

Miscellaneous tools

October 27, 2025

Select a subset of coefficients

Complex models may include a large set of coefficients. Consider for example the estimation of a probit model with instrumental variables. The exogenous covariates are denoted x_1 , the endogenous covariates e and the external instruments x_2 . The density that enters the log-likelihood function is:

$$f(y_n | d_n) = \Phi \left(q_n \frac{\theta^\top u_n}{\sigma} \right) = \Phi \left(q_n \frac{\gamma^\top z_n + \sigma_{\epsilon\nu}^\top \Sigma_\nu^{-1} (e_n - \Pi w_n)}{\sqrt{\sigma_\epsilon^2 - \sigma_{\epsilon\nu}^\top \Sigma_\nu^{-1} \sigma_{\epsilon\nu}}} \right)$$

where $z^\top = (x_1^\top, e^\top)$ are the covariates, $w^\top = (x_1^\top, x_2^\top)$ the instruments, and $e_n - \Pi x_n$ the error vector of the reduced form equation of the endogenous variable.

In this case, coefficients belong to four different groups:

- the coefficients associated with covariates (either exogenous or endogenous), which are the coefficients of main interest (γ),
- the coefficients associated with the errors of the reduced form equation ($\sigma_{\epsilon\nu}$),
- the coefficients of the reduced form equation (Π),
- the matrix of covariance of the errors of the structural equations (Σ_ν).

Actually, a Cholesky decomposition of $\Sigma_\nu^{-1} = C^\top C$ is used so that the second set of coefficients are $\rho^* = C\sigma_{\epsilon\nu}$ and the fourth set are the elements of C . The four sets are respectively denoted "covariates", "resid", "instruments" and "chol".

We use as an example the `federiv` data set where the response is a dummy for foreign exchange derivatives use. The model is fitted using `binomreg` with a two-part formula which defines the covariates and the instruments:

```
library(micsr)
bank <- binomreg(federiv ~ eqrat + optval + mktbk +
  perfor + dealdum | . - eqrat - optval +
  no_emp + no_subs + no_off,
  data = federiv, method = "ml")
```

The full set of coefficients is stored as a named vector in the `coefficients` element of `bank`:

```
names(bank$coefficients)
```

```
[1] "(Intercept)"          "mktbk"
[3] "perfor"               "dealdum"
[5] "eqrat"                "optval"
[7] "rho_eqrat"            "rho_optval"
[9] "instr_eqrat_(Intercept)" "instr_eqrat_mktbk"
[11] "instr_eqrat_perfor"   "instr_eqrat_dealdum"
[13] "instr_eqrat_no_emp"   "instr_eqrat_no_subs"
[15] "instr_eqrat_no_off"   "instr_optval_(Intercept)"
[17] "instr_optval_mktbk"   "instr_optval_perfor"
[19] "instr_optval_dealdum" "instr_optval_no_emp"
[21] "instr_optval_no_subs" "instr_optval_no_off"
[23] "eqrat|eqrat"          "optval|eqrat"
[25] "optval|optval"
```

Therefore, there are a total of 25 coefficients. Note that some of them are prefixed by "rho_", some other by "instr_" and some other are named as the concatenation of two series separated by "|". They form respectively the second, third, and fourth groups of parameters. This structure is stored in the `npar` element of `bank`:

```
bank$npar
```

```
  covariates      resid instruments      chol
        6          2          14          3
attr(,"default")
[1] "covariates" "resid"
```

This is a named vector, the names being the name of the groups of parameters and the values the number of parameters for each group, in their order of appearance in `bank$coefficients`. This vector has an attribute called "default" which indicates which group, by default, should be considered in the printing of the model. Therefore, while using `coef`, `vcov` or `summary` methods, by default only the two first groups will be considered:

```
coef(bank)
```

(Intercept)	mktbk	perfor	dealdum	eqrat
-3.0493980864	0.0007126479	4.6919607187	0.8644131095	20.7983728258
optval	rho_eqrat	rho_optval		
0.0887552265	-0.4478945968	-0.1407430396		

but the subset argument can be filled in order to indicate a specific set of parameters:

```
coef(bank, subset = c("chol", "resid"))
```

rho_eqrat	rho_optval	eqrat eqrat	optval eqrat	optval optval
-0.44789460	-0.14074304	56.80434859	-0.01092933	0.14746163

The default value of **subset** is NA and in this case the default set of parameters is considered. If **subset** is set to NULL or "all", all the parameters are returned

```
coef(bank, subset = NULL)
```

Once the groups of parameters have been define using **subset**, a more precise selection can be using either a vector of names of coefficients or a regular expression, using respectively the **coef** and the **grep** arguments. For example, to select only the coefficients of **mktbk** and **perfor**:

```
coef(bank, coef = c("mktbk", "perfor"))
```

mktbk	perfor
0.0007126479	4.6919607187

To select all the coefficients that contains "eqrat" or "perfor":

```
coef(bank, grep = "eqrat|perfor")
```

perfor	eqrat	rho_eqrat
4.6919607	20.7983728	-0.4478946

Note that only the parameters of the default subset of coefficients are considered.

```
coef(bank, grep = "eqrat|perfor", subset = c("instruments", "chol"))
```

eqrat_(Intercept)	eqrat_mktbk	eqrat_perfor	eqrat_dealdum
4.953602088	-0.001084146	-0.890152444	-0.379929494
eqrat_no_emp	eqrat_no_subs	eqrat_no_off	optval_perfor
-0.014485297	0.031015758	0.504809295	-3.615871902
eqrat eqrat	optval eqrat		
56.804348592	-0.010929330		

Finally, the `invert` argument (by default `FALSE`) can be used to return the coefficients that don't match regular expression. For example, to return all the coefficients of the reduced form equations of the instruments except the intercepts:

```
coef(bank, subset = "instruments", grep = "Intercept", invert = TRUE)
```

eqrat_mktbk	eqrat_perfor	eqrat_dealdum	eqrat_no_emp	eqrat_no_subs
-0.0010841460	-0.8901524443	-0.3799294937	-0.0144852974	0.0310157576
eqrat_no_off	optval_mktbk	optval_perfor	optval_dealdum	optval_no_emp
0.5048092952	0.0001569365	-3.6158719015	-0.0813282274	0.1153214463
optval_no_subs	optval_no_off			
-0.0032324833	-3.2591910753			

Covariance matrix and standard errors

The `vcov` method for `micsr` objects return the covariance matrix. A subset of coefficients can be selected using the same syntax as previously:

```
vcov(bank, subset = "resid")
```

	rho_eqrat	rho_optval
rho_eqrat	0.0021818615	-0.0005167884
rho_optval	-0.0005167884	0.0045537894

The `vcov` argument enables to use different flavors of covariance matrices:

- "info" uses the information matrix,
- "hessian" uses the hessian,
- "opg" uses the outer product of the gradient,

- "hc" uses the heteroscedastic consistent estimator.

All these estimators are not available for all the models. If the argument `vcov` is left empty, the default behavior is to choose "info" if it exists, then "hessian" if not and "opg" if the previous two estimators are not available. Using `vcov = "hc"` results in an internal call to the `sandwich::vcovHC` function:

```
vcov(bank, subset = "resid")
```

```

      rho_eqrat    rho_optval
rho_eqrat  0.0021818615 -0.0005167884
rho_optval -0.0005167884  0.0045537894

```

```
vcov(bank, subset = "resid", vcov = "hessian")
```

```

      rho_eqrat    rho_optval
rho_eqrat  0.0021818615 -0.0005167884
rho_optval -0.0005167884  0.0045537894

```

```
vcov(bank, subset = "resid", vcov = "opg")
```

```

      rho_eqrat    rho_optval
rho_eqrat  0.0025591467 -0.0001606321
rho_optval -0.0001606321  0.0030387065

```

```
vcov(bank, subset = "resid", vcov = "hc")
```

```

      rho_eqrat    rho_optval
rho_eqrat  0.1611708894 -0.0001254363
rho_optval -0.0001254363  0.0075335973

```

micsr provides the convenient `stder` function to extract the standards errors of the coefficients. It can be used with the same argument as the `vcov` method:

```
stder(bank, subset = "resid")
```

```

      rho_eqrat rho_optval
0.04671040 0.06748177

```

```
stder(bank, vcov = "hc", subset = "resid")
```

```
rho_eqrat rho_optval
0.4014609 0.0867963
```

Dummies

Categorical variables are stored with **R** using **factors** with a limited set of **levels**. For the purpose of estimation, using `model.matrix`, factors are translated into a set of contrasts. The most common contrast is **treatment** contrast, it consists on generating a set of dummy variables for every level, except the first. However, it is sometimes easier to work directly with the dummy variables, and the `micsr::dummy` enables to create simply such dummies. Consider for example the `charitable` data set that contains two factors, `education` and `religion`. `dummy` can be used with a data frame as first arguments and factors for which one wants to create dummies as further unnamed arguments:

```
charitable |> dummy( education, religion) |> head(2)
```

	donation	donparents	income	married	south	post_college	college	some_college
1	335	5210	21955.13	0	0	0	0	0
2	75	13225	22103.82	0	0	0	0	0

	high_school	other	jewish	protestant	catholic
1	0	1	0	0	0
2	1	0	0	1	0

Two further named boolean arguments (`FALSE` by default) are available:

- `keep` to indicate whether the factor should be kept in the resulting data frame,
- `ref` to indicate whether a dummy should be generated for the reference level.

```
charitable |> dummy(religion, keep = TRUE) |> head(2)
```

	donation	donparents	education	religion	income	married	south	other
1	335	5210	less_high_school	other	21955.13	0	0	1
2	75	13225	high_school	protestant	22103.82	0	0	0

	jewish	protestant	catholic
1	0	0	0
2	0	1	0

```
charitable |> dummy(religion, ref = TRUE) |> head(2)
```

	donation	donparents	education	income	married	south	other	jewish
1	335	5210	less_high_school	21955.13	0	0	1	0
2	75	13225	high_school	22103.82	0	0	0	0
	protestant	catholic	none					
1	0	0	0					
2	1	0	0					

Short summary

R is not a verbose language as every function returns an object that can be stored by the user. Then, special `print` methods (or `print.summary` methods) are used to print some of the result. However, the output is quite heavy and take too much place to be used in a document. For this reason, `micsr` provides a `gaze` function with several methods to print the results of a model or of a test. Consider for example a t-test for charitable donations with two subsamples defined by the `married` dummy variable:

```
t.test(donation ~ married, charitable)
```

Welch Two Sample t-test

```
data: donation by married
t = -12.903, df = 1750, p-value < 2.2e-16
alternative hypothesis: true difference in means between group 0 and group 1 is not equal to
95 percent confidence interval:
 -1629.240 -1199.302
sample estimates:
mean in group 0 mean in group 1
    334.9068      1749.1776
```

The result of `t.test` is an object of class `htest` and the `print` method results in 12 lines, including 2 blank lines. `gaze` returns a one line result, with the essential informations:

```
t.test(donation ~ married, charitable) |> gaze()
## diff = -1414.271 (109.604), t = -12.903, df: 1750, pval = 0.000
```

For a fitted model, `gaze` returns the table of coefficients for all the fitted parameters and the value of the log-likelihood:

```
tobit1(log(donation / 25) ~ married, charitable) |> gaze()
```

	Estimate	Std. Error	z-value	Pr(> z)
(Intercept)	0.14939	0.09992	1.495	0.135
married	2.32903	0.11887	19.594	<2e-16
sigma	2.56014	0.05029	50.905	<2e-16

```
'log Lik.' -4391.015 (df=3)
```