only a subset of variables.
                This should be a vector of regular expressions (see [regex](#) help for more info).
                Each variable satisfying any of the regular expressions will be kept. This ar-
                gument is applied post aliasing (see argument dict). Example: you have the
                variable x1 to x55 and want to display only x1 to x9, then you could use keep
                = "x[[:digit:]]$". If the first character is an exclamation mark, the effect is
                reversed (e.g. keep = "!Intercept" means: every variable that does not contain
                "Intercept" is kept). See details.

ci.width   The width of the extremities of the confidence intervals. Default is 0.1.

ci_level   Scalar between 0 and 1: the level of the CI. By default it is equal to 0.95.

pt.pch   The patch of the coefficient estimates. Default is 1 (circle).

pt.bg   The background color of the point estimate (when the pt.pch is in 21 to 25).
                Defaults to NULL.

cex   Numeric, default is 1. Expansion factor for the points

pt.cex   The size of the coefficient estimates. Default is the other argument cex.

col   The color of the points and the confidence intervals. Default is 1 ("black"). Note
                that you can set the colors separately for each of them with pt.col and ci.col.

pt.col   The color of the coefficient estimates. Default is equal to the other argument
                col.

ci.col   The color of the confidence intervals. Default is equal to the other argument
                col.

lwd   General liwe with. Default is 1.

pt.lwd   The line width of the coefficient estimates. Default is equal to the other argument
                lwd.

ci.lwd   The line width of the confidence intervals. Default is equal to the other argument
                lwd.

ci.lty   The line type of the confidence intervals. Default is 1.

grid   Logical, default is TRUE. Whether a grid should be displayed. You can set the
                display of the grid with the argument grid.par.

grid.par   List. Parameters of the grid. The default values are: lty = 3 and col = "gray".
                You can add any graphical parameter that will be passed to [abline](#). You also
                have two additional arguments: use horiz = FALSE to disable the horizontal
                lines, and use vert = FALSE to disable the vertical lines. Eg: grid.par = list(vert
                = FALSE,col = "red",lwd = 2).

zero   Logical, default is TRUE. Whether the 0-line should be emphasized. You can set
                the parameters of that line with the argument zero.par.

zero.par	List. Parameters of the zero-line. The default values are col = "black" and lwd = 1. You can add any graphical parameter that will be passed to [abline](#). Example: zero.par = list(col = "darkblue", lwd = 3).
pt.join	Logical, default depends on the situation. If TRUE, then the coefficient estimates are joined with a line. By default, it is equal to TRUE only if: i) interactions are plotted, ii) the x values are numeric and iii) a reference is found.
pt.join.par	List. Parameters of the line joining the cofficients. The default values are: col = pt.col and lwd = lwd. You can add any graphical parameter that will be passed to [lines](#). Eg: pt.join.par = list(lty = 2).
ci.join	Logical default to FALSE. Whether to join the extremities of the confidence intervals. If TRUE, then you can set the graphical parameters with the argument ci.join.par.
ci.join.par	A list of parameters to be passed to [lines](#). Only used if ci.join=TRUE. By default it is equal to list(lwd = lwd, col = col, lty = 2).
ci.fill	Logical default to FALSE. Whether to fille the confidence intervals with a color. If TRUE, then you can set the graphical parameters with the argument ci.fill.par.
ci.fill.par	A list of parameters to be passed to [polygon](#). Only used if ci.fill=TRUE. By default it is equal to list(col = "lightgray", alpha = 0.5). Note that alpha is a special parameter that adds transparency to the color (ranges from 0 to 1).
ref.line	Logical, default is "auto", the behavior depending on the situation. It is TRUE only if: i) interactions are plotted, ii) the x values are numeric and iii) a reference is found. If TRUE, then a vertical line is drawn at the level of the reference value. You can set the parameters of this line with the argument ref.line.par.
ref.line.par	List. Parameters of the vertical line on the reference. The default values are: col = "black" and lty = 2. You can add any graphical parameter that will be passed to [abline](#). Eg: ref.line.par = list(lty = 1, lwd = 3).
lab.cex	The size of the labels of the coefficients. Default is missing. It is automatically set by an internal algorithm which can go as low as lab.min.cex (another argument).
lab.min.cex	The minimum size of the coefficients labels, as set by the internal algorithm. Default is 0.85.
lab.max.mar	The maximum size the left margin can take when trying to fit the coefficient labels into it (only when horiz = TRUE). This is used in the internal algorithm fitting the coefficient labels. Default is 0.25.
lab.fit	The method to fit the coefficient labels into the plotting region (only when horiz = FALSE). Can be "auto" (the default), "simple", "multi" or "tilted". If "simple", then the classic axis is drawn. If "multi", then the coefficient labels are fit horizontally across several lines, such that they don't collide. If "tilted", then the labels are tilted. If "auto", an automatic choice between the three is made.
xlim.add	A numeric vector of length 1 or 2. It represents an extension factor of xlim, in percentage. Eg: xlim.add = c(0,0.5) extends xlim of 50% on the right. If of lentgh 1, positive values represent the right, and negative values the left (Eg: xlim.add = -0.5 is equivalent to xlim.add = c(0.5,0)).

ylim.add	A numeric vector of length 1 or 2. It represents an extension factor of ylim, in percentage. Eg: ylim.add = c(0,0.5) extends ylim of 50% on the top. If of lentgh 1, positive values represent the top, and negative values the bottom (Eg: ylim.add = -0.5 is equivalent to ylim.add = c(0.5,0)).
sep	The distance between two estimates – only when argument object is a list of estimation results.
bg	Background color for the plot. By default it is white.
group	A list, default is missing. Each element of the list reports the coefficients to be grouped while the name of the element is the group name. Each element of the list can be either: i) a character vector of length 1, ii) of length 2, or ii) a numeric vector. If equal to: i) then it is interpreted as a pattern: all element fitting the regular expression will be grouped, if ii) it corrsponds to the first and last elements to be grouped, if iii) it corresponds to the coefficients numbers to be grouped. If equal to a character vector, you can use a percentage to tell the algorithm to look at the coefficients before aliasing (e.g. "%varname"). Example of valid uses: group=list(group_name=\"pattern\"), group=list(group_name=c(\"var_start\",\"var_end\" group=list(group_name=1:2)). See details.
group.par	A list of parameters controlling the display of the group. The parameters controlling the line are: lwd, tcl (length of the tick), line.adj (adjustment of the position, default is 0), tick (whether to add the ticks), lwd.ticks, col.ticks. Then the parameters controlling the text: text.adj (adjustment of the position, default is 0), text.cex, text.font, text.col.
main	The title of the plot. Default is "Effect on __depvar__". You can use the special variable __depvar__ to set the title (useful when you set the plot default with [setFixest_coefplot](#)).
value.lab	The label to appear on the side of the coefficient values. If horiz = FALSE, the label appears in the y-axis. If horiz = TRUE, then it appears on the x-axis. The default is equal to "Estimate and __ci__ Conf. Int.", with __ci__ a special variable giving the value of the confidence interval.
ylab	The label of the y-axis, default is NULL. Note that if horiz = FALSE, it overrides the value of the argument value.lab.
xlab	The label of the x-axis, default is NULL. Note that if horiz = TRUE, it overrides the value of the argument value.lab.
sub	A subtitle, default is NULL.
reset	Logical, default is TRUE. If TRUE, then the arguments that *are not* set during the call are reset to their "factory"-default values. If FALSE, on the other hand, arguments that have already been modified are not changed.

Value

Doesn't return anything.

See Also

[coefplot](#)

Examples

```
# coefplot has many arguments, which makes it highly flexible.
# If you don't like the default style of coefplot. No worries,
# you can set *your* default by using the function
# setFixest_coefplot()

# Estimation
est = feols(Petal.Length ~ Petal.Width + Sepal.Length +
                 Sepal.Width | Species, iris)

# Plot with default style
coefplot(est)

# Now we permanently change some arguments
dict = c("Petal.Length"="Length (Petal)", "Petal.Width"="Width (Petal)",
         "Sepal.Length"="Length (Sepal)", "Sepal.Width"="Width (Sepal)")

setFixest_coefplot(ci.col = 2, pt.col = "darkblue", ci.lwd = 3,
                   pt.cex = 2, pt.pch = 15, ci.width = 0, dict = dict)

# Tadaaa!
coefplot(est)

# To reset to the default settings:
setFixest_coefplot()
coefplot(est)
```

---

setFixest_dict   *Sets/gets the dictionary relabeling the variables*

Description

Sets/gets the default dictionary used in the function [etable](#), [did_means](#) and [coefplot](#). The dictionaries are used to relabel variables (usually towards a fancier, more explicit formatting) when exporting them into a Latex table or displaying in graphs. By setting the dictionary with setFixest_dict, you can avoid providing the argument dict.

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="$bonus^3$").

This dictionary is used in Latex tables or in graphs by the function [coefplot](#). If you want to separate Latex rendering from rendering in graphs, use an ampersand first to make the variable specific to coefplot.

Format

An object of class `function` of length 1.

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.

Format

An object of class `function` of length 1.

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
try(feols(Sepal.Length ~ Sepal.Width, base, na_inf.rm = FALSE))

# to set no tolerance as default:
setFixest_na_inf.rm(FALSE)
try(feols(Sepal.Length ~ Sepal.Width, base))

# Reset it on:
setFixest_na_inf.rm(TRUE)
```

---

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.

Format

An object of class function of length 1.

Author(s)

Laurent Berge

Examples

```
# Change default with
setFixest_notes(FALSE)

# Back to default which is TRUE
getFixest_notes()
```

---

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.

Format

An object of class function of length 1.

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".

Format

An object of class function of length 1.

Author(s)

Laurent Berge

Examples

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

# back to default
setFixest_print.type("table")
```

setFixest_se
Sets the default type of standard errors to be used

Description

This functions defines or extracts the default type of standard-errors to computed in fixest summary, and vcov.

Usage

```
setFixest_se(no_FE = ″standard″, one_FE = ″cluster″, two_FE = ″cluster″)

getFixest_se()
```

Arguments

no_FE   Character scalar equal to either: ″standard″ (default), or ″white″. The type of
               standard-errors to use by default for estimations without fixed-effects.

one_FE   Character scalar equal to either: ″standard″, ″white″, or ″cluster″ (default).
               The type of standard-errors to use by default for estimations with *one* fixed-
               effect.

two_FE   Character scalar equal to either: ″standard″, ″white″, ″cluster″, or ″twoway″
               (default). The type of standard-errors to use by default for estimations with *two
               or more* fixed-effects.

Value

The function getFixest_se() returns a list with three elements containing the default for estima-
tions i) wihtout, ii) with one, or iii) with two or more fixed-effects.

Examples

```
# By default:
# - no fixed-effect (FE): standard
# - one or more FEs: cluster

data(base_did)
est_no_FE  = feols(y ~ x1, base_did)
est_one_FE = feols(y ~ x1 | id, base_did)
est_two_FE = feols(y ~ x1 | id + period, base_did)

etable(est_no_FE, est_one_FE, est_two_FE)

# Changing the default standard-errors
setFixest_se(no_FE = ″white″, one_FE = ″standard″, two_FE = ″twoway″)
etable(est_no_FE, est_one_FE, est_two_FE)
```

```
# Reseting the defaults
setFixest_se()
```

---

sigma.fixest   *Residual standard deviation of fixest estimations*

Description

Extract the estimated standard deviation of the errors from `fixest` estimations.

Usage

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

Arguments

object   A `fixest` object.

...   Not currently used.

Value

Returns a numeric scalar.

See Also

[feols](), [fepois](), [feglm](), [fenegbin](), [feNmlm]().

Examples

```
est = feols(Petal.Length ~ Petal.Width, iris)
sigma(est)
```

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 = getFixest_dof(),
  .vcov,
  forceCovariance = FALSE,
  keepBounded = FALSE,
  n,
  ...
)

summ
```

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 vec-
              tor. 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 argu-
              ments 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   An object of class dof.type obtained with the function [dof](#). Represents how the
              degree of freedom correction should be done.You must use the function [dof](#) for
              this argument. The arguments and defaults of the function [dof](#) are: adj = TRUE,
              fixef.K="nested", cluster.adj = TRUE, cluster.df = "conventional", t.df

= "conventional", fixef.force_exact=FALSE). See the help of the function [dof](#) for details.

.vcov
: A user provided covariance matrix. Must be a square matrix of the same number of rows as the number of variables estimated.

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).

n
: Integer, default is missing (means Inf). Number of coefficients to display when the print method is used.

...
: Not currently used.

Format

An object of class function of length 1.

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 fixed-effects coefficients, and the function [etable](#) to visualize the results of multiple estimations.

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)

# Comparing different types of standard errors
```

```
sum_standard = summary(est_pois, se = "standard")
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")

etable(sum_standard, 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 fixed-effects coefficients*

Description

This function summarizes the main characteristics of the fixed-effects 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 fixed-effect dimension are reported.
...	Not currently used.

Value

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

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 fixed-effects 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 fixed-effects removed per fixed-effect 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)
```

---

terms.fixest   *Extract the terms*

Description

This function extracts the terms of a fixest estimation, excluding the fixed-effects part.

Usage

```
## S3 method for class 'fixest'
terms(x, ...)
```

Arguments

x   A fixest object. Obtained using the functions [femlm](), [feols]() or [feglm]().

...   Not currently used.

Value

An object of class c("terms","formula") which contains the terms representation of a symbolic model.

Examples

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

# Terms of the linear part
terms(res)
```

---

to_integer   *Fast transform of any type of vector(s) into an integer vector*

Description

Tool to transform any type of vector, or even combination of vectors, into an integer vector ranging from 1 to the number of unique values. This actually creates an unique identifier vector.

Usage

```
to_integer(
  ...,
  sorted = FALSE,
  add_items = FALSE,
  items.list = FALSE,
  multi.join = FALSE
)
```

Arguments

...	Vectors of any type, to be transformed in integer.
sorted	Logical, default is FALSE. Whether the integer vector should make reference to sorted values?
add_items	Logical, default is FALSE. Whether to add the unique values of the original vector(s). If requested, an attribute items is created containing the values (alternatively, they can appear in a list if items.list=TRUE).
items.list	Logical, default is FALSE. Only used if add_items=TRUE. If TRUE, then a list of length 2 is returned with x the integer vector and items the vector of items.
multi.join	Logical, or character, scalar, defaults to FALSE. Only used if multiple vectors are to be transformed into integers. If multi.join is not FALSE, the the values of the different vectors will be collated using [paste](#) with collapse=multi.join.

Details

If multiple vectors have to be combined and add_items=TRUE, to have user readable values in the items, you should add the argument multi.join so that the values of the vectors are combined in a "user-readable" way. Note that in the latter case, the algorithm is much much slower.

Value

Reruns a vector of the same length as the input vectors. If add_items=TRUE and items.list=TRUE, a list of two elements is returned: x being the integer vector and items being the unique values to which the values in x make reference.

Examples

```
x1 = iris$Species
x2 = as.integer(iris$Sepal.Length)

# transforms the species vector into integers
to_integer(x1)

# To obtain the "items":
to_integer(x1, add_items = TRUE)
# same but in list form
to_integer(x1, add_items = TRUE, items.list = TRUE)

# transforms x2 into an integer vector from 1 to 4
to_integer(x2, add_items = TRUE)

# To have the sorted items:
to_integer(x2, add_items = TRUE, sorted = TRUE)

# The result can safely be used as an index
res = to_integer(x2, add_items = TRUE, sorted = TRUE, items.list = TRUE)
all(res$items[res$x] == x2)


#
# Multiple vectors
#

# by default, the two vector are fast combined, and items are meaningless
to_integer(x1, x2, add_items = TRUE)

# You can use multi.join to have human-readable values for the items:
to_integer(x1, x2, add_items = TRUE, multi.join = TRUE)

to_integer(x1, x2, add_items = TRUE, multi.join = "; ")
```

---

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.

---

unpanel	*Dissolves a* fixest *panel*

Description

Transforms a fixest_panel object into a regular data.frame.

Usage

```
unpanel(x)
```

Arguments

x   A fixest_panel object (obtained from function [panel](#)).

Value

Returns a data set of the exact same dimension. Only the attribute 'panel_info' is erased.

Author(s)

Laurent Berge

See Also

Alternatively, the function [panel](#) changes a data.frame into a panel from which the functions l and f (creating leads and lags) can be called. Otherwise you can set the panel 'live' during the estimation using the argument panel.id (see for example in the function [feols](#)).

Examples

```
data(base_did)

# Setting a data set as a panel
pdat = panel(base_did, ~id+period)

# ... allows you to use leads and lags in estimations
feols(y~l(x1, 0:1), pdat)

# Now unpanel => returns the initial data set
class(pdat) ; dim(pdat)
new_base = unpanel(pdat)
class(new_base) ; dim(new_base)
```

---

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 fixed-effects. 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 fixed-effect: "Product"
est_3 <- update(est_2, . ~ . | . + Product)

# we remove the fixed-effect "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 = getFixest_dof(),
  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	An object of class dof.type obtained with the function [dof](). Represents how the degree of freedom correction should be done.You must use the function [dof]() for this argument. The arguments and defaults of the function [dof]() are: adj = TRUE, fixef.K="nested", cluster.adj = TRUE, cluster.df = "conventional", t.df = "conventional", fixef.force_exact=FALSE). See the help of the function [dof]() for details.
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 fixed-effects:
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)
```

---

weights.fixest   *Extracts the weights from a fixest object*

Description

Simply extracts the weights used to estimate a `fixest` model.

Usage

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

Arguments

object   A `fixest` object.

...   Not currently used.

Value

Returns a vector of the same length as the number of observations in the original data set. Ignored observations due to NA or perfect fit are re-introduced and their weights set to NA.

See Also

[feols](), [fepois](), [feglm](), [fenegbin](), [feNmlm]().

Examples

```
est = feols(Petal.Length ~ Petal.Width, iris, weights = ~as.integer(Sepal.Length) - 3.99)
weights(est)
```

---

xpd   *Sets/gets and expands formula macros*

Description

You can set formula macros globally with setFixest_fml. These macros can then be used in fixest estimations or when using the function [xpd]().

Usage

```
xpd(fml, ...)

setFixest_fml(..., reset = FALSE)

getFixest_fml()
```

Arguments

fml   A formula containing macros variables. The macro variables can be set globally
                using setFixest_fml, or can be defined in ....

...   Definition of the macro variables. Each argument name corresponds to the
                name of the macro variable. It is required that each macro variable name starts
                with two dots (e.g. ..ctrl). The value of each argument must be a one-
                sided formula, it is the definition of the macro variable. Example of a valid
                call: setFixest_fml(..ctrl = ~ var1 + var2). In the function xpd, the de-
                fault macro variables are taken from getFixest_fml, any variable in ... will
                replace these values.

reset   A logical scalar, defaults to FALSE. If TRUE, all macro variables are first reset
                (i.e. deleted).

Details

In xpd, the default macro variables are taken from getFixest_fml. Any value in the ... argument of xpd will replace these default values.

The definitions of the macro variables will replace in verbatim the macro variables. Therefore, you can include multipart formulas if you wish but then beware of the order the the macros variable in the formula. For example, using the airquality data, say you want to set as controls the variable Temp and Day fixed-effects, you can do setFixest_fml(..ctrl = ~Temp | Day), but then feols(Ozone ~ Wind + ..ctrl,airquality) will be quite different from feols(Ozone ~ ..ctrl + Wind,airquality), so beware!

Value

The function getFixest_fml() returns a list of character strings, the names corresponding to the macro variable names, the character strings corresponding to their definition.

Examples

```
# Small examples with airquality data
data(airquality)
# we set two macro variables
setFixest_fml(..ctrl = ~ Temp + Day,
              ..ctrl_long = ~ poly(Temp, 2) + poly(Day, 2))

# Using the macro in lm with xpd:
lm(xpd(Ozone ~ Wind + ..ctrl), airquality)
lm(xpd(Ozone ~ Wind + ..ctrl_long), airquality)

# You can use the macros without xpd() in fixest estimations
a <- feols(Ozone ~ Wind + ..ctrl, airquality)
b <- feols(Ozone ~ Wind + ..ctrl_long, airquality)
etable(a, b, keep = "Int|Win")
```

`[.fixest_panel`   *Method to subselect from a* `fixest_panel`

Description

Subselection from a fixest_panel which has been created with the function panel. Also allows to create lag/lead variables with functions l()/f() if the fixest_panel is also a data.table.

Usage

```
## S3 method for class 'fixest_panel'
x[i, j, ...]
```

Arguments

x	A `fixest_panel` object, created with the function [`panel`](#).
i	Row subselection. Allows [`data.table`](#) style selection (provided the data is also a data.table).
j	Variable selection. Allows [`data.table`](#) style selection/variable creation (provided the data is also a data.table).
...	Other arguments to be passed to `[.data.frame` or [`data.table`](#) (or whatever the class of the initial data).

Details

If the original data was also a data.table, some calls to `[.fixest_panel` may dissolve the `fixest_panel` object and return a regular data.table. This is the case for subselections with additional arguments. If so, a note is displayed on the console.

Value

It returns a `fixest_panel` data base, with the attributes allowing to create lags/leads properly bookkeeped.

Author(s)

Laurent Berge

See Also

Alternatively, the function [`panel`](#) changes a `data.frame` into a panel from which the functions `l` and `f` (creating leads and lags) can be called. Otherwise you can set the panel 'live' during the estimation using the argument `panel.id` (see for example in the function [`feols`](#)).

Examples

```
data(base_did)

# Creating a fixest_panel object
pdat = panel(base_did, ~id+period)

# Subselections of fixest_panel objects bookkeeps the leads/lags engine
pdat_small = pdat[!pdat$period %in% c(2, 4), ]
a = feols(y~l(x1, 0:1), pdat_small)

# we obtain the same results, had we created the lags "on the fly"
base_small = base_did[!base_did$period %in% c(2, 4), ]
b = feols(y~l(x1, 0:1), base_small, panel.id = ~id+period)
etable(a, b)


# Using data.table to create new lead/lag variables
if(require("data.table")){
```

```
  pdat_dt = panel(as.data.table(base_did), ~id+period)

  # Variable creation
  pdat_dt[, x_l1 := l(x1)]
  pdat_dt[, c("x_l1", "x_f1_2") := .(l(x1), f(x1)**2)]

  # Estimation on a subset of the data
  #  (the lead/lags work appropriately)
  feols(y~l(x1, 0:1), pdat_dt[!period %in% c(2, 4)])
}
```

index   *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.

- To get started, look at the introduction.

Details

The main features are:

- Estimation. 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.

- Easy and flexible clustering of standard-errors. By using the arguments se and dof (see `summary.fixest`). To have a sense of how the standard errors are computed, see the vignette On standard-errors.

- Visualization and exportation of results. You can visualize the results of multiple estimations in R, or export them in Latex using the function `etable`. This vignette details how to customize the Latex tables: Exporting estimation tables.

- Plot multiple results. You can plot the coefficients and confidence intervals of estimations easily with the function `coefplot`. This function also offers a specific layout for interactions.

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`).

Index