rreg — Robust regression

Description

rreg performs one version of robust regression of depvar on indepvars.

Also see Robust standard errors in [R] regress for standard regression with robust variance estimates and [R] qreg for quantile (including median or least-absolute-residual) regression.

Quick start

Robust regression of y on x

rreg y x

Add a categorical covariate a using factor-variable syntax

rreg y x i.a

As above, but set the tuning parameter to 8

rreg y x i.a, tune(8)

Generate a new variable wvar containing the final weight for each observation

rreg y x i.a, genwt(wvar)

Set confidence level to 99%

rreg y x i.a, level(99)

Menu

Statistics > Linear models and related > Other > Robust regression
Syntax

```
rreg depvar [ indepvars ] [ if ] [ in ] [ , options ]
```

<table>
<thead>
<tr>
<th>options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td></td>
</tr>
<tr>
<td><code>tune(#)</code></td>
<td>use # as the biweight tuning constant; default is <code>tune(7)</code></td>
</tr>
<tr>
<td><strong>Reporting</strong></td>
<td></td>
</tr>
<tr>
<td><code>level(#)</code></td>
<td>set confidence level; default is <code>level(95)</code></td>
</tr>
<tr>
<td><code>genwt(newvar)</code></td>
<td>create <code>newvar</code> containing the weights assigned to each observation</td>
</tr>
<tr>
<td><code>display_options</code></td>
<td>control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling</td>
</tr>
<tr>
<td><strong>Optimization</strong></td>
<td></td>
</tr>
<tr>
<td><code>optimization_options</code></td>
<td>control the optimization process; seldom used</td>
</tr>
<tr>
<td><code>graph</code></td>
<td>graph weights during convergence</td>
</tr>
<tr>
<td><code>coeflegend</code></td>
<td>display legend instead of statistics</td>
</tr>
</tbody>
</table>

* `indepvars` may contain factor variables; see [U] 11.4.3 Factor variables.
* `depvar` and `indepvars` may contain time-series operators; see [U] 11.4.4 Time-series varlists.
* `by`, `mfp`, `mi estimate`, `rolling`, and `statsby` are allowed; see [U] 11.1.10 Prefix commands.
* `coeflegend` does not appear in the dialog box.
* See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

**Options**

* **Model**
  * `tune(#)` is the biweight tuning constant. The default is 7, meaning seven times the median absolute deviation (MAD) from the median residual; see Methods and formulas. Lower tuning constants downweight outliers rapidly but may lead to unstable estimates (less than 6 is not recommended). Higher tuning constants produce milder downweighting.

* **Reporting**
  * `level(#)`; see [R] Estimation options.
  * `genwt(newvar)` creates the new variable `newvar` containing the weights assigned to each observation.
  * `display_options`: `noci`, `nopvalues`, `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fwwrap(#)`, `fwwrapon(style)`, `cformat(%,fmt)`, `pformat(%,fmt)`, `sformat(%,fmt)`, and `nolstretch`; see [R] Estimation options.

* **Optimization**
  * `optimization_options`: `iterate(#)`, `tolerance(#)`, `[no]log.iterate()` specifies the maximum number of iterations; iterations stop when the maximum change in weights drops below tolerance(); and `log/nolog` specifies whether to show the iteration log (see `set iterlog` in [R] set `iter`). These options are seldom used.
graph allows you to graphically watch the convergence of the iterative technique. The weights obtained from the most recent round of estimation are graphed against the weights obtained from the previous round.

The following option is available with rreg but is not shown in the dialog box: coeflegend; see [R] Estimation options.

Remarks and examples

rreg first performs an initial screening based on Cook’s distance > 1 to eliminate gross outliers before calculating starting values and then performs Huber iterations followed by biweight iterations, as suggested by Li (1985).

Example 1

We wish to examine the relationship between mileage rating, weight, and location of manufacture for the 74 cars in the auto.dta. As a point of comparison, we begin by fitting an ordinary regression:

```
. use https://www.stata-press.com/data/r16/auto
(1978 Automobile Data)
. regress mpg weight foreign
```

```
Source | SS       df       MS
--------|----------|----------|---------
Model   | 1619.2877 2  809.643849
Residual| 824.171761 71 11.6080531
Total   | 2443.45946 73 33.4720474

F(2, 71) = 69.75, Prob > F = 0.0000
```

```
mpg    Coef.  Std. Err.   t    P>|t|    [95% Conf. Interval]
weight | -.0065879 .0006371 -10.34 0.000   -.0078583 -.0053175
foreign | -1.650029 1.075994 -1.53 0.130   -3.7955  .4954422
_cons   | 41.6797   2.165547 19.25 0.000   37.36172 45.99768
```

We now compare this with the results from rreg:

```
. rreg mpg weight foreign
```

```
Huber iteration 1: maximum difference in weights = .80280176
Huber iteration 2: maximum difference in weights = .2915438
Huber iteration 3: maximum difference in weights = .08911171
Huber iteration 4: maximum difference in weights = .02697328
Biweight iteration 5: maximum difference in weights = .29186818
Biweight iteration 6: maximum difference in weights = .11988101
Biweight iteration 7: maximum difference in weights = .03315872
Biweight iteration 8: maximum difference in weights = .00721325
Robust regression Number of obs = 74
F( 2, 71) = 168.32, Prob > F = 0.0000
```

```
mpg    Coef.  Std. Err.   t    P>|t|    [95% Conf. Interval]
weight | -.0063976 .0003718 -17.21 0.000   -.007139 -.0056562
foreign | -3.182639 .627964 -5.07 0.000   -4.434763 -1.930514
_cons   | 40.64022 1.263841 32.16 0.000   38.1202  43.16025
```

Note the large change in the foreign coefficient.
Technical note

It would have been better if we had fit the previous robust regression by typing `rreg mpg weight foreign, genwt(w)`. The new variable, \(w\), would then contain the estimated weights. Let’s pretend that we did this:

```
. rreg mpg weight foreign, genwt(w)
(output omitted)
. summarize w, detail
```

<table>
<thead>
<tr>
<th>Robust Regression Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentiles</td>
</tr>
<tr>
<td>1%</td>
</tr>
<tr>
<td>5%</td>
</tr>
<tr>
<td>10%</td>
</tr>
<tr>
<td>25%</td>
</tr>
<tr>
<td>50%</td>
</tr>
<tr>
<td>75%</td>
</tr>
<tr>
<td>90%</td>
</tr>
<tr>
<td>95%</td>
</tr>
<tr>
<td>99%</td>
</tr>
</tbody>
</table>

We discover that 3 observations in our data were dropped altogether (they have weight 0). We could further explore our data:

```
. sort w
. list make mpg weight w if w <.467, sep(0)
```

<table>
<thead>
<tr>
<th>make</th>
<th>mpg</th>
<th>weight</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>VW Diesel</td>
<td>41</td>
<td>2,040</td>
<td>0</td>
</tr>
<tr>
<td>Datsun 210</td>
<td>35</td>
<td>2,020</td>
<td>0</td>
</tr>
<tr>
<td>Subaru</td>
<td>35</td>
<td>2,050</td>
<td>0</td>
</tr>
<tr>
<td>Plym. Arrow</td>
<td>28</td>
<td>3,260</td>
<td>.04429567</td>
</tr>
<tr>
<td>Cad. Seville</td>
<td>21</td>
<td>4,290</td>
<td>.08241943</td>
</tr>
<tr>
<td>Toyota Corolla</td>
<td>31</td>
<td>2,200</td>
<td>.10443129</td>
</tr>
<tr>
<td>Olds 98</td>
<td>21</td>
<td>4,060</td>
<td>.28141296</td>
</tr>
</tbody>
</table>

Being familiar with the automobile data, we immediately spotted two things: the VW is the only diesel car in our data, and the weight recorded for the Plymouth Arrow is incorrect.
Example 2

If we specify no explanatory variables, `rreg` produces a robust estimate of the mean:

```
rreg mpg
```

Huber iteration 1: maximum difference in weights = .64471879
Huber iteration 2: maximum difference in weights = .05098336
Huber iteration 3: maximum difference in weights = .0099887
Biweight iteration 4: maximum difference in weights = .25197391
Biweight iteration 5: maximum difference in weights = .00358606

Robust regression
Number of obs = 74
F( 0, 73) = 0.00
Prob > F = .

|                      | Coef. | Std. Err. | t     | P>|t|  | [95% Conf. Interval] |
|----------------------|-------|-----------|-------|------|---------------------|
| _cons                | 20.68825 | .641813 | 32.23 | 0.000 | [19.40912, 21.96738] |

The estimate is given by the coefficient on `_cons`. The mean is 20.69 with an estimated standard error of 0.6418. The 95% confidence interval is [19.4, 22.0]. By comparison, `ci means` (see [R] `ci`) gives us the standard calculation:

```
ci means mpg
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Obs</th>
<th>Mean</th>
<th>Std. Err.</th>
<th>[95% Conf. Interval]</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpg</td>
<td>74</td>
<td>21.2973</td>
<td>.6725511</td>
<td>[19.9569, 22.63769]</td>
</tr>
</tbody>
</table>

Stata has an active user community working in the area of robust statistics. You may also be interested in robust estimators and outlier detection tools from Verardi and McCathie (2012), Verardi and Dehon (2010), Jann (2010), and Verardi and Croux (2009), to name a few. See [R] `ssc` for how to install community-contributed packages.
rreg stores the following in e():

Scalars
- e(N): number of observations
- e(mss): model sum of squares
- e(df_m): model degrees of freedom
- e(rss): residual sum of squares
- e(df_r): residual degrees of freedom
- e(r2): \( R^2 \)
- e(r2_a): adjusted \( R^2 \)
- e(F): F statistic
- e(rmse): root mean squared error
- e(rank): rank of e(V)

Macros
- e(cmd): rreg
- e(cmdline): command as typed
- e(depvar): name of dependent variable
- e(genwt): variable containing the weights
- e(title): title in estimation output
- e(model): ols
- e(properties): b V
- e(predict): program used to implement predict
- e(marginsok): predictions allowed by margins
- e(asbalanced): factor variables fvset as asbalanced
- e(asobserved): factor variables fvset as asobserved

Matrices
- e(b): coefficient vector
- e(V): variance–covariance matrix of the estimators

Functions
- e(sample): marks estimation sample

Methods and formulas


rreg begins by fitting the regression (see [R] regress), calculating Cook’s \( D \) (see [R] predict and [R] regress postestimation), and excluding any observation for which \( D > 1 \).

Thereafter, rreg works iteratively: it performs a regression, calculates case weights from absolute residuals, and regresses again using those weights. Iterations stop when the maximum change in weights drops below \texttt{tolerance()}. Weights derive from one of two weight functions, Huber weights and biweights. Huber weights (Huber 1964) are used until convergence, and then, from that result, biweights are used until convergence. The biweight was proposed by Beaton and Tukey (1974, 151–152) after the Princeton robustness study (Andrews et al. 1972) had compared various estimators. Both weighting functions are used because Huber weights have problems dealing with severe outliers, whereas biweights sometimes fail to converge or have multiple solutions. The initial Huber weighting should improve the behavior of the biweight estimator.

In Huber weighting, cases with small residuals receive weights of 1; cases with larger residuals receive gradually smaller weights. Let \( e_i = y_i - X_i b \) represent the \( i \)th-case residual. The \( i \)th scaled residual \( u_i = e_i / s \) is calculated, where \( s = M / 0.6745 \) is the residual scale estimate and \( M = \text{med}(|e_i - \text{med}(e_i)|) \) is the median absolute deviation from the median residual. Huber estimation obtains case weights:
$$w_i = \begin{cases} 1 & \text{if } |u_i| \leq c_h \\ c_h/|u_i| & \text{otherwise} \end{cases}$$

**rreg** defines $c_h = 1.345$, so downweighting begins with cases whose absolute residual exceeds $(1.345/0.6745)M \approx 2M$.

With biweights, all cases with nonzero residuals receive some downweighting, according to the smoothly decreasing biweight function

$$w_i = \begin{cases} \{1 - (u_i/c_b)^2\}^2 & \text{if } |u_i| \leq c_b \\ 0 & \text{otherwise} \end{cases}$$

where $c_b = 4.685 \times \text{tune()}/7$. Thus when `tune()` = 7, cases with absolute residuals of $(4.685/0.6745)M \approx 7M$ or more are assigned 0 weight and thus are effectively dropped. Goodall (1983, 377) suggests using a value between 6 and 9, inclusive, for `tune()` in the biweight case and states that performance is good between 6 and 12, inclusive.

The tuning constants $c_h = 1.345$ and $c_b = 4.685$ (assuming `tune()` is set at the default 7) give **rreg** about 95% of the efficiency of OLS when applied to data with normally distributed errors (Hamilton 1991b). Lower tuning constants downweight outliers more drastically (but give up Gaussian efficiency); higher tuning constants make the estimator more like OLS.

Standard errors are calculated using the pseudovalues approach described in Street, Carroll, and Ruppert (1988).

**Acknowledgment**

The current version of **rreg** is due to the work of Lawrence Hamilton of the Department of Sociology at the University of New Hampshire.

**References**


Also see

[R] `rreg postestimation` — Postestimation tools for `rreg`

[R] `qreg` — Quantile regression

[R] `regress` — Linear regression

[MI] `Estimation` — Estimation commands for use with `mi estimate`

[U] 20 `Estimation and postestimation commands`