nlsur — Estimation of nonlinear system of equations

Description

`nlsur` fits a system of nonlinear equations by feasible generalized nonlinear least squares (FGNLS). With the interactive version of the command, you enter the system of equations on the command line or in the dialog box by using *substitutable expressions*. If you have a system that you use regularly, you can write a *substitutable expression program* and use the second syntax to avoid having to reenter the system every time. The function evaluator program version gives you the most flexibility in exchange for increased complexity; with this version, your program is given a vector of parameters and a variable list, and your program computes the system of equations.

When you write a substitutable expression program or a function evaluator program, the first five letters of the name must be `nlsur`. *sexp_prog* and *func_prog* refer to the name of the program without the first five letters. For example, if you wrote a function evaluator program named `nlsurregss`, you would type `nlsur regss @` ... to estimate the parameters.

Quick start

Two-parameter exponential model regressing `y1` on `x` using the default FGNLS estimator

```
nlsur (y1 = {b1}*{b2}^x)
```

Add the `variables()` option to allow for missing values of `y1` and `x`

```
nlsur (y1 = {b1}*{b2}^x), variables(y1 x)
```

Two-equation, two-parameter exponential model

```
nlsur (y1 = {b1}*{b2}^x) (y2 = {g1}*{g2}^x), variables(y1 y2 x)
```

Same as above, but use the iterative FGNLS estimator

```
nlsur (y1 = {b1}*{b2}^x) (y2 = {g1}*{g2}^x), variables(y1 y2 x) ifgnls
```

Specify starting values for parameters `b1` and `g2`

```
nlsur (y1 = {b1=.5}*{b2}^x) (y2 = {g1}*{g2=1}^x), variables(y1 y2 x)
```

Same as above

```
nlsur (y1 = {b1}*{b2}^x) (y2 = {g1}*{g2}^x), variables(y1 y2 x)  ///
   initial(b1 .5 g2 1)
```

Same as above, but specify starting values using the matrix `i`

```
matrix i = (.5,0,0,1)
nlsur (y1 = {b1}*{b2}^x) (y2 = {g1}*{g2}^x), variables(y1 y2 x)  ///
   initial(i)
```

Menu

Statistics > Linear models and related > Multiple-equation models > Nonlinear seemingly unrelated regression

Syntax

Interactive version

```
nlsur (depvar_1 = <sexp_1>) (depvar_2 = <sexp_2>) ... [if] [in] [weight]
    [, options]
```

Programmed substitutable expression version

```
nlsur sexp_prog : depvar_1 depvar_2 ... [varlist] [if] [in] [weight] [, options]
```

Function evaluator program version

```
nlsur func_prog @ depvar_1 depvar_2 ... [varlist] [if] [in] [weight] ,
    nequations(#) { parameters(namelist) | nparameters(#) } [options]
```

where

depvar_j is the dependent variable for equation *j*;

<sexp>_j is the substitutable expression for equation *j*;

sexp_prog is a substitutable expression program; and

func_prog is a function evaluator program.

options	Description
Model	
`fgnls`	use two-step FGNLS estimator; the default
`ifgnls`	use iterative FGNLS estimator
`nls`	use NLS estimator
`variables(`*varlist*`)`	variables in model
`initial(`*initial_values*`)`	initial values for parameters
* `nequations(#)`	number of equations in model (function evaluator program version only)
* `parameters(`*namelist*`)`	parameters in model (function evaluator program version only)
* `nparameters(#)`	number of parameters in model (function evaluator program version only)
sexp_options	options for substitutable expression program
func_options	options for function evaluator program
SE/Robust	
`vce(`*vcetype*`)`	*vcetype* may be `gnr`, `robust`, `cluster` *clustvar*, `bootstrap`, or `jackknife`
Reporting	
`level(#)`	set confidence level; default is `level(95)`
`title(`*string*`)`	display *string* as title above the table of parameter estimates
`title2(`*string*`)`	display *string* as subtitle
display_options	control columns and column formats and line width
Optimization	
optimization_options	control the optimization process; seldom used
`coeflegend`	display legend instead of statistics

*You must specify nequations(#) and one of parameters(*namelist*) or nparameters(#) or both.

bayesboot, bootstrap, by, collect, jackknife, rolling, and statsby are allowed; see **[U] 11.1.10 Prefix commands**.

Weights are not allowed with the bootstrap prefix; see [R] **bootstrap**.

aweights are not allowed with the jackknife prefix; see [R] **jackknife**.

aweights, fweights, iweights, and pweights are allowed; see **[U] 11.1.6 weight**.

coeflegend does not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

fgnls requests the two-step FGNLS estimator; this is the default.

ifgnls requests the iterative FGNLS estimator. For the nonlinear systems estimator, this is equivalent to maximum likelihood estimation.

nls requests the nonlinear least-squares (NLS) estimator.

variables(*varlist*) specifies the variables in the system. nlsur ignores observations for which any of these variables has missing values. If you do not specify variables(), nlsur issues an error message if the estimation sample contains any missing values.

initial(*initial_values*) specifies the initial values to begin the estimation. You can specify a $1 \times k$ matrix, where k is the total number of parameters in the system, or you can specify a parameter name, its initial value, another parameter name, its initial value, and so on. For example, to initialize alpha to 1.23 and delta to 4.57, you would type

```
. nlsur ..., initial(alpha 1.23 delta 4.57) ...
```

Initial values declared using this option override any that are declared within substitutable expressions. If you specify a matrix, the values must be in the same order in which the parameters are declared in your model. nlsur ignores the row and column names of the matrix.

nequations(#) specifies the number of equations in the system.

parameters(*namelist*) specifies the names of the parameters in the system. The names of the parameters must adhere to the naming conventions of Stata's variables; see **[U] 11.3 Naming conventions**. If you specify both parameters() and nparameters(), the number of names in the former must match the number specified in the latter.

nparameters(#) specifies the number of parameters in the system. If you do not specify names with the parameters() option, nlsur names them b1, b2, ..., b#. If you specify both parameters() and nparameters(), the number of names in the former must match the number specified in the latter.

sexp_options refer to any options allowed by your *sexp_prog*.

func_options refer to any options allowed by your *func_prog*.

SE/Robust

vce(*vcetype*) specifies the type of standard error reported, which includes types that are derived from asymptotic theory (gnr), that are robust to some kinds of misspecification (robust), that allow for intragroup correlation (cluster *clustvar*), and that use bootstrap or jackknife methods (bootstrap, jackknife); see [R] ***vce_option***.

vce(gnr), the default, uses the conventionally derived variance estimator for nonlinear models fit using Gauss–Newton regression.

Reporting

level(#); see [R] **Estimation options**.

title(*string*) specifies an optional title that will be displayed just above the table of parameter estimates.

title2(*string*) specifies an optional subtitle that will be displayed between the title specified in title() and the table of parameter estimates. If title2() is specified but title() is not, title2() has the same effect as title().

display_options: noci, nopvalues, cformat(%*fmt*), pformat(%*fmt*), sformat(%*fmt*), and nolstretch; see [R] **Estimation options**.

Optimization

optimization_options: iterate(#), [no]log, trace, eps(#), ifgnlsiterate(#), ifgnlseps(#), delta(#), noconstants, and hasconstants(*namelist*).

iterate() specifies the maximum number of iterations to use for NLS at each round of FGNLS estimation. This option is different from ifgnlsiterate(), which controls the maximum rounds of FGNLS estimation to use when the ifgnls option is specified.

log and nolog specify whether to show the iteration log; see set iterlog in [R] ***set iter***.

trace specifies that the iteration log should include the current parameter vector.

eps(#) specifies the convergence criterion for successive parameter estimates and for the residual sum of squares (RSS). The default is eps(1e-5) (0.00001). eps() also specifies the convergence criterion for successive parameter estimates between rounds of iterative FGNLS estimation when ifgnls is specified.

ifgnlsiterate(#) specifies the maximum number of FGNLS iterations to perform. The default is the number set using set maxiter, which is 300 by default. To use this option, you must also specify the ifgnls option.

ifgnlseps(#) specifies the convergence criterion for successive estimates of the error covariance matrix during iterative FGNLS estimation. The default is ifgnlseps(1e-10). To use this option, you must also specify the ifgnls option.

delta(#) specifies the relative change in a parameter, δ, to be used in computing the numeric derivatives. The derivative for parameter β_i is computed as

$$\{f_i(\mathbf{x}_i, \beta_1, \beta_2, \ldots, \beta_i + d, \beta_{i+1}, \ldots) - f_i(\mathbf{x}_i, \beta_1, \beta_2, \ldots, \beta_i, \beta_{i+1}, \ldots)\}/d$$

where $d = \delta(|\beta_i| + \delta)$. The default is delta(4e-7).

noconstants indicates that none of the equations in the system includes constant terms. This option is generally not needed, even if there are no constant terms in the system; though in rare cases without this option, nlsur may claim that there is one or more constant terms even if there are none.

hasconstants(*namelist*) indicates the parameters that are to be treated as constant terms in the system of equations. The number of elements of *namelist* must equal the number of equations in the system. The ith entry of *namelist* specifies the constant term in the ith equation. If an equation

does not include a constant term, specify a period (.) instead of a parameter name. This option is seldom needed with the interactive and programmed substitutable expression versions, because in those cases `nlsur` can almost always find the constant terms automatically.

The following options are available with `nlsur` but are not shown in the dialog box:

`coeflegend`; see [R] **Estimation options**.

Remarks and examples

Remarks are presented under the following headings:

Introduction

`nlsur` fits a system of nonlinear equations by FGNLS. It can be viewed as a nonlinear variant of Zellner's seemingly unrelated regression model (Zellner 1962; Zellner and Huang 1962; Zellner 1963) and is therefore commonly called nonlinear SUR or nonlinear SURE. The model is also discussed in textbooks such as Davidson and MacKinnon (1993) and Greene (2012, 305–306). Formally, the model fit by `nlsur` is

$$\begin{aligned} y_{i1} &= f_1(\mathbf{x}_i, \boldsymbol{\beta}) + u_{i1} \\ y_{i2} &= f_2(\mathbf{x}_i, \boldsymbol{\beta}) + u_{i2} \\ \vdots &= \vdots \\ y_{iM} &= f_M(\mathbf{x}_i, \boldsymbol{\beta}) + u_{iM} \end{aligned}$$

for $i = 1, \ldots, N$ observations and $m = 1, \ldots, M$ equations. The errors for the ith observation, $u_{i1}, u_{i2}, \ldots, u_{iM}$, may be correlated, so fitting the m equations jointly may lead to more efficient estimates. Moreover, fitting the equations jointly allows us to impose cross-equation restrictions on the parameters. Not all elements of the parameter vector $\boldsymbol{\beta}$ and data vector $\mathbf{x}_i$ must appear in all the equations, though each element of $\boldsymbol{\beta}$ must appear in at least one equation for $\boldsymbol{\beta}$ to be identified. For this model, iterative FGNLS estimation is equivalent to maximum likelihood estimation with multivariate normal disturbances.

The syntax you use with `nlsur` closely mirrors that used with `nl`. In particular, you use substitutable expressions with the interactive and programmed substitutable expression versions to define the functions in your system. See [R] **nl** for more information on substitutable expressions. Here we reiterate the three rules that you must follow:

1. Parameters of the model are bound in braces: `{b0}`, `{param}`, etc.

2. Initial values for parameters are given by including an equal sign and the initial value inside the braces: `{b0=1}`, `{param=3.571}`, etc. If you do not specify an initial value, that parameter is initialized to zero. The `initial()` option overrides initial values in substitutable expressions.

3. Linear combinations of variables can be included using the notation {*eqname*`:`*varlist*}, for example, `{xb: mpg price weight}`, `{score: w x z}`, etc. Parameters of linear combinations are initialized to zero.

Example 1: Interactive version using two-step FGNLS estimator

We have data from an experiment in which two closely related types of bacteria were placed in a Petri dish, and the number of each type of bacteria were recorded every hour. We suspect a two-parameter exponential growth model can be used to model each type of bacteria, but because they shared the same dish, we want to allow for correlation in the error terms. We want to fit the system of equations

$$p_1 = \beta_1 {\beta_2}^t + u_1$$
$$p_2 = \gamma_1 {\gamma_2}^t + u_2$$

where p_1 and p_2 are the two populations and t is time, and we want to allow for nonzero correlation between u_1 and u_2. We type

```
. use https://www.stata-press.com/data/r19/petridish
. nlsur (p1 = {b1}*{b2}^t) (p2 = {g1}*{g2}^t)
(obs = 25)
Calculating NLS estimates:
Iteration 0:  Residual SS = 335.5286
Iteration 1:  Residual SS = 333.8583
Iteration 2:  Residual SS = 219.9233
Iteration 3:  Residual SS = 127.9355
Iteration 4:  Residual SS = 14.86765
Iteration 5:  Residual SS = 8.628459
Iteration 6:  Residual SS = 8.281268
Iteration 7:  Residual SS =  8.28098
Iteration 8:  Residual SS = 8.280979
Iteration 9:  Residual SS = 8.280979
Calculating FGNLS estimates:
Iteration 0:  Scaled RSS = 49.99892
Iteration 1:  Scaled RSS = 49.99892
Iteration 2:  Scaled RSS = 49.99892
FGNLS regression
```

Equation	Obs	Parms	RMSE	R-sq	Constant
1 p1	25	2	.4337019	0.9734*	(none)
2 p2	25	2	.3783479	0.9776*	(none)

* Uncentered R-sq

	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
/b1	.3926631	.064203	6.12	0.000	.2668275	.5184987
/b2	1.119593	.0088999	125.80	0.000	1.102149	1.137036
/g1	.5090441	.0669495	7.60	0.000	.3778256	.6402626
/g2	1.102315	.0072183	152.71	0.000	1.088167	1.116463

The header of the output contains a summary of each equation, including the number of observations and parameters and the root mean squared error of the residuals. `nlsur` checks to see whether each equation contains a constant term, and if an equation does contain a constant term, an R^2 statistic is presented. If an equation does not have a constant term, an uncentered R^2 is instead reported. The R^2 statistic for each equation measures the percentage of variance explained by the nonlinear function and may be useful for descriptive purposes, though it does not have the same formal interpretation in the context of FGNLS as it does with NLS estimation. As we would expect, β_2 and γ_2 are both greater than one, indicating the two bacterial populations increased in size over time.

The model we fit in the next three examples is in fact linear in the parameters, so it could be fit using the `sureg` command. However, we will fit the model using `nlsur` so that we can focus on the mechanics of using the command. Moreover, using `nlsur` will obviate the need to generate several variables as well as the need to use the `constraint` command to impose parameter restrictions.

Example 2: Interactive version using iterative FGNLS estimator—the translog production function

Greene (1997, sec. 15.6) discusses the transcendental logarithmic (translog) cost function and provides cost and input price data for capital, labor, energy, and materials for the US economy. One way to fit the translog production function to these data is to fit the system of three equations

$$s_k = \beta_k + \delta_{kk} \ln\left(\frac{p_k}{p_m}\right) + \delta_{kl} \ln\left(\frac{p_l}{p_m}\right) + \delta_{ke} \ln\left(\frac{p_e}{p_m}\right) + u_1$$
$$s_l = \beta_l + \delta_{kl} \ln\left(\frac{p_k}{p_m}\right) + \delta_{ll} \ln\left(\frac{p_l}{p_m}\right) + \delta_{le} \ln\left(\frac{p_e}{p_m}\right) + u_2$$
$$s_e = \beta_e + \delta_{ke} \ln\left(\frac{p_k}{p_m}\right) + \delta_{le} \ln\left(\frac{p_l}{p_m}\right) + \delta_{ee} \ln\left(\frac{p_e}{p_m}\right) + u_3$$

where s_k is capital's cost share, s_l is labor's cost share, and s_e is energy's cost share; p_k, p_l, p_e, and p_m are the prices of capital, labor, energy, and materials, respectively; the u's are regression error terms; and the β's and δ's are parameters to be estimated. There are three cross-equation restrictions on the parameters: δ_{kl}, δ_{ke}, and δ_{le} each appear in two equations. To fit this model by using the iterative FGNLS estimator, we type

```
. use https://www.stata-press.com/data/r19/mfgcost
(Manufacturing cost)
. nlsur (s_k = {bk} + {dkk}*ln(pk/pm) + {dkl}*ln(pl/pm) + {dke}*ln(pe/pm))
>       (s_l = {bl} + {dkl}*ln(pk/pm) + {dll}*ln(pl/pm) + {dle}*ln(pe/pm))
>       (s_e = {be} + {dke}*ln(pk/pm) + {dle}*ln(pl/pm) + {dee}*ln(pe/pm)),
>       ifgnls
(obs = 25)
Calculating NLS estimates:
Iteration 0:  Residual SS = .0009989
Iteration 1:  Residual SS = .0009989
Calculating FGNLS estimates:
Iteration 0:  Scaled RSS = 65.45197
Iteration 1:  Scaled RSS = 65.45197
  (output omitted)
FGNLS iteration 10:
Iteration 0:  Scaled RSS =       75
Iteration 1:  Scaled RSS =       75
Parameter change         = 4.08e-06
Covariance matrix change = 6.26e-10
FGNLS regression
```

Equation	Obs	Parms	RMSE	R-sq	Constant
1 s_k	25	4	.0031722	0.4776	bk
2 s_l	25	4	.0053963	0.8171	bl
3 s_e	25	4	.00177	0.6615	be

	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
/bk	.0568925	.0013454	42.29	0.000	.0542556	.0595294
/dkk	.0294833	.0057956	5.09	0.000	.0181241	.0408425
/dkl	-.0000471	.0038478	-0.01	0.990	-.0075887	.0074945
/dke	-.0106749	.0033882	-3.15	0.002	-.0173157	-.0040341
/bl	.253438	.0020945	121.00	0.000	.2493329	.2575432
/dll	.0754327	.0067572	11.16	0.000	.0621889	.0886766
/dle	-.004756	.002344	-2.03	0.042	-.0093501	-.0001619
/be	.0444099	.0008533	52.04	0.000	.0427374	.0460823
/dee	.0183415	.0049858	3.68	0.000	.0085694	.0281135

We draw your attention to the iteration log at the top of the output. When iterative FGNLS estimation is used, the final scaled RSS will equal the product of the number of observations in the estimation sample and the number of equations; see *Methods and formulas* for details. Because the RSS is scaled by the error covariance matrix during each round of FGNLS estimation, the scaled RSS is not comparable from one FGNLS iteration to the next.

◁

❑ Technical note

You may have noticed that we mentioned having data for four factors of production, yet we fit only three share equations. Because the four shares sum to one, we must drop one of the equations to avoid having a singular error covariance matrix. The iterative FGNLS estimator is equivalent to maximum likelihood estimation, and thus it is invariant to which one of the four equations we choose to drop. The (linearly restricted) parameters of the fourth equation can be obtained using the `lincom` command. Nonlinear functions of the parameters, such as the elasticities of substitution, can be computed using `nlcom`.

❑

Substitutable expression programs

If you fit the same model repeatedly or you want to share code with colleagues, you can write a *substitutable expression program* to define your system of equations and avoid having to retype the system every time. The first five letters of the program's name must be `nlsur`, and the program must set the r-class macro `r(n_eq)` to the number of equations in your system. The first equation's substitutable expression must be returned in `r(eq_1)`, the second equation's in `r(eq_2)`, and so on. You may optionally set `r(title)` to label your output; that has the same effect as specifying the `title()` option.

▷ Example 3: Programmed substitutable expression version

We return to our translog cost function, for which a substitutable expression program is

```
program nlsurtranslog, rclass
     version 19.5      // (or version 19 if you do not have StataNow)
     syntax varlist(min=7 max=7) [if]
     tokenize ‘varlist’
     args sk sl se pk pl pe pm
     local pkpm ln(‘pk’/‘pm’)
     local plpm ln(‘pl’/‘pm’)
     local pepm ln(‘pe’/‘pm’)
     return scalar n_eq = 3
     return local eq_1 "‘sk’= {bk} + {dkk}*‘pkpm’ + {dkl}*‘plpm’ + {dke}*‘pepm’"
     return local eq_2 "‘sl’= {bl} + {dkl}*‘pkpm’ + {dll}*‘plpm’ + {dle}*‘pepm’"
     return local eq_3 "‘se’= {be} + {dke}*‘pkpm’ + {dle}*‘plpm’ + {dee}*‘pepm’"
     return local title "4-factor translog cost function"
end
```

We made our program accept seven variables, for the three dependent variables s_k, s_l, and s_e, and the four factor prices p_k, p_l, p_m, and p_e. The `tokenize` command assigns to macros `‘1’`, `‘2’`, …, `‘7’` the seven variables stored in `‘varlist’`, and the `args` command transfers those numbered macros to macros `‘sk’`, `‘sl’`, …, `‘pm’`. Because we knew our substitutable expressions were going to be somewhat long, we created local macros to hold the log price ratios. These are simply macros that hold strings such as `ln(pk/pm)`, not variables, and they will save us some repetitious typing when we define our substitutable expressions. Our program returns the number of equations in `r(n_eq)`, and we defined our substitutable expressions in `eq_1`, `eq_2`, and `eq_3`. We do not bind the expressions in parentheses as we do with the interactive version of `nlsur`. Finally, we put a title in `r(title)` to label our output.

Our `syntax` command also accepts an `if` clause, and that is how `nlsur` indicates the estimation sample to our program. In this application, we can safely ignore it because our program does not compute initial values. However, had we used commands such as `summarize` or `regress` to obtain initial values, then we would need to restrict those commands to analyze only the estimation sample. In those cases, typically, you simply need to include `‘if’` with the commands you are using. For example, instead of the command

```
summarize ‘depvar’, meanonly
```

you would use

```
summarize ‘depvar’ ‘if’, meanonly
```

We can check our program by typing

```
. nlsurtranslog s_k s_l s_e pk pl pe pm
. return list
scalars:
                 r(n_eq) =  3
macros:
                r(title) : "4-factor translog cost function"
                 r(eq_3) : "s_e= {be} + {dke}*ln(pk/pm) + {dle}*ln(pl/pm) + {.."
                 r(eq_2) : "s_l= {bl} + {dkl}*ln(pk/pm) + {dll}*ln(pl/pm) + {.."
                 r(eq_1) : "s_k= {bk} + {dkk}*ln(pk/pm) + {dkl}*ln(pl/pm) + {.."
```

Now that we know that our program works, we fit our model by typing

```
. nlsur translog: s_k s_l s_e pk pl pe pm, ifgnls
(obs = 25)
Calculating NLS estimates:
Iteration 0:  Residual SS = .0009989
Iteration 1:  Residual SS = .0009989
Calculating FGNLS estimates:
Iteration 0:  Scaled RSS = 65.45197
Iteration 1:  Scaled RSS = 65.45197
FGNLS iteration 2:
Iteration 0:  Scaled RSS = 73.28311
Iteration 1:  Scaled RSS = 73.28311
Iteration 2:  Scaled RSS = 73.28311
Parameter change         = 6.54e-03
Covariance matrix change = 1.00e-06
  (output omitted)
FGNLS iteration 10:
Iteration 0:  Scaled RSS =       75
Iteration 1:  Scaled RSS =       75
Parameter change         = 4.08e-06
Covariance matrix change = 6.26e-10
FGNLS regression
```

Equation		Obs	Parms	RMSE	R-sq	Constant
1	s_k	25	4	.0031722	0.4776	bk
2	s_l	25	4	.0053963	0.8171	bl
3	s_e	25	4	.00177	0.6615	be

4-factor translog cost function

	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
/bk	.0568925	.0013454	42.29	0.000	.0542556	.0595294
/dkk	.0294833	.0057956	5.09	0.000	.0181241	.0408425
/dkl	-.0000471	.0038478	-0.01	0.990	-.0075887	.0074945
/dke	-.0106749	.0033882	-3.15	0.002	-.0173157	-.0040341
/bl	.253438	.0020945	121.00	0.000	.2493329	.2575432
/dll	.0754327	.0067572	11.16	0.000	.0621889	.0886766
/dle	-.004756	.002344	-2.03	0.042	-.0093501	-.0001619
/be	.0444099	.0008533	52.04	0.000	.0427374	.0460823
/dee	.0183415	.0049858	3.68	0.000	.0085694	.0281135

Because we set `r(title)` in our substitutable expression program, the coefficient table has a title attached to it. The estimates are identical to those we obtained in example 2.

◁

❑ Technical note

nlsur accepts frequency and analytic weights as well as pweights (sampling weights) and iweights (importance weights). You do not need to modify your substitutable expressions in any way to perform weighted estimation, though you must make two changes to your substitutable expression program. The general outline of a *sexp_prog* program is

```
program nlsur name, rclass
    version 19.5      // (or version 19 if you do not have StataNow)
    syntax varlist [fw aw pw iw] [if]
    // Obtain initial values incorporating weights.  For example,
    summarize varname ['weight''exp'] 'if'
    ...
    // Return n_eqn and substitutable expressions
    return scalar n_eq = #
    return local eq_1 = ...
    ...
end
```

First, we wrote the syntax statement to accept a weight expression. Here we allow all four types of weights, but if you know that your estimator is valid, say, for only frequency weights, then you should modify the syntax line to accept only fweights. Second, if your program computes starting values, then any commands you use must incorporate the weights passed to the program; you do that by including ['weight''exp'] when calling those commands.

❑

Function evaluator programs

Although substitutable expressions are extremely flexible, there are some problems for which the nonlinear system cannot be defined using them. You can use the function evaluator program version of nlsur in these cases. We present two examples, a simple one to illustrate the mechanics of function evaluator programs and a more complicated one to illustrate the power of nlsur.

▷ Example 4: Function evaluator program version

Here we write a function evaluator program to fit the translog cost function used in examples 2 and 3. The function evaluator program is

```
program nlsurtranslog2
        version 19.5      // (or version 19 if you do not have StataNow)
        syntax varlist(min=7 max=7) [if], at(name)
        tokenize ‘varlist’
        args sk sl se pk pl pe pm
        tempname bk dkk dkl dke bl dll dle be dee
        scalar ‘bk’  = ‘at’[1,1]
        scalar ‘dkk’ = ‘at’[1,2]
        scalar ‘dkl’ = ‘at’[1,3]
        scalar ‘dke’ = ‘at’[1,4]
        scalar ‘bl’  = ‘at’[1,5]
        scalar ‘dll’ = ‘at’[1,6]
        scalar ‘dle’ = ‘at’[1,7]
        scalar ‘be’  = ‘at’[1,8]
        scalar ‘dee’ = ‘at’[1,9]
        local pkpm ln(‘pk’/‘pm’)
        local plpm ln(‘pl’/‘pm’)
        local pepm ln(‘pe’/‘pm’)
        quietly {
                replace ‘sk’ = ‘bk’ + ‘dkk’*‘pkpm’ + ‘dkl’*‘plpm’ +    ///
                                      ‘dke’*‘pepm’ ‘if’
                replace ‘sl’ = ‘bl’ + ‘dkl’*‘pkpm’ + ‘dll’*‘plpm’ +    ///
                                      ‘dle’*‘pepm’ ‘if’
                replace ‘se’ = ‘be’ + ‘dke’*‘pkpm’ + ‘dle’*‘plpm’ +    ///
                                      ‘dee’*‘pepm’ ‘if’
        }
end
```

Unlike the substitutable expression program we wrote in example 3, `nlsurtranslog2` is not declared as r-class because we will not be returning any stored results. We are again expecting seven variables: three shares and four factor prices, and `nlsur` will again mark the estimation sample with an `if` expression.

Our function evaluator program also accepts an option named `at()`, which will receive a parameter vector at which we are to evaluate the system of equations. All function evaluator programs must accept this option. Our model has nine parameters to estimate, and we created nine temporary scalars to hold the elements of the `‘at’` matrix.

Because our model has three equations, the first three variables passed to our program are the dependent variables that we are to fill in with the function values. We replaced only the observations in our estimation sample by including the `‘if’` qualifier in the `replace` statements. Here we could have ignored the `‘if’` qualifier because `nlsur` will skip over observations not in the estimation sample and we did not perform any computations requiring knowledge of the estimation sample. However, including the `‘if’` is good practice and may result in a slight speed improvement if the functions of your model are complicated and the estimation sample is much smaller than the dataset in memory.

We could have avoided creating temporary scalars to hold our individual parameters by writing the `replace` statements as, for example,

```
replace `sk' = `at'[1,1] + `at'[1,2]*`pkpm' + `at'[1,3]*`plpm' + `at'[1,4]*`pepm' `if'
```

You can use whichever method you find more appealing, though giving the parameters descriptive names reduces the chance for mistakes and makes debugging easier.

To fit our model by using the function evaluator program version of `nlsur`, we type

```
. nlsur translog2 @ s_k s_l s_e pk pl pe pm, ifgnls nequations(3)
>         parameters(bk dkk dkl dke bl dll dle be dee)
>         hasconstants(bk bl be)
(obs = 25)

Calculating NLS estimates:
Iteration 0:  Residual SS = .0009989
Iteration 1:  Residual SS = .0009989

Calculating FGNLS estimates:
Iteration 0:  Scaled RSS = 65.45197
Iteration 1:  Scaled RSS = 65.45197

FGNLS iteration 2:
Iteration 0:  Scaled RSS = 73.28311
Iteration 1:  Scaled RSS = 73.28311
Iteration 2:  Scaled RSS = 73.28311
Parameter change         = 6.54e-03
Covariance matrix change = 1.00e-06

FGNLS iteration 3:
Iteration 0:  Scaled RSS =  74.7113
Iteration 1:  Scaled RSS =  74.7113
Parameter change         = 2.58e-03
Covariance matrix change = 3.96e-07

FGNLS iteration 4:
Iteration 0:  Scaled RSS = 74.95356
Iteration 1:  Scaled RSS = 74.95356
Parameter change         = 1.02e-03
Covariance matrix change = 1.57e-07

FGNLS iteration 5:
Iteration 0:  Scaled RSS = 74.99261
Iteration 1:  Scaled RSS = 74.99261
Parameter change         = 4.07e-04
Covariance matrix change = 6.25e-08

FGNLS iteration 6:
Iteration 0:  Scaled RSS = 74.99883
Iteration 1:  Scaled RSS = 74.99883
Parameter change         = 1.62e-04
Covariance matrix change = 2.49e-08

FGNLS iteration 7:
Iteration 0:  Scaled RSS = 74.99981
Iteration 1:  Scaled RSS = 74.99981
Iteration 2:  Scaled RSS = 74.99981
Parameter change         = 6.45e-05
Covariance matrix change = 9.91e-09

FGNLS iteration 8:
Iteration 0:  Scaled RSS = 74.99997
Iteration 1:  Scaled RSS = 74.99997
Iteration 2:  Scaled RSS = 74.99997
Iteration 3:  Scaled RSS = 74.99997
Parameter change         = 2.57e-05
Covariance matrix change = 3.95e-09
```

```
FGNLS iteration 9:
Iteration 0:  Scaled RSS =       75
Iteration 1:  Scaled RSS =       75
Iteration 2:  Scaled RSS =       75
Parameter change         = 1.02e-05
Covariance matrix change = 1.57e-09
FGNLS iteration 10:
Iteration 0:  Scaled RSS =       75
Iteration 1:  Scaled RSS =       75
Parameter change         = 4.08e-06
Covariance matrix change = 6.26e-10
FGNLS regression
```

Equation		Obs	Parms	RMSE	R-sq	Constant
1	s_k	25	.	.0031722	0.4776	bk
2	s_l	25	.	.0053963	0.8171	bl
3	s_e	25	.	.00177	0.6615	be

	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
/bk	.0568925	.0013454	42.29	0.000	.0542556	.0595294
/dkk	.0294833	.0057956	5.09	0.000	.0181241	.0408425
/dkl	-.0000471	.0038478	-0.01	0.990	-.0075887	.0074945
/dke	-.0106749	.0033882	-3.15	0.002	-.0173157	-.0040341
/bl	.253438	.0020945	121.00	0.000	.2493329	.2575432
/dll	.0754327	.0067572	11.16	0.000	.0621889	.0886766
/dle	-.004756	.002344	-2.03	0.042	-.0093501	-.0001619
/be	.0444099	.0008533	52.04	0.000	.0427374	.0460823
/dee	.0183415	.0049858	3.68	0.000	.0085694	.0281135

When we use the function evaluator program version, `nlsur` requires us to specify the number of equations in `nequations()`, and it requires us to specify either the names for each of our parameters or the number of parameters in the model. Here we used the `parameters()` option to name our parameters; the order in which we specified them in this option is the same as the order in which we extracted them from the `'at'` matrix in our program. Had we instead specified `nparameters(9)`, our parameters would have been labeled `/b1`, `/b2`, …, `/b9` in the output.

`nlsur` has no way of telling how many parameters appear in each equation, so the `Parms` column in the header contains missing values. Moreover, the function evaluator program version of `nlsur` does not attempt to identify constant terms, so we used the `hasconstant` option to tell `nlsur` which parameter in each equation is a constant term.

The estimates are identical to those we obtained in examples 2 and 3.

◁

❑ Technical note

As with substitutable expression programs, if you intend to do weighted estimation with a function evaluator program, you must modify your *func_prog* program's `syntax` statement to accept weights. Moreover, if you use any statistical commands when computing your nonlinear functions, then you must include the weight expression with those commands.

❑

▷ Example 5: Fitting the basic AIDS model using nlsur

Deaton and Muellbauer (1980) introduce the almost ideal demand system (AIDS), and Poi (2012) presents a set of commands and several extensions for fitting the AIDS automatically. Here we show how to fit the basic AIDS model, which is a common example of a nonlinear system of equations, by manually using `nlsur`. The dataset `food.dta` contains household expenditures, expenditure shares, and log prices for four broad food groups. For a four-good demand system, we need to fit the following system of three equations:

$$w_1 = \alpha_1 + \gamma_{11}\ln p_1 + \gamma_{12}\ln p_2 + \gamma_{13}\ln p_3 + \beta_1 \ln\left\{\frac{m}{P(\mathbf{p})}\right\} + u_1$$
$$w_2 = \alpha_2 + \gamma_{12}\ln p_1 + \gamma_{22}\ln p_2 + \gamma_{23}\ln p_3 + \beta_2 \ln\left\{\frac{m}{P(\mathbf{p})}\right\} + u_2$$
$$w_3 = \alpha_3 + \gamma_{13}\ln p_1 + \gamma_{23}\ln p_2 + \gamma_{33}\ln p_3 + \beta_3 \ln\left\{\frac{m}{P(\mathbf{p})}\right\} + u_3$$

where w_k denotes a household's fraction of expenditures on good k, $\ln p_k$ denotes the logarithm of the price paid for good k, m denotes a household's total expenditure on all four goods, the u's are regression error terms, and

$$\ln P(\mathbf{p}) = \alpha_0 + \sum_{i=1}^{4}\alpha_i \ln p_i + \frac{1}{2}\sum_{i=1}^{4}\sum_{j=1}^{4}\gamma_{ij}\ln p_i \ln p_j$$

The parameters for the fourth good's share equation can be recovered from the following constraints that are imposed by economic theory:

$$\sum_{i=1}^{4}\alpha_i = 1 \qquad \sum_{i=1}^{4}\beta_i = 0 \qquad \gamma_{ij} = \gamma_{ji} \qquad \text{and} \qquad \sum_{i=1}^{4}\gamma_{ij} = 0 \ \text{ for all } j$$

Our model has a total of 12 unrestricted parameters. We will not estimate α_0 directly. Instead, we will set it equal to 5; see Deaton and Muellbauer (1980) for a discussion of why treating α_0 as fixed is acceptable.

Our function evaluator program is

```
program nlsuraids
        version 19.5      // (or version 19 if you do not have StataNow)
        syntax varlist(min=8 max=8) if, at(name)
        tokenize `varlist'
        args w1 w2 w3 lnp1 lnp2 lnp3 lnp4 lnm
        tempname a1 a2 a3 a4
        scalar `a1' = `at'[1,1]
        scalar `a2' = `at'[1,2]
        scalar `a3' = `at'[1,3]
        scalar `a4' = 1 - `a1' - `a2' - `a3'
        tempname b1 b2 b3
        scalar `b1' = `at'[1,4]
        scalar `b2' = `at'[1,5]
        scalar `b3' = `at'[1,6]
        tempname g11 g12 g13 g14
        tempname g21 g22 g23 g24
        tempname g31 g32 g33 g34
        tempname g41 g42 g43 g44
        scalar `g11' = `at'[1,7]
        scalar `g12' = `at'[1,8]
        scalar `g13' = `at'[1,9]
        scalar `g14' = -`g11'-`g12'-`g13'
        scalar `g21' = `g12'
        scalar `g22' = `at'[1,10]
        scalar `g23' = `at'[1,11]
        scalar `g24' = -`g21'-`g22'-`g23'
        scalar `g31' = `g13'
        scalar `g32' = `g23'
        scalar `g33' = `at'[1,12]
        scalar `g34' = -`g31'-`g32'-`g33'
        scalar `g41' = `g14'
        scalar `g42' = `g24'
        scalar `g43' = `g34'
        scalar `g44' = -`g41'-`g42'-`g43'
        quietly {
                tempvar lnpindex
                gen double `lnpindex' = 5 + `a1'*`lnp1' + `a2'*`lnp2' + ///
                                            `a3'*`lnp3' + `a4'*`lnp4'
                forvalues i = 1/4 {
                        forvalues j = 1/4 {
                                replace `lnpindex' = `lnpindex' + ///
                                        0.5*`g`i''`j''*`lnp`i''*`lnp`j''
                        }
                }
                replace `w1' = `a1' + `g11'*`lnp1' + `g12'*`lnp2' + ///
                                      `g13'*`lnp3' + `g14'*`lnp4' + ///
                                       `b1'*(`lnm' - `lnpindex')
                replace `w2' = `a2' + `g21'*`lnp1' + `g22'*`lnp2' + ///
                                      `g23'*`lnp3' + `g24'*`lnp4' + ///
                                      `b2'*(`lnm' - `lnpindex')
                replace `w3' = `a3' + `g31'*`lnp1' + `g32'*`lnp2' + ///
                                      `g33'*`lnp3' + `g34'*`lnp4' + ///
                                      `b3'*(`lnm' - `lnpindex')
        }
end
```

The `syntax` statement accepts eight variables: three expenditure share variables, all four log-price variables, and a variable for log expenditures ($\ln m$). Most of the code simply extracts the parameters from the `'at'` matrix. Although we are estimating only 12 parameters, to calculate the price index term and the expenditure share equations, we need the restricted parameters as well. Notice how we impose the constraints on the parameters. We then created a temporary variable to hold $\ln P(\mathbf{p})$, and we filled the three dependent variables with the predicted expenditure shares.

To fit our model, we type

```
. use https://www.stata-press.com/data/r19/food
(Four food groups)
. nlsur aids @ w1 w2 w3 lnp1 lnp2 lnp3 lnp4 lnexp,
>                 parameters(a1 a2 a3 b1 b2 b3
>                            g11 g12 g13 g22 g32 g33)
>                 neq(3) ifgnls
(obs = 4,048)
Calculating NLS estimates:
Iteration 0:  Residual SS = 126.9713
Iteration 1:  Residual SS =  125.669
Iteration 2:  Residual SS =  125.669
Iteration 3:  Residual SS =  125.669
Iteration 4:  Residual SS =  125.669
Calculating FGNLS estimates:
Iteration 0:  Scaled RSS = 12080.14
Iteration 1:  Scaled RSS = 12080.14
Iteration 2:  Scaled RSS = 12080.14
Iteration 3:  Scaled RSS = 12080.14
FGNLS iteration 2:
Iteration 0:  Scaled RSS = 12143.99
Iteration 1:  Scaled RSS = 12143.99
Iteration 2:  Scaled RSS = 12143.99
Parameter change         = 1.97e-04
Covariance matrix change = 2.94e-06
FGNLS iteration 3:
Iteration 0:  Scaled RSS =    12144
Iteration 1:  Scaled RSS =    12144
Parameter change         = 2.18e-06
Covariance matrix change = 3.47e-08
FGNLS regression
```

Equation		Obs	Parms	RMSE	R-sq	Constant
1	w1	4,048	.	.1333175	0.9017*	(none)
2	w2	4,048	.	.1024166	0.8480*	(none)
3	w3	4,048	.	.053777	0.7906*	(none)

```
* Uncentered R-sq
```

	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
/a1	.3163958	.0073871	42.83	0.000	.3019175	.3308742
/a2	.2712501	.0056938	47.64	0.000	.2600904	.2824097
/a3	.1039898	.0029004	35.85	0.000	.0983051	.1096746
/b1	.0161044	.0034153	4.72	0.000	.0094105	.0227983
/b2	-.0260771	.002623	-9.94	0.000	-.0312181	-.0209361
/b3	.0014538	.0013776	1.06	0.291	-.0012463	.004154
/g11	.1215838	.0057186	21.26	0.000	.1103756	.1327921
/g12	-.0522943	.0039305	-13.30	0.000	-.0599979	-.0445908
/g13	-.0351292	.0021788	-16.12	0.000	-.0393996	-.0308588
/g22	.0644298	.0044587	14.45	0.000	.0556909	.0731687
/g32	-.0011786	.0019767	-0.60	0.551	-.0050528	.0026957
/g33	.0424381	.0017589	24.13	0.000	.0389909	.0458854

To get the restricted parameters for the fourth share equation, we can use `lincom`. For example, to obtain α_4, we type

```
. lincom 1 - [a1]_cons - [a2]_cons - [a3]_cons
 ( 1)  - [a1]_cons - [a2]_cons - [a3]_cons = -1
```

	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
(1)	.3083643	.0052611	58.61	0.000	.2980528	.3186758

For more information on `lincom`, see [R] **lincom**.

◁

Stored results

`nlsur` stores the following in `e()`:

Scalars

`e(N)`	number of observations
`e(k)`	number of parameters
`e(k_#)`	number of parameters for equation #
`e(k_eq)`	number of equation names in `e(b)`
`e(k_eq_model)`	number of equations in overall model test
`e(n_eq)`	number of equations
`e(mss_#)`	model sum of squares for equation #
`e(rss_#)`	RSS for equation #
`e(rmse_#)`	root mean squared error for equation #
`e(r2_#)`	R^2 for equation #
`e(ll)`	Gaussian log likelihood (`iflgs` version only)
`e(N_clust)`	number of clusters
`e(rank)`	rank of `e(V)`
`e(converged)`	1 if converged, 0 otherwise

Macros

`e(cmd)`	`nlsur`
`e(cmdline)`	command as typed
`e(method)`	`fgnls`, `ifgnls`, or `nls`
`e(depvar)`	names of dependent variables
`e(depvar_#)`	dependent variable for equation #
`e(wtype)`	weight type
`e(wexp)`	weight expression

e(title)	title in estimation output
e(title_2)	secondary title in estimation output
e(clustvar)	name of cluster variable
e(vce)	*vcetype* specified in vce()
e(vcetype)	title used to label Std. err.
e(type)	1 = interactively entered expression 2 = substitutable expression program 3 = function evaluator program
e(sexpprog)	substitutable expression program
e(sexp_#)	substitutable expression for equation #
e(params)	names of all parameters
e(params_#)	parameters in equation #
e(funcprog)	function evaluator program
e(rhs)	contents of variables()
e(constants)	identifies constant terms
e(properties)	b V
e(predict)	program used to implement predict

Matrices

e(b)	coefficient vector
e(init)	initial values vector
e(Sigma)	error covariance matrix ($\widehat{\boldsymbol{\Sigma}}$)
e(V)	variance–covariance matrix of the estimators

Functions

e(sample)	marks estimation sample

In addition to the above, the following is stored in r():

Matrices

r(table)	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in r() are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

Write the system of equations for the ith observation as

$$\mathbf{y}_i = \mathbf{f}(\mathbf{x}_i, \boldsymbol{\beta}) + \mathbf{u}_i \tag{1}$$

where $\mathbf{y}_i$ and $\mathbf{u}_i$ are $1 \times M$ vectors, for $i = 1, \ldots, N$; $\mathbf{f}$ is a function that returns a $1 \times M$ vector; $\mathbf{x}_i$ represents all the exogenous variables in the system; and $\boldsymbol{\beta}$ is a $1 \times k$ vector of parameters. The generalized nonlinear least-squares system estimator is defined as

$$\widehat{\boldsymbol{\beta}} \equiv \text{argmin}_{\boldsymbol{\beta}} \sum_{i=1}^{N} \{\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i, \boldsymbol{\beta})\} \, \boldsymbol{\Sigma}^{-1} \, \{\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i, \boldsymbol{\beta})\}'$$

where $\boldsymbol{\Sigma} = E(\mathbf{u}_i' \mathbf{u}_i)$ is an $M \times M$ positive-definite weight matrix. Let $\mathbf{T}$ be the Cholesky decomposition of $\boldsymbol{\Sigma}^{-1}$; that is, $\mathbf{T}\mathbf{T}' = \boldsymbol{\Sigma}^{-1}$. Postmultiply (1) by $\mathbf{T}$:

$$\mathbf{y}_i \mathbf{T} = \mathbf{f}(\mathbf{x}_i, \boldsymbol{\beta})\mathbf{T} + \mathbf{u}_i \mathbf{T} \tag{2}$$

Because $E(\mathbf{T}'\mathbf{u}_i'\mathbf{u}_i\mathbf{T}) = \mathbf{I}$, we can "stack" the columns of (2) and write

$$
\begin{aligned}
\mathbf{y}_1\mathbf{T}_1 &= \mathbf{f}(\mathbf{x}_1,\boldsymbol{\beta})\mathbf{T}_1 + \tilde{u}_{11}\\
\mathbf{y}_1\mathbf{T}_2 &= \mathbf{f}(\mathbf{x}_1,\boldsymbol{\beta})\mathbf{T}_2 + \tilde{u}_{12}\\
&\vdots = \vdots\\
\mathbf{y}_1\mathbf{T}_M &= \mathbf{f}(\mathbf{x}_1,\boldsymbol{\beta})\mathbf{T}_M + \tilde{u}_{1M}\\
&\vdots = \vdots\\
\mathbf{y}_N\mathbf{T}_1 &= \mathbf{f}(\mathbf{x}_N,\boldsymbol{\beta})\mathbf{T}_1 + \tilde{u}_{N1}\\
\mathbf{y}_N\mathbf{T}_2 &= \mathbf{f}(\mathbf{x}_N,\boldsymbol{\beta})\mathbf{T}_2 + \tilde{u}_{N2}\\
&\vdots = \vdots\\
\mathbf{y}_N\mathbf{T}_M &= \mathbf{f}(\mathbf{x}_N,\boldsymbol{\beta})\mathbf{T}_M + \tilde{u}_{NM}
\end{aligned}
\tag{3}
$$

where $\mathbf{T}_j$ denotes the jth column of $\mathbf{T}$. By construction, all $\tilde{u}_{ij}$ are independently distributed with unit variance. As a result, by transforming the model in (1) to that shown in (3), we have reduced the multivariate generalized nonlinear least-squares system estimator to a univariate nonlinear least-squares problem; and the same parameter estimation technique used by `nl` can be used here. See [R] **nl** for the details. Moreover, because the $\tilde{u}_{ij}$ all have variance 1, the final scaled RSS reported by `nlsur` is equal to NM.

To make the estimator feasible, we require an estimate $\widehat{\boldsymbol{\Sigma}}$ of $\boldsymbol{\Sigma}$. `nlsur` first sets $\widehat{\boldsymbol{\Sigma}} = \mathbf{I}$. Although not efficient, the resulting estimate, $\widehat{\boldsymbol{\beta}}_{\text{NLS}}$, is consistent. If the `nls` option is specified, estimation is complete. Otherwise, the residuals

$$\hat{\mathbf{u}}_i = \mathbf{y}_i - \mathbf{f}(\mathbf{x}_i, \widehat{\boldsymbol{\beta}}_{\text{NLS}})$$

are calculated and used to compute

$$\widehat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N}\hat{\mathbf{u}}_i'\hat{\mathbf{u}}_i$$

With $\widehat{\boldsymbol{\Sigma}}$ in hand, a new estimate $\widehat{\boldsymbol{\beta}}$ is then obtained.

If the `ifgnls` option is specified, the new $\widehat{\boldsymbol{\beta}}$ is used to recompute the residuals and obtain a new estimate of $\widehat{\boldsymbol{\Sigma}}$, from which $\widehat{\boldsymbol{\beta}}$ can then be reestimated. Iterations stop when the relative change in $\widehat{\boldsymbol{\beta}}$ is less than `eps()`, the relative change in $\widehat{\boldsymbol{\Sigma}}$ is less than `ifgnlseps()`, or if `ifgnlsiterate()` iterations have been performed.

If the `vce(robust)` and `vce(cluster` *clustvar*`)` options were not specified, then

$$V(\widehat{\boldsymbol{\beta}}) = \left(\sum_{i=1}^{N}\mathbf{X}_i'\widehat{\boldsymbol{\Sigma}}^{-1}\mathbf{X}_i\right)^{-1}$$

where the $M \times k$ matrix $\mathbf{X}_i$ has typical element X_{ist}, the derivative of the sth element of $\mathbf{f}$ with respect to the tth element of $\boldsymbol{\beta}$, evaluated at $\mathbf{x}_i$ and $\widehat{\boldsymbol{\beta}}$. As a practical matter, once the model is written in the form of (3), the variance–covariance matrix can be calculated via a Gauss–Newton regression; see Davidson and MacKinnon (1993, chap. 6).

If `robust` is specified, then

$$V_R(\widehat{\boldsymbol{\beta}}) = \left(\sum_{i=1}^{N}\mathbf{X}_i'\widehat{\boldsymbol{\Sigma}}^{-1}\mathbf{X}_i\right)^{-1}\sum_{i=1}^{N}\mathbf{X}_i'\widehat{\boldsymbol{\Sigma}}^{-1}\hat{\mathbf{u}}_i'\hat{\mathbf{u}}_i\widehat{\boldsymbol{\Sigma}}^{-1}\mathbf{X}_i\left(\sum_{i=1}^{N}\mathbf{X}_i'\widehat{\boldsymbol{\Sigma}}^{-1}\mathbf{X}_i\right)^{-1}$$

The cluster–robust variance matrix is

$$V_C(\widehat{\boldsymbol{\beta}}) = \left(\sum_{i=1}^{N} \mathbf{X}_i' \widehat{\boldsymbol{\Sigma}}^{-1} \mathbf{X}_i\right)^{-1} \sum_{c=1}^{N_C} \mathbf{w}_c \mathbf{w}_c' \left(\sum_{i=1}^{N} \mathbf{X}_i' \widehat{\boldsymbol{\Sigma}}^{-1} \mathbf{X}_i\right)^{-1}$$

where N_C is the number of clusters and

$$\mathbf{w}_c = \sum_{j \in C_k} \mathbf{X}_j' \widehat{\boldsymbol{\Sigma}}^{-1} \hat{\mathbf{u}}_j'$$

with C_k denoting the set of observations in the kth cluster. In evaluating these formulas, we use the value of $\widehat{\boldsymbol{\Sigma}}$ used in calculating the final estimate of $\widehat{\boldsymbol{\beta}}$. That is, we do not recalculate $\widehat{\boldsymbol{\Sigma}}$ after we obtain the final value of $\widehat{\boldsymbol{\beta}}$.

The RSS for the jth equation, RSS_j, is

$$\text{RSS}_j = \sum_{i=1}^{N} \left(\hat{y}_{ij} - y_{ij}\right)^2$$

where $\hat{y}_{ij}$ is the predicted value of the ith observation on the jth dependent variable; the total sum of squares (TSS) for the jth equation, TSS_j, is

$$\text{TSS}_j = \sum_{i=1}^{N} \left(y_{ij} - \overline{y}_j\right)^2$$

if there is a constant term in the jth equation, where $\overline{y}_j$ is the sample mean of the jth dependent variable, and

$$\text{TSS}_j = \sum_{i=1}^{N} y_{ij}^2$$

if there is no constant term in the jth equation; and the model sum of squares (MSS) for the jth equation, MSS_j, is $\text{TSS}_j - \text{RSS}_j$.

The R^2 for the jth equation is $\text{MSS}_j/\text{TSS}_j$. If an equation does not have a constant term, then the reported R^2 for that equation is "uncentered" and based on the latter definition of TSS_j.

Under the assumption that the $\mathbf{u}_i$ are independent and identically distributed $N(\mathbf{0}, \widehat{\boldsymbol{\Sigma}})$, the log likelihood for the model is

$$\ln L = -\frac{MN}{2}\{1 + \ln(2\pi)\} - \frac{N}{2}\ln\left|\widehat{\boldsymbol{\Sigma}}\right|$$

The log likelihood is reported only when the `ifgnls` option is specified.

References

Canette, I. 2011. A tip to debug your nl/nlsur function evaluator program. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2011/12/05/a-tip-to-debug-your-nlnlsur-function-evaluator-program/.

Davidson, R., and J. G. MacKinnon. 1993. *Estimation and Inference in Econometrics*. New York: Oxford University Press.

Deaton, A. S., and J. Muellbauer. 1980. An almost ideal demand system. *American Economic Review* 70: 312–326.

Greene, W. H. 1997. *Econometric Analysis.* 3rd ed. Upper Saddle River, NJ: Prentice Hall.

———. 2012. *Econometric Analysis*. 7th ed. Upper Saddle River, NJ: Prentice Hall.

Poi, B. P. 2012. Easy demand-system estimation with quaids. *Stata Journal* 12: 433–446.

Zellner, A. 1962. An efficient method of estimating seemingly unrelated regressions and tests for aggregation bias. *Journal of the American Statistical Association* 57: 348–368. https://doi.org/10.2307/2281644.

———. 1963. Estimators for seemingly unrelated regression equations: Some exact finite sample results. *Journal of the American Statistical Association* 58: 977–992. https://doi.org/10.2307/2283326.

Zellner, A., and D. S. Huang. 1962. Further properties of efficient estimators for seemingly unrelated regression equations. *International Economic Review* 3: 300–313. https://doi.org/10.2307/2525396.

Also see

[R] **nlsur postestimation** — Postestimation tools for nlsur

[R] **nl** — Nonlinear least-squares estimation

[R] **demandsys** — Estimation of flexible demand systems

[R] **gmm** — Generalized method of moments estimation

[R] **ml** — Maximum likelihood estimation

[R] **mlexp** — Maximum likelihood estimation of user-specified expressions

[R] **reg3** — Three-stage estimation for systems of simultaneous equations

[R] **sureg** — Zellner's seemingly unrelated regression

[U] 20 Estimation and postestimation commands