lm function

July 9, 2025

OLS is by far the most important estimation method used in econometrics. It is implemented in **R** by the `lm` function. A deep understanding of the way this function is implemented, its arguments, its results and how to use them is particularly useful as other estimation functions (for example `glm` for generalized linear models) have a lot in common with `lm`. Moreover, if one is willing to implement its own estimation function, it is advisable to use at least some of the elements of `lm`.

For illustration, we use the `random_group` data set of the **micsr** package that analyzes the effect of a training on wage.

```
library(micsr)
random_group |> head()
```

```
  female age child migrant single temp ten          edu         fsize samplew
1      1  45     3       0      0    0  23 Intermediate      up to 50   0.730
2      0  39     2       0      0    0 251 Intermediate more than 200   0.968
3      1  52     0       0      0    1  18 Intermediate      up to 50   1.282
4      1  39     1       0      0    0  18          Low      up to 50   0.692
5      0  39     0       0      1    0  68         High more than 200   1.352
6      0  40     3       0      0    0  26 Intermediate     50 to 200   0.818
       wage group
1 31.468530     2
2 32.051276    -2
3  8.571425     1
4 14.423070     1
5 37.393167     1
6 31.882587    -1
```

The response is hourly wage `wage` and the main covariate is a dummy for training. The `group` variable contains integers from -2 to 3, with negative values for treated individuals and positive values for untreated individuals. We first create a dummy for treatment, called `treated`:

```r
random_group <- transform(random_group, treated = ifelse(group < 0, 1, 0))
```

A basic model

We then estimate the model, using as control the number of children `child` and the `age`.

```r
mod0 <- lm(log(wage) ~ treated + age + child, data = random_group)
```

The result is a an object of class `lm`, with the 12 following elements:

```r
names(mod0)
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"         
 [5] "fitted.values" "assign"        "qr"            "df.residual"  
 [9] "xlevels"       "call"          "terms"         "model"        
```

We won't say much about `"qr"` and `"effects"` which are technical elements of the computation of least squares. `"coefficients"`, `"residuals"` and `"fitted.values"` don't deserve further comments. The length of `"residuals"` and `"fitted.values"` is N, the one of `"coefficients"` is $K+1$ (if there is an intercept). `"rank"` is the rank of the model matrix. If the matrix has full rank, the rank equals the number of columns of the model matrix $(K+1)$, which is also the number of fitted coefficients.[1]

```r
mod0$rank
## [1] 4
length(mod0$coefficients)
## [1] 4
```

`"df.residual"` is then $N-K-1$. `"terms"` is a term object, which is a kind of representation of the formula of the model:

```r
mod0$terms
```

```
log(wage) ~ treated + age + child
attr(,"variables")
list(log(wage), treated, age, child)
attr(,"factors")
          treated age child
```

[1]Actually the number of not `NA` coefficients.

```
log(wage)       0   0     0
treated         1   0     0
age             0   1     0
child           0   0     1
attr(,"term.labels")
[1] "treated" "age"     "child"
attr(,"order")
[1] 1 1 1
attr(,"intercept")
[1] 1
attr(,"response")
[1] 1
attr(,".Environment")
<environment: R_GlobalEnv>
attr(,"predvars")
list(log(wage), treated, age, child)
attr(,"dataClasses")
log(wage)   treated       age     child
"numeric" "numeric" "numeric" "numeric"
```

`"xlevels"` is for this model a useless list of length 0. We'll see latter the usefulness of this element. `assign` is a numeric vector indicating the relation between the variables and the coefficients:

```
mod0$assign
## [1] 0 1 2 3
```

Once again, in our simple case, it is useless: the intercept is at position 0, the first covariate at position 1 and so on. All the elements of the model can be extracted using the `$` operator, for example `mod0$coefficients` for the coefficients. A safer method to extract elements of the model is to use generic functions with homonym names:

```
coef(mod0)
## (Intercept)     treated         age       child
##  2.69603688  0.19107776  0.01496236  0.01003474
df.residual(mod0)
## [1] 2162
resid(mod0) |> head()
##           1           2           3           4           5           6
##  0.04954051 -0.02337931 -1.32564580 -0.62077481  0.34191893 -0.05365342
fitted(mod0) |> head()
##        1        2        3        4        5        6
```

```
## 3.399447 3.490716 3.474080 3.289604 3.279569 3.515713
```

```
terms(mod0)
```

Several other generics enables to extract information of the model that are not elements of the results, for example the covariance matrix of the estimates, the residual standard error and the number of observations:

```
vcov(mod0)
##               (Intercept)       treated           age         child
## (Intercept)  2.272281e-03 -3.249881e-04 -5.020516e-05 -2.766710e-05
## treated     -3.249881e-04  5.234244e-04  1.867944e-06  5.201122e-06
## age         -5.020516e-05  1.867944e-06  1.314397e-06 -1.994140e-06
## child       -2.766710e-05  5.201122e-06 -1.994140e-06  1.015276e-04
sigma(mod0)
## [1] 0.5298291
nobs(mod0)
## [1] 2166
```

The next two sections are devoted to the last two elements of the `lm` object, `"model"` and `"call"`.

Model frame

The element `"model"` of the `lm` object is the model frame. It's a special data frame:

```
head(mod0$model, 3)
```

```
  log(wage) treated age child
1  3.448988       0  45     3
2  3.467337       1  39     2
3  2.148434       0  52     0
```

Instead of using the `$` operator, the `model.frame` function can be used to extract the model frame:

```
model.frame(mod0)
```

For this basic model, it looks like a subset of columns of the original data frame, but note that the first column contains the response which is `log(wage)` and not `wage`. A model frame has several attributes:

```r
names(attributes(mod0$model))
## [1] "names"     "terms"     "row.names" "class"
```

The only interesting attribute is `"terms"`, the two other being the rows and columns' names (like for ordinary data frames). From the model frame, the elements of the model can be extracted. For this basic model, there are two elements, the matrix of covariates X and the vector of response y. To extract the model matrix, we use `model.matrix` with two arguments, the model terms and the model frame:

```r
model.matrix(terms(mod0), model.frame(mod0)) |> head(3)
```

```
  (Intercept) treated age child
1           1       0  45     3
2           1       1  39     2
3           1       0  52     0
```

More simply, the same model matrix can be extracted using the fitted object as the unique argument:

```r
model.matrix(mod0)
```

The model matrix looks like the model frame, except that the response has been removed and a column of ones (called `"(Intercept)"`) has been added. Actually there is a fundamental difference between the model frame and the model matrix, even in this very simple example. A model frame is a data frame, which is a list of vectors of the same length. Because all the vectors have the same length, it has a tabular representation and therefore looks like a matrix. A model matrix is a matrix, which is for **R** a vector with an attribute indicating the dimension, ie., the number of rows and of columns.

```r
dim(model.matrix(mod0))
```

```
[1] 2166    4
```

Being a vector, a matrix contains only elements of the same mode, for example numeric. The model response is extracted using `model.response` with the model frame as unique argument:

```r
mod0 |> model.frame() |> model.response() |> head()
##        1        2        3        4        5        6
## 3.448988 3.467337 2.148434 2.668829 3.621488 3.462060
```

Call and update

"call" is an object of class "call", that corresponds to the call to the function that resulted in the mod0 object:

```
class(mod0$call)
## [1] "call"
mod0$call
## lm(formula = log(wage) ~ treated + age + child, data = random_group)
```

Note that all the arguments are named, although we didn't name the first argument (formula) while calling "lm". The call is very useful as it enables to update the model. The default update function is: update(object, formula., ..., evaluate = TRUE). Its first argument is a fitted model and the second one is a formula. For example, to update mod0 by adding the female, single, migrant and temp covariates (dummies for females, singles, migrants and temporary jobs), instead of rewriting the whole command:

```
mod1 <- lm(log(wage) ~ treated + age + child + female + single + migrant +
               temp, data = random_group)
```

we can update mod0 with a new formula:

```
mod1 <- mod0 |> update(log(wage) ~ treated + age + child +
                           female + single + migrant + temp)
```

or even more simply, we can use dots in the formula, which means "as before" and enables to add or to remove variables from the initial formula:

```
mod1 <- update(mod0, . ~ . + female + single + migrant + temp)
```

We have seen previously that the default update function has a ... argument. This means that we can use in a call to update an argument of lm which is then passed to lm while updating the model. As an example, lm has a subset argument which is a logical expression that select a subset of the sample for which the estimation has to be performed. For example, starting from mod0, if one wishes to add the four dummies as previously and also to select the individuals aged at least 20, we can use:

```
lm(log(wage) ~ treated + age + child + female + single + migrant + temp,
          data = random_group, subset = age >= 20)
```

Or much more simply:

```r
update(mod0, . ~ . + female + single + migrant + temp, subset = age >= 20)
```

Missing values

When values of some variables used in the regression are missing for some observations, one has to indicate how to deal with this problem. The default behavior corresponds to the `na.action` element of `options()` which is a list containing options:

```r
options()$na.action
## [1] "na.omit"
```

`na.omit` means remove from the model frame all the lines for which there is at least a missing value. There are a couple of other possible values, one of them being `na.fail` which stops the estimation and returns an error. `lm` has a `na.action` argument where the desired action can be indicated. For example, if we update the model with `na.action` set to `"na.fail"`:

```r
mod1 |> update(na.action = "na.fail")
```

we get exactly the same results as there are no missing values for the subset of variables we used. Now consider adding the `ten` covariate which is the tenure in month and has some missing values. Setting as previously `na.action = "na.fail"`, we'll then get an error message:

```r
mod1 |> update(. ~ . + ten, na.action = "na.fail")
```

Using the default `"na.omit"` value, we get:

```r
mod2 <- mod1 |> update(. ~ . + ten)
names(mod2)
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "na.action"     "xlevels"       "call"          "terms"
[13] "model"
```

and there is now a 13^{th} element named `na.action`:

```r
unname(mod2$na.action)
```

```
 [1]  61  66  87 126 131 149 182 213 291 372 390 448 566 635 655
```

```
[16]  717  783  852  866  877  910 1008 1024 1027 1079 1252 1261 1319
attr(,"class")
[1] "omit"
```

This is an object of class `"omit"` containing a vector of integers indicating the positions of the lines that have been removed because of missing values. We used `unname` because actually this is a named vector and, as in our case, when there is no row names, the “name” is the number (so that the name of the first element of the vector, 61, is “61”). This object is also returned as an attribute of the model frame:

```
mod2$model |> attributes() |> names()
## [1] "names"     "terms"     "row.names" "class"     "na.action"
```

Factors

Now we introduce two further covariates `fsize` (firm size) and `edu` (education). These covariates are **factors** (categorical variables) and the modalities are called **levels**. The modalities can be extracted using the `levels` function:

```
levels(random_group$fsize)
## [1] "up to 50"      "50 to 200"     "more than 200"
levels(random_group$edu)
## [1] "Low"          "Intermediate" "High"
```

We introduce these two factors in the regression and we also introduce a quadratic term for `age`. This is done using the `poly` function inside the formula:

```
mod3 <- lm(log(wage) ~ treated + poly(age, 2) + child + fsize + edu +
               female + single + migrant + temp + ten,
           data = random_group)
```

There is now a supplementary element called `"contrasts"`. But first consider the already existing `"assign"` and `"xlevels"` elements:

```
mod3$assign
```

```
 [1]  0  1  2  2  3  4  4  5  5  6  7  8  9 10
```

`"assign"` is now a vector of length 14. The first element is 0 (for the intercept), the 13 remaining indicate the link between the position of the coefficients and the position of the covariate in the formula:

```
names(coef(mod3))
```

```
 [1] "(Intercept)"          "treated"              "poly(age, 2)1"
 [4] "poly(age, 2)2"        "child"                "fsize50 to 200"
 [7] "fsizemore than 200" "eduIntermediate"    "eduHigh"
[10] "female"               "single"               "migrant"
[13] "temp"                 "ten"
```

for example, the 3rd and 4th values of `"assign"` are 2, which is the position of the `"age"` covariate, with now two coefficients. The 8th and 9th values of `"assign"` are 5, they correspond to the two dummies introduced in the regression for the 5th covariate in the formula which is `edu`. Let's now have a look to the `"xlevels"` element which used to be in the previous example a list of length 0:

```
mod3$xlevels
```

```
$fsize
[1] "up to 50"       "50 to 200"      "more than 200"

$edu
[1] "Low"           "Intermediate" "High"
```

This is now a named list containing character vectors indicating the different levels of the factors. Note that for a factor with J levels, only $J-1$ dummies are introduced in the regression, the one corresponding to the first level being omitted. The `contrasts` element is:

```
mod3$contrasts
```

```
$fsize
[1] "contr.treatment"

$edu
[1] "contr.treatment"
```

It's a named list indicating how the contrasts are computed from the factors. The default value is `"contr.treatment"` and consist, as we seen previously, to create $J-1$ dummies for all the levels of the factor but the first. Other values are possible and can be set individually for every factor using the `contrasts` argument of `lm`:

```
mod4 <- update(mod3, contrasts = list(edu = "contr.sum",
                                      fsize = "contr.helmert"))
mod4$contrasts
```

```
$fsize
[1] "contr.helmert"

$edu
[1] "contr.sum"
```

The model frame now looks like:

```
head(mod4$model, 3)
```

```
  log(wage) treated poly(age, 2).1 poly(age, 2).2 child         fsize
1  3.448988       0   0.0126077067  -0.0120734877     3      up to 50
2  3.467337       1  -0.0001432471  -0.0180851447     2 more than 200
3  2.148434       0   0.0274838194   0.0110421946     0      up to 50
           edu female single migrant temp ten
1 Intermediate      1      0       0    0  23
2 Intermediate      0      0       0    0 251
3 Intermediate      1      0       0    1  18
```

The `age` column has been replaced by a $N \times 2$ matrix containing the two terms of the polynomial. The two factors are in their own column. The model matrix X is now:

```
head(model.matrix(mod3), 3)
```

```
  (Intercept) treated poly(age, 2)1 poly(age, 2)2 child fsize50 to 200
1           1       0  0.0126077067   -0.01207349     3              0
2           1       1 -0.0001432471   -0.01808514     2              0
3           1       0  0.0274838194    0.01104219     0              0
  fsizemore than 200 eduIntermediate eduHigh female single migrant temp ten
1                  0               1       0      1      0       0    0  23
2                  1               1       0      0      0       0    0 251
3                  0               1       0      1      0       0    1  18
```

```r
head(model.matrix(mod4), 3)
```

```
  (Intercept) treated poly(age, 2)1 poly(age, 2)2 child fsize1 fsize2 edu1 edu2
1           1       0  0.0126077067   -0.01207349     3     -1     -1    0    1
2           1       1 -0.0001432471   -0.01808514     2      0      2    0    1
3           1       0  0.0274838194    0.01104219     0     -1     -1    0    1
  female single migrant temp ten
1      1      0       0    0  23
2      0      0       0    0 251
3      1      0       0    1  18
```

The fundamental difference between the model frame (a list) and the model matrix (a vector with a dimension) is even clearer than in the previous simple example. The model frame contains a column called `"edu"` which is a factor. In the model matrix, this column is replaced by $J - 1 = 2$ columns that contain numeric and we can see that the numerical coding of the variable depends on the contrast used.

Weights and offset

Instead of minimizing the sum of squares residuals $\sum_n (y_n - \beta^\top x_n)^2$, **weights** w_n can be used so that the objective function becomes $\sum_n w_n(y_n - \beta^\top x_n)^2$, leading to the weighted least squares estimator. Weights can be indicating in `lm` using the `weights` argument, which is the unquoted name of the column of the data frame that contains the weights. For the `random_group` data set, the variable that contains the sample weights is `samplew`:

```r
mod4 <- update(mod3, weights = samplew)
```

The `lm` object now has a 15^{th} element called `"weights"` which contains the vector of weights of length N. The model frame is now:

```r
model.frame(mod4) |> head(3)
```

```
  log(wage) treated poly(age, 2).1 poly(age, 2).2 child          fsize
1  3.448988       0   0.0126077067  -0.0120734877     3      up to 50
2  3.467337       1  -0.0001432471  -0.0180851447     2 more than 200
3  2.148434       0   0.0274838194   0.0110421946     0      up to 50
           edu female single migrant temp ten (weights)
1 Intermediate      1      0       0    0  23     0.730
2 Intermediate      0      0       0    0 251     0.968
3 Intermediate      1      0       0    1  18     1.282
```

Note the last column called "(weights)" that contains the weights. The weights can be extracted from the model frame using model.weights:

```
model.weights(model.frame(mod4)) |> head(3)
```

```
[1] 0.730 0.968 1.282
```

An **offset** is a variable added to the regression equation, but with no associated coefficients (or with a coefficient set to 1). For example, in the equation: $y_n = \alpha + \beta_1 x_{n1} + \beta_2 x_{n2} + x_{n3} + \epsilon_n$, x_{n3} is an offset. Note also that the same equation can be rewritten as: $y_n - x_{n3} = \alpha + \beta_1 x_{n1} + \beta_2 x_{n2} + \epsilon_n$ and that in this case, the new response is $y_n - x_{n3}$ and there is no offset anymore. We have seen in mod3 that the coefficients for the "Intermediate" and "High" education values are respectively 0.23 and 0.50, which means approximately 25 and 50% more wage compared to the reference level of this factor which is "Low". Coercing the three levels of the factor to $z = 1, 2, 3$ (which is actually the internal representation of the factor) and denoting $v = 3/4 + 1/4z$, we get the values of 1, 1.25 and 1.50. Therefore, we could get approximately the same results as mod3 by removing "edu" and adding v as an offset. There are two equivalent ways to introduce an offset: in the formula, using offset(v) or by setting the offset argument of lm to v:

```
random_group <- transform(random_group, v = 3/4 + 1/4 * as.numeric(edu))
mod5 <- update(mod4, . ~ . + offset(v) - edu)
mod6 <- update(mod4, . ~ . - edu, offset = v)
```

The two fitted models are identical, they contain a supplementary element called "offset". The offset can be extract from the model frame using model.offset:

```
model.offset(model.frame(mod5)) |> head(3)
```

```
[1] 1.25 1.25 1.25
```

The internal

To conclude the presentation of the lm function, we'll describe the internal of the function. We first create a simple mylm function with the same argument as lm, but which returns only the call, obtained using the function match.call:

```
mylm <- function (formula, data, subset, weights, na.action, method = "qr",
                  model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
                  singular.ok = TRUE, contrasts = NULL, offset, ...){
```

```
    mf <- match.call()
    mf
}
```

We then use this function with our most advanced example:

```
mf <- mylm(log(wage) ~ treated + poly(age, 2) + child + fsize + female +
               single + migrant + temp + ten,
           random_group,
           subset = age > 20,
           weights = samplew,
           contrasts = list(fsize = "contr.helmert"),
           offset = v
           )
mf
```

```
mylm(formula = log(wage) ~ treated + poly(age, 2) + child + fsize +
    female + single + migrant + temp + ten, data = random_group,
    subset = age > 20, weights = samplew, contrasts = list(fsize = "contr.helmert"),
    offset = v)
```

```
cl <- mf
```

We saved a copy of the result as we'll need it latter on. The next lines of `lm` work on the call (the `mf` object returned by our `mylm` function). The call being a list, the idea is first to keep only the arguments in the call that are useful to compute the model frame. This is done using the `match` function on the names of the call:

```
names(mf)
```

```
[1] ""           "formula"    "data"       "subset"     "weights"    "contrasts"
[7] "offset"
```

```
m <- match(c("formula", "data", "subset", "weights", "na.action",
             "offset"), names(mf), 0L)
m
```

```
[1] 2 3 4 5 0 7
```

`m` is an integer vector indicating the position of the arguments we wish to keep in the call; note that the position of `na.action` is 0 because we didn't use this argument in our call to `mylm`. The first element of `mf` is unnamed, this is the name of the function (`mylm`). We then extract from `mf` its first element and those given by vector `m`:

```
mf <- mf[c(1L, m)]
mf
```

```
mylm(formula = log(wage) ~ treated + poly(age, 2) + child + fsize +
    female + single + migrant + temp + ten, data = random_group,
    subset = age > 20, weights = samplew, offset = v)
```

Then, we change the first argument (the name of the function) from `mylm` to `model.frame`, using the `quote` function:

```
mf[[1L]] <- quote(stats::model.frame)
mf
```

```
stats::model.frame(formula = log(wage) ~ treated + poly(age,
    2) + child + fsize + female + single + migrant + temp + ten,
    data = random_group, subset = age > 20, weights = samplew,
    offset = v)
```

The call is then evaluated using the `eval` function and the result is the model frame:

```
mf <- eval(mf)
head(mf, 3)
```

```
  log(wage) treated poly(age, 2).1 poly(age, 2).2 child          fsize female
1  3.448988       0   0.0126077067  -0.0120734877     3       up to 50      1
2  3.467337       1  -0.0001432471  -0.0180851447     2 more than 200      0
3  2.148434       0   0.0274838194   0.0110421946     0       up to 50      1
  single migrant temp ten (weights) (offset)
1      0       0    0  23     0.730     1.25
2      0       0    0 251     0.968     1.25
3      0       0    1  18     1.282     1.25
```

```
names(attributes(mf))
```

```
[1] "names"     "terms"     "row.names" "class"     "na.action"
```

We then extract the terms, and the different components of the model:

```
mt <- attr(mf, "terms")
y <- model.response(mf)
w <- model.weights(mf)
offset <- model.offset(mf)
x <- model.matrix(mt, mf, list(fsize = "contr.helmert"))
```

Note that the model matrix is constructed using as third argument the `contrasts` argument of `lm`. The estimation is then performed by `lm.wfit` (if there were no weights, `lm.fit` would have been used), that take as arguments the components of the model previously extracted:

```
z <- lm.wfit(x, y, w, offset = offset)
names(z)
```

```
[1] "coefficients"  "residuals"     "fitted.values" "effects"
[5] "weights"       "rank"          "assign"        "qr"
[9] "df.residual"
```

We can see that the resulting object contains a subset of the elements returned by `lm`. We then assign to the object the `"lm"` class:

```
class(z) <- "lm"
```

And we add to `z` the elements that are not returned by `lm.wfit`:

```
z$na.action <- attr(mf, "na.action")
z$offset <- offset
z$contrasts <- attr(x, "contrasts")
z$xlevels <- .getXlevels(mt, mf)
z$call <- cl
z$terms <- mt
z$model <- mf
```

`na.action` is an attribute of the model frame, `contrasts` an attribute of the model matrix and `xlevels` is obtained using the `.getXlevels` function with the terms and the model frame as arguments. We then get our `lm` object that contains the fitted model:

```
z
```

```
Call:
mylm(formula = log(wage) ~ treated + poly(age, 2) + child + fsize +
    female + single + migrant + temp + ten, data = random_group,
    subset = age > 20, weights = samplew, contrasts = list(fsize = "contr.helmert"),
    offset = v)

Coefficients:
  (Intercept)        treated  poly(age, 2)1  poly(age, 2)2          child
    2.1459138      0.0912960      3.2498758     -1.8725685      0.0053146
       fsize1         fsize2         female         single        migrant
    0.0289841      0.0321877     -0.2337914     -0.0990943     -0.1255097
         temp            ten
    0.0019268      0.0004435
```

Finally, we include in our `mylm` function all the lines we just have described:

```r
mylm <- function (formula, data, subset, weights, na.action, method = "qr",
                  model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
                  singular.ok = TRUE, contrasts = NULL, offset, ...){
    mf <- match.call(expand.dots = FALSE)
    m <- match(c("formula", "data", "subset", "weights", "na.action",
                 "offset"), names(mf), 0L)
    mf <- mf[c(1L, m)]
    mf[[1L]] <- quote(stats::model.frame)
    mf <- eval(mf, parent.frame())
    mt <- attr(mf, "terms")
    y <- model.response(mf, "numeric")
    w <- as.vector(model.weights(mf))
    offset <- model.offset(mf)
    x <- model.matrix(mt, mf, contrasts)
    z <- lm.wfit(x, y, w, offset = offset)
    class(z) <- "lm"
    z$na.action <- attr(mf, "na.action")
    z$offset <- offset
    z$contrasts <- attr(x, "contrasts")
    z$xlevels <- .getXlevels(mt, mf)
    z$call <- cl
    z$terms <- mt
    z$model <- mf
    z
}
```

to get a simplified clone of `lm`:

```
mod6 <- mylm(log(wage) ~ treated + poly(age, 2) + child + fsize + female +
                 single + migrant + temp + ten,
             random_group,
             subset = age > 20,
             weights = samplew,
             contrasts = list(fsize = "contr.helmert"),
             offset = v)
```