Description

_robust_ is a programmer’s command that computes a robust variance estimator based on _varlist_ of equation-level scores and a covariance matrix. It produces estimators for ordinary data (each observation independent), clustered data (data not independent within groups, but independent across groups), and complex survey data from one stage of stratified cluster sampling.

_robust_ helps implement estimation commands and is rarely used. That is because other commands are implemented in terms of it and are easier and more convenient to use. For instance, if all you want to do is make your estimation command allow the _vce(robust)_ and _vce(cluster clustvar)_ options, see [R] _ml_. If you want to make your estimation command work with survey data, it is easier to make your command work with the _svy_ prefix—see [P] _program properties_—rather than to use _robust_.

If you really want to understand what _ml_ and _svy_ are doing, however, this is the section for you. Or, if you have an estimation problem that does not fit with the _ml_ or _svy_ framework, then _robust_ may be able to help.

Syntax

```
_robust_ _varlist_ [if] [in] [weight] [ , variance(matname) minus(#) ]
```

```
    strata(varname) psu(varname) cluster(varname) fpc(varname)
    subpop(varname) vsrs(matname) srssubpop zeroweight
```

_robust_ works with models that have all types of varlists, including those with factor variables and time-series operators; see [U] 11.4.3 _Factor variables_ and [U] 11.4.4 _Time-series varlists_.

_pweights, aweights, fweights, and iweights_ are allowed; see [U] 11.1.6 _weight_.

Options

_variance(matname)_ specifies a matrix containing the unadjusted “covariance” matrix, that is, the _D_ in _V = DMD_. The matrix must have its rows and columns labeled with the appropriate corresponding variable names, that is, the names of the _x’s_ in _xβ_. If there are multiple equations, the matrix must have equation names; see [P] _matrix rownames_. The _D_ matrix is overwritten with the robust covariance matrix _V_. If variance() is not specified, Stata assumes that _D_ has been posted using _ereturn post_; _robust_ will then automatically post the robust covariance matrix _V_ and replace _D_.

_minus(#)_ specifies _k = #_ for the multiplier _n/(n – k)_ of the robust variance estimator. Stata’s maximum likelihood commands use _k = 1_, and so does the _svy_ prefix. _regress, vce(robust)_ uses, by default, this multiplier with _k_ equal to the number of explanatory variables in the model, including the constant. The default is _minus(1)_.

See _Methods and formulas_ for details.
robust — Robust variance estimates

strata(varname) specifies the name of a variable (numeric or string) that contains stratum identifiers.

psu(varname) specifies the name of a variable (numeric or string) that contains identifiers for the primary sampling unit (PSU). psu() and cluster() are synonyms; they both specify the same thing.

ccluster(varname) is a synonym for psu().

fpc(varname) requests a finite population correction for the variance estimates. If the variable specified has values less than or equal to 1, it is interpreted as a stratum sampling rate \( f_h = n_h/N_h \), where \( n_h \) is the number of PSUs sampled from stratum \( h \) and \( N_h \) is the total number of PSUs in the population belonging to stratum \( h \). If the variable specified has values greater than 1, it is interpreted as containing \( N_h \).

subpop(varname) specifies that estimates be computed for the single subpopulation defined by the observations for which \( varname \neq 0 \) (and is not missing). This option would typically be used only with survey data; see [SVY] subpopulation estimation.

vsrs(matname) creates a matrix containing \( \hat{V}_{srswor} \), an estimate of the variance that would have been observed had the data been collected using simple random sampling without replacement. This is used to compute design effects for survey data; see [SVY] estat for details.

srssubpop can be specified only if vsrs() and subpop() are specified. srssubpop requests that the estimate of simple-random-sampling variance, vsrs(), be computed assuming sampling within a subpopulation. If srssubpop is not specified, it is computed assuming sampling from the entire population.

zeroweight specifies whether observations with weights equal to zero should be omitted from the computation. This option does not apply to frequency weights; observations with zero frequency weights are always omitted. If zeroweight is specified, observations with zero weights are included in the computation. If zeroweight is not specified (the default), observations with zero weights are omitted. Including the observations with zero weights affects the computation in that it may change the counts of PSUs (clusters) per stratum. Stata’s svy prefix command includes observations with zero weights; all other commands exclude them. This option is typically used only with survey data.

Remarks and examples

Remarks are presented under the following headings:

- Introduction
- Formulas and simple examples
- Clustered data
- Survey data
- Controlling the header display
- Maximum likelihood estimators
- Multiple-equation estimators

Introduction

Before reading this section, you should be familiar with [U] 20.22 Obtaining robust variance estimates and the Methods and formulas section of [R] regress. We assume that you have already programmed an estimator in Stata and now wish to have it compute robust variance estimates. If you have not yet programmed your estimator, see [U] 18 Programming Stata, [R] ml, and [P] ereturn.
The robust variance estimator goes by many names: Huber/White/sandwich are typically used in the context of robustness against heteroskedasticity. Survey statisticians often refer to this variance calculation as a first-order Taylor-series linearization method. Despite the different names, the estimator is the same.

The equation-level score variables (varlist) consist of one variable for single-equation models or multiple variables for multiple-equation models, one variable for each equation. The “covariance” matrix before adjustment is either posted using ereturn post (see [P] ereturn) or specified with the variance(matname) option. In the former case, _robust replaces the covariance in the post with the robust covariance matrix. In the latter case, the matrix matname is overwritten with the robust covariance matrix.

If you wish to program an estimator for survey data, then you should write the estimator for nonsurvey data first and then use the instructions in [P] program properties (making programs svyable) to get your estimation command to work properly with the svy prefix. See [SVY] variance estimation for a discussion of variance estimation for survey data.

Formulas and simple examples

This section explains the formulas behind the robust variance estimator and how to use _robust through an informal development with some simple examples. For an alternative discussion, see [U] 20.22 Obtaining robust variance estimates. See the references cited at the end of this entry for more formal expositions.

First, consider ordinary least-squares regression. The estimator for the coefficients is

$$\hat{\beta} = (X'X)^{-1}X'y$$

where y is an \(n \times 1\) vector representing the dependent variable and X is an \(n \times k\) matrix of covariates.

Because everything is considered conditional on X, \((X'X)^{-1}\) can be regarded as a constant matrix. Hence, the variance of \(\hat{\beta}\) is

$$V(\hat{\beta}) = (X'X)^{-1}V(X'y)(X'X)^{-1}$$

What is the variance of \(X'y\), a \(k \times 1\) vector? Look at its first element; it is

$$X_1'y = x_{11}y_1 + x_{21}y_2 + \cdots + x_{n1}y_n$$

where \(X_1\) is the first column of \(X\). Because \(X\) is treated as a constant, you can write the variance as

$$V(X_1'y) = x_{11}^2V(y_1) + x_{21}^2V(y_2) + \cdots + x_{n1}^2V(y_n)$$

The only assumption made here is that the \(y_j\) are independent.

The obvious estimate for \(V(y_j)\) is \(\hat{e}_j^2\), the square of the residual \(\hat{e}_j = y_j - x_j\hat{\beta}\), where \(x_j\) is the \(j\)th row of \(X\). You must estimate the off-diagonal terms of the covariance matrix for \(X'y\), as well. Working this out, you have

$$\hat{V}(X'y) = \sum_{j=1}^{n} \hat{e}_j^2 x_j'x_j$$

\(x_j\) is defined as a row vector so that \(x_j'x_j\) is a \(k \times k\) matrix.
You have just derived the robust variance estimator for linear regression coefficient estimates for independent observations:

$$\hat{V}(\hat{\beta}) = (X'X)^{-1} \left( \sum_{j=1}^{n} \hat{e}_j^2 x'_j x_j \right) (X'X)^{-1}$$

You can see why it is called the sandwich estimator.

**Technical note**

The only detail not discussed is the multiplier. You will see later that survey statisticians like to view the center of the sandwich as a variance estimator for totals. They use a multiplier of $n/(n - 1)$, just as $1/(n - 1)$ is used for the variance estimator of a mean. However, for survey data, $n$ is no longer the total number of observations but is the number of clusters in a stratum. See *Methods and formulas* at the end of this entry.

Linear regression is, however, special. Assuming homoskedasticity and normality, you can derive the expectation of $\hat{e}_j^2$ for finite $n$. This is discussed in [R] `regress`. Under the assumptions of homoskedasticity and normality, $n/(n - k)$ is a better multiplier than $n/(n - 1)$.

If you specify the minus(#) option, _robust_ will use $n/(n - #)$ as the multiplier. `regress, vce(robust)` also gives two other options for the multiplier: `hc2` and `hc3`. Because these multipliers are special to linear regression, _robust_ does not compute them.

**Example 1**

Before we show how _robust_ is used, let’s compute the robust variance estimator “by hand” for linear regression for the case in which observations are independent (that is, no clusters).

We need to compute $D = (X'X)^{-1}$ and the residuals $\hat{e}_j$. `regress` with the `mse1` option will allow us to compute both easily; see [R] `regress`.

```
. use http://www.stata-press.com/data/r15/_robust
(1978 Automobile Data -- modified)
. regress mpg weight gear_ratio foreign, mse1
(output omitted)
. matrix D = e(V)
. predict double e, residual
```

We can write the center of the sandwich as

$$M = \sum_{j=1}^{n} \hat{e}_j^2 x'_j x_j = X'WX$$

where $W$ is a diagonal matrix with $\hat{e}_j^2$ on the diagonal. `matrix accum` with `iweights` can be used to calculate this (see [P] `matrix accum`):

```
. matrix accum M = weight gear_ratio foreign [iweight=e^2]
(obs=813.7814109)
```
We now assemble the sandwich. To match `regress, vce(robust)`, we use a multiplier of $n/(n-k)$.

\[
\begin{align*}
\text{matrix } & V = 74/70 \ast D \ast M \ast D \\
\text{matrix list } & V
\end{align*}
\]

\[
\begin{array}{cccc}
\text{weight} & \text{gear_ratio} & \text{foreign} & \_\text{cons} \\
\text{weight} & 3.788e-07 \\
\text{gear_ratio} & 0.00039798 & 1.9711317 \\
\text{foreign} & 0.0008463 & -.55488334 & 1.4266939 \\
\_\text{cons} & -.00236851 & -6.9153285 & 1.2149035 & 27.536291
\end{array}
\]

The result is the same as that from `regress, vce(robust)`:

\[
\begin{align*}
\text{matrix } & V_{\text{reg}} = e(V) \\
\text{matrix list } & V_{\text{reg}}
\end{align*}
\]

\[
\begin{array}{cccc}
\text{weight} & \text{gear_ratio} & \text{foreign} & \_\text{cons} \\
\text{weight} & 3.788e-07 \\
\text{gear_ratio} & 0.00039798 & 1.9711317 \\
\text{foreign} & 0.0008463 & -.55488334 & 1.4266939 \\
\_\text{cons} & -.00236851 & -6.9153285 & 1.2149035 & 27.536291
\end{array}
\]

If we use `_robust`, the initial steps are the same. We still need $D$, the “bread” of the sandwich, and the residuals. The residuals $e$ are the varlist for `_robust`. $D$ is overwritten and contains the robust variance estimate.

\[
\begin{align*}
\text{drop } & e \\
\text{regress mpg weight gear_ratio foreign, mse1} \\
\text{matrix } & D = e(V) \\
\text{predict double } & e, \text{ residual} \\
\text{_robust } & e, \text{ v(D) minus(4)} \\
\text{matrix list } & D
\end{align*}
\]

\[
\begin{array}{cccc}
\text{weight} & \text{gear_ratio} & \text{foreign} & \_\text{cons} \\
\text{weight} & 3.788e-07 \\
\text{gear_ratio} & 0.00039798 & 1.9711317 \\
\text{foreign} & 0.0008463 & -.55488334 & 1.4266939 \\
\_\text{cons} & -.00236851 & -6.9153285 & 1.2149035 & 27.536291
\end{array}
\]

Rather than specifying the `variance()` option, we can use `ereturn post` to post $D$ and the point estimates. `_robust` alters the post, substituting the robust variance estimates.

\[
\begin{align*}
\text{drop } & e \\
\text{regress mpg weight gear_ratio foreign, mse1} \\
\text{matrix } & D = e(V) \\
\text{matrix } & b = e(b) \\
\text{local } & n = e(N) \\
\text{local } & k = \text{colsof(D)} \\
\text{local } & \text{dof} = \text{‘n’} - \text{‘k’} \\
\text{predict double } & e, \text{ residual} \\
\text{ereturn post } & b, D, \text{ dof(‘dof’)} \\
\text{_robust } & e, \text{ minus(‘k’)}
\end{align*}
\]
Again what we did matches `regress, vce(robust)`:

```stata
. regress mpg weight gear_ratio foreign, vce(robust)
```

Linear regression  
Number of obs = 74  
F(3, 70) = 48.30  
Prob > F = 0.0000  
R-squared = 0.6670  
Root MSE = 3.4096

| Coef. Std. Err. | t | P>|t| | [95% Conf. Interval] |
|----------------|---|-----|-----------------------------|
| weight | -0.006139 | 0.0006155 | -9.97 | 0.000 | -0.0073666 | -0.0049115 |
| gear_ratio | 1.457113 | 1.40397 | 1.04 | 0.303 | -1.343016 | 4.257243 |
| foreign | -2.221682 | 1.194443 | -1.86 | 0.067 | -4.603923 | 0.1605598 |
| _cons | 36.10135 | 5.247503 | 6.88 | 0.000 | 25.63554 | 46.56717 |

Technical note

Note the simple ways in which `_robust` was called. When we used the `variance()` option, we called it by typing

```
. _robust e, v(D) minus(4)
```

As we described, `_robust` computed

$$
\hat{V}(\hat{\beta}) = D \left( \frac{n}{n-k} \sum_{j=1}^{n} \hat{e}_j^2 x'_j x_j \right) D
$$

We passed $D$ to `_robust` by using the `v(D)` option and specified $\hat{e}_j$ as the variable `e`. So how did `_robust` know what variables to use for $x_j$? It got them from the row and column names of the matrix $D$. Recall how we generated $D$ initially:

```
. regress mpg weight gear_ratio foreign, mse1  
(output omitted)  
. matrix D = e(V)  
. matrix list D  
symmetric D[4,4]  
weight gear_ratio foreign _cons  
weight  5.436e-08  
gear_ratio  0.000629 0.20434146  
foreign  0.0001032 0.08016692 0.1311889  
_cons  0.0035697 0.782292 0.17154326 3.3988878
```
Stata’s estimation commands and the \texttt{ml} commands produce matrices with appropriately labeled rows and columns. If that is how we generate our \texttt{D}, this will be taken care of automatically. But if we generate \texttt{D} in another manner, we must be sure to label it appropriately; see \cite{matrix rownames}.

When \_\texttt{robust} is used after \texttt{ereturn post}, it gets the variable names from the row and column names of the posted matrices. So again, the matrices must be labeled appropriately.

Let us make another rather obvious comment. \_\texttt{robust} uses the variables from the row and column names of the \texttt{D} matrix at the time \_\texttt{robust} is called. It is the programmer’s responsibility to ensure that the data in these variables have not changed and that \_\texttt{robust} selects the appropriate observations for the computation, using an \texttt{if} restriction if necessary (for instance, \texttt{if e(sample)}).

---

### Clustered data

#### Example 2

To get robust variance estimates for clustered data or for complex survey data, simply use the \texttt{cluster()}, \texttt{strata()}, etc., options when you call \_\texttt{robust}.

The first steps are the same as before. For clustered data, the number of degrees of freedom of the \textit{t} statistic is the number of clusters minus one (we will discuss this later).

```stata
. drop e  
. quietly regress mpg weight gear_ratio foreign, mse1  
. generate byte samp = e(sample)  
. matrix D = e(V)  
. matrix b = e(b)  
. predict double e, residual  
. local k = colsof(D)  
. tabulate rep78

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2.90</td>
<td>2.90</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>11.59</td>
<td>14.49</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>43.48</td>
<td>57.97</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>26.09</td>
<td>84.06</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>15.94</td>
<td>100.00</td>
</tr>
<tr>
<td>Total</td>
<td>69</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>
```

. local nclust = r(r)  
. display ‘nclust’  
5

. local dof = ‘nclust’ - 1  
. ereturn post b D, dof(‘dof’) esample(samp)  
. _robust e, minus(‘k’) cluster(rep78)
What you get is, of course, the same as `regress, vce(cluster rep78)`. Wait a minute. It is not the same!

```
. regress mpg weight gear_ratio foreign, vce(cluster rep78)
Linear regression
Number of obs = 69
F(3, 4) = 78.61
Prob > F = 0.0005
R-squared = 0.6631
Root MSE = 3.4827
(Std. Err. adjusted for 5 clusters in rep78)

mpg          Coef.     Std. Err.     t    P>|t|     [95% Conf. Interval]
-------------  --------  --------  ------  --------  ------------------
weight        -0.05893   0.008214    -7.17  0.002     -0.081735    -0.036126
gear_ratio    1.904503   2.183222     0.87  0.432     -4.157088    7.966093
foreign       -2.149017   1.204890    -1.78  0.149     -5.49433    1.196295
_cons          34.09959   4.215275     8.09  0.001     22.39611    45.80307
```

Not even the point estimates are the same. This is the classic programmer’s mistake of not using the same sample for the initial `regress, mse1` call as done with `_robust`. The cluster variable `rep78` is missing for 5 observations. `_robust` omitted these observations, but `regress, mse1` did not.

 `_robust` is best used only in programs for just this reason. So, you can write a program and use `marksample` and `markout` (see `[P] mark`) to determine the sample in advance of running `regress` and `_robust`. 
program myreg, eclass sortpreserve
version 15.1
syntax varlist [if] [in] [ , Ccluster(varname) ]
marksample touse
markout 'touse' 'cluster', strok
tempvar e count
tempname D b
quietly {
    regress 'varlist' if 'touse', mse1
    matrix 'D' = e(V)
    matrix 'b' = e(b)
    local n = e(N)
    local k = colsof('D')
predict double 'e' if 'touse', residual
if "'cluster'"!="" {
    sort 'touse' 'cluster'
    by 'touse' 'cluster': gen byte 'count' = 1 if _n==1 & 'touse'
    summarize 'count', meanonly
    local nclust = r(sum)
    local dof = 'nclust' - 1
    local clopt "cluster('cluster')"
}
else
    local dof = 'n' - 'k'
ereturn post 'b' 'D', dof('dof') esample('touse')
_robust 'e' if e(sample), minus('k') 'clopt'
}
ereturn display
end

Running this program produces the same results as regress, vce(cluster clustvar).

. myreg mpg weight gear_ratio foreign, cluster(rep78)
(Std. Err. adjusted for 5 clusters in rep78)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Robust</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Coef.</td>
<td>Std. Err.</td>
<td>t</td>
<td>P&gt;</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>-.005893</td>
<td>.0008214</td>
<td>-7.17</td>
<td>0.002</td>
<td>-.0081735</td>
<td>-.0036126</td>
</tr>
<tr>
<td>gear_ratio</td>
<td>1.904503</td>
<td>2.18322</td>
<td>0.87</td>
<td>0.432</td>
<td>-4.157088</td>
<td>7.966093</td>
</tr>
<tr>
<td>foreign</td>
<td>-2.149017</td>
<td>1.20489</td>
<td>-1.78</td>
<td>0.149</td>
<td>-5.49433</td>
<td>1.196295</td>
</tr>
<tr>
<td>_cons</td>
<td>34.09959</td>
<td>4.215275</td>
<td>8.09</td>
<td>0.001</td>
<td>22.39611</td>
<td>45.80307</td>
</tr>
</tbody>
</table>
Survey data

Example 3

We will now modify our `myreg` command so that it handles complex survey data. Our new version will allow pweights and iweights, stratification, and clustering.

```
begin myreg.ado

program myreg, eclass
version 15.1
syntax varlist [if] [in] [pweight iweight] [ , /*
* STRata(varname) CLuster(varname) ]
marksample touse, zeroweight
markout `touse' `cluster' `strata', strok
if "'weight'"!=" {
    tempvar w
    quietly generate double `w' `exp' if `touse'
    local iwexp `[iw=`w']"
    if "'weight'" == "pweight" {
        capture assert `w' >= 0 if `touse'
        if c(rc) error 402
    }
}
if "'cluster'"!=" {
    local clopt "cluster('cluster')"
}
if "'strata'"!=" {
    local stopt "strata('strata')"
}

tempvar e
tempname D b
regress `varlist' `iwexp' if `touse', mse1
matrix `D' = e(V)
matrix `b' = e(b)
predict double `e' if `touse', residual
-_robust `e' `iwexp' if `touse', v('D') clopt 'stopt' zeroweight
local dof = r(N_clust) - r(N_strata)
local depn : word 1 of `varlist'
ereturn post `b' 'D', depn('depn') dof('dof') esample('touse')
}
display
ereturn display
end myreg.ado
```

Note the following details about our version of `myreg` for survey data:

- We called `_robust` before we posted the matrices with `ereturn post`, whereas in our previous version of `myreg`, we called `ereturn post` and then `_robust`. Here we called `_robust` first so that we could use its `r(N_strata)`, containing the number of strata, and `r(N_clust)`, containing the number of clusters; see Stored results at the end of this entry. We did this so that we could pass the correct degrees of freedom (= number of clusters − number of strata) to `ereturn post`. This works even if the `strata()` and `cluster()` options are not specified: `r(N_strata) = 1` if `strata()` is not specified (there truly is one stratum); and `r(N_clust) = number of observations` if cluster() is not specified (each observation is a cluster).

- The call to `_robust` was made with iweights, whether `myreg` was called with pweights or iweights. Computationally, `_robust` treats pweights and iweights the same. The only difference is that it puts out an error message if it encounters a negative pweight, whereas
negative *i*weights are allowed. As good programmers, we put out the error message early before any time-consuming computations are done.

- We used the *zeroweight* option with the *marksample* command so that zero weights would not be excluded from the sample. We gave the *zeroweight* option with *_robust* so that it, too, would not exclude zero weights.

Observations with zero weights affect results only by their effect (if any) on the counts of the clusters. Setting some weights temporarily to zero will, for example, produce subpopulation estimates. If subpopulation estimates are desired, however, it would be better to implement *_robust*’s *subpop()* option and restrict the call to *regress, mse1* to this subpopulation.

- Stata’s *svyset* accepts a *psu* variable rather than having a *cluster()* option. This is only a matter of style. They are synonyms, as far as *_robust* is concerned.

Our program gives the same results as *svy: regress*. For our example, we add a *strata* variable and a *psu* variable to the *auto* dataset.

```
.set seed 1
.generate strata = int(3*runiform()) + 1
.generate psu = int(5*runiform()) + 1
.myreg mpg weight gear_ratio foreign [pw=displ], strata(strata) cluster(psu)
```

```
Coef. Std. Err. t  P>|t|  [95% Conf. Interval]
weight     -.0057248  .000388 -14.75 0.000  -.0065702 -.0048794
gear_ratio  .7775839  1.20131  0.65 0.530 -1.839845  3.395013
foreign    -1.86776  1.122833 -1.66 0.122 -4.314202  .5786828
_cons       36.64061  3.844625  9.53 0.000  28.26389  45.01733
```

```
.svyset psu [pw=displ], strata(strata)
.pweight: displacement
.VCE: linearized
.Single unit: missing
.Strata 1: strata
.SU 1: psu
.FPC 1: <zero>
.svy: regress mpg weight gear_ratio foreign
(running regress on estimation sample)
```

Survey: Linear regression

```
Number of strata  =   3  Number of obs  =   74
Number of PSUs    =  15  Population size  =  14,600
Design df          =   12  F(  3,   10) =   68.37
Prob > F           = 0.0000  R-squared       =  0.6900
```

```
Coef. Std. Err. t  P>|t|  [95% Conf. Interval]
weight     -.0057248  .000388 -14.75 0.000  -.0065702 -.0048794
gear_ratio  .7775839  1.20131  0.65 0.530 -1.839845  3.395013
foreign    -1.86776  1.122833 -1.66 0.122 -4.314202  .5786828
_cons       36.64061  3.844625  9.53 0.000  28.26389  45.01733
```

---

\[\text{Robust variance estimates}\]
Controlling the header display

Example 4

Let’s compare the output for our survey version of `myreg` with the earlier version that handled only clustering. The header for the earlier version was

```
(Std. Err. adjusted for 5 clusters in rep78)
```

| Robust | Coef. | Std. Err. | t | P>|t| | [95% Conf. Interval] |
|--------|-------|-----------|---|------|----------------------|

The header for the survey version lacked the word “Robust” above “Std. Err.”, and it lacked the banner “(Std. Err. adjusted for # clusters in varname)”.

Both of these headers were produced by `ereturn display`, and programmers can control what it produces. The word above “Std. Err.” is controlled by setting `e(vcetype)`. The banner “(Std. Err. adjusted for # clusters in varname)” is controlled by setting `e(clustvar)` to the cluster variable name. These can be set using the `ereturn local` command; see `[P] ereturn`.

When `_robust` is called after `ereturn post` (as it was in the earlier version that produced the above header), it automatically sets these macros. To not display the banner, the code should read

```
ereturn post ... 
_robust ... 
ereturn local clustvar ""
```

We can also change the phrase displayed above “Std. Err.” by resetting `e(vcetype)`. To display nothing there, reset `e(vcetype)` to empty—`ereturn local vcetype ""`.

For our survey version of `myreg`, we called `_robust` before calling `ereturn post`. Here `_robust` does not set these macros. Trying to do so would be futile because `ereturn post` clears all previous estimation results, including all `e()` macros, but you can set them yourself after calling `ereturn post`. We make this addition to our survey version of `myreg`:

```
_robust ... 
ereturn post ... 
ereturn local vcetype "Design-based"
```

The output is

```
. myreg mpg weight gear_ratio foreign [pw=displ], strata(strata) cluster(psu)

| mpg | Coef. | Std. Err. | t | P>|t| | [95% Conf. Interval] |
|-----|-------|-----------|---|------|----------------------|
| weight | -.0057248 | .000388 | -14.75 | 0.000 | -.0065702 | -.0048794 |
| gear_ratio | .7775839 | 1.20131 | 0.65 | 0.530 | -1.839845 | 3.395013 |
| foreign | -1.86776 | 1.122833 | -1.66 | 0.122 | -4.314202 | .5786828 |
| _cons | 36.64061 | 3.844625 | 9.53 | 0.000 | 28.26389 | 45.01733 |
```
Maximum likelihood estimators

Maximum likelihood estimators are basically no different from linear regression when it comes to the use of _robust_. We will first do a little statistics and then give a simple example.

We can write our maximum-likelihood estimation equation as

\[ G(\beta) = \sum_{j=1}^{n} S(\beta; y_j, x_j) = 0 \]

where \( S(\beta; y_j, x_j) = \partial \ln L_j / \partial \beta \) is the score and \( \ln L_j \) is the log likelihood for the \( j \)th observation. Here \( \beta \) represents all the parameters in the model, including any auxiliary parameters. We will discuss how to use _robust_ when there are auxiliary parameters or multiple equations in the next section. But for now, all the theory works out fine for any set of parameters.

Using a first-order Taylor-series expansion (that is, the delta method), we can write the variance of \( G(\beta) \) as

\[ \hat{V}\{G(\beta)\}_{\beta=\hat{\beta}} = \frac{\partial G(\beta)}{\partial \beta} \bigg|_{\beta=\hat{\beta}} \hat{V}\{\beta\} \frac{\partial G(\beta)}{\partial \beta} \bigg|_{\beta=\hat{\beta}} \]

Solving for \( \hat{V}(\beta) \) gives

\[ \hat{V}(\beta) = \left[ \left\{ \frac{\partial G(\beta)}{\partial \beta} \right\}^{-1} \hat{V}\{G(\beta)\} \left\{ \frac{\partial G(\beta)}{\partial \beta} \right\}^{-1} \right]_{\beta=\hat{\beta}} \]

but

\[ H = \frac{\partial G(\beta)}{\partial \beta} \]

is the Hessian (matrix of second derivatives) of the log likelihood. Thus we can write

\[ \hat{V}(\beta) = D \hat{V}\{G(\beta)\}_{\beta=\hat{\beta}} D \]

where \( D = -H^{-1} \) is the traditional covariance estimate.

Now \( G(\beta) \) is simply a sum, and we can estimate its variance just as we would the sum of any other variable—it is \( n^2 \) times the standard estimator of the variance of a mean:

\[ \frac{n}{n-1} \sum_{j=1}^{n} (z_j - \bar{z})^2 \]

But here, the scores \( u_j = S(\hat{\beta}; y_j, x_j) \) are (row) vectors. Their sum, and thus their mean, is zero. So, we have

\[ \hat{V}\{G(\beta)\}_{\beta=\hat{\beta}} = \frac{n}{n-1} \sum_{j=1}^{n} u_j' u_j \]

Thus our robust variance estimator is

\[ \hat{V}(\beta) = D \left( \frac{n}{n-1} \sum_{j=1}^{n} u_j' u_j \right) D \]
so we see that the robust variance estimator is just the delta method combined with a simple estimator for totals!

The above estimator for the variance of the total (the center of the sandwich) is appropriate only when observations are independent. For clustered data and complex survey data, this estimator is replaced by one appropriate for the independent units of the data. Clusters (or PSUs) are independent, so we can sum the scores within a cluster to create a “superobservation” and then use the standard formula for a total on these independent superobservations. Our robust variance estimator thus becomes

\[
\hat{V}(\hat{\beta}) = D \left\{ \frac{n_c}{n_c - 1} \sum_{i=1}^{n_c} \left( \sum_{j \in C_i} u_j \right)' \left( \sum_{j \in C_i} u_j \right) \right\} D
\]

where \(C_i\) contains the indices of the observations belonging to the \(i\)th cluster for \(i = 1, 2, \ldots, n_c\), with \(n_c\) the total number of clusters.

See \([SVY]\) variance estimation for the variance estimator for a total that is appropriate for complex survey data. Our development here has been heuristic. We have, for instance, purposefully omitted sampling weights from our discussion; see \([SVY]\) variance estimation for a better treatment.

See Gould, Pitblado, and Poi (2010) for a discussion of maximum likelihood and of Stata’s \texttt{ml} command.

\(\blacksquare\) Technical note

It is easy to see where the appropriate degrees of freedom for the robust variance estimator come from: the center of the sandwich is \(n^2\) times the standard estimator of the variance for the mean of \(n\) observations. A mean divided by its standard error has exactly a Student’s \(t\) distribution with \(n - 1\) degrees of freedom for normal i.i.d. variables but also has approximately this distribution under many other conditions. Thus a point estimate divided by the square root of its robust variance estimate is approximately distributed as a Student’s \(t\) with \(n - 1\) degrees of freedom.

More importantly, this also applies to clusters, where each cluster is considered a “superobservation”. Here the degrees of freedom is \(n_c - 1\), where \(n_c\) is the number of clusters (superobservations). If there are only a few clusters, confidence intervals using \(t\) statistics can become quite large. It is just like estimating a mean with only a few observations.

When there are strata, the degrees of freedom is \(n_c - L\), where \(L\) is the number of strata; see \([SVY]\) variance estimation for details.

Not all of Stata’s maximum likelihood estimators that produce robust variance estimators for clustered data use \(t\) statistics. Obviously, this matters only when the number of clusters is small. Users who want to be rigorous in handling clustered data should use the \texttt{svy} prefix, which always uses \(t\) statistics and adjusted Wald tests (see \([R]\) \texttt{test}). Programmers who want to impose similar rigor should do likewise.

\(\blacksquare\)

We have not yet given any details about the functional form of our scores \(u_j = \partial \ln L_j / \partial \beta\). The log likelihood \(\ln L_j\) is a function of \(x_j \beta\) (the “index”). Logistic regression, probit regression, and Poisson regression are examples. There are no auxiliary parameters, and there is only one equation.

We can then write \(u_j = \hat{s}_j x_j\), where

\[
\hat{s}_j = \left. \frac{\partial \ln L_j}{\partial (x_j \beta)} \right|_{\beta = \hat{\beta}}
\]
We refer to $s_j$ as the equation-level score. Our formula for the robust estimator when observations are independent becomes

$$\hat{V}(\hat{\beta}) = D \left( \frac{n}{n-1} \sum_{j=1}^{n} \hat{s}_j^2 x'_j x_j \right) D$$

This is precisely the formula that we used for linear regression, with $\hat{e}_j$ replaced by $\hat{s}_j$ and $k = 1$ in the multiplier.

Before we discuss auxiliary parameters, let's show how to implement \_robust for single-equation models.

**Example 5**

The robust variance implementation for single-equation maximum-likelihood estimators with no auxiliary parameters is almost the same as it is for linear regression. The only differences are that $D$ is now the traditional covariance matrix (the negative of the inverse of the matrix of second derivatives) and that the variable passed to \_robust is the equation-level score $\hat{s}_j$ rather than the residuals $\hat{e}_j$.

Let's alter our last myreg program for survey data to make a program that does logistic regression for survey data. We have to change only a few lines of the program.

```
begin mylogit.ado

program mylogit, eclass
version 15.1
syntax varlist [if] [in] [pweight] [, /*
*/ STRata(varname) CLuster(varname) ]
marksample touse, zeroweight
markout 'touse' 'strata' 'cluster', strok
if "'weight'"!="" {
    tempvar w
    quietly generate double 'w' 'exp' if 'touse'
    local iwexp "[iw='w']"
    capture assert 'w' >= 0 if 'touse'
    if c(rc) error 402
}
if "'cluster'"!="" {
    local clopt "cluster('cluster')"
}
if "'strata'"!="" {
    local stopt "strata('strata')"
}
tempvar s
tempname D b
quietly {
    logit 'varlist' 'iwexp' if 'touse'
    matrix 'D' = e(V)
    matrix 'b' = e(b)
    predict double 's' if e(sample), score
    _robust 's' 'iwexp' if e(sample), v('D') 'clopt' 'stopt' zeroweight
    local dof = r(N_clust) - r(N_strata)
    local depn : word 1 of 'varlist'
    replace 'touse' = e(sample)
    ereturn post 'b' 'D', depn('depn') dof('dof') esample('touse')
    ereturn local vcetype "Design-based"
}
display
ereturn display
end
```

Note the following about our program:

- We use the `score` option of `predict` after `logit` to obtain the equation-level scores. If `predict` does not have a `score` option, then we must generate the equation-level score variable some other way.

- `logit` is a unique command in that it will sometimes drop observations for reasons other than missing values (for example, when success or failure is predicted perfectly), so our ‘touse’ variable may not represent the true estimation sample. That is why we used the `if e(sample)` condition with the `predict` and `_robust` commands. Then, to provide `ereturn post` with an appropriate `esample()` option, we set the ‘touse’ variable equal to the `e(sample)` from the `logit` command and then use this ‘touse’ variable in the `esample()` option.

Our `mylogit` program gives the same results as `svy: logit`:

```
    . mylogit foreign mpg weight gear_ratio [pw=displ], strata(strata) cluster(psu)

                  Design-based                     [95% Conf. Interval]
                   Coef.    Std. Err.     t    P>|t|     [5% Conf. Interval]
          foreign
             mpg   -.3489011    .1258802   -2.77   0.017    -.6231705   -.0746317
            weight   -.0040789    .0012508   -3.26   0.007    -.0068042   -.0013536
          gear_ratio   6.324169    1.729436    3.66   0.003    2.556051   10.09229
          _cons   -2.189748    7.754278   -0.28   0.782   -19.08485    14.70536
```

```
    . svyset psu [pw=displ], strata(strata)
    pweight: displacement
     VCE: linearized
    Single unit: missing
    Strata 1: strata
    SU 1: psu
    FPC 1: <zero>
    . svy: logit foreign mpg weight gear_ratio
    (running logit on estimation sample)

Survey: Logistic regression
Number of strata   =   3
Number of PSUs     =   15
Population size    = 14,600
Design df          =   12
F(  3,   10)       =   6.89
Prob > F           =   0.0085

                  Linearized                     [95% Conf. Interval]
                   Coef.    Std. Err.     t    P>|t|     [5% Conf. Interval]
          foreign
             mpg   -.3489011    .1258802   -2.77   0.017    -.6231705   -.0746317
            weight   -.0040789    .0012508   -3.26   0.007    -.0068042   -.0013536
          gear_ratio   6.324169    1.729436    3.66   0.003    2.556051   10.09229
          _cons   -2.189748    7.754278   -0.28   0.782   -19.08485    14.70536
```

## Technical note

The theory developed here applies to full-information maximum-likelihood estimators. Conditional likelihoods, such as conditional (fixed-effects) logistic regression (`clogit`) and Cox regression (`stcox`), use variants on this theme. The `vce(robust)` option on `stcox` uses a similar, but not identical, formula; see [ST] `stcox` and Lin and Wei (1989) for details.
On the other hand, the theory developed here applies not only to maximum likelihood estimators but also to general estimating equations:

$$G(\beta) = \sum_{j=1}^{n} g(\beta; y_j, x_j) = 0$$

See Binder (1983) for a formal development of the theory.

Programmers: You are responsible for the theory behind your implementation.

Multiple-equation estimators

The theory for auxiliary parameters and multiple-equation models is no different from that described earlier. For independent observations, just as before, the robust variance estimator is

$$\hat{V}(\hat{\beta}) = D \left( \frac{n}{n-1} \sum_{j=1}^{n} u'_j u_j \right) D$$

where $u_j = \partial \ln L_j / \partial \beta$ is the score (row) vector and $D$ is the traditional covariance estimate (the negative of the inverse of the matrix of second derivatives).

With auxiliary parameters and multiple equations, $\beta$ can be viewed as the vector of all the parameters in the model. Without loss of generality, you can write the log likelihood as

$$\ln L_j = \ln L_j(x_j^{(1)} \beta^{(1)}, x_j^{(2)} \beta^{(2)}, \ldots, x_j^{(p)} \beta^{(p)})$$

An auxiliary parameter is regarded as $x_j^{(i)} \beta^{(i)}$ with $x_j \equiv 1$ and $\beta^{(i)}$ a scalar. The score vector becomes

$$u_j = (s_j^{(1)} x_j^{(1)}, s_j^{(2)} x_j^{(2)}, \ldots, s_j^{(p)} x_j^{(p)})$$

where $s_j^{(i)} = \partial \ln L_j / \partial (x_j \beta^{(i)})$ is the equation-level score for the $i$th equation.

This notation has been introduced so that it is clear how to call \_robust. You use

```
    . _robust s_j^{(1)} s_j^{(2)} \ldots s_j^{(p)}, options
```

where $s_j^{(1)}$, etc., are variables that contain the equation-level score values. The $D$ matrix that you pass to \_robust or post with \_ereturn \_post must be labeled with exactly $p$ equation names.

\_robust takes the first equation-level score variable, $s_j^{(1)}$, and matches it to the first equation on the $D$ matrix to determine $x_j^{(1)}$, takes the second equation-level score variable and matches it to the second equation, etc. Some examples will make this perfectly clear.

Example 6

Here is what a matrix with equation names looks like, ending with a call to \_robust

```
    . generate cat = rep78 - 3
    (5 missing values generated)
    . replace cat = 2 if cat < 0
    (10 real changes made)
    . mlogit cat price foreign, base(0)
    (output omitted)
    . matrix D = e(V)
```
robust — Robust variance estimates

```
. matrix list D
symmetric D[9,9]
  0.  0.  0.  1.  1.
  o.  o.  o.  o.  o.
  price foreign _cons price foreign
0:  0.  0.  0.  1.24e-08
0:o.price 0
0:o.foreign 0 0
0:o._cons 0 0 0 1.240e-08
1:price 0 0 0 1.240e-08
1:foreign 0 0 0 -1.401e-06
1:_cons 0 0 0 -1.3992997
2:price 0 0 0 4.265e-09
2:foreign 0 0 0 -1.590e-06
2:_cons 0 0 0 -0.000265
1: 2: 2: 2:
_cons 61347545
price .61347545
foreign 1.207e-08
_cons -.00002693
2:price -.00002693
2:foreign -3.184e-06
2:_cons -3.184e-06
1:_cons .56833686
weight .00069222
foreign 2.2471526
gear_ratio .00011344
_eq1:_cons 1.4426905
_eq1:weight -8.6029018
_eq1:gear_ratio 1.8864693
_eq1:foreign -37.377729
_eq1:_cons .54017838
_sigma:_cons -3.527e-14
_weight -3.915e-10
_foreign -1.035e-10
_gear_ratio -4.552e-09
_cons .07430437
2:weight -.00392566
2:foreign -8.8159935
2:_cons 1.4426905
1:_cons -3.527e-14
```

where s1, s2, and s3 are the equation-level score variables.

Covariance matrices from models with auxiliary parameters look just like multiple-equation matrices. The second equation consists of the auxiliary parameter only. We again end with a call to `robust`.

```
. _robust s1 s2 s3, v(D)
```

Example 7

We will now give an example using `ml` and `robust` to produce an estimation command that has `vce(robust)` and `vce(cluster clustvar)` options. You can actually accomplish all of this easily by using `ml` without using the `robust` command because `ml` has `robust` and `cluster()` options. We will pretend that these two options are unavailable to illustrate the use of `robust`.

To keep the example simple, we will do linear regression as a maximum likelihood estimator. Here the log likelihood is

$$
\ln L_j = -\frac{1}{2} \left\{ \left( \frac{y_j - x_j \beta}{\sigma} \right)^2 + \ln(2\pi\sigma^2) \right\}
$$
There is an auxiliary parameter, $\sigma$, and thus we have two equation-level scores:

$$\frac{\partial \ln L_j}{\partial (x_j \beta)} = \frac{y_j - x_j \beta}{\sigma^2}$$

$$\frac{\partial \ln L_j}{\partial \sigma} = \frac{1}{\sigma} \left\{ \left( \frac{y_j - x_j \beta}{\sigma} \right)^2 - 1 \right\}$$

Here are programs to compute this estimator. We have two ado-files: `mymle.ado` and `likereg.ado`. The first ado-file contains two programs, `mymle` and `Scores`. `mymle` is the main program, and `Scores` is a subprogram that computes the equation-level scores after we compute the maximum likelihood solution. Because `Scores` is called only by `mymle`, we can nest it in the `mymle.ado` file; see [U] 17 Ado-files.
Our `robust` program computes the likelihood. Because it is called by Stata’s `ml` commands, we cannot nest it in the other file.

Note the following:

- Our command `mymle` will produce robust variance estimates if either the `robust` or the `cluster()` option is specified. Otherwise, it will display the traditional estimates.

- We used the `lf` method with `ml`; see [R] `ml`. We could have used the `d1` or `d2` methods. Because we would probably include code to compute the first derivatives analytically for the `vce(robust)` option, there is no point in using `d0`. (However, we could compute the first derivatives numerically and pass these to `robust`.)

- Our `Scores` program uses `predict` to compute the index $x_j \beta$. Because we had already posted the results using `ml`, `predict` is available to us. By default, `predict` computes the index for the first equation.

- Again because we had already posted the results by using `ml`, we can use `[sigma]_[_cons]` to get the value of $\sigma$; see [U] 13.5 Accessing coefficients and standard errors for the syntax used to access coefficients from multiple-equation models.

- `ml` calls `ereturn post`, so when we call `robust`, it alters the posted covariance matrix, replacing it with the robust covariance matrix. `robust` also sets `e(vcetype)`, and if the `cluster()` option is specified, it sets `e(clustvar)` as well.

- We let `ml` produce $z$ statistics, even when we specified the `cluster()` option. If the number of clusters is small, it would be better to use $t$ statistics. To do this, we could specify the `dof()` option on the `ml` command, but we would have to compute the number of clusters in advance. We could also get the number of clusters from `robust`’s `r(N_clust)` and then repost the matrices by using `ereturn repost`. 

If we run our command with the `cluster()` option, we get

```
   . mymle mpg weight gear_ratio foreign, cluster(rep78)

initial:  log likelihood = -219.4845
rescale:  log likelihood = -219.4845
rescale eq:  log likelihood = -219.4845
Iteration 0:  log likelihood = -219.4845  (not concave)
Iteration 1:  log likelihood = -207.02829  (not concave)
Iteration 2:  log likelihood = -202.6134
Iteration 3:  log likelihood = -190.01198
Iteration 4:  log likelihood = -181.94871
Iteration 5:  log likelihood = -181.94473
Iteration 6:  log likelihood = -181.94473

MLE linear regression  Number of obs = 69
Wald chi2(3) = 135.82
Log likelihood = -181.94473  Prob > chi2 = 0.0000

(Std. Err. adjusted for 5 clusters in rep78)

Robust
Coef.  Std. Err.  z  P>|z|  [95% Conf. Interval]

   eq1
weight        -.005893  .000803  -7.34  0.000  -.0074669  -.0043191
gear_ratio   1.904503  2.134518   0.89  0.372  -2.279075   6.08808
foreign     -2.149017  1.178012  -1.82  0.068  -4.457879   .1598441
   _cons     34.09959  4.121243   8.27  0.000   26.02211  42.17708

sigma
   _cons     3.380223  .8840543   3.82  0.000   1.647508   5.112937
```

These results are similar to the earlier results that we got with our first `myreg` program and `regress, vce(cluster rep78)`.

Our likelihood is not globally concave. Linear regression is not globally concave in $\beta$ and $\sigma$. `ml`'s `lf` convergence routine encountered a little trouble in the beginning but had no problem coming to the right solution.

---

**Stored results**

`_robust` stores the following in `r()`:

**Scalars**

- `r(N)`  number of observations
- `r(N_sub)`  subpopulation observations
- `r(N_strata)`  number of strata
- `r(N_clust)`  number of clusters (PSUs)
- `r singleton)`  1 if singleton strata, 0 otherwise
- `r(census)`  1 if census data, 0 otherwise
- `r(df_r)`  variance degrees of freedom
- `r(sum_w)`  sum of weights
- `r(N_subpop)`  number of observations for subpopulation (`subpop()` only)
- `r(sum_wsub)`  sum of weights for subpopulation (`subpop()` only)

**Macros**

- `r(subpop)`  `subpop` from `subpop()`

`r(N_strata)` and `r(N_clust)` are always set. If the `strata()` option is not specified, then `r(N_strata)=1` (there truly is one stratum). If neither the `cluster()` nor the `psu()` option is specified, then `r(N_clust)` equals the number of observations (each observation is a PSU).
When `robust` alters the post of `ereturn post`, it also stores the following in `e()`:

- `e(vcetype)`: Robust
- `e(clustvar)`: name of cluster (PSU) variable

`e(vcetype)` controls the phrase that `ereturn display` displays above “Std. Err.”; `e(vcetype)` can be set to another phrase (or to empty for no phrase). `e(clustvar)` displays the banner “(Std. Err. adjusted for # clusters in `varname`),” or it can be set to empty (`ereturn local clustvar ""`).

## Methods and formulas

We give the formulas here for complex survey data from one stage of stratified cluster sampling, as this is the most general case.

Our parameter estimates, $\hat{\beta}$, are the solution to the estimating equation

$$G(\beta) = \sum_{h=1}^{L} \sum_{i=1}^{n_h} \sum_{j=1}^{m_{hi}} w_{hij} S(\beta; y_{hij}, x_{hij}) = 0$$

where $(h, i, j)$ index the observations: $h = 1, \ldots, L$ are the strata; $i = 1, \ldots, n_h$ are the sampled PSUs (clusters) in stratum $h$; and $j = 1, \ldots, m_{hi}$ are the sampled observations in PSU $(h, i)$. The outcome variable is represented by $y_{hij}$; the explanatory variables are $x_{hij}$ (a row vector); and $w_{hij}$ are the weights.

If no weights are specified, $w_{hij} = 1$. If the weights are `aweights`, they are first normalized to sum to the total number of observations in the sample: $n = \sum_{h=1}^{L} \sum_{i=1}^{n_h} m_{hi}$. If the weights are `fweights`, the formulas below do not apply; `fweights` are treated in such a way to give the same results as unweighted observations duplicated the appropriate number of times.

For maximum likelihood estimators, $S(\beta; y_{hij}, x_{hij}) = \partial \ln L_j / \partial \beta$ is the score vector, where $\ln L_j$ is the log likelihood. For survey data, this is not a true likelihood, but a “pseudolikelihood”; see [SVY] survey.

Let

$$D = -\frac{\partial G(\beta)}{\partial \beta} \bigg|_{\beta=\hat{\beta}}^{-1}$$

For maximum likelihood estimators, $D$ is the traditional covariance estimate—the negative of the inverse of the Hessian. In the following, the sign of $D$ does not matter.

The robust covariance estimate calculated by `robust` is

$$\hat{V}(\beta) = DMD$$

where $M$ is computed as follows. Let $u_{hij} = S(\beta; y_{hij}, x_{hij})$ be a row vector of scores for the $(h, i, j)$ observation. Let

$$u_{h*} = \sum_{j=1}^{m_{hi}} w_{hij} u_{hij} \quad \text{and} \quad \bar{u}_{h*} = \frac{1}{n_h} \sum_{i=1}^{n_h} u_{h*}$$
$M$ is given by

$$
M = \frac{n - 1}{n - k} \sum_{h=1}^{L} (1 - f_h) \frac{n_h}{n - 1} \sum_{i=1}^{n_h} \left( \bar{u}_{hi} - \bar{u}_{h\bullet} \right) \left( \bar{u}_{hi} - \bar{u}_{h\bullet} \right)'
$$

where $k$ is the value given in the `minus()` option. By default, $k = 1$, and the term $(n - 1)/(n - k)$ vanishes. Stata’s `regress, vce(robust)` and `regress, vce(cluster clustvar)` commands use $k$ equal to the number of explanatory variables in the model, including the constant (Fuller et al. 1986). The `svy` prefix uses $k = 1$.

The specification $k = 0$ is handled differently. If `minus(0)` is specified, $(n - 1)/(n - k)$ and $n_h/(n_h - 1)$ are both replaced by 1.

The factor $(1 - f_h)$ is the finite population correction. If the `fpc()` option is not specified, $f_h = 0$ is used. If `fpc()` is specified and the variable is greater than or equal to $n_h$, it is assumed to contain the values of $N_h$, and $f_h$ is given by $f_h = n_h/N_h$, where $N_h$ is the total number of PSUs in the population belonging to the $h$th stratum. If the `fpc()` variable is less than or equal to 1, it is assumed to contain the values of $f_h$. See [SVY] variance estimation for details.

For the `vsrs()` option and the computation of $\hat{V}_{srswor}$, the `subpop()` option, and the `srssubpop` option, see [SVY] estat and [SVY] subpopulation estimation.

References


Also see

[P]  ereturn  —  Post the estimation results

[R]  ml  —  Maximum likelihood estimation

[R]  regress  —  Linear regression

[SVY]  variance estimation  —  Variance estimation for survey data

[U]  18  Programming Stata

[U]  20.22  Obtaining robust variance estimates

[U]  26  Overview of Stata estimation commands