Package 'fixest'

March 29, 2021

Type Package

Title Fast Fixed-Effects Estimations

Version 0.8.4

Imports stats, graphics, grDevices, utils, methods, numDeriv, nlme, sandwich, Rcpp, dreamerr(>= 1.2.3)

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

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

URL https://lrberge.github.io/fixest/, https://github.com/lrberge/fixest

SystemRequirements C++11

VignetteBuilder knitr

LazyData true

RoxygenNote 7.1.1

Encoding UTF-8

NeedsCompilation yes

Author Laurent Berge [aut, cre], Sebastian Krantz [ctb]

Maintainer Laurent Berge <laurent.berge@u-bordeaux.fr>

Repository CRAN

Date/Publication 2021-03-29 17:00:02 UTC

R topics documented:

`aggregate.fixest` *Aggregates the values of DiD coefficients a la Sun and Abraham*

Description

Simple tool that aggregates the value of CATT coefficients in staggered difference-in-difference setups (see details).

Usage

```
## S3 method for class 'fixest'
aggregate(x, agg, full = FALSE, use_weights = TRUE, ...)
```

Arguments

`x`	A `fixest` object.
`agg`	A character scalar describing the variable names to be aggregated, it is pattern-based. All variables that match the pattern will be aggregated. It must be of the form `"(root)"`, the parentheses must be there and the resulting variable name will be `"root"`. You can add another root with parentheses: `"(root1)regex(root2)"`, in which case the resulting name is `"root1::root2"`. To name the resulting

	variable differently you can pass a named vector: `c("name" = "pattern")` or `c("name" = "pattern(root2)")`. It's a bit intricate sorry, please see the examples.
`full`	Logical scalar, defaults to `FALSE`. If `TRUE`, then all coefficients are returned, not only the aggregated coefficients.
`use_weights`	Logical, default is `TRUE`. If the estimation was weighted, whether the aggregation should take into account the weights. Basically if the weights reflected frequency it should be `TRUE`.
`...`	Arguments to be passed to `summary.fixest`.

Details

This is a function helping to replicate the estimator from Sun and Abraham (2020). You first need to perform an estimation with cohort and relative periods dummies (typically using the function `i`), this leads to estimators of the cohort average treatment effect on the treated (CATT). Then you can use this function to retrieve the average treatment effect on each relative period, or for any other way you wish to aggregate the CATT.

Note that contrary to the SA article, here the cohort share in the sample is considered to be a perfect measure for the cohort share in the population.

Value

It returns a matrix representing a table of coefficients.

Author(s)

Laurent Berge

References

Liyang Sun and Sarah Abraham, forthcoming, "Estimating Dynamic Treatment Effects in Event Studies with Heterogeneous Treatment Effects". Journal of Econometrics.

Examples

```
#
# DiD example
#

# first we set up the data

set.seed(1)
n_group = 20
n_per_group = 5

id_i = paste0((1:n_group), ":", rep(1:n_per_group, each = n_group))
id_t = 1:10

base = expand.grid(id = id_i, year = id_t)
```

```
base$group = as.numeric(gsub(":.+", "", base$id))

base$year_treated = base$group
base$year_treated[base$group > 10] = 10000
base$treat_post = (base$year >= base$year_treated) * 1
base$time_to_treatment = pmax(base$year - base$year_treated, -1000)
base$treated = (base$year_treated < 10000) * 1

# The effect of the treatment is cohort specific and increases with time
base$y_true = base$treat_post * (1 + 1 * base$time_to_treatment - 1 * base$group)
base$y = base$y_true + rnorm(nrow(base))


# The controls have a time_to_treatment equal to -1000

# we drop the always treated
base = base[base$group > 1,]

# Now we perform the estimation
res_naive = feols(y ~ i(treated, time_to_treatment,
                        ref = -1, drop = -1000) | id + year, base)

res_cohort = feols(y ~ i(time_to_treatment, f2 = group,
                         drop = c(-1, -1000)) | id + year, base)

coefplot(res_naive, ylim = c(-6, 8))
att_true = tapply(base$y_true, base$time_to_treatment, mean)[-1]
points(-9:8 + 0.15, att_true, pch = 15, col = 2)

# The aggregate effect for each period
agg_coef = aggregate(res_cohort, "(ti.*nt)::(-?[[:digit:]]+)")
x = c(-9:-2, 0:8) + .35
points(x, agg_coef[, 1], pch = 17, col = 4)
ci_low = agg_coef[, 1] - 1.96 * agg_coef[, 2]
ci_up = agg_coef[, 1] + 1.96 * agg_coef[, 2]
segments(x0 = x, y0 = ci_low, x1 = x, y1 = ci_up, col = 4)

legend("topleft", col = c(1, 2, 4), pch = c(20, 15, 17),
       legend = c("Naive", "True", "Sun & Abraham"))


# The ATT
aggregate(res_cohort, c("ATT" = "treatment::[^-]"))
mean(base[base$treat_post == 1, "y_true"])

# With etable
etable(res_naive, res_cohort, agg = "(ti.*nt)::(-?[[:digit:]]+):gro")
```

`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)
```

`as.list.fixest_multi` *Transforms a fixest_multi object into a list*

Description

Extracts the results from a `fixest_multi` object and place them into a list.

Usage

```
## S3 method for class 'fixest_multi'
as.list(x, ...)
```

Arguments

`x`	A `fixest_multi` object, obtained from a `fixest` estimation leading to multiple results.
`...`	Not currently used.

Value

Returns a list containing all the results of the multiple estimations.

See Also

The main fixest estimation functions: `feols`, `fepois`, `fenegbin`, `feglm`, `feNmlm`. Tools for mutliple fixest estimations: `summary.fixest_multi`, `print.fixest_multi`, `as.list.fixest_multi`, `sub-sub-.fixest_multi`, `sub-.fixest_multi`, `cash-.fixest_multi`.

Examples

```
base = iris
names(base) = c("y", "x1", "x2", "x3", "species")

# Multiple estimation
res = feols(y ~ csw(x1, x2, x3), base, split = ~species)

# All the results at once
as.list(res)
```

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**.

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 sandwich 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, ...)

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

Arguments

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

Details

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

Note that if the model has been estimated with 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 function. 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, it is equal to getFixest_dict(), a default dictionary which can be set with setFixest_dict. You can use dict = FALSE to disable it..
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 (note that you can use the special character "^^" to clean the beginning of the names, see example), 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\",`
`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 "%" (percentage) to make reference to the original variable name instead of the aliased name. For example, you have a variable named "Month6", and use a dictionary dict = c(Month6="June"). Thus the variable will be displayed as "June". If you want to delete that variable, you can use either drop="June", or drop="%Month6" (which makes reference to its original name).

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)

#
# Example 4: group + cleaning
#

# You can use the argument group to group variables
# You can further use the special character "^^" to clean
#  the beginning of the coef. name: particularly useful for factors

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

# No grouping:
coefplot(est)

# now we group by Sepal and Species
coefplot(est, group = list(Sepal = "Sepal", Species = "Species"))

# now we group + clean the beginning of the names using the special character ^^
coefplot(est, group = list(Sepal = "^^Sepal.", Species = "^^Species"))
```

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

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", "hetero", "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.

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 parameters are used.
level	The confidence level. Default is 0.95.
se	Character scalar. Which kind of standard error should be computed: "standard", "hetero", "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")], 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). You can interact two variables using ^ with the following syntax: cluster = ~var1^var2 or cluster = "var1^var2".
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

User-level access to internal demeaning algorithm of `fixest`.

Usage

```
demean(
  X,
  f,
  slope.vars,
  slope.flag,
  data,
  weights,
  nthreads = getFixest_nthreads(),
  notes = getFixest_notes(),
  iter = 2000,
  tol = 1e-06,
  na.rm = TRUE,
  as.matrix = is.atomic(X),
  im_confident = FALSE
)
```

Arguments

`X`	A matrix, vector, data.frame or a list OR a formula. If equal to a formula, then the argument `data` is required, and it must be of the type: `x1 + x2 ~ f1 + fe2` with on the LHS the variables to be centered, and on the RHS the factors used for centering. Note that you can use variables with varying slopes with the syntax `fe[v1,v2]` (see details in `feols`). If not a formula, it must represent the data to be centered. Of course the dimension of that data must be the same as the factors used for centering (argument `f`).

`f` A matrix, vector, data.frame or list. The factors used to center the variables in argument `X`. Matrices will be coerced using `as.data.frame`.

`slope.vars` A vector, matrix or list representing the variables with varying slopes. Matrices will be coerced using `as.data.frame`. Note that if this argument is used it MUST be in conjunction with the argument `slope.flag` that maps the factors to which the varying slopes are attached. See examples.

`slope.flag` An integer vector of the same length as the number of variables in `f` (the factors used for centering). It indicates for each factor the number of variables with varying slopes to which it is associated. Positive values mean that the raw factor should also be included in the centering, negative values that it should be excluded. Sorry it's complicated... but see the examples it may get clearer.

`data` A data.frame containing all variables in the argument `X`. Only used if `X` is a formula, in which case `data` is mandatory.

`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`.

`na.rm` Logical, default is `TRUE`. If `TRUE` and the input data contains any NA value, then any observation with NA will be discarded leading to an output with less observations than the input. If `FALSE`, if NAs are present the output will also be filled with NAs for each NA observation in input.

`as.matrix` Logical, if `TRUE` a matrix is returned, if `FALSE` it will be a data.frame. The default depends on the input, if atomic then a matrix will be returned.

`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 vector/matrix/list (not a formula!), `f` MUST be a list, `slope.vars` MUST be a list, `slope.vars` MUST be consistent with `slope.flag`, and `weights`, if given, MUST be numeric (not integer!). Further there MUST be not any NA value, and the number of observations of each element MUST be consistent. Non compliance to these rules may simply lead your R session to break.

Value

It returns a data.frame of the same number of columns as the number of variables to be centered.

If `na.rm = TRUE`, then the number of rows is equal to the number of rows in input minus the number of NA values (contained in `X`, `f`, `slope.vars` or `weights`). The default is to have an output of the same number of observations as the input (filled with NAs where appropriate).

A matrix can be returned if `as.matrix = TRUE`.

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]]

In general, for convergence reasons, it is recommended to always add the fixed-effect and avoid using only the variable with varying slope (i.e. use single square brackets).

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")],
                  f = 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")

#
# Variables with varying slopes
#

# You can center on factors but also on variables with varying slopes

# Let's have an illustration
base = iris
names(base) = c("y", "x1", "x2", "x3", "species")

#
# We center y and x1 on species and x2 * species

# using a formula
```

```
base_dm = demean(y + x1 ~ species[x2], data = base)

# using vectors
base_dm_bis = demean(X = base[, c("y", "x1")], f = base$species,
                     slope.vars = base$x2, slope.flag = 1)

# Let's look at the equivalences
res_vs_1 = feols(y ~ x1 + species + x2:species, base)
res_vs_2 = feols(y ~ x1, base_dm)
res_vs_3 = feols(y ~ x1, base_dm_bis)

# only the small sample adj. differ in the SEs
etable(res_vs_1, res_vs_2, res_vs_3, keep = "x1")

#
# center on x2 * species and on another FE

base$fe = rep(1:5, 10)

# using a formula => double square brackets!
base_dm = demean(y + x1 ~ fe + species[[x2]], data = base)

# using vectors => note slope.flag!
base_dm_bis = demean(X = base[, c("y", "x1")], f = base[, c("fe", "species")],
                     slope.vars = base$x2, slope.flag = c(0, -1))

# Explanations slope.flag = c(0, -1):
# - the first 0: the first factor (fe) is associated to no variable
# - the "-1":
#    * |-1| = 1: the second factor (species) is associated to ONE variable
#    *   -1 < 0: the second factor should not be included as such

# Let's look at the equivalences
res_vs_1 = feols(y ~ x1 + i(fe) + x2:species, base)
res_vs_2 = feols(y ~ x1, base_dm)
res_vs_3 = feols(y ~ x1, base_dm_bis)

# only the small sample adj. differ in the SEs
etable(res_vs_1, res_vs_2, res_vs_3, keep = "x1")
```

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

`prepostnames` Only if there is a 'post' variable. The names of the pre and post periods to be displayed in Latex. Default is `c("Before","After")`.

`diff.inv` Logical, default to `FALSE`. Whether to inverse the difference.

Details

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

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

Value

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

Examples

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

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

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

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

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

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

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

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 = "min",
  t.df = "min",
  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" or "min" (default). 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 a unique adjustment is made, of the form G_min/(G_min-1) with G_min the smallest G_i. If cluster.df="conventional" then the i-th "sandwich" matrix is adjusted with G_i/(G_i-1) with G_i the number of unique clusters.

`t.df` Either "conventional" or "min" (default). 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 minimum size of the clusters with which the VCOV has been clustered. If `t.df="conventional"`, then the degrees of freedom of the Student t distribution is equal to the number of observations minus the number of estimated variables.

`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, attr = TRUE))[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"), attr = TRUE), "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"), attr = TRUE), "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), attr = TRUE), "dof.K")

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

#
# To permanently set the default dof:
#

# eg no small sample adjustment:
setFixest_dof(dof(adj = FALSE))

# Factory default
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 = NULL,
  dof = NULL,
  cluster = NULL,
  stage = 2,
  agg = NULL,
  .vcov,
  .vcov_args = NULL,
  digits = 4,
  digits.stats = 5,
  tex,
  fitstat,
  title,
  coefstat = "se",
  ci = 0.95,
  sdBelow = TRUE,
  keep,
  drop,
  order,
  dict,
  file,
  replace = FALSE,
  convergence,
  signifCode,
  label,
  float,
  subtitles = list("auto"),
  fixef_sizes = FALSE,
  fixef_sizes.simplify = TRUE,
  keepFactors = TRUE,
  family,
  powerBelow = -5,
  interaction.combine = " $\\times $ ",
  depvar = TRUE,
  style.tex = NULL,
```

```
  style.df = NULL,
  notes = NULL,
  group = NULL,
  extraline = NULL,
  placement = "htbp",
  drop.section = NULL,
  poly_dict = c("", " square", " cube"),
  postprocess.tex = NULL,
  postprocess.df = NULL,
  fit_format = "__var__"
)

setFixest_etable(
  digits = 4,
  digits.stats = 5,
  fitstat,
  coefstat = c("se", "tstat", "confint"),
  ci = 0.95,
  sdBelow = TRUE,
  keep,
  drop,
  order,
  dict,
  signifCode,
  float,
  fixef_sizes = FALSE,
  fixef_sizes.simplify = TRUE,
  family,
  powerBelow = -5,
  interaction.combine = " $\\times $ ",
  depvar,
  style.tex = NULL,
  style.df = NULL,
  notes = NULL,
  group = NULL,
  extraline = NULL,
  placement = "htbp",
  drop.section = NULL,
  postprocess.tex = NULL,
  postprocess.df = NULL,
  fit_format = "__var__",
  reset = FALSE
)

getFixest_etable()

esttable(
  ...,
```

```
  se = NULL,
  dof = NULL,
  cluster = NULL,
  stage = 2,
  agg = NULL,
  .vcov,
  .vcov_args = NULL,
  digits = 4,
  digits.stats = 5,
  fitstat,
  coefstat = "se",
  ci = 0.95,
  sdBelow = TRUE,
  keep,
  drop,
  order,
  dict,
  file,
  replace = FALSE,
  convergence,
  signifCode,
  subtitles = list("auto"),
  fixef_sizes = FALSE,
  fixef_sizes.simplify = TRUE,
  keepFactors = TRUE,
  family,
  powerBelow = -5,
  interaction.combine = " $\\times $ ",
  depvar = TRUE,
  style.df = NULL,
  group = NULL,
  extraline = NULL,
  drop.section = NULL,
  poly_dict = c("", " square", " cube"),
  postprocess.df = NULL,
  fit_format = "__var__"
)

esttex(
  ...,
  se = NULL,
  dof = NULL,
  cluster = NULL,
  stage = 2,
  agg = NULL,
  .vcov,
  .vcov_args = NULL,
  digits = 4,
```

```
  digits.stats = 5,
  fitstat,
  title,
  coefstat = "se",
  ci = 0.95,
  sdBelow = TRUE,
  keep,
  drop,
  order,
  dict,
  file,
  replace = FALSE,
  convergence,
  signifCode,
  label,
  float,
  subtitles = list("auto"),
  fixef_sizes = FALSE,
  fixef_sizes.simplify = TRUE,
  keepFactors = TRUE,
  family,
  powerBelow = -5,
  interaction.combine = " $\\times $ ",
  depvar = TRUE,
  style.tex = NULL,
  notes = NULL,
  group = NULL,
  extraline = NULL,
  placement = "htbp",
  drop.section = NULL,
  poly_dict = c("", " square", " cube"),
  postprocess.tex = NULL,
  fit_format = "__var__"
)
```

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", "hetero", "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")], 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). You can interact two variables using ^ with the following syntax: cluster = ~var1^var2 or cluster = "var1^var2".

stage — Can be equal to 2 (default), 1, 1:2 or 2:1. Only used if the object is an IV estimation: defines the stage to which summary should be applied. If stage = 1 and there are multiple endogenous regressors or if stage is of length 2, then an object of class fixest_multi is returned.

agg — A character scalar describing the variable names to be aggregated, it is pattern-based. All variables that match the pattern will be aggregated. It must be of the form "(root)", the parentheses must be there and the resulting variable name will be "root". You can add another root with parentheses: "(root1)regex(root2)", in which case the resulting name is "root1::root2". To name the resulting variable differently you can pass a named vector: c("name" = "pattern") or c("name" = "pattern(root2)"). It's a bit intricate sorry, please see the examples.

.vcov — A function to be used to compute the standard-errors of each fixest object. You can pass extra arguments to this function using the argument .vcov_args. See the example.

.vcov_args — A list containing arguments to be passed to the function .vcov.

digits — Integer or character scalar. Default is 4 and represents the number of significant digits to be displayed for the coefficients and standard-errors. To apply rounding instead of significance use, e.g., digits = "r3" which will round at the first 3 decimals. If character, it must be of the form "rd" or "sd" with d a digit (r is for round and s is for significance). For the number of digits for the fit statistics, use digits.stats. Note that when significance is used it does not exactly display the number of significant digits: see details for its exact meaning.

digits.stats — Integer or character scalar. Default is 5 and represents the number of significant digits to be displayed for the fit statistics. To apply rounding instead of significance use, e.g., digits = "r3" which will round at the first 3 decimals. If character, it must be of the form "rd" or "sd" with d a digit (r is for round and s is for significance). Note that when significance is used it does not exactly display the number of significant digits: see details for its exact meaning.

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 (both with only lowercase letters). A vector listing which fit statistics to display. The valid types are 'n', 'll', 'aic',

	'bic' and r2 types like 'r2', 'pr2', 'war2', etc (see all valid types in r2). Also accepts valid types from the function fitstat. The default value depends on the models to display. Example of use: fitstat=c('n','cor2','ar2','war2'), or fitstat=~n+cor2+ar2+war2 using a formula. You can use the dot to refer to default values: ~ . + ll would add the log-likelihood to the default fit statistics.
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, it is equal to getFixest_dict(), a default dictionary which can be set with setFixest_dict. You can use dict = FALSE to disable it..
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 suppress 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 or list. The elements should be of length 1 or of the same length as the number of models. If a list, the names of the list will be displayed on the leftmost column. By default it is equal to `list("auto")` which means that if the object is a split sample estimation, the sample will be automatically added as a sub-title.

`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).

`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` Logical, default is `TRUE`. Whether a first line containing the dependent variables should be shown.

`style.tex` An object created by the function `style.tex`. It represents the style of the Latex table, see the documentation of `style.tex`.

`style.df` An object created by the function `style.df`. It represents the style of the data frame returned (if `tex = FALSE`), see the documentation of `style.df`.

`notes` (Tex only.) 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 discared, 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`. Starting the list name with an underscore is equivalent to adding `"{where:stats}"`: eg `list("_Controls"="x")` is equivalent to `list("{where:stats}Controls"="x")`.

`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 `extraline=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`. Starting the list name with an underscore is equivalent to adding `"{where:stats}"`: eg `list("_Controls"=TRUE)` is equivalent to `list("{where:stats}Controls"=TRUE)`.

`placement` (Tex only.) Character string giving the position of the float in Latex. Default is "htbp". It must consist of only the characters 'h', 't', 'b', 'p', 'H' and '!'. Reminder: h: here; t: top; b: bottom; p: float page; H: definitely here; !: prevents Latex to look for other positions. Note that it can be equal to the empty string (and you'll get the default placement).

`drop.section` Character vector which can be of length 0 (i.e. equal to `NULL`). Can contain the values "fixef", "slopes" or "stats". It would drop, respectively, the fixed-effects section, the variables with varying slopes section or the fit statistics section.

`poly_dict` Character vector, default is `c("","square","cube")`. When raw polynomials (`x^2`, etc) are used, the variables are automatically renamed and `poly_dict` rules the display of the power. For powers greater than the number of elements of the vector, the value displayed is `$^{pow}$` in Latex and `^pow` in the R console.

`postprocess.tex`
: A function that will postprocess the character vector defining the latex table. Only when `tex = TRUE`. By default it is equal to `NULL`, meaning that there is no postprocessing. When `tex = FALSE`, see the argument `postprocess.df`. See details.

`postprocess.df`
: A function that will postprocess.tex the resulting data.frame. Only when `tex = FALSE`. By default it is equal to `NULL`, meaning that there is no postprocessing. When `tex = TRUE`, see the argument `postprocess.tex`.

`fit_format`
: Character scalar, default is `"__var__"`. Only used in the presence of IVs. By default the endogenous regressors are named `fit_varname` in the second stage. The format of the endogenous regressor to appear in the table is governed by `fit_format`. For instance, by default, the prefix `"fit_"` is removed, leading to only `varname` to appear. If `fit_format = "$\\hat{__var__}$"`, then `"$\hat{varname}$"` will appear in the table.

`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`. This function is deprecated.

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

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

When the argument `postprocessing.tex` is not missing, two additional tags will be included in the character vector returned by `etable`: `"%start:tab\n"` and `"%end:tab\n"`. These can be used to identify the start and end of the tabular and are useful to insert code within the `table` environment.

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 printed and returned invisibly.

Functions

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

How does `digits` handle the number of decimals displayed?

The default display of decimals is the outcome of an algorithm. Let's take the example of `digits = 3` which "kind of" requires 3 significant digits to be displayed.

For numbers greater than 1 (in absolute terms), their integral part is always be displayed and the number of decimals shown is equal to `digits` minus the number of digits in the integral part. This means that `12.345` will be displayed as `12.3`. If the number of decimals should be 0, then a single decimal is displayed to suggest that the number is not whole. This means that `1234.56` will be displayed as `1234.5`. Note that if the number is whole, no decimals are shown.

For numbers lower than 1 (in absolute terms), the number of decimals displayed is equal to `digits` except if there are only 0s in which case the first significant digit is shown. This means that `0.01234` will be displayed as `0.012` (first rule), and that 0.000123 will be displayed as `0.0001` (second rule).

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 "%" (percentage) to make reference to the original variable name instead of the aliased name. For example, you have a variable named `"Month6"`, and use a dictionary `dict = c(Month6="June")`. Thus the variable will be displayed as `"June"`. If you want to delete that variable, you can use either `drop="June"`, or `drop="%Month6"` (which makes reference to its original name).

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

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

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

# keep/drop: keeping only interactions
etable(est1, est2, keep = " x ")
# or using drop  (see regexp help):
etable(est1, est2, drop = "^(Month|Temp|\\()")

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

# order: Wind variable first, intercept last (note the "!" to reverse the effect)
etable(est1, est2, order = c("Wind", "!Inter"))
# Month, then interactions, then the rest
etable(est1, est2, order = c("^Month", " x "))

#
# dict
#

# You can rename variables with dict = c(var1 = alias1, var2 = alias2, etc)
# You can also rename values taken by factors.
# Here's a full example:
dict = c(Temp = "Temperature", "Month::5"="May", "6"="Jun")
etable(est1, est2, dict = dict)
# Note the difference of treatment between Jun and May

# Assume the following dictionnary:
dict = c("Month::5"="May", "Month::6"="Jun", "Month::7"="Jul",
         "Month::8"="Aug", "Month::9"="Sep")

# We would like to keep only the Months, but now the names are all changed...
# How to do?
# We can use the special character '%' to make reference to the original names.

etable(est1, est2, dict = dict, keep = "%Month")

#
# signifCode
#

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

```
#
# 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:
# Note that a space adds an extra line
style_noHeaders = style.tex(var.title = "", fixef.title = "", stats.title = " ")
etable(est1, est2, dict = dict, tex = TRUE, style.tex = style_noHeaders)

# To change the lines of the table + dropping the table footer
style_lines = style.tex(line.top = "\\toprule", line.bottom = "\\bottomrule",
                    tablefoot = FALSE)
etable(est1, est2, dict = dict, tex = TRUE, style.tex = style_lines)

# Or you have the predefined type "aer"
etable(est1, est2, dict = dict, tex = TRUE, style.tex = style.tex("aer"))

#
# 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 + 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
# Note that the "%" is needed because polnomials are automatically renamed
#  so we need to make reference to the original name.

# '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"}
```

```
# (the enclosing curly braces are only here to make Rd work, please ignore them)
# You can use the shortcut "_Group name" which is equivalent to {"{where:stats}Group name"}

# 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.")))


#
# Using custom functions to compute the standard errors
#

# You can customize the way you compute the SEs with the argument .vcov
# Let's use some covariances from the sandwich package

etable(est_c0, est_c1, est_c2, .vcov = sandwich::vcovHC)

# To add extra arguments to vcovHC, you need to use .vcov_args
etable(est_c0, est_c1, est_c2, .vcov = sandwich::vcovHC, .vcov_args = list(type = "HC0"))


#
# Customize which fit statistic to display
#

# You can change the fit statistics with the argument fitstat
# and you can rename them with the dictionnary
etable(est1, est2, fitstat = ~ r2 + n + G)

# If you use a formula, '.' means the default:
etable(est1, est2, fitstat = ~ ll + .)


#
# Computing a different SE for each model
#

est = feols(Ozone ~ Solar.R + Wind + Temp, data = aq)

#
# Method 1: use summary

s1 = summary(est, "standard")
s2 = summary(est, cluster = ~ Month)
s3 = summary(est, cluster = ~ Day)
s4 = summary(est, cluster = ~ Day + Month)

etable(list(s1, s2, s3, s4))

#
# Method 2: using a list in the argument 'cluster'
```

```
est_bis = feols(Ozone ~ Solar.R + Wind + Temp | Month, data = aq)
etable(list(est, est_bis), cluster = list("standard", ~ Month))

#
# Method 3: Using rep()

etable(rep(est, cluster = list("standard", ~ Month)))

# When using rep on 2 or more objects, you need to embed them in .l()
etable(rep(.l(est, est_bis), cluster = list("standard", ~ Month, ~ Day)))

# Using each to order differently
etable(rep(.l(est, est_bis), each = 3, cluster = list("standard", ~ Month, ~ Day)))
```

---

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)

d(x, lag = 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
- `d`: Creates differences (i.e. x - lag(x)) 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)
feols(d(y) ~ d(x1), 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[, x1_d1 := d(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,
  subset,
  split,
  fsplit,
  cluster,
  se,
  dof,
  panel.id,
  start = NULL,
  etastart = NULL,
  mustart = NULL,
  fixef,
  fixef.rm = "perfect",
  fixef.tol = 1e-06,
  fixef.iter = 10000,
  collin.tol = 1e-10,
  glm.iter = 25,
  glm.tol = 1e-08,
  nthreads = getFixest_nthreads(),
  lean = FALSE,
  warn = TRUE,
  notes = getFixest_notes(),
  verbose = 0,
  combine.quick,
  mem.clean = FALSE,
  only.env = FALSE,
  env,
  ...
)

feglm.fit(
  y,
  X,
```

```
  fixef_mat,
  family = "poisson",
  offset,
  split,
  fsplit,
  cluster,
  se,
  dof,
  weights,
  subset,
  start = NULL,
  etastart = NULL,
  mustart = NULL,
  fixef.rm = "perfect",
  fixef.tol = 1e-06,
  fixef.iter = 10000,
  collin.tol = 1e-10,
  glm.iter = 25,
  glm.tol = 1e-08,
  nthreads = getFixest_nthreads(),
  lean = FALSE,
  warn = TRUE,
  notes = getFixest_notes(),
  mem.clean = FALSE,
  verbose = 0,
  only.env = FALSE,
  env,
  ...
)

fepois(
  fml,
  data,
  offset,
  weights,
  subset,
  split,
  fsplit,
  cluster,
  se,
  dof,
  panel.id,
  start = NULL,
  etastart = NULL,
  mustart = NULL,
  fixef,
  fixef.rm = "perfect",
  fixef.tol = 1e-06,
```

```
  fixef.iter = 10000,
  collin.tol = 1e-10,
  glm.iter = 25,
  glm.tol = 1e-08,
  nthreads = getFixest_nthreads(),
  lean = FALSE,
  warn = TRUE,
  notes = getFixest_notes(),
  verbose = 0,
  combine.quick,
  mem.clean = FALSE,
  only.env = FALSE,
  env,
  ...
)
```

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`, see details. To add IVs, insert the endogenous vars./instruments after a pipe, like in `y ~ x | c(x_endo1,x_endo2) ~ x_inst1 + x_inst2`. Note that it should always be the last element, see details. Multiple estimations can be performed at once: for multiple dep. vars, wrap them in `c()`: ex `c(y1,y2)`. For multiple indep. vars, use the stepwise functions: ex `x1 + csw(x2,x3)`. The formula `fml = c(y1,y2) ~ x1 + cw0(x2,x3)` leads to 6 estimation, see details.
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 one-sided: for example `~ var_weight`.
subset	A vector (logical or numeric) or a one-sided formula. If provided, then the estimation will be performed only on the observations defined by this argument.
split	A one sided formula representing a variable (eg `split = ~var`) or a vector. If provided, the sample is split according to the variable and one estimation is performed for each value of that variable. If you also want to include the estimation for the full sample, use the argument `fsplit` instead.

`fsplit` A one sided formula representing a variable (eg `split = ~var`) or a vector. If provided, the sample is split according to the variable and one estimation is performed for each value of that variable. This argument is the same as split but also includes the full sample as the first estimation.

`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")]`, `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). You can interact two variables using `^` with the following syntax: `cluster = ~var1^var2` or `cluster = "var1^var2"`.

`se` Character scalar. Which kind of standard error should be computed: "standard", "hetero", "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.

`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.rm` Can be equal to "perfect" (default), "singleton", "both" or "none". Controls which observations are to be removed. If "perfect", then observations having a fixed-effect with perfect fit (e.g. only 0 outcomes in Poisson estimations) will be removed. If "singleton", all observations for which a fixed-effect appears only once will be removed. The meaning of "both" and "none" is direct.

`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.

`collin.tol` Numeric scalar, default is `1e-10`. Threshold deciding when variables should be considered collinear and subsequently removed from the estimation. Higher values means more variables will be removed (if there is presence of collinearity). One signal of presence of collinearity is t-stats that are extremely low (for instance when t-stats < 1e-3).

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

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

`nthreads` The number of threads. Can be: a) an integer lower than, or equal to, the maximum number of threads; b) 0: meaning all available threads will be used; c) a number strictly between 0 and 1 which represents the fraction of all threads to use. The default is to use 50% of all threads. You can set permanently the number of threads used within this package using the function `setFixest_nthreads`.

`lean` Logical, default is `FALSE`. If `TRUE` then all large objects are removed from the returned result: this will save memory but will block the possibility to use many methods. It is recommended to use the arguments `se` or `cluster` to obtain the appropriate standard-errors at estimation time, since obtaining different SEs won't be possible afterwards.

`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.

`mem.clean` Logical, default is `FALSE`. Only to be used if the data set is large compared to the available RAM. If `TRUE` then intermediary objects are removed as much as possible and `gc` is run before each substantial C++ section in the internal code to avoid memory issues.

only.env	(Advanced users.) Logical, default is FALSE. If TRUE, then only the environment used to make the estimation is returned.
env	(Advanced users.) A fixest environment created by a fixest estimation with only.env = TRUE. Default is missing. If provided, the data from this environment will be used to perform the estimation.
...	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. Note that fixest objects contain many elements and most of them are for internal use, they are presented here only for information. To access them, it is safer to use the user-level methods (e.g. vcov.fixest, resid.fixest, etc) or functions (like for instance fitstat to access any fit statistic).

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_all	A list containing different parts of the formula. Always contain the linear formula. Then, if relevant: fixef: 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	(When 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	(When relevant.) Vector of observations that were removed because of NA values or because of only 0/1 outcome within a fixed-effect (depends on the family though).

`fixef_removed` (When 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` (When relevant.) The offset formula.

`weights` (When relevant.) The weights formula.

`collin.var` (When relevant.) Vector containing the variables removed because of collinearity.

`collin.coef` (When 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]]

In general, for convergence reasons, it is recommended to always add the fixed-effect and avoid using only the variable with varying slope (i.e. use single square brackets).

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`, `f` and `d`.

You can provide several leads/lags/differences 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)`.

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.

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”).

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.

Multiple estimations

Multiple estimations can be performed at once, they just have to be specified in the formula. Multiple estimations yield a `fixest_multi` object which is 'kind of' a list of all the results but includes specific methods to access the results in a handy way.

To include mutliple dependent variables, wrap them in `c()` (`list()` also works). For instance `fml = c(y1,y2) ~ x1` would estimate the model `fml = y1 ~ x1` and then the model `fml = y2 ~ x1`.

To include multiple independent variables, you need to use the stepwise functions. There are 4 stepwise functions associated to 4 short aliases. These are a) stepwise, stepwise0, cstepwise, cstepwise0, and b) sw, sw0, csw, csw0. Let's explain that. Assume you have the following formula: `fml = y ~ x1 + sw(x2,x3)`. The stepwise function `sw` will estimate the following two models: `y ~ x1 + x2` and `y ~ x1 + x3`. That is, each element in `sw()` is sequentially, and separately, added to the formula. Would have you used `sw0` in lieu of `sw`, then the model `y ~ x1` would also have been estimated. The `0` in the name means that the model wihtout any stepwise element also needs to be estimated. Finally, the prefix `c` means cumulative: each stepwise element is added to the next. That is, `fml = y ~ x1 + csw(x2,x3)` would lead to the following models `y ~ x1 + x2` and `y ~ x1 + x2 + x3`. The `0` has the same meaning and would also lead to the model without the stepwise elements to be estimated: in other words, `fml = y ~ x1 + csw0(x2,x3)` leads to the following three models: `y ~ x1`, `y ~ x1 + x2` and `y ~ x1 + x2 + x3`.

Multiple independent variables can be combined with multiple dependent variables, as in `fml = c(y1,y2) ~ cw(x1,x2,x3)` which would lead to 6 estimations. Multiple estimations can also be combined to split samples (with the arguments `split`, `fsplit`).

Fixed-effects cannot be included in a stepwise fashion: they are there or not and stay the same for all estimations.

A note on performance. The feature of multiple estimations has been highly optimized for `feols`, in particular in the presence of fixed-effects. It is faster to estimate multiple models using the formula rather than with a loop. For non-`feols` models using the formula is roughly similar to using a loop performance-wise.

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)

# Note that you have more examples in feols

#
# Multiple estimations:
#

# 6 estimations
est_mult = fepois(c(Ozone, Solar.R) ~ Wind + Temp + csw0(Wind:Temp, Day), airquality)

# We can display the results for the first lhs:
etable(est_mult[lhs = 1])

# And now the second (access can be made by name)
etable(est_mult[lhs = "Solar.R"])

# Now we focus on the two last right hand sides
# (note that .N can be used to specify the last item)
etable(est_mult[rhs = 2:.N])

# Combining with split
est_split = fepois(c(Ozone, Solar.R) ~ sw(poly(Wind, 2), poly(Temp, 2)),
                   airquality, split = ~ Month)

# You can display everything at once with the print method
est_split

# Different way of displaying the results with "compact"
summary(est_split, "compact")

# You can still select which sample/LHS/RHS to display
est_split[sample = 1:2, lhs = 1, rhs = 1]
```

`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,
  fixef.rm = "perfect",
  offset,
  subset,
  split,
  fsplit,
  cluster,
  se,
  dof,
  panel.id,
  fixef.tol = 1e-05,
  fixef.iter = 10000,
  nthreads = getFixest_nthreads(),
  lean = FALSE,
  verbose = 0,
  warn = TRUE,
  notes = getFixest_notes(),
  theta.init,
  combine.quick,
  mem.clean = FALSE,
  only.env = FALSE,
  env,
  ...
)

fenegbin(
  fml,
  data,
  theta.init,
  start = 0,
  fixef,
  fixef.rm = "perfect",
  offset,
  subset,
  split,
  fsplit,
  cluster,
  se,
```

```
  dof,
  panel.id,
  fixef.tol = 1e-05,
  fixef.iter = 10000,
  nthreads = getFixest_nthreads(),
  lean = FALSE,
  verbose = 0,
  warn = TRUE,
  notes = getFixest_notes(),
  combine.quick,
  mem.clean = FALSE,
  only.env = FALSE,
  env,
  ...
)
```

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|fixef_1+fixef_2`. Multiple estimations can be performed at once: for multiple dep. vars, wrap them in `c()`: ex `c(y1,y2)`. For multiple indep. vars, use the stepwise functions: ex `x1 + csw(x2,x3)`. The formula `fml = c(y1,y2) ~ x1 + cw0(x2,x3)` leads to 6 estimation, see details.
data	A data.frame containing the necessary variables to run the model. The variables of the non-linear right hand side of the formula are identified with this `data.frame` names. Can also be a matrix.
family	Character scalar. It should provide the family. The possible values are "poisson" (Poisson model with log-link, the default), "negbin" (Negative Binomial model with log-link), "logit" (LOGIT model with log-link), "gaussian" (Gaussian model).
start	Starting values for the coefficients. Can be: i) a numeric of length 1 (e.g. `start = 0`, the default), ii) a numeric vector of the exact same length as the number of variables, or iii) a named vector of any length (the names will be used to initialize the appropriate coefficients).
fixef	Character vector. The 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.rm	Can be equal to "perfect" (default), "singleton", "both" or "none". Controls which observations are to be removed. If "perfect", then observations having a fixed-effect with perfect fit (e.g. only 0 outcomes in Poisson estimations) will be removed. If "singleton", all observations for which a fixed-effect appears only once will be removed. The meaning of "both" and "none" is direct.
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'.

`subset` A vector (logical or numeric) or a one-sided formula. If provided, then the estimation will be performed only on the observations defined by this argument.

`split` A one sided formula representing a variable (eg `split = ~var`) or a vector. If provided, the sample is split according to the variable and one estimation is performed for each value of that variable. If you also want to include the estimation for the full sample, use the argument `fsplit` instead.

`fsplit` A one sided formula representing a variable (eg `split = ~var`) or a vector. If provided, the sample is split according to the variable and one estimation is performed for each value of that variable. This argument is the same as split but also includes the full sample as the first estimation.

`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")]`, `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). You can interact two variables using `^` with the following syntax: `cluster = ~var1^var2` or `cluster = "var1^var2"`.

`se` Character scalar. Which kind of standard error should be computed: "standard", "hetero", "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.

`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.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.

`nthreads` The number of threads. Can be: a) an integer lower than, or equal to, the maximum number of threads; b) 0: meaning all available threads will be used; c) a number strictly between 0 and 1 which represents the fraction of all threads to use. The default is to use 50% of all threads. You can set permanently the number of threads used within this package using the function `setFixest_nthreads`.

`lean` Logical, default is `FALSE`. If `TRUE` then all large objects are removed from the returned result: this will save memory but will block the possibility to use many methods. It is recommended to use the arguments `se` or `cluster` to obtain the appropriate standard-errors at estimation time, since obtaining different SEs won't be possible afterwards.

`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.

`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)`.

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

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

`mem.clean` Logical, default is `FALSE`. Only to be used if the data set is large compared to the available RAM. If `TRUE` then intermediary objects are removed as much as possible and `gc` is run before each substantial C++ section in the internal code to avoid memory issues.

`only.env` (Advanced users.) Logical, default is `FALSE`. If `TRUE`, then only the environment used to make the estimation is returned.

`env` (Advanced users.) A `fixest` environment created by a `fixest` estimation with `only.env = TRUE`. Default is missing. If provided, the data from this environment will be used to perform the estimation.

`...` 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. Note that `fixest` objects contain many elements and most of them are for internal use, they are presented here only for information. To access them, it is safer to use the user-level methods (e.g. `vcov.fixest`, `resid.fixest`, etc) or functions (like for instance `fitstat` to access any fit statistic).

`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_all`	A list containing different parts of the formula. Always contain the linear formula. Then, if relevant: `fixef`: the fixed-effects; `NL`: the non linear part of the formula.
`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`	(When 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`	(When 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	(When relevant.) The offset formula.
weights	(When 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`, `f` and `d`.

You can provide several leads/lags/differences 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)`.

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.

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").

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.

Multiple estimations

Multiple estimations can be performed at once, they just have to be specified in the formula. Multiple estimations yield a fixest_multi object which is 'kind of' a list of all the results but includes specific methods to access the results in a handy way.

To include mutliple dependent variables, wrap them in c() (list() also works). For instance fml = c(y1,y2) ~ x1 would estimate the model fml = y1 ~ x1 and then the model fml = y2 ~ x1.

To include multiple independent variables, you need to use the stepwise functions. There are 4 stepwise functions associated to 4 short aliases. These are a) stepwise, stepwise0, cstepwise, cstepwise0, and b) sw, sw0, csw, csw0. Let's explain that. Assume you have the following formula: fml = y ~ x1 + sw(x2,x3). The stepwise function sw will estimate the following two models: y ~ x1 + x2 and y ~ x1 + x3. That is, each element in sw() is sequentially, and separately, added to the formula. Would have you used sw0 in lieu of sw, then the model y ~ x1 would also have been estimated. The 0 in the name means that the model wihtout any stepwise element also needs to be estimated. Finally, the prefix c means cumulative: each stepwise element is added to the next. That is, fml = y ~ x1 + csw(x2,x3) would lead to the following models y ~ x1 + x2 and y ~ x1 + x2 + x3. The 0 has the same meaning and would also lead to the model without the stepwise elements to be estimated: in other words, fml = y ~ x1 + csw0(x2,x3) leads to the following three models: y ~ x1, y ~ x1 + x2 and y ~ x1 + x2 + x3.

Multiple independent variables can be combined with multiple dependent variables, as in fml = c(y1,y2) ~ cw(x1,x2,x3) which would lead to 6 estimations. Multiple estimations can also be combined to split samples (with the arguments split, fsplit).

Fixed-effects cannot be included in a stepwise fashion: they are there or not and stay the same for all estimations.

A note on performance. The feature of multiple estimations has been highly optimized for feols, in particular in the presence of fixed-effects. It is faster to estimate multiple models using the formula rather than with a loop. For non-feols models using the formula is roughly similar to using a loop performance-wise.

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

```
# 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 etable
etable(est_pois, est_gaus)
# Now using two way clustered standard-errors
etable(est_pois, est_gaus, se = "twoway")

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

etable(sum_hetero, sum_oneway, sum_twoway, sum_threeway)


#
# Multiple estimations:
#

# 6 estimations
est_mult = femlm(c(Ozone, Solar.R) ~ Wind + Temp + csw0(Wind:Temp, Day), airquality)
```

```
# We can display the results for the first lhs:
etable(est_mult[lhs = 1])

# And now the second (access can be made by name)
etable(est_mult[lhs = "Solar.R"])

# Now we focus on the two last right hand sides
# (note that .N can be used to specify the last item)
etable(est_mult[rhs = 2:.N])

# Combining with split
est_split = fepois(c(Ozone, Solar.R) ~ sw(poly(Wind, 2), poly(Temp, 2)),
                   airquality, split = ~ Month)

# You can display everything at once with the print method
est_split

# Different way of displaying the results with "compact"
summary(est_split, "compact")

# You can still select which sample/LHS/RHS to display
est_split[sample = 1:2, lhs = 1, rhs = 1]
```

---

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 (their design is simpler).

Usage

```
feNmlm(
  fml,
  data,
  family = c("poisson", "negbin", "logit", "gaussian"),
  NL.fml,
  fixef,
  fixef.rm = "perfect",
  NL.start,
  lower,
  upper,
```

```
  NL.start.init,
  offset,
  subset,
  split,
  fsplit,
  cluster,
  se,
  dof,
  panel.id,
  start = 0,
  jacobian.method = "simple",
  useHessian = TRUE,
  hessian.args = NULL,
  opt.control = list(),
  nthreads = getFixest_nthreads(),
  lean = FALSE,
  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,
  mem.clean = FALSE,
  only.env = FALSE,
  env,
  ...
)
```

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, insert them in this formula using a pipe (e.g. `fml = z~x+y|fixef_1+fixef_2`). To include a non-linear in parameters element, you must use the argment `NL.fml`. Multiple estimations can be performed at once: for multiple dep. vars, wrap them in `c()`: ex `c(y1,y2)`. For multiple indep. vars, use the stepwise functions: ex `x1 + csw(x2,x3)`. This leads to 6 estimation `fml = c(y1,y2) ~ x1 + cw0(x2,x3)`. See details.
`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.

`fixef.rm` Can be equal to "perfect" (default), "singleton", "both" or "none". Controls which observations are to be removed. If "perfect", then observations having a fixed-effect with perfect fit (e.g. only 0 outcomes in Poisson estimations) will be removed. If "singleton", all observations for which a fixed-effect appears only once will be removed. The meaning of "both" and "none" is direct.

`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 a numeric scalar.

`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'.

`subset` A vector (logical or numeric) or a one-sided formula. If provided, then the estimation will be performed only on the observations defined by this argument.

`split` A one sided formula representing a variable (eg `split = ~var`) or a vector. If provided, the sample is split according to the variable and one estimation is performed for each value of that variable. If you also want to include the estimation for the full sample, use the argument `fsplit` instead.

`fsplit` A one sided formula representing a variable (eg `split = ~var`) or a vector. If provided, the sample is split according to the variable and one estimation is

performed for each value of that variable. This argument is the same as split but also includes the full sample as the first estimation.

`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")]`, `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). You can interact two variables using `^` with the following syntax: `cluster = ~var1^var2` or `cluster = "var1^var2"`.

`se` Character scalar. Which kind of standard error should be computed: "standard", "hetero", "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.

`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` The number of threads. Can be: a) an integer lower than, or equal to, the maximum number of threads; b) 0: meaning all available threads will be used; c) a number strictly between 0 and 1 which represents the fraction of all threads to

use. The default is to use 50% of all threads. You can set permanently the number of threads used within this package using the function `setFixest_nthreads`.

`lean` Logical, default is `FALSE`. If `TRUE` then all large objects are removed from the returned result: this will save memory but will block the possibility to use many methods. It is recommended to use the arguments `se` or `cluster` to obtain the appropriate standard-errors at estimation time, since obtaining different SEs won't be possible afterwards.

`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.

`mem.clean` Logical, default is `FALSE`. Only to be used if the data set is large compared to the available RAM. If `TRUE` then intermediary objects are removed as much as possible and `gc` is run before each substantial C++ section in the internal code to avoid memory issues.

only.env	(Advanced users.) Logical, default is FALSE. If TRUE, then only the environment used to make the estimation is returned.
env	(Advanced users.) A fixest environment created by a fixest estimation with only.env = TRUE. Default is missing. If provided, the data from this environment will be used to perform the estimation.
...	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. Note that `fixest` objects contain many elements and most of them are for internal use, they are presented here only for information. To access them, it is safer to use the user-level methods (e.g. `vcov.fixest`, `resid.fixest`, etc) or functions (like for instance `fitstat` to access any fit statistic).

`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.
`fml_all`	A list containing different parts of the formula. Always contain the linear formula. Then, if relevant: `fixef`: the fixed-effects; `NL`: the non linear part of the formula.
`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`, `f` and `d`.

You can provide several leads/lags/differences 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)`.

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.

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”).

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.

Multiple estimations

Multiple estimations can be performed at once, they just have to be specified in the formula. Multiple estimations yield a `fixest_multi` object which is 'kind of' a list of all the results but includes specific methods to access the results in a handy way.

To include mutliple dependent variables, wrap them in `c()` (`list()` also works). For instance `fml = c(y1,y2) ~ x1` would estimate the model `fml = y1 ~ x1` and then the model `fml = y2 ~ x1`.

To include multiple independent variables, you need to use the stepwise functions. There are 4 stepwise functions associated to 4 short aliases. These are a) stepwise, stepwise0, cstepwise, cstepwise0, and b) sw, sw0, csw, csw0. Let's explain that. Assume you have the following formula: `fml = y ~ x1 + sw(x2,x3)`. The stepwise function `sw` will estimate the following two models: `y ~ x1 + x2` and `y ~ x1 + x3`. That is, each element in `sw()` is sequentially, and separately, added to the formula. Would have you used `sw0` in lieu of `sw`, then the model `y ~ x1` would also have been estimated. The `0` in the name means that the model wihtout any stepwise element also needs to be estimated. Finally, the prefix `c` means cumulative: each stepwise element is added to the next. That is, `fml = y ~ x1 + csw(x2,x3)` would lead to the following models `y ~ x1 + x2` and `y ~ x1 + x2 + x3`. The `0` has the same meaning and would also lead to the model without the stepwise elements to be estimated: in other words, `fml = y ~ x1 + csw0(x2,x3)` leads to the following three models: `y ~ x1`, `y ~ x1 + x2` and `y ~ x1 + x2 + x3`.

Multiple independent variables can be combined with multiple dependent variables, as in `fml = c(y1,y2) ~ cw(x1,x2,x3)` which would lead to 6 estimations. Multiple estimations can also be combined to split samples (with the arguments `split`, `fsplit`).

Fixed-effects cannot be included in a stepwise fashion: they are there or not and stay the same for all estimations.

A note on performance. The feature of multiple estimations has been highly optimized for `feols`, in particular in the presence of fixed-effects. It is faster to estimate multiple models using the formula rather than with a loop. For non-`feols` models using the formula is roughly similar to using a loop performance-wise.

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)
etable(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 = 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:
etable(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,
  subset,
  split,
  fsplit,
  cluster,
  se,
  dof,
  panel.id,
  fixef,
  fixef.rm = "none",
  fixef.tol = 1e-06,
  fixef.iter = 10000,
  collin.tol = 1e-10,
  nthreads = getFixest_nthreads(),
  lean = FALSE,
  verbose = 0,
  warn = TRUE,
  notes = getFixest_notes(),
  combine.quick,
  demeaned = FALSE,
  mem.clean = FALSE,
  only.env = FALSE,
  env,
  ...
)
```

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`, see details. To add IVs, insert the endogenous vars./instruments after a pipe, like in `y ~ x | c(x_endo1,x_endo2) ~ x_inst1 + x_inst2`. Note that it should always be the last element, see details. Multiple estimations can be performed at once: for multiple dep. vars, wrap them in `c()`: ex `c(y1,y2)`. For multiple indep. vars, use the stepwise functions: ex `x1 + csw(x2,x3)`. The formula `fml = c(y1,y2) ~ x1 + cw0(x2,x3)` leads to 6 estimation, see details.

`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 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'.

`subset` A vector (logical or numeric) or a one-sided formula. If provided, then the estimation will be performed only on the observations defined by this argument.

`split` A one sided formula representing a variable (eg `split = ~var`) or a vector. If provided, the sample is split according to the variable and one estimation is performed for each value of that variable. If you also want to include the estimation for the full sample, use the argument `fsplit` instead.

`fsplit` A one sided formula representing a variable (eg `split = ~var`) or a vector. If provided, the sample is split according to the variable and one estimation is performed for each value of that variable. This argument is the same as split but also includes the full sample as the first estimation.

`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")]`, `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). You can interact two variables using `^` with the following syntax: `cluster = ~var1^var2` or `cluster = "var1^var2"`.

`se` Character scalar. Which kind of standard error should be computed: "standard", "hetero", "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.

`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.rm` Can be equal to "perfect" (default), "singleton", "both" or "none". Controls which observations are to be removed. If "perfect", then observations having a fixed-effect with perfect fit (e.g. only 0 outcomes in Poisson estimations) will be removed. If "singleton", all observations for which a fixed-effect appears only once will be removed. The meaning of "both" and "none" is direct.

`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.

`collin.tol` Numeric scalar, default is `1e-10`. Threshold deciding when variables should be considered collinear and subsequently removed from the estimation. Higher values means more variables will be removed (if there is presence of collinearity). One signal of presence of collinearity is t-stats that are extremely low (for instance when t-stats < 1e-3).

`nthreads` The number of threads. Can be: a) an integer lower than, or equal to, the maximum number of threads; b) 0: meaning all available threads will be used; c) a number strictly between 0 and 1 which represents the fraction of all threads to use. The default is to use 50% of all threads. You can set permanently the number of threads used within this package using the function `setFixest_nthreads`.

`lean` Logical, default is `FALSE`. If `TRUE` then all large objects are removed from the returned result: this will save memory but will block the possibility to use many methods. It is recommended to use the arguments `se` or `cluster` to obtain the appropriate standard-errors at estimation time, since obtaining different SEs won't be possible afterwards.

`verbose` Integer. Higher values give more information. In particular, it can detail the number of iterations in the demeaning algorithm (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.
mem.clean	Logical, default is FALSE. Only to be used if the data set is large compared to the available RAM. If TRUE then intermediary objects are removed as much as possible and gc is run before each substantial C++ section in the internal code to avoid memory issues.
only.env	(Advanced users.) Logical, default is FALSE. If TRUE, then only the environment used to make the estimation is returned.
env	(Advanced users.) A fixest environment created by a fixest estimation with only.env = TRUE. Default is missing. If provided, the data from this environment will be used to perform the estimation.
...	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. Note that fixest objects contain many elements and most of them are for internal use, they are presented here only for information. To access them, it is safer to use the user-level methods (e.g. vcov.fixest, resid.fixest, etc) or functions (like for instance fitstat to access any fit statistic).

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_all	A list containing different parts of the formula. Always contain the linear formula. Then depending on the cases: fixef: the fixed-effects, iv: the IV part of the formula.
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.

`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` (When relevant.) The offset formula.

`weights` (When relevant.) The weights formula.

`obsRemoved` (When relevant.) Vector of observations that were removed because of NA values.

`collin.var` (When relevant.) Vector containing the variables removed because of collinearity.

`collin.coef` (When relevant.) Vector of coefficients, where the values of the variables removed because of collinearity are NA.

`collin.min_norm` The minimal diagonal value of the Cholesky decomposition. Small values indicate possible presence collinearity.

`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`). 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]]

In general, for convergence reasons, it is recommended to always add the fixed-effect and avoid using only the variable with varying slope (i.e. use single square brackets).

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`, `f` and `d`.

You can provide several leads/lags/differences 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)`.

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.

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”).

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.

Instrumental variables

To estimate two stage least square regressions, insert the relationship between the endogenous regressor(s) and the instruments in a formula, after a pipe.

For example, `fml = y ~ x1 | x_endo ~ x_inst` will use the variables `x1` and `x_inst` in the first stage to explain `x_endo`. Then will use the fitted value of `x_endo` (which will be named `fit_x_endo`) and `x1` to explain `y`. To include several endogenous regressors, just use "+", like in: `fml = y ~ x1 | x_endo1 + x_end2 ~ x_inst1 + x_inst2`.

Of course you can still add the fixed-effects, but the IV formula must always come last, like in `fml = y ~ x1 | fe1 + fe2 | x_endo ~ x_inst`.

By default, the second stage regression is returned. You can access the first stage(s) regressions either directly in the slot `iv_first_stage` (not recommended), or using the argument `stage = 1` from the function `summary.fixest`. For example `summary(iv_est,stage = 1)` will give the first stage(s). Note that using summary you can display both the second and first stages at the same time using, e.g., `stage = 1:2` (using `2:1` would reverse the order).

Multiple estimations

Multiple estimations can be performed at once, they just have to be specified in the formula. Multiple estimations yield a `fixest_multi` object which is 'kind of' a list of all the results but includes specific methods to access the results in a handy way.

To include mutliple dependent variables, wrap them in `c()` (`list()` also works). For instance `fml = c(y1,y2) ~ x1` would estimate the model `fml = y1 ~ x1` and then the model `fml = y2 ~ x1`.

To include multiple independent variables, you need to use the stepwise functions. There are 4 stepwise functions associated to 4 short aliases. These are a) stepwise, stepwise0, cstepwise, cstepwise0, and b) sw, sw0, csw, csw0. Let's explain that. Assume you have the following formula: `fml = y ~ x1 + sw(x2,x3)`. The stepwise function `sw` will estimate the following two models: `y ~ x1 + x2` and `y ~ x1 + x3`. That is, each element in `sw()` is sequentially, and separately, added to the formula. Would have you used `sw0` in lieu of `sw`, then the model `y ~ x1` would also have been estimated. The `0` in the name means that the model wihtout any stepwise element also needs to be estimated. Finally, the prefix `c` means cumulative: each stepwise element is added to the next. That is, `fml = y ~ x1 + csw(x2,x3)` would lead to the following models `y ~ x1 + x2` and `y ~ x1 + x2 + x3`. The `0` has the same meaning and would also lead to the model without the stepwise elements to be estimated: in other words, `fml = y ~ x1 + csw0(x2,x3)` leads to the following three models: `y ~ x1`, `y ~ x1 + x2` and `y ~ x1 + x2 + x3`.

Multiple independent variables can be combined with multiple dependent variables, as in `fml = c(y1,y2) ~ cw(x1,x2,x3)` which would lead to 6 estimations. Multiple estimations can also be combined to split samples (with the arguments `split`, `fsplit`).

Fixed-effects cannot be included in a stepwise fashion: they are there or not and stay the same for all estimations.

A note on performance. The feature of multiple estimations has been highly optimized for `feols`, in particular in the presence of fixed-effects. It is faster to estimate multiple models using the formula rather than with a loop. For non-`feols` models using the formula is roughly similar to using a loop performance-wise.

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

```
#
# Basic estimation
#

res = feols(Sepal.Length ~ Sepal.Width + Petal.Length, iris)
# You can specify clustered standard-errors in summary:
summary(res, cluster = ~Species)

#
# Just one set of fixed-effects:
#

res = feols(Sepal.Length ~ Sepal.Width + Petal.Length | Species, iris)
# By default, the SEs are clustered according to the first fixed-effect
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:
#

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)

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

#
# Instrumental variables
#

# To estimate Two stage least squares,
# insert a formula describing the endo. vars./instr. relation after a pipe:

base = iris
names(base) = c("y", "x1", "x2", "x3", "fe1")
base$x_inst1 = 0.2 * base$x1 + 0.7 * base$x2 + rpois(150, 2)
base$x_inst2 = 0.2 * base$x2 + 0.7 * base$x3 + rpois(150, 3)
base$x_endo1 = 0.5 * base$y + 0.5 * base$x3 + rnorm(150, sd = 2)
base$x_endo2 = 1.5 * base$y + 0.5 * base$x3 + 3 * base$x_inst1 + rnorm(150, sd = 5)

# Using 2 controls, 1 endogenous var. and 1 instrument
res_iv = feols(y ~ x1 + x2 | x_endo1 ~ x_inst1, base)

# The second stage is the default
summary(res_iv)

# To show the first stage:
summary(res_iv, stage = 1)

# To show both the first and second stages:
summary(res_iv, stage = 1:2)
```

```
# Adding a fixed-effect => IV formula always last!
res_iv_fe = feols(y ~ x1 + x2 | fe1 | x_endo1 ~ x_inst1, base)

# With two endogenous regressors
res_iv2 = feols(y ~ x1 + x2 | x_endo1 + x_endo2 ~ x_inst1 + x_inst2, base)

# Now there's two first stages => a fixest_multi object is returned
sum_res_iv2 = summary(res_iv2, stage = 1)

# You can navigate through it by subsetting:
sum_res_iv2[iv = 1]

# The stage argument also works in etable:
etable(res_iv, res_iv_fe, res_iv2, order = "endo")

etable(res_iv, res_iv_fe, res_iv2, stage = 1:2, order = c("endo", "inst"),
       group = list(control = "!endo|inst"))

#
# Multiple estimations:
#

# 6 estimations
est_mult = feols(c(Ozone, Solar.R) ~ Wind + Temp + csw0(Wind:Temp, Day), airquality)

# We can display the results for the first lhs:
etable(est_mult[lhs = 1])

# And now the second (access can be made by name)
etable(est_mult[lhs = "Solar.R"])

# Now we focus on the two last right hand sides
# (note that .N can be used to specify the last item)
etable(est_mult[rhs = 2:.N])

# Combining with split
est_split = feols(c(Ozone, Solar.R) ~ sw(poly(Wind, 2), poly(Temp, 2)),
                  airquality, split = ~ Month)

# You can display everything at once with the print method
est_split

# Different way of displaying the results with "compact"
summary(est_split, "compact")

# You can still select which sample/LHS/RHS to display
est_split[sample = 1:2, lhs = 1, rhs = 1]
```

`fitstat` *Computes fit statistics of fixest objects*

Description

Computes various fit statistics for `fixest` estimations.

Usage

```
fitstat(x, type, simplify = FALSE, verbose = TRUE, show_types = FALSE, ...)
```

Arguments

`x`	A `fixest` estimation.
`type`	Character vector or one sided formula. The type of fit statistic to be computed. The classic ones are: n, rmse, r2, pr2, f, wald, ivf, ivwald. You have the full list in the details section or use `show_types = TRUE`. Further, you can register your own types with `fitstat_register`.
`simplify`	Logical, default is `FALSE`. By default a list is returned whose names are the selected types. If `simplify = TRUE` and only one type is selected, then the element is directly returned (ie will not be nested in a list).
`verbose`	Logical, default is `TRUE`. If `TRUE`, an object of class `fixest_fitstat` is returned (so its associated print method will be triggered). If `FALSE` a simple list is returned instead.
`show_types`	Logical, default is `FALSE`. If `TRUE`, only prompts all available types.
`...`	Other elements to be passed to other methods and may be used to compute the statistics (for example you can pass on arguments to compute the VCOV when using `type = "g"` or `type = "wald"`.).

Value

By default an object of class `fixest_fitstat` is returned. Using `verbose = FALSE` returns a simple a list. Finally, if only one type is selected, `simplify = TRUE` leads to the selected type to be returned.

Registering your own types

You can register custom fit statistics with the function `fitstat_register`.

Available types

The types are case sensitive, please use lower case only. The types available are:

- `n`, `ll`, `aic`, `bic`, `rmse`: The number of observations, the log-likelihood, the AIC, the BIC and the root mean squared error, respectively.
- `g`: The degrees of freedom used to compute the t-test (it influences the p-values of the coefficients). When the VCOV is clustered, this value is equal to the minimum cluster size, otherwise, it is equal to the sample size minus the number of variables.
- `r2`, `ar2`, `wr2`, `awr2`, `pr2`, `apr2`, `wpr2`, `awpr2`: All r2 that can be obtained with the function `r2`. The `a` stands for 'adjusted', the `w` for 'within' and the `p` for 'pseudo'. Note that the order of the letters `a`, `w` and `p` does not matter.

- `theta`: The over-dispersion parameter in Negative Binomial models. Low values mean high overdispersion.
- `f`, `wf`: The F-tests of nullity of the coefficients. The `w` stands for 'within'. These types return the following values: `stat`, `p`, `df1` and `df2`. If you want to display only one of these, use their name after a dot: e.g. `f.stat` will give the statistic of the F-test, or `wf.p` will give the p-values of the F-test on the projected model (i.e. projected onto the fixed-effects).
- `wald`: Wald test of joint nullity of the coefficients. This test always excludes the intercept and the fixed-effects. These type returns the following values: `stat`, `p`, `df1`, `df2` and `vcov`. The element `vcov` reports the way the VCOV matrix was computed since it directly influences this statistic.
- `ivf`, `ivf1`, `ivf2`, `ivfall`: These statistics are specific to IV estimations. They report either the IV F-test (namely the Cragg-Donald F statistic) of the first stage (`ivf` or `ivf1`), of the second stage (`ivf2`) or of both (`ivfall`). The F-test of the first stage is commonly named weak instrument test. The value of `ivfall` is only useful in `etable` when both the 1st and 2nd stages are displayed (it leads to the 1st stage F-test(s) to be displayed on the 1st stage estimation(s), and the 2nd stage one on the 2nd stage estimation – otherwise, `ivf1` would also be displayed on the 2nd stage estimation). These types return the following values: `stat`, `p`, `df1` and `df2`.
- `ivwald`, `ivwald1`, `ivwald2`, `ivwaldall`: These statistics are specific to IV estimations. They report either the IV Wald-test of the first stage (`ivwald` or `ivwald1`), of the second stage (`ivwald2`) or of both (`ivwaldall`). The Wald-test of the first stage is commonly named weak instrument test. Note that if the estimation was done with a robust VCOV, this is also known as the Kleibergen-Paap Wald test. The value of `ivwaldall` is only useful in `etable` when both the 1st and 2nd stages are displayed (it leads to the 1st stage Wald-test(s) to be displayed on the 1st stage estimation(s), and the 2nd stage one on the 2nd stage estimation – otherwise, `ivwald1` would also be displayed on the 2nd stage estimation). These types return the following values: `stat`, `p`, `df1`, `df2`, and `vcov`.
- `wh`: This statistic is specific to IV estimations. Wu-Hausman endogeneity test. H0 is the absence of endogeneity of the instrumented variables. It returns the following values: `stat`, `p`, `df1`, `df2`.
- `sargan`: Sargan test of overidentifying restrictions. H0: the instruments are not correlated with the second stage residuals. It returns the following values: `stat`, `p`, `df`.
- `lr`, `wlr`: Likelihood ratio and within likelihood ratio tests. It returns the following elements: `stat`, `p`, `df`. Concerning the within-LR test, note that, contrary to estimations with `femlm` or `feNmlm`, estimations with `feglm`/`fepois` need to estimate the model with fixed-effects only which may prove time-consuming (depending on your model). Bottom line, if you really need the within-LR and estimate a Poisson model, use `femlm` instead of `fepois` (the former uses direct ML maximization for which the only FEs model is a by product).

Examples

```
data(trade)
gravity = feols(log(Euros) ~ log(dist_km) | Destination + Origin, trade)

# Extracting the 'working' number of observations used to compute the pvalues
fitstat(gravity, "g", simplify = TRUE)
```

```
# Some fit statistics
fitstat(gravity, ~ rmse + r2 + wald + wf)

# You can use them in etable
etable(gravity, fitstat = ~ rmse + r2 + wald + wf)

# For wald and wf, you could show the pvalue instead:
etable(gravity, fitstat = ~ rmse + r2 + wald.p + wf.p)

# Now let's display some statistics that are not built-in
# => we use fitstat_register to create them

# We need: a) type name, b) the function to be applied
#          c) (optional) an alias

fitstat_register("tstand", function(x) tstat(x, se = "stand")[1], "t-stat (regular)")
fitstat_register("thc", function(x) tstat(x, se = "heter")[1], "t-stat (HC1)")
fitstat_register("t1w", function(x) tstat(x, se = "clus")[1], "t-stat (clustered)")
fitstat_register("t2w", function(x) tstat(x, se = "twow")[1], "t-stat (2-way)")

# Now we can use these keywords in fitstat:
etable(gravity, fitstat = ~ . + tstand + thc + t1w + t2w)

# Note that the custom stats we created are can easily lead
# to errors, but that's another story!
```

`fitstat_register` *Register custom fit statistics*

Description

Enables the registration of custom fi statistics that can be easily summoned with the function `fitstat`.

Usage

```
fitstat_register(type, fun, alias)
```

Arguments

`type`	A character scalar giving the type-name.
`fun`	A function to be applied to a `fixest` estimation. It must return either a scalar, either a list. Note that for the print method to work correctly, the names of the items of the list must be one of: `stat`, `p`, `df`, `df1`, `df2`, `vcov`. Only the print method is affected by this.
`alias`	An alias to be used in lieu of the type name in the display methods (ie when used in the function `print.fixest_fitstat` or `etable`).

Examples

```
data(trade)
gravity = feols(log(Euros) ~ log(dist_km) | Destination + Origin, trade)

# Extracting the 'working' number of observations used to compute the pvalues
fitstat(gravity, "g", simplify = TRUE)

# Some fit statistics
fitstat(gravity, ~ rmse + r2 + wald + wf)

# You can use them in etable
etable(gravity, fitstat = ~ rmse + r2 + wald + wf)

# For wald and wf, you could show the pvalue instead:
etable(gravity, fitstat = ~ rmse + r2 + wald.p + wf.p)

# Now let's display some statistics that are not built-in
# => we use fitstat_register to create them

# We need: a) type name, b) the function to be applied
#          c) (optional) an alias

fitstat_register("tstand", function(x) tstat(x, se = "stand")[1], "t-stat (regular)")
fitstat_register("thc", function(x) tstat(x, se = "heter")[1], "t-stat (HC1)")
fitstat_register("t1w", function(x) tstat(x, se = "clus")[1], "t-stat (clustered)")
fitstat_register("t2w", function(x) tstat(x, se = "twow")[1], "t-stat (2-way)")

# Now we can use these keywords in fitstat:
etable(gravity, fitstat = ~ . + tstand + thc + t1w + t2w)

# Note that the custom stats we created are can easily lead
# to errors, but that's another story!
```

`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"), na.rm = TRUE, ...)
```

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

Arguments

object	A fixest object. Obtained using the functions femlm, feols or feglm.
type	Character either equal to "response" (default) or "link". If type="response", then the output is at the level of the response variable, i.e. it is the expected predictor $E(Y\|X)$. If "link", then the output is at the level of the explanatory variables, i.e. the linear predictor $X \cdot \beta$.
na.rm	Logical, default is TRUE. If FALSE the number of observation returned will be the number of observations in the original data set, otherwise it will be the number of observations used in the estimation.
...	Not currently used.

Details

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

Value

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

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

Author(s)

Laurent Berge

See Also

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

Examples

```
# simple estimation on iris data, 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(), sorted = TRUE, ...)
```

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.
`sorted`	Logical, default is `TRUE`. Whether to order the fixed-effects by their names. If `FALSE`, then the order used in the demeaning algorithm is used.
`...`	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", "iv", "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 IV 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 *Create, or interact variables with, factors*

Description

Treat a variable as a factor, or interacts a variable with another treated as a factor. Values to be dropped/kept from the factor can be easily set. Note that to interact fixed-effects, this function should not be used: instead use directly the syntax `fe1^fe2`.

Usage

```
i(var, f, f2, ref, drop, keep, drop2, keep2)

interact(var, f, f2, ref, drop, keep, drop2, keep2)
```

Arguments

`var`	A vector to be interacted with `f`. If the other argument `f` is missing, then this vector will be treated as the argument `f`.
`f`	A vector (of any type) that will be treated as a factor. Must be of the same length as `var` if `var` is not missing.
`f2`	A vector (of any type) that will be treated as a factor. Must be of the same length as `f`.
`ref`	A single value that belongs to the interacted variable (`f`). Can be missing, can also be a logical: if `TRUE`, then the first value of `f` will be removed..
`drop`	A vector of regular expressions or integers (if `f` is integer). If provided, all values from `f` that match `drop` will be removed.

keep	A vector of regular expressions or integers (if f is integer). If provided, only the values from f that match keep will be kept.
drop2	A vector of regular expressions or integers (if f2 is integer). If provided, all values from f2 that match drop2 will be removed.
keep2	A vector of regular expressions or integers (if f2 is integer). If provided, only the values from f2 that match keep2 will be kept.

Details

To interact fixed-effects, this function should not be used: instead use directly the syntax fe1^fe2 in the fixed-effects part of the formula. Please see the details and examples in the help page of feols.

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 f minus the reference(s).

Shorthand in fixest estimations

In fixest estimations, instead of using i(var,f,ref), you can instead use the following writing var::f(ref). Note that this way of doing interactions is not endorsed any more and will likely be deprecated in the future.

Author(s)

Laurent Berge

See Also

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

Examples

```
#
# Simple illustration
#

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

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

# without interaction
cbind(x, y, i(y, ref = 1))

# You can interact factors too
z = rep(c("a", "b", "c"), c(5, 3, 2))
data.frame(z, y, i(z, y))
```

```
#
# In fixest estimations
#

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)

# => special treatment in coefplot
coefplot(est_did)

# Using i() for factors
est_bis = feols(y ~ x1 + i(period, keep = 3:6) + i(treat, period, 5) | id, base_did)

coefplot(est_bis, only.inter = FALSE)

# => special treatment in etable
etable(est_bis, dict = c("6" = "six"))

#
# Interact two factors => f2
#

# To interact two factor, use the argument f2
data(airquality)
aq = airquality
aq$week = aq$Day %/% 7 + 1

# Interacting Month and week:
res_2F = feols(Ozone ~ Solar.R + i(Month, f2 = week), aq)

# Same but dropping the 5th Month and 1st week
res_2F_bis = feols(Ozone ~ Solar.R + i(Month, f2 = week, drop = 5, drop2 = 1), aq)

etable(res_2F, res_2F_bis)
```

`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 = NULL,
  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, default is `NULL` (which means automatically set). Can be equal to: `"unitary"`, `"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* `fixest` *object*

Description

This function creates the left-hand-side or the right-hand-side(s) of a `femlm`, `feols` or `feglm` estimation.

Usage

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

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`.
`type`	Character vector or one sided formula, default is "rhs". Contains the type of matrix/data.frame to be returned. Possible values are: "lhs", "rhs", "fixef", "iv.rhs1", "iv.rhs2".
`na.rm`	Default is `TRUE`. Should observations with NAs be removed from the matrix?
`subset`	Logical or character vector. Default is `FALSE`. If `TRUE`, then the matrix created will be restricted only to the variables contained in the argument `data`, which can then contain a subset of the variables used in the estimation. If a character vector, then only the variables matching the elements of the vector via regular expressions will be created.
`...`	Not currently used.

Value

It returns either a matrix or a data.frame. It returns a matrix for the "rhs", "iv.rhs1" and "iv.rhs2" parts. A data.frame for "lhs" and "fixef".

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

```
base = iris
names(base) = c("y", "x1", "x2", "x3", "species")

est = feols(y ~ poly(x1, 2) + x2, base)
head(model.matrix(est))

# Illustration of subset

# subset => character vector
head(model.matrix(est, subset = "x1"))

# subset => TRUE, only works with data argument!!
head(model.matrix(est, data = base[, "x1", drop = FALSE], subset = TRUE))
```

---

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)
```

---

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 = NULL, 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 dedicated section in `feols`).
`time.step`	The method to compute the lags, default is `NULL` (which means automatically set). Can be equal to: `"unitary"`, `"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.

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.

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"), na.rm = TRUE, ...)
```

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 $X \cdot \beta$.

na.rm	Logical, default is TRUE. Only used when the argument newdata is missing. If FALSE the number of observation returned will be the number of observations in the original data set, otherwise it will be the number of observations used in the estimation.
...	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 can be customized by the user (using function `setFixest_print`).

Usage

```
## S3 method for class 'fixest'
print(x, n, type = "table", fitstat = NULL, ...)

setFixest_print(type = "table", fitstat = NULL)

getFixest_print()
```

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.
`fitstat`	A formula or a character vector representing which fit statistic to display. The types must be valid types of the function `fitstat`. The default fit statistics depend on the type of estimation (OLS, GLM, IV, with/without fixed-effect). Providing the argument `fitstat` overrides the default fit statistics, you can however use the point "." to summon them back. Ex 1: `fitstat = ~ . + ll` adds the log-likelihood to the default values. Ex 2: `fitstat = ~ ll + pr2` only displays the log-likelihood and the pseudo-R2.
`...`	Other arguments to be passed to `vcov.fixest`.

Details

It is possible to set the default values for the arguments `type` and `fitstat` by using the function `setFixest_print`.

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 = fepois(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:
setFixest_print(type = "coef")
est_pois
# back to default:
setFixest_print(type = "table")

#
# fitstat
#

# We modify which fit statistic to display
print(est_pois, fitstat = ~ . + lr)

# We add the LR test to the default (represented by the ".")

# to show only the LR stat:
print(est_pois, fitstat = ~ . + lr.stat)

# To modify the defaults:
setFixest_print(fitstat = ~ . + lr.stat + rmse)
est_pois

# Back to default (NULL == default)
setFixest_print(fitstat = NULL)
```

---

print.fixest_fitstat *Print method for fit statistics of fixest estimations*

Description

Displays a brief summary of selected fit statistics from the function `fitstat`.

Usage

```
## S3 method for class 'fixest_fitstat'
print(x, na.rm = FALSE, ...)
```

Arguments

x	An object resulting from the fitstat function.
na.rm	Logical, default is FALSE. If TRUE, the statistics that are missing are not displayed.
...	Not currently used.

Examples

```
data(trade)
gravity = feols(log(Euros) ~ log(dist_km) | Destination + Origin, trade)

# Extracting the 'working' number of observations used to compute the pvalues
fitstat(gravity, "g", simplify = TRUE)

# Some fit statistics
fitstat(gravity, ~ rmse + r2 + wald + wf)

# You can use them in etable
etable(gravity, fitstat = ~ rmse + r2 + wald + wf)

# For wald and wf, you could show the pvalue instead:
etable(gravity, fitstat = ~ rmse + r2 + wald.p + wf.p)

# Now let's display some statistics that are not built-in
# => we use fitstat_register to create them

# We need: a) type name, b) the function to be applied
#          c) (optional) an alias

fitstat_register("tstand", function(x) tstat(x, se = "stand")[1], "t-stat (regular)")
fitstat_register("thc", function(x) tstat(x, se = "heter")[1], "t-stat (HC1)")
fitstat_register("t1w", function(x) tstat(x, se = "clus")[1], "t-stat (clustered)")
fitstat_register("t2w", function(x) tstat(x, se = "twow")[1], "t-stat (2-way)")

# Now we can use these keywords in fitstat:
etable(gravity, fitstat = ~ . + tstand + thc + t1w + t2w)

# Note that the custom stats we created are can easily lead
# to errors, but that's another story!
```

print.fixest_multi *Print method for fixest_multi objects*

Description

Displays summary information on fixest_multi objects in the R console.

Usage

```
## S3 method for class 'fixest_multi'
print(x, ...)
```

Arguments

x	A `fixest_multi` object, obtained from a `fixest` estimation leading to multiple results.
...	Other arguments to be passed to `summary.fixest_multi`.

See Also

The main fixest estimation functions: `feols`, `fepois`, `fenegbin`, `feglm`, `feNmlm`. Tools for mutliple fixest estimations: `summary.fixest_multi`, `print.fixest_multi`, `as.list.fixest_multi`, `sub-sub-.fixest_multi`, `sub-.fixest_multi`, `cash-.fixest_multi`.

Examples

```
base = iris
names(base) = c("y", "x1", "x2", "x3", "species")

# Multiple estimation
res = feols(y ~ csw(x1, x2, x3), base, split = ~species)

# Let's print all that
res
```

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) possibly missing. E.g. to get the regular R2: use type = "r2", the within adjusted R2: use type = "war2", the pseudo R2: use type = "pr2", etc. Use "cor2" 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 cor2, 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 number 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, "cor2")

# "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("cor2", "r2", "pr2", "war2"))

# same with full names instead of codes
r2(est, c("cor2", "r2", "pr2", "war2"), full_names = TRUE)
```

`rep.fixest` *Replicates fixest objects*

Description

Simple function that replicates fixest objects while (optionally) computing different standard-errors. Useful mostly in combination with `etable` or `coefplot`.

Usage

```
## S3 method for class 'fixest'
rep(x, times = 1, each = 1, cluster, ...)

## S3 method for class 'fixest_list'
rep(x, times = 1, each = 1, cluster, ...)

.l(...)
```

Arguments

`x`	Either a fixest object, either a list of fixest objects created with `.l()`.
`times`	Integer vector giving the number of repetitions of the vector of elements. By default `times = 1`. It must be either of length 1, either of the same length as the argument `x`.
`each`	Integer scalar indicating the repetition of each element. Default is 1.
`cluster`	A list containing the types of standard-error to be computed, default is missing. If not missing, it must be of the same length as `times`, `each`, or the final vector. Note that if the arguments `times` and `each` are missing, then `times` becomes equal to the length of `cluster`. (Note that `cluster` accepts the character values `"standard"` or `"hetero"` to compute non-clustered SEs.)
`...`	In `.l()`: `fixest` objects. In `rep()`: not currently used.

Details

To apply `rep.fixest` on a list of fixest objects, it is absolutely necessary to use `.l()` and not `list()`.

Value

Returns a list of the appropriate length. Each element of the list is a fixest object.

Examples

```
# Let's show results with different standard-errors

est = feols(Ozone ~ Solar.R + Wind + Temp, data = airquality)

my_cluster = list("Month", "Day", ~ Day + Month)

etable(rep(est, cluster = my_cluster))

coefplot(rep(est, cluster = my_cluster), drop = "Int")

#
# To rep multiple objects, you need to use .l()
#

est_bis = feols(Ozone ~ Solar.R + Wind + Temp | Month, airquality)

etable(rep(.l(est, est_bis), cluster = my_cluster))

# using each
etable(rep(.l(est, est_bis), each = 3, cluster = my_cluster))
```

---

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"),
  na.rm = TRUE,
  ...
)

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

```
  type = c("response", "deviance", "pearson", "working"),
  na.rm = TRUE,
  ...
)
```

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.
na.rm	Logical, default is TRUE. Whether to remove the observations with NAs from the original data set. If FALSE, then the vector returned is always of the same length as the original data set.
...	Not currently used.

Value

It returns a numeric vector of the length the number of observations used for the estimation (if na.rm = TRUE) or of the length of the original data set (if na.rm = FALSE).

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 function. 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, it is equal to getFixest_dict(), a default dictionary which can be set with setFixest_dict. You can use dict = FALSE to disable it..
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.
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 (note that you can use the special character "^^" to clean the beginning of the names, see example), 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\",` `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_estimation` *Default arguments for fixest estimations*

Description

This function sets globally the default arguments of fixest estimations.

Usage

```
setFixest_estimation(
  fixef.rm = "perfect",
  fixef.tol = 1e-06,
  fixef.iter = 10000,
  collin.tol = 1e-10,
  lean = FALSE,
  verbose = 0,
  warn = TRUE,
  combine.quick = NULL,
  demeaned = FALSE,
  mem.clean = FALSE,
  glm.iter = 25,
  glm.tol = 1e-08,
  panel.id = NULL,
  reset = FALSE
)

getFixest_estimation()
```

Arguments

`fixef.rm`	Can be equal to "perfect" (default), "singleton", "both" or "none". Controls which observations are to be removed. If "perfect", then observations having a fixed-effect with perfect fit (e.g. only 0 outcomes in Poisson estimations) will be removed. If "singleton", all observations for which a fixed-effect appears only once will be removed. The meaning of "both" and "none" is direct.
`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.
`collin.tol`	Numeric scalar, default is `1e-10`. Threshold deciding when variables should be considered collinear and subsequently removed from the estimation. Higher values means more variables will be removed (if there is presence of collinearity). One signal of presence of collinearity is t-stats that are extremely low (for instance when t-stats < 1e-3).

`lean` Logical, default is `FALSE`. If `TRUE` then all large objects are removed from the returned result: this will save memory but will block the possibility to use many methods. It is recommended to use the arguments `se` or `cluster` to obtain the appropriate standard-errors at estimation time, since obtaining different SEs won't be possible afterwards.

`verbose` Integer. Higher values give more information. In particular, it can detail the number of iterations in the demeaning algorithm (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).

`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`.

`mem.clean` Logical, default is `FALSE`. Only to be used if the data set is large compared to the available RAM. If `TRUE` then intermediary objects are removed as much as possible and `gc` is run before each substantial C++ section in the internal code to avoid memory issues.

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

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

`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`).

`reset` Logical, default to `FALSE`. Whether to reset all values.

Value

The function `getFixest_estimation` returns the currently set global defaults.

Examples

```
#
# Example: removing singletons is FALSE by default
#

# => changing this default
```

```
# Let's create data with singletons
base = iris
names(base) = c("y", "x1", "x2", "x3", "species")
base$fe_singletons = as.character(base$species)
base$fe_singletons[1:5] = letters[1:5]

res          = feols(y ~ x1 + x2 | fe_singletons, base)
res_noSingle = feols(y ~ x1 + x2 | fe_singletons, base, fixef.rm = "single")

# New defaults
setFixest_estimation(fixef.rm = "single")
res_newDefault = feols(y ~ x1 + x2 | fe_singletons, base)

etable(res, res_noSingle, res_newDefault)

# Resetting the defaults
setFixest_estimation(reset = TRUE)
```

`setFixest_fml` *Sets/gets 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

```
setFixest_fml(..., reset = FALSE)

getFixest_fml()
```

Arguments

`...`	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 or a character vector, it is the definition of the macro variable. Example of a valid call: `setFixest_fml(..ctrl = ~ var1 + var2)`. In the function xpd, the default 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")

#
# You can use xpd for stepwise estimations
#

# we want to look at the effect of x1 on y
# controlling for different variables

base = iris
names(base) = c("y", "x1", "x2", "x3", "species")

# We first create a matrix with all possible combinations of variables
my_args = lapply(names(base)[-(1:2)], function(x) c("", x))
(all_combs = as.matrix(do.call("expand.grid", my_args)))

res_all = list()
for(i in 1:nrow(all_combs)){
  res_all[[i]] = feols(xpd(y ~ x1 + ..v, ..v = all_combs[i, ]), base)
}
```

```
etable(res_all)
coefplot(res_all, group = list(Species = "^^species"))

#
# You can use macros to grep variables in your data set
#

# Example 1: setting a macro variable globally

data(longley)
setFixest_fml(..many_vars = grep("GNP|ployed", names(longley), value = TRUE))
feols(Armed.Forces ~ Population + ..many_vars, longley)

# Example 2: using ..("regex") to grep the variables "live"

feols(Armed.Forces ~ Population + ..("GNP|ployed"), longley)

# Example 3: same as Ex.2 but without using a fixest estimation

# Here we need to use xpd():
lm(xpd(Armed.Forces ~ Population + ..("GNP|ployed"), data = longley), longley)

#
# You can also put numbers in macros
#

res_all = list()
for(p in 1:3){
  res_all[[p]] = feols(xpd(Ozone ~ Wind + poly(Temp, ..p), ..p = p), airquality)
}

etable(res_all)

#
# lhs and rhs arguments
#

# to create a one sided formula from a character vector
vars = letters[1:5]
xpd(rhs = vars)

# Alternatively, to replace the RHS
xpd(y ~ 1, rhs = vars)

# To create a two sided formula
xpd(lhs = "y", rhs = vars)
```

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	The number of threads. Can be: a) an integer lower than, or equal to, the maximum number of threads; b) 0: meaning all available threads will be used; c) a number strictly between 0 and 1 which represents the fraction of all threads to use. If missing, the default is to use 50% of all threads.

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_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",
  all = NULL,
  reset = FALSE
)

getFixest_se()
```

Arguments

no_FE	Character scalar equal to either: "standard" (default), or "hetero". The type of standard-errors to use by default for estimations without fixed-effects.
one_FE	Character scalar equal to either: "standard", "hetero", 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", "hetero", "cluster" (default), or "twoway". The type of standard-errors to use by default for estimations with *two or more* fixed-effects.
all	Character scalar equal to either: "standard", or "hetero". By default is is NULL. If provided, it sets all the SEs to that value.
reset	Logical, default is FALSE. Whether to reset to the default values.

Value

The function getFixest_se() returns a list with three elements containing the default for estimations i) without, 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 = "hetero", one_FE = "standard", two_FE = "twoway")
etable(est_no_FE, est_one_FE, est_two_FE)

# Resetting 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)
```

---

stepwise *Stepwise estimation tools*

Description

Functions to perform stepwise estimations.

Usage

```
stepwise(...)

sw(...)

cstepwise(...)

csw(...)

stepwise0(...)

sw0(...)
```

```
cstepwise0(...)

csw0(...)
```

Arguments

...	Represents formula variables to be added in a stepwise fashion to an estimation.

Details

To include multiple independent variables, you need to use the stepwise functions. There are 4 stepwise functions associated to 4 short aliases. These are a) stepwise, stepwise0, cstepwise, cstepwise0, and b) sw, sw0, csw, csw0. Let's explain that.

Assume you have the following formula: `fml = y ~ x1 + sw(x2,x3)`. The stepwise function `sw` will estimate the following two models: `y ~ x1 + x2` and `y ~ x1 + x3`. That is, each element in `sw()` is sequentially, and separately, added to the formula. Would have you used `sw0` in lieu of `sw`, then the model `y ~ x1` would also have been estimated. The `0` in the name means that the model wihtout any stepwise element also needs to be estimated.

Finally, the prefix `c` means cumulative: each stepwise element is added to the next. That is, `fml = y ~ x1 + csw(x2,x3)` would lead to the following models `y ~ x1 + x2` and `y ~ x1 + x2 + x3`. The `0` has the same meaning and would also lead to the model without the stepwise elements to be estimated: in other words, `fml = y ~ x1 + csw0(x2,x3)` leads to the following three models: `y ~ x1`, `y ~ x1 + x2` and `y ~ x1 + x2 + x3`.

Examples

```
base = iris
names(base) = c("y", "x1", "x2", "x3", "species")

# Regular stepwise
feols(y ~ sw(x1, x2, x3), base)

# Cumulative stepwise
feols(y ~ csw(x1, x2, x3), base)

# Using the 0
feols(y ~ x1 + x2 + sw0(x3), base)
```

`style.df` *Style of data.frames created by etable*

Description

This function describes the style of data.frames created with the function `etable`.

Usage

```
style.df(
  depvar.title = "Dependent Var.:",
  fixef.title = "Fixed-Effects:",
  fixef.line = "-",
  fixef.prefix = "",
  fixef.suffix = "",
  slopes.title = "Varying Slopes:",
  slopes.line = "-",
  slopes.format = "__var__ (__slope__)",
  stats.title = "_",
  stats.line = "_",
  yesNo = c("Yes", "No")
)
```

Arguments

depvar.title	Character scalar. Default is "Dependent Var.:". The row name of the dependent variables.
fixef.title	Character scalar. Default is "Fixed-Effects:". The header preceding the fixed-effects. If equal to the empty string, then this line is removed.
fixef.line	A single character. Default is "-". A character that will be used to create a line of separation for the fixed-effects header. Used only if fixef.title is not the empty string.
fixef.prefix	Character scalar. Default is "". A prefix to appear before each fixed-effect name.
fixef.suffix	Character scalar. Default is "". A suffix to appear after each fixed-effect name.
slopes.title	Character scalar. Default is "Varying Slopes:". The header preceding the variables with varying slopes. If equal to the empty string, then this line is removed.
slopes.line	Character scalar. Default is "-". A character that will be used to create a line of separation for the variables with varying slopes header. Used only if slopes.line is not the empty string.
slopes.format	Character scalar. Default is "__var__ (__slope__)". The format of the name of the varying slopes. The values __var__ and __slope__ are special characters that will be replaced by the value of the variable name and slope name, respectively.
stats.title	Character scalar. Default is "_". The header preceding the statistics section. If equal to the empty string, then this line is removed. If equal to single character (like in the default), then this character will be expanded to take the full column width.
stats.line	Character scalar. Default is "_". A character that will be used to create a line of separation for the statistics header. Used only if stats.title is not the empty string.
yesNo	Character vector of length 1 or 2. Default is c("Yes","No"). Used to inform on the presence or absence of fixed-effects in the estimation. If of length 1, then automatically the second value is considered as the empty string.

Details

The title elements (`depvar.title`, `fixef.title`, `slopes.title` and `stats.title`) will be the row names of the returned data.frame. Therefore keep in mind that any two of them should not be identical (since identical row names are forbidden in data.frames).

Value

It returns an object of class `fixest_style_df`.

Examples

```
# Multiple estimations => see details in feols
aq = airquality
est = feols(c(Ozone, Solar.R) ~
               Wind + csw(Temp, Temp^2, Temp^3) | Month + Day,
            data = aq)


# Default result
etable(est)

# Playing a bit with the styles
etable(est, style_df = style.df(fixef.title = "", fixef.suffix = " FE",
                                stats.line = " ", yesNo = "yes"))
```

---

style.tex *Style definitions for Latex tables*

Description

This function describes the style of Latex tables to be exported with the function `etable`.

Usage

```
style.tex(
  main = "base",
  depvar.title,
  model.title,
  model.format,
  line.top,
  line.bottom,
  var.title,
  fixef.title,
  fixef.prefix,
  fixef.suffix,
```

```
  fixef.where,
  slopes.title,
  slopes.format,
  fixef_sizes.prefix,
  fixef_sizes.suffix,
  stats.title,
  notes.title,
  tablefoot,
  tablefoot.title,
  tablefoot.value,
  yesNo,
  tabular = "normal"
)
```

Arguments

main	Either "base", "aer" or "qje". Defines the basic style to start from. The styles "aer" and "qje" are almost identical and only differ on the top/bottom lines.
depvar.title	A character scalar. The title of the line of the dependent variables (defaults to "Dependent variable(s):" if main = "base" (the 's' appears only if just one variable) and to "" if main = "aer").
model.title	A character scalar. The title of the line of the models (defaults to "Model:" if main = "base" and to "" if main = "aer").
model.format	A character scalar. The value to appear on top of each column. It defaults to "(1)". Note that 1, i, I, a and A are special characters: if found, their values will be automatically incremented across columns.
line.top	A character scalar. The line at the top of the table (defaults to "\tabularnewline\toprule\toprule" if main = "base" and to "\toprule" if main = "aer").
line.bottom	A character scalar. The line at the bottom of the table (defaults to "" if main = "base" and to "\bottomrule" if main = "aer").
var.title	A character scalar. The title line appearing before the variables (defaults to "\midrule \emph{Variables}" if main = "base" and to "\midrule" if main = "aer"). Note that the behavior of var.title = " " (a space) is different from var.title = "" (the empty string): in the first case you will get an empty row, while in the second case you get no empty row. To get a line without an empty row, use "\midrule" (and not "\midrule "!–the space!).
fixef.title	A character scalar. The title line appearing before the fixed-effects (defaults to "\midrule \emph{Fixed-effects}" if main = "base" and to " " if main = "aer"). Note that the behavior of fixef.title = " " (a space) is different from fixef.title = "" (the empty string): in the first case you will get an empty row, while in the second case you get no empty row. To get a line without an empty row, use "\midrule" (and not "\midrule "!–the space!).
fixef.prefix	A prefix to add to the fixed-effects names. Defaults to "" (i.e. no prefix).
fixef.suffix	A suffix to add to the fixed-effects names. Defaults to "" if main = "base") and to "fixed-effects" if main = "aer").

`fixef.where` Either "var" or "stats". Where to place the fixed-effects lines? Defaults to `"var"`, i.e. just after the variables, if `main = "base"`) and to `"stats"`, i.e. just after the statistics, if `main = "aer"`).

`slopes.title` A character scalar. The title line appearing before the variables with varying slopes (defaults to `"\midrule \emph{Varying Slopes}"` if `main = "base"` and to `""` if `main = "aer"`). Note that the behavior of `slopes.title = " "` (a space) is different from `slopes.title = ""` (the empty string): in the first case you will get an empty row, while in the second case you get no empty row. To get a line without an empty row, use `"\midrule"` (and not `"\midrule "`!–the space!).

`slopes.format` Character scalar representing the format of the slope variable name. There are two special characters: "__var__" and "__slope__", placeholers for the variable and slope names. Defaults to `"__var__ (__slope__)"` if `main = "base"`) and to `"__var__ $\times $ __slope__"` if `main = "aer"`).

`fixef_sizes.prefix`
A prefix to add to the fixed-effects names. Defaults to `"# "`.

`fixef_sizes.suffix`
A suffix to add to the fixed-effects names. Defaults to `""` (i.e. no suffix).

`stats.title` A character scalar. The title line appearing before the statistics (defaults to `"\midrule \emph{Fit statistics}"` if `main = "base"` and to `" "` if `main = "aer"`). Note that the behavior of `stats.title = " "` (a space) is different from `stats.title = ""` (the empty string): in the first case you will get an empty row, while in the second case you get no empty row. To get a line without an empty row, use `"\midrule"` (and not `"\midrule "`!–the space!).

`notes.title` A character scalar. The title appearing just before the notes, defaults to `"\medskip \emph{Notes:} "`.

`tablefoot` A logical scalar. Whether or not to display a footer within the table. Defaults to `TRUE` if `main = "aer"`) and `FALSE` if `main = "aer"`).

`tablefoot.title`
A character scalar. Only if `tablefoot = TRUE`, value to appear before the table footer. Defaults to `"\bottomrule\bottomrule"` if `main = "base"`.

`tablefoot.value`
A character scalar. The notes to be displayed in the footer. Defaults to `"default"` if `main = "base"`, which leads to custom footers informing on the type of standard-error and significance codes, depending on the estimations.

`yesNo` A character vector of length 1 or 2. Defaults to `"Yes"` if `main = "base"` and to `"$\checkmark$"` if `main = "aer"` (from package `amssymb`). 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.

`tabular` Character scalar equal to "normal" (default), "*" or "X". Represents the type of tabular to export.

Details

The `\checkmark` command, used in the "aer" style (in argument `yesNo`), is in the `amssymb` package.

The commands `\toprule`, `\midrule` and `\bottomrule` are in the `booktabs` package. You can set the width of the top/bottom rules with \setlength\heavyrulewidth{wd}, and of the midrule with \setlength\lightrulewidth{wd}.

Value

Returns a list containing the style parameters.

See Also

etable

Examples

```
# Multiple estimations => see details in feols
aq = airquality
est = feols(c(Ozone, Solar.R) ~
               Wind + csw(Temp, Temp^2, Temp^3) | Month + Day,
            data = aq)

# Playing a bit with the styles
etable(est, tex = TRUE)
etable(est, tex = TRUE, style.tex = style.tex("aer"))

etable(est, tex = TRUE, style.tex = style.tex("aer",
                                    var.title = "\\emph{Expl. Vars.}",
                                    model.format = "[i]",
                                    yesNo = "x",
                                    tabular = "*"))
```

---

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 = NULL,
  cluster = NULL,
  dof = NULL,
  .vcov,
  stage = 2,
  lean = FALSE,
```

```
  agg = NULL,
  forceCovariance = FALSE,
  keepBounded = FALSE,
  n,
  nthreads = getFixest_nthreads(),
  ...
)

summ(
  object,
  se,
  cluster,
  dof = getFixest_dof(),
  forceCovariance = FALSE,
  keepBounded = FALSE,
  ...
)

## S3 method for class 'fixest_list'
summary(
  object,
  se,
  cluster,
  dof = getFixest_dof(),
  .vcov,
  stage = 2,
  lean = FALSE,
  n,
  ...
)
```

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", "hetero", "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")]`, `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). You can interact two variables using `^` with the following syntax: `cluster = ~var1^var2` or `cluster = "var1^var2"`.

`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 or a function computing this matrix. If a matrix, it must be a square matrix of the same number of rows as the number of variables estimated. If a function, it must return the previously mentioned matrix.

`stage` Can be equal to `2` (default), `1`, `1:2` or `2:1`. Only used if the object is an IV estimation: defines the stage to which `summary` should be applied. If `stage = 1` and there are multiple endogenous regressors or if `stage` is of length 2, then an object of class `fixest_multi` is returned.

`lean` Logical, default is `FALSE`. Used to reduce the (memory) size of the summary object. If `TRUE`, then all objects of length N (the number of observations) are removed from the result. Note that some `fixest` methods may consequently not work when applied to the summary.

`agg` A character scalar describing the variable names to be aggregated, it is pattern-based. All variables that match the pattern will be aggregated. It must be of the form `"(root)"`, the parentheses must be there and the resulting variable name will be `"root"`. You can add another root with parentheses: `"(root1)regex(root2)"`, in which case the resulting name is `"root1::root2"`. To name the resulting variable differently you can pass a named vector: `c("name" = "pattern")` or `c("name" = "pattern(root2)")`. It's a bit intricate sorry, please see the examples.

`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.

`nthreads` The number of threads. Can be: a) an integer lower than, or equal to, the maximum number of threads; b) 0: meaning all available threads will be used; c) a number strictly between 0 and 1 which represents the fraction of all threads to use. The default is to use 50% of all threads. You can set permanently the number of threads used within this package using the function `setFixest_nthreads`.

`...` Only used if the argument `.vocv` is provided and is a function: extra arguments to be passed to that function.

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.

Compatibility with sandwich package

The VCOVs from sandwich can be used with feols, feglm and fepois estimations. If you want to have a sandwich VCOV when using summary.fixest, you can use the argument .vcov to specify the VCOV function to use (see examples). Note that if you do so and you use a formula in the cluster argument, an innocuous warning can pop up if you used several non-numeric fixed-effects in the estimation (this is due to the function expand.model.frame used in sandwich).

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

# Comparing different types of standard errors
sum_standard = summary(est_pois, se = "standard")
sum_hetero   = summary(est_pois, se = "hetero")
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_hetero, 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)

# You can interact the clustering variables "live" using the var1 ^ var2 syntax.

summary(est_pois, cluster = "Destination^Product")
summary(est_pois, cluster = ~Destination^Product)
# Equivalent to
summary(est_pois, cluster = paste(trade$Destination, trade$Product))


#
# Compatibility with sandwich
#

# You can use the VOCVs from sandwich by using the argument .vcov:
library(sandwich)
summary(est_pois, .vcov = vcovCL, cluster = trade[, c("Destination", "Product")])
```

`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_multi *Summary for fixest_multi objects*

Description

Summary information for fixest_multi objects. In particular, this is used to specify the type of standard-errors to be computed.

Usage

```
## S3 method for class 'fixest_multi'
summary(
  object,
  type = "short",
  se = NULL,
  cluster = NULL,
  dof = NULL,
  .vcov,
  stage = 2,
  lean = FALSE,
  n,
  ...
)
```

Arguments

object	A fixest_multi object, obtained from a fixest estimation leading to multiple results.
type	A character either equal to "short", "long", "compact", or "se_compact". If short, only the table of coefficients is displayed for each estimation. If long, then the full results are displayed for each estimation. If compact, a data.frame is returned with one line per model and the formatted coefficients + standard-errors in the columns. If se_compact, a data.frame is returned with one line per model, one numeric column for each coefficient and one numeric column for each standard-error.
se	Character scalar. Which kind of standard error should be computed: "standard", "hetero", "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")], 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). You can interact two variables using ^ with the following syntax: cluster = ~var1^var2 or cluster = "var1^var2".
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 or a function computing this matrix. If a matrix, it must be a square matrix of the same number of rows as the number of variables estimated. If a function, it must return the previously mentioned matrix.
stage	Can be equal to 2 (default), 1, 1:2 or 2:1. Only used if the object is an IV estimation: defines the stage to which summary should be applied. If stage = 1 and there are multiple endogenous regressors or if stage is of length 2, then an object of class fixest_multi is returned.
lean	Logical, default is FALSE. Used to reduce the (memory) size of the summary object. If TRUE, then all objects of length N (the number of observations) are removed from the result. Note that some fixest methods may consequently not work when applied to the summary.
n	Integer, default is missing (means Inf). Number of coefficients to display when the print method is used.
...	Not currently used.

Value

It returns either an object of class `fixest_multi` (if `type` equals `short` or `long`), either a `data.frame` (if type equals `compact` or `se_compact`).

See Also

The main fixest estimation functions: `feols`, `fepois`, `fenegbin`, `feglm`, `feNmlm`. Tools for mutliple fixest estimations: `summary.fixest_multi`, `print.fixest_multi`, `as.list.fixest_multi`, `sub-sub-.fixest_multi`, `sub-.fixest_multi`, `cash-.fixest_multi`.

Examples

```
base = iris
names(base) = c("y", "x1", "x2", "x3", "species")

# Multiple estimation
res = feols(y ~ csw(x1, x2, x3), base, split = ~species)

# By default, the type is "short"
# You can stil use the arguments from summary.fixest
summary(res, cluster = ~ species)

summary(res, type = "long")

summary(res, type = "compact")

summary(res, type = "se_compact")
```

---

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`, then 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, evaluate = TRUE, ...)
```

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.
evaluate	Logical, default is TRUE. If FALSE, only the updated call is returned.
...	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 *Computes the variance/covariance of a* fixest *object*

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 = NULL,
  attr = FALSE,
  forceCovariance = FALSE,
  keepBounded = FALSE,
  nthreads = getFixest_nthreads(),
  ...
)
```

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", "hetero", "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")], 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). You can interact two variables using ^ with the following syntax: cluster = ~var1^var2 or cluster = "var1^var2".
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.

`attr` Logical, defaults to `FALSE`. Whether to include the attributes describing how the VCOV was computed.

`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).

`nthreads` The number of threads. Can be: a) an integer lower than, or equal to, the maximum number of threads; b) 0: meaning all available threads will be used; c) a number strictly between 0 and 1 which represents the fraction of all threads to use. The default is to use 50% of all threads. You can set permanently the number of threads used within this package using the function `setFixest_nthreads`.

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

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

Details

For an explanation on how the standard-errors are computed and what is the exact meaning of the arguments, please have a look at the dedicated vignette: On standard-errors.

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. if it was created using a heteroskedascity-robust correction, or if it was 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)

# Heteroskedasticity-robust VCOV
vcov(est_pois, se = "hetero")

# "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)
```

`wald` *Wald test of nullity of coefficients*

Description

Wald test used to test the joint nullity of a set of coefficients.

Usage

```
wald(x, keep = NULL, drop = NULL, print = TRUE, se, cluster, ...)
```

Arguments

`x`	A `fixest` object. Obtained using the methods `femlm`, `feols` or `feglm`.
`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.

print — Logical, default is TRUE. If TRUE, then a verbose description of the test is prompted on the R console. Otherwise only a named vector containing the test statistics is returned.

se — Character scalar. Which kind of standard error should be computed: "standard", "hetero", "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")], 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). You can interact two variables using ^ with the following syntax: cluster = ~var1^var2 or cluster = "var1^var2".

... — Any other element to be passed to summary.fixest.

Details

The type of VCOV matrix plays a crucial role in this test. Use the arguments se and cluster to change the type of VCOV for the test.

Value

A named vector containing the following elements is returned: stat, p, df1, and df2. They correspond to the test statistic, the p-value, the first and second degrees of freedoms.

If no valud coefficient is found, the value NA is returned.

Examples

```
data(airquality)

est = feols(Ozone ~ Solar.R + Wind + poly(Temp, 3), airquality)

# Testing the joint nullity of the Temp polynomial
wald(est, "poly")
```

```
# Same but with clustered SEs
wald(est, "poly", cluster = "Month")

# Now: all vars but the polynomial and the intercept
wald(est, drop = "Inte|poly")

#
# Toy example: testing pre-trends
#

data(base_did)

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

# The graph of the coefficients
coefplot(est_did)

# The pre-trend test
wald(est_did, "period::[1234]$")

# If "period::[1234]$" looks weird to you, check out
# regular expressions: e.g. see ?regex.
# Learn it, you won't regret it!
```

`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 *Expands formula macros*

Description

Create macros within formulas and expand them with character vectors or other formulas.

Usage

```
xpd(fml, ..., lhs, rhs, data = NULL)
```

Arguments

fml	A formula containing macros variables. Each macro variable must start with two dots. The macro variables can be set globally using setFixest_fml, or can be defined in .... Special macros of the form ..("regex") can be used to fetch, through a regular expression, variables directly in a character vector (or in column names) given in the argument data. See examples.
...	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 or a character vector, it is the definition of the macro variable. Example of a valid call: setFixest_fml(..ctrl = ~ var1 + var2). In the function xpd, the default macro variables are taken from getFixest_fml, any variable in ... will replace these values.
lhs	If present then a formula will be constructed with lhs as the full left-hand-side. The value of lhs can be a one-sided formula, a call, or a character vector. Note that the macro variables wont be applied. You can use it in combination with the argument rhs. Note that if fml is not missing, its LHS will be replaced by lhs.
rhs	If present, then a formula will be constructed with rhs as the full right-hand-side. The value of rhs can be a one-sided formula, a call, or a character vector. Note that the macro variables wont be applied. You can use it in combination with the argument lhs. Note that if fml is not missing, its RHS will be replaced by rhs.
data	Either a character vector or a data.frame. This argument will only be used if a macro of the type ..("regex") is used in the formula of the argument fml. If so, any variable name from data that matches the regular expression will be added to the formula.

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 multi-part formulas if you wish but then beware of the order of 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

It returns a formula where all macros have been expanded.

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")

#
# You can use xpd for stepwise estimations
#

# we want to look at the effect of x1 on y
# controlling for different variables

base = iris
names(base) = c("y", "x1", "x2", "x3", "species")

# We first create a matrix with all possible combinations of variables
my_args = lapply(names(base)[-(1:2)], function(x) c("", x))
(all_combs = as.matrix(do.call("expand.grid", my_args)))

res_all = list()
for(i in 1:nrow(all_combs)){
  res_all[[i]] = feols(xpd(y ~ x1 + ..v, ..v = all_combs[i, ]), base)
}
```

```
etable(res_all)
coefplot(res_all, group = list(Species = "^^species"))

#
# You can use macros to grep variables in your data set
#

# Example 1: setting a macro variable globally

data(longley)
setFixest_fml(..many_vars = grep("GNP|ployed", names(longley), value = TRUE))
feols(Armed.Forces ~ Population + ..many_vars, longley)

# Example 2: using ..("regex") to grep the variables "live"

feols(Armed.Forces ~ Population + ..("GNP|ployed"), longley)

# Example 3: same as Ex.2 but without using a fixest estimation

# Here we need to use xpd():
lm(xpd(Armed.Forces ~ Population + ..("GNP|ployed"), data = longley), longley)

#
# You can also put numbers in macros
#

res_all = list()
for(p in 1:3){
  res_all[[p]] = feols(xpd(Ozone ~ Wind + poly(Temp, ..p), ..p = p), airquality)
}

etable(res_all)

#
# lhs and rhs arguments
#

# to create a one sided formula from a character vector
vars = letters[1:5]
xpd(rhs = vars)

# Alternatively, to replace the RHS
xpd(y ~ 1, rhs = vars)

# To create a two sided formula
xpd(lhs = "y", rhs = vars)
```

`[.fixest_multi` *Subset a fixest_multi object*

Description

Subset a fixest_multi object using different keys.

Usage

```
## S3 method for class 'fixest_multi'
x[i, sample, lhs, rhs, fixef, iv, I, reorder = TRUE, drop = TRUE]
```

Arguments

x	A fixest_multi object, obtained from a fixest estimation leading to multiple results.
i	An integer vector. Represents the estimations to extract.
sample	An integer vector, a logical scalar, or a character vector. It represents the sample identifiers for which the results should be extracted. Only valid when the fixest estimation was a split sample. You can use .N to refer to the last element. If logical, all elements are selected in both cases, but FALSE leads sample to become the rightmost key (just try it out).
lhs	An integer vector, a logical scalar, or a character vector. It represents the left-hand-sides identifiers for which the results should be extracted. Only valid when the fixest estimation contained multiple left-hand-sides. You can use .N to refer to the last element. If logical, all elements are selected in both cases, but FALSE leads lhs to become the rightmost key (just try it out).
rhs	An integer vector or a logical scalar. It represents the right-hand-sides identifiers for which the results should be extracted. Only valid when the fixest estimation contained multiple right-hand-sides. You can use .N to refer to the last element. If logical, all elements are selected in both cases, but FALSE leads rhs to become the rightmost key (just try it out).
fixef	An integer vector or a logical scalar. It represents the fixed-effects identifiers for which the results should be extracted. Only valid when the fixest estimation contained fixed-effects in a stepwise fashion. You can use .N to refer to the last element. If logical, all elements are selected in both cases, but FALSE leads fixef to become the rightmost key (just try it out).
iv	An integer vector or a logical scalar. It represent the stages of the IV. Note that the length can be greater than 2 when there are multiple endogenous regressors (the first stage corresponding to multiple estimations). Note that the order of the stages depends on the stage argument from summary.fixest. If logical, all elements are selected in both cases, but FALSE leads iv to become the rightmost key (just try it out).
I	An integer vector. Represents the root element to extract.
reorder	Logical, default is TRUE. Indicates whether reordering of the results should be performed depending on the user input.
drop	Logical, default is TRUE. If the result contains only one estimation, then if drop = TRUE it will be transformed into a fixest object (instead of fixest_multi).

Details

The order with we we use the keys matter. Every time a key `sample`, `lhs`, `rhs`, `fixef` or `iv` is used, a reordering is performed to consider the leftmost-side key to be the new root.

Use logical keys to easily reorder. For example, say the object `res` contains a multiple estimation with multiple left-hand-sides, right-hand-sides and fixed-effects. By default the results are ordered as follows: `lhs`, `fixef`, `rhs`. If you use `res[lhs = FALSE]`, then the new order is: `fixef`, `rhs`, `lhs`. With `res[rhs = TRUE, lhs = FALSE]` it becomes: `rhs`, `fixef`, `lhs`. In both cases you keep all estimations.

Value

It returns a `fixest_multi` object. If there is only one estimation left in the object, then the result is simplified into a `fixest` object.

See Also

The main fixest estimation functions: `feols`, `fepois`, `fenegbin`, `feglm`, `feNmlm`. Tools for mutliple fixest estimations: `summary.fixest_multi`, `print.fixest_multi`, `as.list.fixest_multi`, `sub-sub-.fixest_multi`, `sub-.fixest_multi`, `cash-.fixest_multi`.

Examples

```
# Estimation with multiple samples/LHS/RHS
aq = airquality[airquality$Month %in% 5:6, ]
est_split = feols(c(Ozone, Solar.R) ~ sw(poly(Wind, 2), poly(Temp, 2)),
                  aq, split = ~ Month)

# By default: sample is the root
etable(est_split)

# Let's reorder, by considering lhs the root
etable(est_split[lhs = 1:.N])

# Selecting only one LHS and RHS
etable(est_split[lhs = "Ozone", rhs = 1])

# Taking the first root (here sample = 5)
etable(est_split[I = 1])

# The first and last estimations
etable(est_split[i = c(1, .N)])
```

`[.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)])
}
```

[[.fixest_multi *Extracts one element from a* fixest_multi *object*

Description

Extracts single elements from multiple `fixest` estimations.

Usage

```
## S3 method for class 'fixest_multi'
x[[i]]
```

Arguments

x	A `fixest_multi` object, obtained from a `fixest` estimation leading to multiple results.
i	An integer scalar. The identifier of the estimation to extract.

Value

A `fixest` object is returned.

See Also

The main fixest estimation functions: `feols`, `fepois`, `fenegbin`, `feglm`, `feNmlm`. Tools for mutliple fixest estimations: `summary.fixest_multi`, `print.fixest_multi`, `as.list.fixest_multi`, `sub-sub-.fixest_multi`, `sub-.fixest_multi`, `cash-.fixest_multi`.

Examples

```
base = iris
names(base) = c("y", "x1", "x2", "x3", "species")

# Multiple estimation
res = feols(y ~ csw(x1, x2, x3), base, split = ~species)

# The first estimation
res[[1]]

# The second one, etc
res[[2]]
```

$.fixest_multi *Extracts the root of a fixest_multi object*

Description

Extracts an element at the root of a fixest_multi object.

Usage

```
## S3 method for class 'fixest_multi'
x$name
```

Arguments

x	A `fixest_multi` object, obtained from a `fixest` estimation leading to multiple results.
name	The name of the root element to select.

Value

It either returns a `fixest_multi` object or a `fixest` object it there is only one estimation associated to the root element.

See Also

The main fixest estimation functions: feols, fepois, fenegbin, feglm, feNmlm. Tools for mutliple fixest estimations: summary.fixest_multi, print.fixest_multi, as.list.fixest_multi, sub-sub-.fixest_multi, sub-.fixest_multi, cash-.fixest_multi.

Examples

```
base = iris
names(base) = c("y", "x1", "x2", "x3", "species")

# Multiple estimation
res = feols(y ~ csw(x1, x2, x3), base, split = ~species)

# Let's the results for the setosa species
res$setosa

# now for versicolor
etable(res$versicolor)
```

---

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.
- Multiple estimationsYou can perform multiple estimations at once with the stepwise functions. It's then very easy to manipulate multiple results with the associated methods. See an introduction in the dedicated vignette: Multiple estimations
- 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