

Object oriented S-Plus functions for Huber type robust regression

Alfio Marazzi
March 1997

1 Introductory example

Enter the following data into S-plus

```
y      <- c(0.8, 1.2, 1.8, 2.2, 2.8, 3.2, 0.8, 1.2, 1.8, 2.2, 2.8, 3.2)
Dose   <- c( 1,  1,  2,  2,  3,  3,  4,  4,  5,  5,  6,  6)
P      <- c(-1, -1, -1, -1, -1, -1,  1,  1,  1,  1,  1,  1)
n      <- 12
```

and suppose that the values in y are responses of different subjects to the doses of a particular stimulus (substance) in a simple analytic dilution biological assay, as described in Finney (1978, Chapter 3). The elements of P indicate the observations obtained using the standard substance S ($P=+1$) and those obtained with the test substance T ($P=-1$). It is assumed that T behaves as if it were a dilution (or a concentration) of S in a diluent that is inert with respect of the response. Therefore, the mean response to any dose z of T , on an appropriate logarithmic scale, equals the mean response to the dose $z + \log(\rho)$ of S . ρ is the *potency* of T with respect to S and must be estimated. A graphical display of the data is obtained by opening a graphic windows (e.g., use `win.graph()` under Windows) and typing

```
plot(Dose,y,type="n")
points(Dose[P==-1],y[P==-1],pch=16)
points(Dose[P==+1],y[P==+1],pch=3)
```

The standard analysis of this type of data is based on a multiple regression model that relates the response y (y) to two variables X_S and X_T defined by

X_{Si} = dose value for subject i , if i is treated with S
0, otherwise,

X_{Ti} = dose value for subject i , if i is treated with T
0, otherwise.

The model is

$$\Omega : \quad y = \mu + \pi P + \eta X_S + \theta X_T + error.$$

In other words, two straight lines are used to describe the dose reponse relation. For the purpose of testing the parallelism of the two straight lines (see below), it is convenient to re-express Ω as

$$\Omega : \quad y = \mu + \pi P + \gamma L_1 + \delta L'_1 + error,$$

where $L_1 = X_S + X_T$ and $L'_1 = X_S - X_T$. Thus, $\eta = (\gamma + \delta)/2$ and $\theta = (\gamma - \delta)/2$ (see Marazzi et al., 1986, for more complex models).

In S-plus, we define X_S , X_T , L_1 and L'_1 as follows:

```
XS      <- XT <- rep(0,n)
XS[P==+1] <- Dose[P==+1]
XT[P== -1] <- Dose[P== -1]
L1 <- XS+XT
Lp1 <- XS-XT
```

The function `lm()` can be used to adjust Ω : Type

```
Model <- lm(y~P+L1+Lp1)
summary(Model)
```

We obtain, $\hat{\mu} = -1.5$, $\hat{\pi} = -1.5$, $\hat{\gamma} = 1$, $\hat{\eta} = 0.0$. To visualize the two straight lines, enter the function `DiagrM()` listed in the Appendix and type

```
diagrM(Dose,y,P,Model)
```

We now test the hypothesis that the two straight lines are parallel: Type

```
model <- update(Model,.~.-Lp1)
anova(Model,model)
```

As the hypothesis is not rejected, the horizontal distance between the two straight lines is an estimate of $\log(\rho)$. The function `potcy()` (Appendix) computes this distance as $\hat{\kappa} = 2\hat{\pi}/\hat{\gamma}$ and the estimate $\hat{\rho} = \exp(\hat{\kappa})$ of ρ . Moreover, it computes the corresponding confidence interval, with coefficient of confidence 95%, using Fieller's theorem. Enter `potcy()` and type

```
Potency <- potcy(model,summary(model)$sigma)
Potency$R; Potency$Rl; Potency$Ru
```

We obtain $\hat{\kappa} = -3$, $\hat{\rho} = 0.05$ and the confidence interval $[Rl, Ru] = [0.038, 0.065]$ for ρ .

We now change one observation and repeat the analysis again: Type

```
y[12] <- 0
Model <- lm(y~P+L1+Lp1)
summary(Model)
diagrM(Dose,y,P,Model)
model <- update(Model,.~.-Lp1)
anova(Model,model)
Potency <- potcy(model,summary(model)$sigma)
Potency$R; Potency$Rl; Potency$Ru
```

The results are completely changed: The fitted lines are no longer parallel (although the test is still non significant) and the confidence interval for ρ is much larger. A single *outlier* has distorted the parameter estimates and inflated the confidence interval.

Observation 12 is an outlier because it has a discordant value of the dependent variable (the response). We call it an outlier *in y-direction*. This kind of outlier is often observed when the values of the explanatory variables are carefully fixed according to some experimental design, as in most bioassay experiments. In general, outliers are individual observations far removed from the pattern set by the majority of the data. They are mainly due to gross errors, such as recording errors or inadvertent observations of members

of a different population (i.e., anomalous subjects in biological material) or legitimate extreme observations. They are usually influential, that is, their deletion often causes major changes in the estimates, confidence regions, tests, and so on. As the values and frequency of outliers strongly fluctuate from sample to sample, outliers can make the conclusions of a statistical analysis unreliable. For these reasons, we usually want to bound the influence of outliers on the results of the statistical analysis. Methods designed for this purpose are called *robust*.

A robust method for bounding the influence of outliers in y -direction has been proposed by Huber (see Huber, 1973). To use this method with S-plus you need to load two libraries:

```
library(robeth, T)
library(Huber, T)
```

Then type:

```
Model <- lm(y~P+L1+Lp1, method="Huber")
diagrM(Dose,y,P,Model)
model <- update(Model, .~.-Lp1)
anova(Model,model)
Potency <- potcy(model,summary(model)$sigma)
Potency$R; Potency$Rl; Potency$Ru
```

Thanks to the object oriented language of S-plus, the command lines required by the robust method are the same as those needed for the classical one, except for the additional parameter `method="Huber"` in `lm()`. To visualize the robust estimates you can use the same `print`, `plot`, and `extractor` functions you have used for classical estimates. For example, type

```
summary(Model)
coef(Model)
plot(Model)
```

and observe that the robust results, obtained with `y[12]=0`, are very similar to those of the classical analysis when `y[12]=3.2`.

Remark. Traditional advice has been to compute robust estimates and, if they differ little from the least squares estimates, then use the least squares estimates. On the other hand, if the difference is great, then eliminate the outliers and recompute all estimates. Once again, if the difference between the least squares and robust estimates is not too great, use the least squares estimates. This advice requires an answer to the question, “how do I tell if the robust estimates and the least squares estimates are close?”. This question is not answered here. Rather, S-Plus provides mechanisms through which estimates and graphical figures from different fits can be compared. The user must then decide if the robust and least squares estimates are too different. To use this mechanism, type

```
lsfit <- lm(y~P+L1)
hbfit <- update(lsfit, .~.,method="Huber")
comparison <- fits.compare(lsfit,hbfit)
comparison
plot(comparison)
```

Section 2 briefly describes the Huber estimate, some procedures for testing linear hypotheses based on Huber estimates, and the computational algorithms. The level of this section should be appropriate for users that are more interested in practical applications than in theoretical aspects. The description refers to Huber (1981) and Hampel et al. (1986) for theoretical details. Most algorithms have been taken from the ROBETH library described in Marazzi (1993), which makes available many more computational options than those implemented by means of the object oriented functions. Section 3 gives a complete description of the object oriented functions and Section 4 completes the bioassay example.

2 The Huber estimate

Let $X = (x_{ij})$ denote a real $n \times p$ matrix, $y = (y_1, \dots, y_n)^T$ denote an observed n -vector of responses, and $\theta = (\theta_1, \dots, \theta_p)^T$ denote an unknown p -vector of coefficients to be estimated. We consider the usual linear model

$$y = X\theta + e,$$

where $e = (e_1, \dots, e_n)^T$ is an n -vector of unknown errors. It is assumed that, for given X , the components e_i of e are independent and identically distributed according to a distribution $\mathcal{L}(\cdot/\sigma)$, where σ is an unknown scale parameter. Denote the i th row of the matrix X by x_i^T .

2.1 Estimation procedure

The classical estimate of θ is the value of θ that minimizes the sum of squares $\sum_i (y_i - \sum_j x_{ij}\theta_j)^2$. This value is usually computed by solving the system of p normal equations obtained by differentiating this sum with respect to the parameters and equating the partial derivatives to 0. The scale parameter σ is then estimated by $\hat{\sigma} = (\sum_i r_i^2 / (n-p))^{1/2}$, where the r_i are the residuals with respect to the least squares fit.

Suppose for simplicity that σ is known; in this case Huber proposes to minimize:

$$Q(\theta) = \sum_{i=1}^n \rho\left(\frac{y_i - \sum_{j=1}^p x_{ij}\theta_j}{\sigma}\right), \quad (1)$$

where ρ is a convex function of the residuals that increases less rapidly than the squared function. By taking partial derivatives of Q with respect to $\theta_1, \dots, \theta_p$ and defining $\psi = \rho'$, we obtain the following system of equations for $\theta_1, \dots, \theta_p$:

$$\sum_{i=1}^n \psi\left(\frac{y_i - \sum_{j=1}^p x_{ij}\theta_j}{\sigma}\right) x_{ik} = 0, \quad k = 1, \dots, p. \quad (2)$$

The solution $\hat{\theta}_1, \dots, \hat{\theta}_p$ of these equations is called an *M-estimate of $\theta_1, \dots, \theta_p$ of Huber-type*. At first, the function ρ is arbitrary and must be chosen. On theoretical grounds, Huber proposes

$$\rho(r) = \begin{cases} c|r| - c^2/2 & \text{for } |r| \geq c \\ r^2/2 & \text{for } |r| < c, \end{cases}$$

which corresponds to $\psi(r) = \max[-c, \min(c, r)]$. In the following, this pair of functions ρ and ψ are denoted by ρ_c and ψ_c and are called the *Huber functions*.

The most common value of the *tuning constant* c used in practice is $c = 1.345$. In this case, the Huber estimate is about 95% as precise as the least squares estimate for Gaussian errors. It is (asymptotically), however, the most precise (maximum likelihood) estimate with respect to the worst distribution (i.e., the least favorable distribution in a two-person game between nature and statistician) of the form $F(s) = (1 - \epsilon)\Phi(s) + \epsilon H(s)$, where H is an arbitrary symmetric distribution (*contamination*) and $\epsilon \approx 6\%$: (In other words, the Huber estimate is minimax. There are also justifications for the Huber estimate not requiring the symmetry assumption.) Different values of c correspond to different efficiency losses at the Gaussian model and to different values of ϵ , that is, to different degrees of robustness. Some S-plus functions that may help in determining other values of c are described in Marazzi (1993, Chapter 5).

In S-plus, we define a Huber-type estimate $\hat{\theta}$ as follows. If σ is known, $\hat{\theta}$ is the solution of the system of p equations (2), where ψ is a suitable user-defined function. As σ is usually unknown, three options are made available in order to estimate its value:

1. Use the value of a preliminary estimate. This value is kept fixed when the solution of (2) is computed.
2. Solve the system formed by (2) and the supplementary equation (for σ):

$$\frac{1}{n-p} \sum_{i=1}^n \chi\left(\frac{r_i}{\sigma}\right) = \beta. \quad (3)$$

Here, χ is another suitable function and $r = y - X\theta$ is the residual vector $r = (r_1, \dots, r_n)^T$. The constant β is chosen so that the solution $\hat{\sigma}$ is an asymptotically consistent estimate of σ when $\mathcal{L}(\cdot/\sigma) = \Phi(\cdot)$, i.e., $\beta = \int \chi(s)d\Phi(s)$. A common choice of χ is $\chi_c = \psi_c^2/2$ which gives the classical estimate of σ in the limit case $c \rightarrow \infty$.

3. Solve the system formed by (2) and the supplementary equation:

$$\sigma = \text{med}_i |y_i - x_i^T \theta| / \beta_0. \quad (4)$$

The solution $\hat{\sigma}$ is known as the *median absolute deviation* of the residuals. In this case, $\beta_0 = \Phi^{-1}(0.75)$ makes the estimate $\hat{\sigma}$ asymptotically unbiased for normal errors.

Various choices of functions ψ , χ , and ρ have been considered in the literature (Huber (1981, Chapter 4; Hampel et al., 1986, Chapter 6). For example, Hampel et al. (p. 149) consider *redescending ψ -functions* (such that $\psi(s) = 0$ for $|s|$ larger than a given value) that completely eliminate the influence of very distant outliers. A well-known particular case is Tukey's biweight $\psi(s) = s[\max(1 - s^2, 0)]^2$. A repertory of functions ψ , χ , and ρ is made available in the ROBETH library (Marazzi, 1993, Chapter 11) and can be used with the object oriented functions.

2.2 Covariance matrix of the estimated coefficients

Under regularity conditions the Huber estimate $\hat{\theta}$ is asymptotically normal with expected value θ and asymptotic covariance matrix

$$C = \sigma^2 f_H (X^T X)^{-1},$$

where $f_H = \int \psi^2(s)d\Phi(s) / (\int \psi'(s)d\Phi(s))^2$. This matrix is estimated by

$$\hat{C} = \hat{\sigma}^2 \hat{f}_H (X^T X)^{-1},$$

where \hat{f}_H is the coefficient defined in Huber (1981, p. 173, formulas (6.5) and (6.8)), i.e.,

$$\hat{f}_H = \hat{f}_0^2 \left(\frac{n}{n-p} \right) \frac{\text{ave } \psi^2}{(\text{ave } \psi')^2},$$

where

$$\begin{aligned} \hat{f}_0 &= 1 + \frac{p}{n} \frac{\text{var } \psi'}{(\text{ave } \psi')^2}, \\ \text{ave } \psi^2 &= \frac{1}{n} \sum \psi^2(r_i/\hat{\sigma}), \\ \text{ave } \psi' &= \frac{1}{n} \sum \psi'(r_i/\hat{\sigma}), \\ \text{var } \psi' &= \frac{1}{n} \sum [\psi'(r_i/\hat{\sigma}) - \text{ave } \psi']^2. \end{aligned}$$

2.3 Proportion of explained variation R^*

This measure corresponds (and reduces) to the R^2 in the classical case. If there is an intercept term, then we compute the location Huber estimate \hat{t} defined by

$$\sum_{i=1}^n \psi((y_i - \hat{t})/\hat{\sigma}) = 0;$$

otherwise, we set $\hat{t} = 0$. Finally, we define

$$R^* = \frac{\sum \rho((y_i - \hat{t})/\hat{\sigma}) - \sum \rho((y_i - x_i^T \theta^{(1)})/\hat{\sigma})}{\sum \rho((y_i - \hat{t})/\hat{\sigma})}.$$

2.4 Deviance D^*

Define the deviance D^* as the optimal value of the objective function on the σ^2 -scale. More precisely let:

$$D^* = 2 \cdot (\hat{\sigma})^2 \sum \rho\left(\frac{(y_i - x_i^T \hat{\theta})}{\hat{\sigma}}\right).$$

This definition coincides with the residual sum of squares in the classical case ($\rho(s) = s^2/2$).

2.5 Test of a linear hypothesis

In order to test a linear hypothesis two options are made available: the τ -test and the R_n^2 -test described in Hampel et al. (1986, Chapter 7). Consider the p parameter model

$$\Omega : \quad y = X\theta + e,$$

let a linear hypothesis be expressed in the canonical form

$$\mathcal{H}_0 : \theta_{q+1} = \theta_{q+2} = \dots = \theta_p, \quad 0 < q < p,$$

and denote by ω the submodel of Ω obtained by imposing the condition \mathcal{H}_0 . Denote by $\hat{\theta}_\omega$, resp. $\hat{\theta}_\Omega$, the Huber estimate of θ in the model ω , resp. Ω , and let $\hat{\sigma}_\Omega$ be the estimate of σ under Ω . The τ -test statistic for testing \mathcal{H}_0 is defined as

$$F_\tau = \frac{2}{p-q} \sum_{i=1}^n (\rho(r_{\omega,i}/\hat{\sigma}_\Omega) - \rho(r_{\Omega,i}/\hat{\sigma}_\Omega)),$$

where $r_{\omega,i} = y_i - x_i^\top \hat{\theta}_\omega$ and $r_{\Omega,i} = y_i - x_i^\top \hat{\theta}_\Omega$. The null distribution of F_τ can be approximately evaluated as the distribution of

$$\left[\int \psi^2(s) d\Phi(s) / \int \psi'(s) d\Phi(s) \right] Z_{p-q},$$

where Z_{p-q} is a χ^2 -distributed random variable with $p-q$ degrees of freedom and Φ denotes the standard cumulative Gaussian distribution function.

Let K_{22} be the $(p-q) \times (p-q)$ lower right block of the asymptotic covariance matrix of the Huber estimate $\hat{\theta}_\Omega$. The R_n^2 -test statistic is defined by

$$R_n^2 = n(\hat{\theta}_{\Omega,q+1}, \dots, \hat{\theta}_{\Omega,p}) K_{22}^{-1} (\hat{\theta}_{\Omega,q+1}, \dots, \hat{\theta}_{\Omega,p})^\top.$$

Under \mathcal{H}_0 , the R_n^2 -test statistic follows approximately a χ^2 -distribution with $(p-q)$ degrees of freedom.

2.6 Computation of the Huber estimate

Suppose that

$$\psi(s) = \rho'(s) \quad \text{and} \quad \chi(s) = s\psi(s) - \rho(s). \quad (5)$$

Then, the solution of (2)–(3) characterizes the minimum of

$$Q(\theta, \sigma) = \sum_{i=1}^n \rho\left(\frac{r_i}{\sigma}\right) \sigma + \sigma(n-p)\beta.$$

Very often $\rho(s) = \rho_c(s)$, $\psi = \psi_c$, $\chi = \psi_c^2/2$, and most algorithms are based on this particular case. These methods are also applied to other situations, in particular with redescending ψ functions such as Tukey's biweight.

The scale step. Iteration cycles split into an improvement step for θ and one for σ . For example, if $\sigma^{(0)}$ denotes the current value of σ , an improved solution of (3) is

$$\sigma^{(1)} := \sigma^{(0)} \sqrt{\frac{1}{(n-p)\beta} \sum \chi\left(\frac{r_i}{\sigma^{(0)}}\right)}.$$

We can replace (3) by (4), but convergence proofs are known only for (2) and (3) under conditions that are somewhat more stringent than (5).

The coefficient step. In what follows, attention is restricted to the θ -step and σ is supposed fixed. The Newton algorithm is a natural choice justified by the approximate quadratic behaviour of Q near its minimum. The gradient and the Hessian matrix are

$$\nabla Q := \frac{\partial Q}{\partial \theta} = -\frac{1}{\sigma} X^\top r^* \quad \text{and} \quad H := \frac{\partial^2 Q}{\partial \theta \partial \theta^\top} = \frac{1}{\sigma} X^\top \mathcal{D} X,$$

where $r^* = (r_1^*, \dots, r_n^*)^T$, and

$$\mathcal{D} = \text{diag} \left(\psi' \left(\frac{r_i}{\sigma} \right) \right),$$

$$r_i^* = \psi \left(\frac{r_i}{\sigma} \right) \sigma \quad \text{for } i = 1, \dots, n.$$

An improvement step δ is therefore found by solving $X^T \mathcal{D} X \delta = X^T r^*$. The well-known drawbacks are (1) H and its inverse must be recalculated at each iteration; (2) H can become singular; (3) in order to prevent oscillations, an adaptive *step length* must be introduced. Various tricks can be devised for overcoming these difficulties. The most popular procedures replace \mathcal{D} by some approximation $\bar{\mathcal{D}}$. Programs that implements these procedures can be found in the ROBETH library (Marazzi, 1993). The object oriented S-plus functions uses the *H-algorithm* or the *W-algorithm*. The H-algorithm sets $\bar{\mathcal{D}} = I$, that is, $\bar{H} = (1/\sigma)X^T X$. With this approximation, \bar{H} must be inverted only once and no adaptation of the step length is needed (Huber, 1981). The W-algorithm (also known as the *iteratively reweighted least squares algorithm*) sets $\bar{\mathcal{D}} = \text{diag}(\psi(r_1/\sigma)/(r_i/\sigma))$. Again, no adaptive step length is needed; \bar{H} changes at each step, but the number of iterations is usually very low.

Initial values. By default, the starting values for the iterative algorithm are the least squares estimates of θ and σ . Optionally, the L_1 -norm estimate of θ (defined by the minimum of $\sum |r_i|$) and the associated median absolute deviation can be used.

Remark. There is a nice geometrical interpretation of the H-algorithm. Diagram a in Figure 1 shows some data points and the least squares line l_0 . Although this line is unsatisfactory, we use it as the starting value for an iterative process. In diagram b, a band centered on l_0 has been indicated by two parallel lines; the band width is not essential in this example but usually depends on the data dispersion. In the same diagram, the points outside the band have been translated, parallel to the y -axis, to the border of the band. In this way, new *pseudoobservations* can be defined: They coincide with the original points inside the band and with the “modified points” on the border. Diagram c shows the least squares line l_1 that has been computed using the pseudoobservations. A new band is placed on l_1 , new pseudoobservations are computed, and so on. After 10 iterations we obtain the line l_{10} indicated in diagram d.

As explained in Huber (1981, p. 19), the idea behind this intuitive procedure is a general recipe that works for many robust procedures: “Clean the data by pulling outliers towards their fitted values, refitting iteratively until convergence is obtained.”

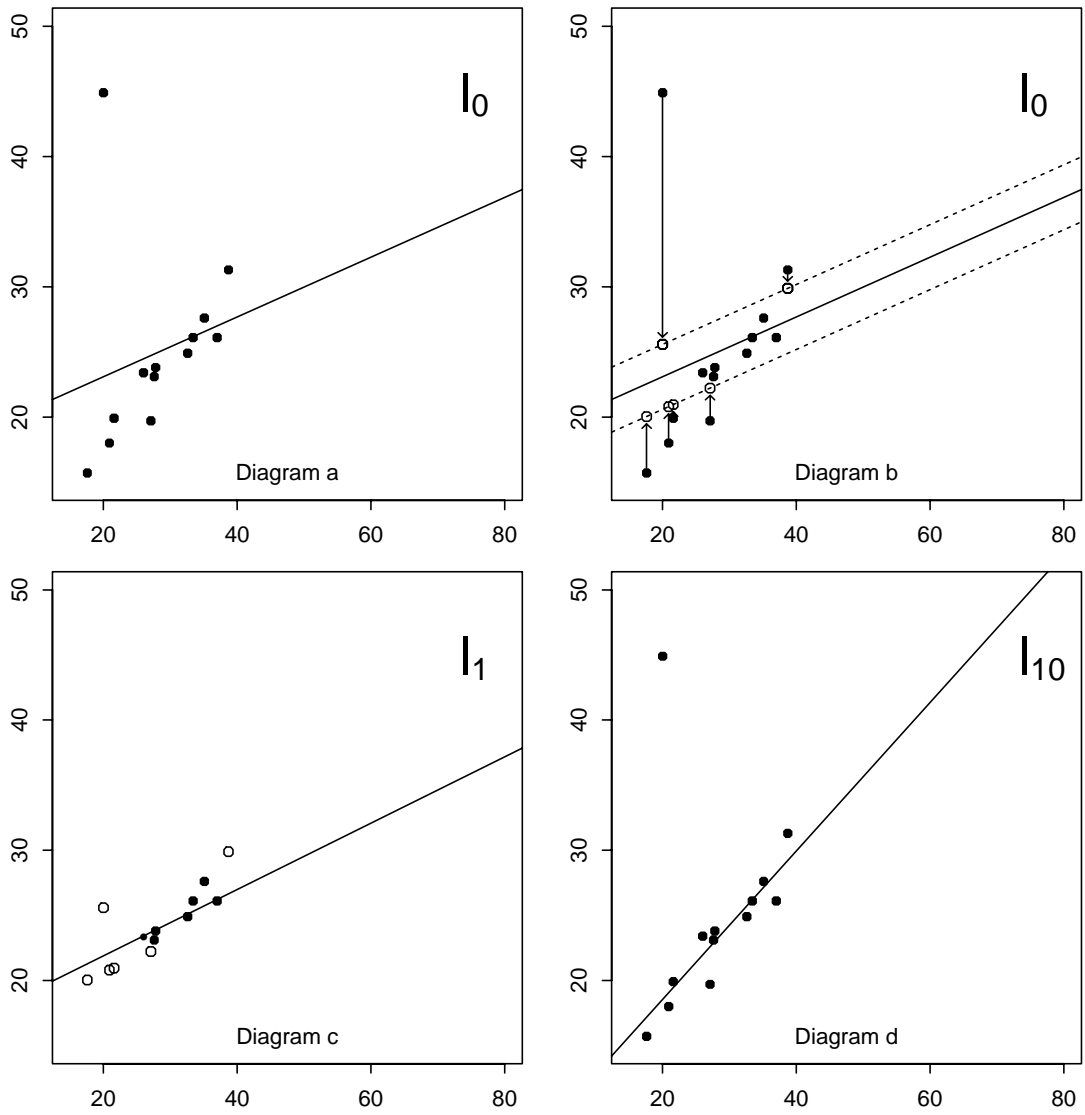


Figure 1. Geometrical interpretation of the H-algorithm

3 S-Plus Functions

Two types of functions are provided:

- (a) Functions for the numerical computations of the estimates;
- (b) Interface functions for the use of the estimation functions within the object oriented paradigm of S-Plus, as described in Chambers and Hastie (1992).

The functions of type (b) are likely to be appropriate for most users and applications. In order to control the computational algorithms in details some users may want to use functions of type (a). In addition there are service functions which are normally intended for programmers and developers. These functions are not necessarily documented elsewhere and are listed here for completeness.

3.1 The estimation functions

The main estimation function is `hubreg()`, which is used by `lm()` to fit the Huber estimate. Alternatively, `hubreg()` can be called directly. `hubreg()` returns a list containing the Huber estimate of coefficients and scale. (see `help(hubreg)` for more details). Computations are performed using routines from the ROBETH library as follows:

1. The initial values of $\hat{\theta}$ and $\hat{\sigma}$ are computed using the subroutines `RICLLS` (default) or `RILARS` (optional) of ROBETH (Marazzi 1993, p.67 and p.71).
2. The least squares problem is solved by means of the QR-decomposition. This is computed using the subroutine `RIMTRF` of ROBETH (Marazzi 1993, p. 64).
3. The Huber estimate is computed by means of the subroutine `RYHALG` (default) or `RYWALG` (optional) of ROBETH (Marazzi 1993, p.78 and p. 83).
4. Covariance matrices of estimated coefficients are obtained using the subroutines `KIASCV`, `KFASCV` and `KFFACV` of ROBETH (Marazzi 1993, p.64 and pp. 150-154).

The following function can be used in order to change defaults:

- `lm.Huber.control()` – There are a few control parameters for the numerical algorithms. Their default values are collected in the auxiliary function `lm.Huber.control()` and may be set with this function. The control structure input in `hubreg()` defaults to the list returned by `lm.Huber.control()`.

Users may wish to set their own control parameters. This is easily accomplished. In either `hubreg()` or `lm()`, simply set `control=xcontrol` in the calling sequence of either function. Here `xcontrol` is any object produced by `lm.Huber.control()` with the desired control parameter values. Use `help(lm.Huber.control)` for additional details.

The following functions are service functions:

- `comval()` and `dfcomn()` — The ROBETH routines use a few common blocks. Common blocks parameters are retrieved and modified by means of the service functions `comval()` and `dfcomn()` (Marazzi 1993, p. 405).
- The functions `rimtrf()`, `riclls()`, `ryhalg()`, `rywalg()`, `rilaris()`, `kiascv()`, `kfascv()`, `kffacv()`, `liepsh()`, `ribet0()`, `ribeth()`, are wrapper functions for the ROBETH routines used by `hubreg()`.
- ROBETH routines call a few auxiliary routines as described in Marazzi (1993).

3.2 The object oriented interface

The function `hubreg()` computes the robust estimates and returns a list of results. These are more easily produced using `method="Huber"` in the `lm()` function. The value returned by `lm()` is an object with class "lm.Huber" inheriting from class "lm".

The functions `print()`, `summary()`, `plot()`, `update()`, and `anova()`, as well as the access functions `coef()`, `residual()`, `fitted()`, `formula()` and `deviance()` have been extended to objects of the class "lm.Huber".

The following access functions are new.

- `scale.estimate()`, `r.squared()`, `covar()`, `correl()`, `weights()`, and `Rank()` — These access functions extract the scale estimate, the proportion of explained variation R^* , the covariance matrix of the estimated coefficients, the corresponding correlation matrix, and the rank of the design matrix from an object of class "lm" or "lm.Huber".

Note. For `method="Huber"`, the weights are set to 1 when the H-algorithm is used and to $\psi(r_i/\hat{\sigma})/(r_i/\hat{\sigma})$ when the W-algorithm is used.

In addition, a function for comparing two fits has been added.

- `fits.compare()` — The `fits.compare()` function accepts a sequence of objects of class "lm", "lm.Huber", or "aov" (with optional names), and creates a class "fits.compare" object. The "fits.compare" object is nothing more than a list of the input objects with names. However, when the "fits.compare" object is printed, summaries of each of the input objects are computed and printed in a manner suitable for comparing the input models. Plotting the "fits.compare" object results in a sequence of graphical displays. These displays are designed to be of use in comparing two sets of parameter estimates in linear models.

Note. The `fits.compare()` and the extractor functions also accepts objects of class "lm.robust" (Clarkson and Marazzi, 1997), "glm" and "cubinf" (Marazzi, 1997). It is not recommended to compare objects with different structure.

Finally, a number of new auxiliary functions are required by the interface. Many of these routines are simple program stubs used to print error messages that the associated `lm()` function is not yet available for objects from the class.

- `lm.fit.Huber()` — Set up to call `hubreg()`. `lm.fit.Huber()` is called by `lm()` when `method="Huber"`. It handles some of the record keeping required by the algorithms, calls `hubreg()`, and assigns the class "lm.Huber" (which inherits from "lm") to the list of parameter estimates returned by `hubreg()`.

Note. Preprocessing before the estimation procedure is done by the function `lm()` which remains unchanged. `lm()` calls `lm.fit()` or `lm.wfit()` (both unchanged). `lm.fit()` and `lm.wfit()` call `lm.fit.Huber()`. Observation weights, if needed, are handled correctly by the existing function `lm.wfit()`.

- `print.lm.Huber()`, `summary.lm.Huber()`, `print.summary.lm.Huber()` — These functions extend the `print.lm()`, `summary.lm()`, and `print.summary.lm()` functions to objects of class "lm.Huber".
- `plot.lm.Huber()` — extends `plot()` to objects of class "lm.Huber".
- `update.lm.Huber()` — extends `update()` to objects of class "lm.Huber".
- `anova.lm.Huber()` — extends `anova()` to objects of class "lm.Huber".

- `plot.fits.compare()` — extends `plot()` to objects of class “fits.compare”.
- `deviance.lm.Huber()`, `scale.estimate.lm.Huber()`, `r.squared.lm.Huber()`, `covar.lm.Huber()`, `correl.lm.Huber()`, `weights.lm.Huber()`. `Rank.lm.Huber()` — These functions are required by the corresponding access functions.
- `add1.lm.Huber()`, `drop1.lm.Huber()`, and `step.lm.Huber()` — These are function stubs used to print error messages that the associated routines are not yet implemented. Stepwise search algorithms have not yet been implemented for objects of class “lm.Huber”.
- Other function stubs — In addition to the stepwise search functions, the class “lm” functions `effects()`, `kappa()`, `proj()`, `alias()`, `lm.influence()`, and `lm.sensitivity()` are not applicable to objects of class “lm.Huber”. Therefore, function stubs `effects.lm.Huber()`, `kappa.lm.Huber()`, `proj.lm.Huber()`, `lm.influence.lm.Huber()`, `alias.lm.Huber()`, and `lm.sensitivity.lm.Huber()` have been added to issue warning messages for these functions.

4 Advanced example

We consider a new set of artificial dose response data and adjust the model Ω of Section 1 by means of the least squares and the Huber estimates. Moreover, we estimate the covariance matrix of the coefficient estimates by bootstrapping residuals as described in Efron and Tibshirani (1993). In order to do efficient computations, we use the low level functions `lm.fit.qr()` (standard S-plus) and `hubreg()` in place of `lm(...)` and `lm(..., method="Huber")`. The functions `bootstrap()` (Efron and Tibshirani 1993), `sample.resid.ls()` and `sample.resid.hb()` listed in the Appendix must be loaded.

Enter the data, define the explanatory variables, and the design matrix:

```

y      <- c(0.8, 1.2, 1.1, 1.8, 2.2, 1.7, 2.8, 3.2, 3.1
           0.8, 1.2, 0.9, 1.8, 2.2, 2.1, 2.8, 3.2, 0.0)
Dose   <- c(1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6)
P      <- c(rep(-1,9),rep(1,9))
n      <- 18
XS     <- XT <- rep(0,n)
Xt[P== -1] <- Dose[P== -1]; Xs[P== +1] <- Dose[P== +1]
L1    <- XS+XT; Lp1 <- XS-XT
Xmat  <- cbind(1,P,L1,Lp1)

```

Open a graphic windows (e.g., use `win.graph()` under Windows). Compute the least squares estimate and bootstrap its residuals:

```

lsfit <- lm(y~P+L1+Lp1)
diagrM(Dose,y,P,lsfit); text(5,3,"ls",cex=2)
cf    <- coef(lsfit); res <- resid(lsfit)
xcf   <- as.matrix(Xmat)%*%cf
.Random.seed <- c(35,51,42,30,21,2,13,63,50,22,6,3); i <- 0
coef.star.ls <- bootstrap(1:n, nboot=100,
                        sample.resid.ls,res,Xmat,xcf)$thetastar

```

Print the usual and the bootstrap estimates of the coefficient standard errors:

```

sqrt(diag(covar(lsfit)))
sqrt(diag(var(t(coef.star.ls[1:4,]))))

```

These estimates do not seem very reliable ! Compute the Huber estimate and bootstrap its residuals:

```
hbfit <- lm(y~P+L1+Lp1,method="Huber")
diagrM(Dose,y,P,hbfit);text(5,3,"Huber",cex=2)
cf <- coef(hbfit); res <- resid(hbfit)
xcf <- as.matrix(Xmat)%*%cf
.Random.seed <- c(35,51,42,30,21,2,13,63,50,22,6,7); i <- 0
coef.star.hb <- bootstrap(1:n, nboot=100,
                          sample.resid.hb,res,Xmat,xcf)$thetastar
```

Print the usual and the bootstrap estimates of the coefficient standard errors:

```
sqrt(diag(covar(hbfit)))
sqrt(diag(var(t(coef.star.hb[1:4,]))))
```

These estimates seem much more reliable !

Acknowledgment. This work was supported by grant No. 21-36521.92. from the Swiss National Science Foundation

References

- Chambers J.M., Hastie T.J., Eds. (1992). *Statistical Models in S*, Wadsworth & Brooks/Cole Computer Science Series, Pacific Grove.
- Clarkson D.B., Marazzi A. (1997). Object oriented S-plus functions for high breakdown point and high efficiency regression with test for bias. Technical report. Inst Univ Med Soc Prev, Bugnon 17, CH-1005 Lausanne.
- Efron B., Tibshirani R. (1993). An introduction to the bootstrap. Chapman and Hall, New York.
- Finney D.J. (1978). *Statistical Method in Biological Assay*, Third edition, Charles Griffin & Company Ltd., London and High Wycombe.
- Hampel F.R., Ronchetti E.M., Rousseeuw P.J., Stahel W.A. (1986). *Robust Statistics: The Approach Based on Influence Functions*, Wiley, New York.
- Huber P. (1973). Robust regression: Asymptotics, conjectures and Monte Carlo. *Ann. Statist.*, 1, 799-821.
- Huber P. (1981). *Robust Statistics*, Wiley, New York.
- Marazzi A. (1993). *Algorithms, Routines and S Functions for Robust Statistics, The FORTRAN library ROBETH with an interface to S-Plus*, Chapman and Hall, New York.
- Marazzi A. (1997). Object oriented S-plus functions for robust generalized linear models. Technical report. Inst Univ Med Soc Prev, Bugnon 17, CH-1005 Lausanne.

Appendix

```
potcy <- function(lmfit,sigma){
  Pi <- lmfit$coeff[2]
  Gam1 <- lmfit$coeff[3]
  K <- 2*Pi/Gam1
  R <- exp(K)
  cov <- summary(lmfit)$cov.unscaled
  M22 <- cov[2,2]; M32 <- cov[3,2]; M33 <- cov[3,3]
  V11 <- 4*M22; V12 <- 2*M32; V22 <- M33
  z <- qnorm(.975)
  a <- z^2*sigma^2*V22/Gam1^2
  b <- z*sigma*(V11-2*K*V12+K^2*V22-a*(V11-V12^2/V22))^0.5/abs(Gam1)
  A <- K-a*V12/V22
  Kl <- (A-b)/(1-a); Ku <- (A+b)/(1-a)
  Rl <- exp(Kl); Ru <- exp(Ku)
  names(K)<-names(Kl)<-names(Ku)<-names(R)<-names(Rl)<-names(Ru)<-""
  list(K=K,Kl=Kl,Ku=Ku,R=R,Rl=Rl,Ru=Ru)}

diagrM <- function(Dose,Resp,Prep,Model){
  X <- seq(min(Dose),max(Dose),length=100)
  cf <- coef(Model)
  Eta1 <- cf[3]+cf[4]; Theta1 <- cf[3]-cf[4]
  Ys <- cf[1]-cf[2]+Eta1*X ; Yt <- cf[1]+cf[2]+Theta1*X
  plot(Dose,Resp,type="n")
  points(Dose[Prep== -1],Resp[Prep== -1],pch=16)
  points(Dose[Prep== +1],Resp[Prep== +1],pch=3)
  points(6,Resp[12],col=2,pch=16)
  lines(X,Ys); lines(X,Yt)}

sample.resid.ls <- function(indices,res,x,xcoef){i <- i+1;cat(i,"\n")
  ystar <- xcoef+res[indices]
  zstar <- lm.fit.qr(x,ystar,singular.ok=T); cf <- zstar$coefficients
  abline(cf[1]-cf[2],cf[3]+cf[4],lty=2)
  abline(cf[1]+cf[2],cf[3]-cf[4],lty=2)
  cf}

sample.resid.hb <- function(indices,res,x,xcoef){i <- i+1;cat(i,"\n")
  ystar <- xcoef+res[indices]
  zstar <- hubreg(x,ystar); cf <- zstar$coefficients
  abline(cf[1]-cf[2],cf[3]+cf[4],lty=2)
  abline(cf[1]+cf[2],cf[3]-cf[4],lty=2)
  cf}

bootstrap <- function(x, nboot, theta, ...){
  call <- match.call(); n <- length(x)
  data <- matrix(sample(x, size=n*nboot, replace=T),
    nrow=nboot, byrow=T)
  thetastar <- apply(data, 1, theta, ...)
  list(thetastar = thetastar, call = call)}
```