This file consists of Chapter 4 of **MacAnova User's Guide** by Gary W. Oehlert and Christopher Bingham, issued as Technical Report Number 617, School of Statistics, University of Minnesota, revised August 1998, describing Version 4.07 of MacAnova.

This manual is Copyright © 1998 Gary W. Oehlert and Christopher Bingham, all rights reserved.

Fonts used in this manual are Palatino, Courier, and Symbol.

For information concerning MacAnova, write University of Minnesota, Department of Applied Statistics, 352 Classroom Office Building, 1994 Buford Avenue, St. Paul, MN 55108-6042.

4. Generalized linear models and robustness (advanced topic.)

4.1 **Alternatives to Least Squares** Linear least squares procedures generally assume that the errors in a model follow a normal distribution. This is often not even approximately the case. Sometimes the data are counts, or take only 0 and 1 as values, or contain outliers – deviant values that are further from their expectations than could reasonably happen with normal (Gaussian) data. In such situations, you may be better off abandoning linear least squares procedures and adopting procedures more appropriate for the kind of data you have.

MacAnova provides several commands, `poisson()`, `ipf()`, `logistic()`, `probit()`, `glmfit()` and `robust()` for fitting models with a linear structure by methods other than least squares. All conform to the pattern for GLM commands (see Sec. 3.1-3.4). The first argument must either be a `CHARACTER` scalar or quoted string specifying a model or be missing, in which case `STRMODEL` is taken to specify the model. All but `robust()` fit what are known as *generalized linear models* ; `robust()` fits the same sort of models (linear) as does `anova()`, but uses an non-least squares estimation method that is less influenced by outliers and data errors.

4.2 **Generalized Linear Model Commands** The essence of a generalized linear model is that some function $\eta = g(\mu)$ of the expectation $\mu = E[Y]$, where Y is the response variable, is a linear combination of the X variables. Using the notation of Sec. 3.2, this means

$$\eta = g(E[Y]) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_k X_k \text{ or } \eta = g(E[Y]) = \beta_1 X_1 + \beta_2 X_2 + ... + \beta_k X_k .$$

Function $g(\mu)$ is the *link function*. A *linear* model can be characterized as a generalized linear model with *identity* link $\eta = g(\mu) = \mu$. A complete description of a generalized linear model also includes specification of the distribution of the response conditional on the X-variables. Common distributions are Poisson, binomial or normal. For a comprehensive introduction to generalized linear models, see McCullagh and Nelder (1983).

Poisson regression (log linear fitting) analyzes data from a generalized linear model with link function $\eta = g(\mu) = \log(\mu)$ and Poisson distributed Y. Inverting the link function, $\mu = g^{-1}(\eta) = e^{\eta}$. Such models may be fitted by MacAnova command `poisson()`. Command `ipf()` uses another algorithm to do log-linear fitting of balanced models with no variates.

Logistic regression is for generalized linear models with link function $\eta = g(\mu) = \text{logit}(\mu) = \log(\mu / (1 - \mu)) = \log(\mu) - \log(1-\mu)$ with response Y/n, where Y is binomial with n trials and $p = P(\text{“success”}) = \mu = g^{-1}(\eta) = \dfrac{e^{\eta}}{1 + e^{\eta}}$. Command `logistic()` performs logistic regression for arbitrary models. You need to specify the sample sizes (number of trials), which can be a small as 1, for each case.

Probit analysis is similar to logistic except $g(\mu) = \text{probit}(\mu)$, where $\eta = \text{probit}(p)$ is such that $P(Z \le \eta) = p$, Z a standard normal random variable. In terms of MacAnova functions, $\eta = \text{probit}(p) = \text{invnor}(p)$ and $p = g^{-1}(\eta) = \text{cumnor}(\eta)$.

Command probit() carries out probit analysis of binomial data. In the traditional definition of a probit, the constant 5 is added to the value so as to make values positive except for extremely small values of p (< 0.00000029). Since this just adds 5 to the intercept of any probit model fitted but doesn't change other coefficients, it is not done by probit().

glmfit() is an omnibus command to fit generalized linear models. You specify the link function and distribution by, for example, link:"log",distrib:"poisson". At present glmfit() does no more than combine several other GLM commands (anova(), poisson(), logistic() and probit()) into a single command, but eventually additional links and distributions will be added to its repertoire. If the link specified is "identity" and the distribution is "normal" (the defaults), glmfit() is equivalent to anova(). This special case is excluded from much of the discussion below.

The algorithm used by all these commands, except ipf(), is iteratively reweighted least squares (McCullagh and Nelder (1983), Sec. 2.5). Each iteration is equivalent to a weighted least squares linear fit, with the weights updated after each fit. The iteration stops when the relative change in the "deviance" (see below) is small enough, or when a limit on the number of iterations is hit. All the commands recognize keyword phrases maxit:m and eps:small, where m is a positive integer and small is a small positive REAL scalar such as .0001. The default values are 50 for m and 10^{-6} for small. Quantity small specifies how small the relative change in deviance has to be to stop iterating and m is the limit on the number of iterations. If there are no variates in the model and it is balanced, ipf() uses iterative proportional fitting. In other situations, ipf() is identical to poisson(). It also recognizes keywords maxit and eps.

4.2.1 **Analysis of Deviance Table** The standard output for poisson(), ipf(), logistic(), probit() and glmfit() (except with identity link and normal distribution) is an *Analysis of Deviance* table. This table has columns labeled DF (degrees of freedom), Deviance and MDev. The MDev column is simply Deviance/DF and is analogous to the MS (Mean Square) column in anova() output. The deviance and degrees of freedom columns are saved in side effect variables SS and DF (see the example in Sec. 4.2.2).

Each entry in the Deviance column is a difference 2 (L_1 - L_0), where L_0 and L_1 are the maximized log likelihoods of two models M_0 and M_1. In the simplest usage, there are only two lines in the table, labelled Overall model and ERROR1.

For the Overall model line M_0 is the model with all parameters 0 (the null model) and M_1 is the complete model being fit. For a log link, M_0 corresponds to E[Y] = 1 for all cases and for a logit or probit link with binomial data it corresponds to constant probability p = .5 for all cases. When all model parameters are 0, the Overall model deviance is approximately distributed as χ^2 with the degrees of freedom from the DF column.

For the ERROR1 line, M_0 is the model being fitted by the command and M_1 is a model that fits the data perfectly, that is, with a parameter for each case. The error deviance can be described as a measure of the "lack of fit" of the model to the data. When the model specified is correct, under certain conditions, the ERROR1 deviance is

approximately distributed as χ^2 with the degrees of freedom from the DF column, and can be used to test for the appropriateness of the model.

You can get a more complete deviance table, with lines for all the usual terms in the model, by including keyword phrase increment:T (or inc:T) as an argument. Designate by Term_i the term in the model associated with line i of the Deviance table. For example, Term_1 would normally correspond to CONSTANT. Then for the deviance in line 1, M_0 is the model with all parameters 0 and M_1 is Term_1; for the deviance in line $i > 1$ M_0 is $\text{Term}_1+...+\text{Term}_{i-1}$ and M_1 is $\text{Term}_1+...+\text{Term}_{i-1}+\text{Term}_i$. Thus the Deviance is the increase in $2L$ "due to" adding Term_i to the model, but not including Term_{i+1}, Term_{i+2}, ... If the model $M_1 = \text{Term}_1+...+\text{Term}_i$ for term i is correct (that is, $\text{Term}_j = 0$, for $j > i$) and also $\text{Term}_i = 0$, so that M_0 is correct, then the corresponding deviance is usually approximately distributed as χ^2 with the degrees of freedom from the DF column and thus you can use a χ^2 critical value to test the significance of Term_i. The computation of the complete table can be slow, since each of the models Term_1 , $\text{Term}_1+\text{Term}_2$, ..., $\text{Term}_1+...+\text{Term}_i$, ... are separately fitted one after another.

Below are several short examples, all using the same data presumed derived from a two factor experiment, factors a and b having 3 and 4 levels respectively, entered as follows:

```
Cmd> y <- vector(34,34,37,23,23,38,12,27,29,17,17,29)
Cmd> n <- vector(87,88,82,82,84,80,87,82,80,84,85,84)
Cmd> a <- factor(rep(run(3),4))
Cmd> b <- factor(rep(run(4),rep(3,4)))
```

In a real data set, probably only one of the analyses would be appropriate.

More examples are in Chapter 10.

4.2.2 **Side effect variables** All the generalized linear model commands create the same side effect variables as anova() (see Sec. 3.6) and most also create WTDRESIDUALS.

Variables STRMODEL, DEPVNAME, TERMNAMES, and DF are identical to what they would be after anova() with the same model.

When keyword phrase increment:T is an argument (Sec. 4.2.1), SS contains the values from the Deviance column in the Analysis of Deviance table. *Without* increment:T, SS is 0 except for the last two elements which are the Overall model and ERROR1 deviances, in that order. In the latter case the non-zero elements of SS can be extracted by SS[length(SS)+vector(-1,0)] with their degrees of freedom computed by vector(sum(DF[run(length(DF)-1)]),DF[length(DF)]).

Vector RESIDUALS consists of Y[i] - fitted[i], where the elements of fitted are the best fitting value to the response Y[i], in the *original* scale of the response, not in the *link scale*, that is the scale of the link function values.

As mentioned in Sec. 4.2, the fitting algorithm involves iteratively reweighted least squares. At each iteration, the dependent variable is the current fitted value in the link

scale plus a r_i, where r_i is a modified residual, also computed in the link scale. The weights W_i used are computed from the current fitted values. Then the i^{th} element of side effect variable `WTDRESIDUALS` is $\sqrt{W_i} \times r_i$ and is not directly related to the the i^{th} element of `RESIDUALS`. This contrasts with the situation after a weighted regression, ANOVA or MANOVA.

The elements of `HII` are the diagonal elements h_{ii} of the hat matrix $H = X(X'WX)^{-1}X'W$, where W is diagonal matrix of weights. The variance of $\sqrt{W_i} \times r_i$ is approximately $(1 - h_{ii})$. This means that you can use `WTDRESIDUALS` together with `HII` in diagnostic procedures much as in weighted regression or ANOVA. In particular, predefined macros `resvsrankits`, `resvsyhat` and `resvsindex` may still be useful for examining residuals (see Sec. 3.17).

4.2.3 **Keywords** Besides `increment:T`, these commands recognize keyword phrases `pvals:T`, `offsets:OffVec`, and `scale:sigma`.

`pvals:T` causes *P* values to be printed based on an assumed null distribution of χ^2 for the Deviance. How appropriate these are will depend on the situation. In particular, for `probit()` and `logistic()`, the *P* value associated with `ERROR1` is unlikely to be meaningful if the values of `n` (number of trials) are too small.

`offsets:OffVec` specifies that `REAL` vector `OffVec` is to be added to the linear predictor in computing the fit. Its elements should be in the units of the *link function* (log, logit, or probit). It effectively changes the model fit to include the term `1*OffVec` with no coefficient to be estimated. The `Overall model` deviance is -2 (L_0' - L_1), where L_0' is the log likelihood of the model $g(\mathrm{E}[Y])$ = `OffVec` or $\mathrm{E}[Y] = g^{-1}$(`OffVec`), where $g()$ is the link function.

When the offset vector is a linear combination of X-variables in the model, the null hypotheses tested are that the coefficients are the same as those used in defining the offset vector rather than that they are zero. A very simple case would be

```
poisson("y=1",offset:rep(log(30),length(y)))
```

The `ERROR1` deviance from this analysis tests whether the mean is constant and the `Overall model` deviance tests the hypothesis that a constant mean is 30, assuming the response is Poisson.

Use of `scale:sigma` as an argument does not affect the output from the GLM command itself. Instead, the value of `sigma` will used by other functions such as `secoefs()`, `predtable()`, and `contrast()` to replace a default multiplier used to compute standard errors. `sigma` must be a positive `REAL` scalar or `?` (`MISSING`). When `sigma` is `MISSING`, the scale factor will be computed as `sqrt(SS[m]/DF[m])`, where `SS[m]` and `DF[m]` are from the last line (`ERROR1`) in the Deviance table. The default scale multiplier `sigma` is 1 except after a `glmfit()` that is equivalent to `anova()` when it is `sqrt(SS[m]/DF[m])`. In `secoefs()`, `sigma` multiplies the square roots of the diagonal values of $(X'WX)^{-1}$, where X is the matrix of X-variables, and W is a diagonal matrix of weights computed using the converged fit.

4.2.4 **logistic() and probit()** Both of these are designed for data where the response vector consists of binomial distributed random variables. Besides the model, you need to provide the sample size (number of trials) for each case. Commands `logistic()` and `probit()` fit generalized linear models that differ only in the link function. `logistic()` assumes a logit link and `probit()` assumes a probit link (see above).

Typical usage is `logistic("y=x1+x2",N)` and `probit("y=x1+x2",N)`, with the sample sizes specified by the `N`. If the number of trials is the same for each case, `N` can be a scalar; otherwise, `N` must be a vector with `length(N)` = `length(y)`, with $0 \le$ `y[i]` $\le$ `N[i]`. Both `logistic()` and `probit()` tolerate *non-integer* values, although they print a warning unless keyword phrase `silent:T`) is an argument.

Instead of `logistic(Model,N)` and `probit(Model,N)` you can use `glmfit(Model, n:N,link:"logit"distrib:"binomial")` and `glmfit(Model,n:N, link:"probit",distrib:"binomial")`, respectively.

```
Cmd> logistic("y=a+b",n,increment:T) # split Overall model deviance
Model used is y=a+b
WARNING: summaries are sequential
                DF     Deviance         MDev
CONSTANT         1       135.64       135.64
a                2       19.216       9.6078
b                3       18.694       6.2314
ERROR1           6       9.8484       1.6414

Cmd> SS
(1)       135.64       19.216       18.694       9.8484

Cmd> SS[1] + SS[2] + SS[3] # total deviance for model
(1)       173.55

Cmd> logistic("y=a+b",n)
Model used is y=a+b
WARNING: summaries are sequential
                DF     Deviance         MDev
Overall model    6       173.55       28.925
ERROR1           6       9.8484       1.6414

Cmd> # Overall model deviance is SS[1]+SS[2]+SS[3] in preceding

Cmd> SS #  All but last two elements are 0
SS:
(1)            0            0       173.55       9.8484

Cmd> DF # Degrees of freedom for 1st 3 terms are not pooled
(1)            1            2            3            6

Cmd> 1 - cumchi(SS[4],DF[4]) # P value for goodness of fit
(1)      0.13118
```

```
Cmd> probit("y=a+b",n)
Model used is y=a+b
WARNING: summaries are sequential
                DF     Deviance         MDev
Overall model    6       173.77       28.962
ERROR1           6       9.6263       1.6044
```

See Sec. 10.18 for another example of the use of `logistic()`.

One computational problem that can occur with both `logistic()` and `probit()` is that the best fitting model may actually have infinite parameters, with some fitted probabilities exactly 0 or 1. To cope with this possibility, at each step of the iteration, if a fitted probability $\hat{p}$ would satisfy $\hat{p} < 10^{-8}$ or $\hat{p} > 1 - 10^{-8}$, it is "clamped" back to one of these limits and a warning message is printed. You can change the threshold to 10^{-10}, say, with keyword phrase `problimit:1e-10`, or any number $> 10^{-15}$, and repeat the computation. You may still get the warning. Whether you do or not, usually the estimated standard errors of coefficients as computed by `secoefs()` (see Sec. 3.13.1) are large compared to the estimated coefficients.

Here is a simple example, with a factor `a` with 3 levels and $n = 10$. Clearly the estimated probabilities for the 3 cells ought to be 0, .7 and 1, respectively. These cannot be reached by either the logit or probit link unless $\eta = \pm\infty$.

```
Cmd> groups <- factor(run(3)); counts <- vector(0, 7, 10)

Cmd> logistic("counts=groups",10,inc:T)
Model used is counts=groups
WARNING: problimit = 1e-08 was hit by logistic() at least once
WARNING: summaries are sequential
                DF     Deviance         MDev
CONSTANT         1      0.53492      0.53492
groups           2       28.837       14.418
ERROR1           0            0    undefined

Cmd> secoefs() # see Sec. 3.13
NOTE: standard errors assume scale parameter is 1
component: CONSTANT
  component: coefs
(1)      0.28243
  component: se
(1)       1490.7
component: groups
  component: coefs
(1)      -19.703      0.56487       19.138
  component: se
(1)         2357       1490.7         2357

Cmd> glmtable() # compute fitted probabilities; see Sec. 3.18
NOTE: standard errors assume scale parameter is 1
component: estimate
(1)   3.6788e-09          0.7            1
component: SEest
(1)   1.1633e-05      0.14491   1.1633e-05
```

4.2.5 **poisson() and ipf()** These fit Poisson log linear regression models which may be appropriate when the elements of the response vector are independent counts with Poisson distributions. The link function is the natural logarithm so that $\log(\mu) = \beta_0 + \beta_1 X_1 + \ldots + \beta_k X_k$. Equivalently $\mu = E[Y] = e^{\beta_0 + \beta_1 X_1 + \ldots + \beta_k X_k}$. `poisson()` is also appropriate for certain multinomial models in which some of the marginals are fixed provided the models fitted match those marginals exactly. Typical usage for `poisson()` is `poisson("y=a*b + c + a.c + b.c")`, where `y[i]` ≥ 0. The elements of `y` are normally integers, and, if any are not, a warning message is printed unless `silent:T` is an argument. For balanced designs with no variates, `ipf()` and `poisson()` use different computational methods but yield the same results except for rounding error. In this special case, `ipf()` may be somewhat faster, but has the disadvantage that you cannot later compute standard errors using functions such as `secoefs()` and `predtable()`.

```
Cmd> poisson("y=a+b",increment:T)
Model used is y=a+b
WARNING: summaries are sequential
                DF     Deviance         MDev
CONSTANT         1       1485.4       1485.4
a                2       10.621       5.3107
b                3         13.1       4.3666
ERROR1           6       7.4863       1.2477

Cmd> ipf("y=a+b")
Model used is y=a+b
WARNING: summaries are sequential
                DF     Deviance         MDev
Overall model    6       1509.1       251.52
ERROR1           6       7.4863       1.2477
```

See Sec. 10.19 for an example of the use of `poisson()`.

4.2.6 **glmfit()** As mentioned above, this is potentially a general function for fitting generalized linear models. Its usage is

```
glmfit(Model,link:linkName,distrib:distName [,parameters:vec])
```

where `linkName` and `distName` are quoted strings or `CHARACTER` scalars specifying the link function and distribution. At the present time only the following combinations of values for `link` and `distrib` are permitted, although further options are planned.

distrib	link	Command equivalent to `glmfit()`
`"normal"`	`"identity"`	`anova()`
`"poisson"`	`"log"`	`poisson()`
`"binomial"`	`"logit"`	`logistic()`
`"binomial"`	`"probit"`	`probit()`

When `distrib` is not used, `distrib:"normal"` is assumed. When `link` is not used, `link:"identity"` is assumed for `distrib:"normal"`, `link:"log"` is assumed for `distrib:"poisson"` and `link:"logit"` are assumed for `distrib:"binomial"`.

With `distrib:"binomial"`, you must also specify the sample sizes by `n:N`, where `N` is either a `REAL` scalar or a `REAL` vector as long as the response. For this purpose, you can also use `parameters:N` instead.

`glmfit()` is designed to provide an interface for fitting additional generalized linear models with different links such as reciprocal and distributions such as gamma or negative binomial, but such extensions have not yet been implemented.

```
Cmd> glmfit("y=a+b") # link:"identity", distrib:"normal" assumed
Model used is y=a+b
WARNING: summaries are sequential
                DF           SS           MS
CONSTANT         1       8533.3       8533.3
a                2       288.17       144.08
b                3          358       119.33
ERROR1           6        136.5        22.75

Response distribution is normal, link function is identity

Cmd> # This table is identical with table from anova("y=a+b")

Cmd> glmfit("y=a+b",distrib:"poisson",link:"log") #same as poisson()
Model used is y=a+b
WARNING: summaries are sequential
                DF     Deviance         MDev
Overall model    6       1509.1       251.52
ERROR1           6       7.4863       1.2477

Response distribution is poisson, link function is log

Cmd> glmfit("y=a+b",n:n,distrib:"binomial",link:"probit") #probit()
Model used is y=a+b
WARNING: summaries are sequential
                DF     Deviance         MDev
Overall model    6       173.77       28.962
ERROR1           6       9.6263       1.6044

Response distribution is binomial, link function is probit

Cmd> glmfit("y=a+b",distrib:"normal",link:"log")
ERROR: glmfit(Model,link:"log",distrib:"normal") is not implemented
```

4.3 **Robust regression – robust()** `robust()` uses an outlier-resistant algorithm to fit linear models, that is, it uses the identity link. Robust fitting of models is appropriate when the response variable may be contaminated by wild values, or when the distribution of errors has long heavy tails. The output is similar to that of `anova()` and, in large samples, it can be used the same way. What is done is closely related to, but *not* the same as, what is known as *M*-estimation.

Let $\mu(\beta) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$, and let σ be the standard deviation of the residual Y_i - $\mu(\beta)$. `robust()` attempts to find β and σ to minimize $\frac{\sigma}{n}\sum \rho\left(\frac{Y_i - \mu_i(\beta)}{\sigma}\right)$, where the function $\rho(x) = \begin{cases} \frac{1}{2}x^2 + \frac{1}{2}\gamma & |x| < c \\ c|x| - \frac{1}{2}c^2 + \frac{1}{2}\gamma & |x| \geq c \end{cases}$ with derivative $\psi(x) = \rho'(x) = \begin{cases} -c & x \leq -c \\ x & -c < x < c \\ c & x \geq c \end{cases}$. c is the *truncation* point since scaled residuals $r_i = \frac{Y_i - \mu_i(\beta)}{\sigma}$ with $|r_i| > c$ are given less weight. Constant $\gamma = E[\psi(x)^2]$, when x is standard normal.

In addition to the analysis of variance table, `robust()` always prints a robust estimate of the of scale which will estimate σ if the response is actually normal. This can be recovered by `modelinfo(sigmahat:T)` (Sec. 3.24), but should not be used in computing standard errors (use `modelinfo(scale:T)` instead). The default value of c used by `robust()` is = .75. The specific computational method is given in Section 7.8 of Huber (1981) from which the notation is also taken.

You can specify a different truncation point using keyword phrase `trunc:c`, where `c` is a positive `REAL` scalar.

The usual side effect variables are computed. `HII` consists of the diagonal elements of $X(X'X)^{-1}X'$. The elements of `RESIDUALS` are $Y_i - \mu_i(\hat{\beta})$ and those of `WTDRESIDUALS` are $\hat{\sigma} \times \rho\left(\frac{Y_i - \mu_i(\hat{\beta})}{\hat{\sigma}}\right)$. Since $\rho(x) = x$ for $|x| \leq c$, when case `i` is not truncated `WTDRESIDUALS[i]/RESIDUALS[i]` = 1; for when it is truncated, this ratio is $\frac{c\hat{\sigma}}{|Y_i - \mu_i(\hat{\beta})|} < 1$ and can be interpreted as the amount of importance given to the i^{th} case in the fitting procedure. When all elements of `WTDRESIDUALS`/`RESIDUALS` are 1, no truncation occurred and the fit is the same as the least squares fit. `modelinfo(weights:T)` returns `(WTDRESIDUALS/RESIDUALS)^2`.

```
Cmd> y1 <- y; y1[12] <- 130 # make modified y with outlier
Cmd> anova("y1=a+b") # fit by least squares
Model used is y1=a+b
                 DF          SS          MS
CONSTANT          1       14770       14770
a                 2      3318.2      1659.1
b                 3      1763.6      587.86
ERROR1            6      5523.2      920.53
```

```
Cmd> robust("y1=a+b") # fit robustly, truncation c = .75
Model used is y1=a+b
WARNING: summaries are sequential
                DF          SS*          MS*
CONSTANT         1       9332.6       9332.6
a                2       516.85       258.42
b                3       245.62       81.873
ERROR1           6       371.15       61.858
* ANOVA is approximate and should be interpreted with caution

Robust estimate of sigma: 11.657

Cmd> WTDRESIDUALS/RESIDUALS # only 1 residual was truncated
 (1)            1            1            1            1            1
 (6)            1            1            1            1            1
(11)            1     0.091936
Cmd> modelinfo(weights:T,scale:T,sigmahat:T)
component: scale
(1)        7.865
component: weights
 (1)            1            1            1            1            1
 (6)            1            1            1            1            1
(11)            1    0.0084523    0.0084523 = 0.091936^2
component: sigmahat
(1)       11.657

Cmd> sqrt(SS[4]/DF[4]) # value of scale
(1)        7.865

Cmd> robust("y1=a+b",trunc:.60) # more severe truncation
Model used is y1=a+b
WARNING: summaries are sequential
                DF          SS*          MS*
CONSTANT         1         9233         9233
a                2       485.06       242.53
b                3        253.6       84.532
ERROR1           6       332.51       55.418
* ANOVA is approximate and should be interpreted with caution

Robust estimate of sigma: 13.078
```

See Sec. 10.20 for an example of the use of `robust()`.