

nag_robust_m_regsn_estim (g02hac)

1. Purpose

nag_robust_m_regsn_estim (g02hac) performs bounded influence regression (M-estimates). Several standard methods are available.

2. Specification

```
#include <nag.h>
#include <nagg02.h>

void nag_robust_m_regsn_estim(Nag_RegType regtype, Nag_PsiFun psifun,
    Nag_sigmaEst sigma_est, Nag_CovMatrixEst covmat_est, Integer n,
    Integer m, double x[], Integer tdx, double y[],
    double cpsi, double hpsi[], double cucv, double dchi,
    double theta[], double *sigma, double c[], Integer tdc,
    double rs[], double wt[], double tol, Integer max_iter,
    Integer print_iter, char *outfile, double info[], NagError *fail)
```

3. Description

For the linear regression model

$$y = X\theta + \epsilon$$

where y is a vector of length n of the dependent variable,

X is a n by m matrix of independent variables of column rank k ,

θ is a vector of length m of unknown parameters,

and ϵ is a vector of length n of unknown errors with $\text{var}(\epsilon_i) = \sigma^2$:

nag_robust_m_regsn_estim calculates the M-estimates given by the solution, $\hat{\theta}$, to the equation

$$\sum_{i=1}^n \psi(r_i/(\sigma w_i)) w_i x_{ij} = 0 \quad j = 1, 2, \dots, m \quad (1)$$

where r_i is the i th residual i.e., the i th element of $r = y - X\hat{\theta}$,

ψ is a suitable weight function,

w_i are suitable weights,

and σ may be estimated at each iteration by the median absolute deviation of the residuals:

$$\hat{\sigma} = \text{med}_i[|r_i|]/\beta_1$$

or as the solution to:

$$\sum_{i=1}^n \chi(r_i/(\hat{\sigma} w_i)) w_i^2 = (n - k)\beta_2$$

for suitable weight function χ , where β_1 and β_2 are constants, chosen so that the estimator of σ is asymptotically unbiased if the errors, ϵ_i , have a Normal distribution. Alternatively σ may be held at a constant value.

The above describes the Schweppe type regression. If the w_i are assumed to equal 1 for all i then Huber type regression is obtained. A third type, due to Mallows, replaces (1) by

$$\sum_{i=1}^n \psi(r_i/\sigma) w_i x_{ij} = 0, \quad j = 1, 2, \dots, m.$$

This may be obtained by use of the transformations

$$\begin{aligned} w_i^* &\leftarrow \sqrt{w_i} \\ y_i^* &\leftarrow y_i \sqrt{w_i} \\ x_{ij}^* &\leftarrow x_{ij} \sqrt{w_i}, \quad j = 1, 2, \dots, m \end{aligned}$$

(see Marazzi (1987a)).

For Huber and Schweppe type regressions, β_1 is the 75th percentile of the standard Normal distribution. For Mallows type regression β_1 is the solution to

$$\frac{1}{n} \sum_{i=1}^n \Phi(\beta_1 / \sqrt{w_i}) = .75$$

where Φ is the standard Normal cumulative distribution function.

β_2 is given by:

$$\begin{aligned} \beta_2 &= \int_{-\infty}^{\infty} \chi(z) \phi(z) dz, && \text{in Huber case;} \\ \beta_2 &= \frac{1}{n} \sum_{i=1}^n w_i \int_{-\infty}^{\infty} \chi(z) \phi(z) dz, && \text{in Mallows case;} \\ \beta_2 &= \frac{1}{n} \sum_{i=1}^n w_i^2 \int_{-\infty}^{\infty} \chi(z/w_i) \phi(z) dz, && \text{in Schweppe case;} \end{aligned}$$

where ϕ is the standard Normal density, i.e.,

$$\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right).$$

The calculation of the estimates of θ can be formulated as an iterative weighted least-squares problem with a diagonal weight matrix G given by

$$G_{ii} = \begin{cases} \frac{\psi(r_i/(\sigma w_i))}{(r_i/(\sigma w_i))}, & r_i \neq 0 \\ \psi'(0), & r_i = 0 \end{cases}$$

where $\psi'(t)$ is the derivative of ψ at the point t .

The value of θ at each iteration is given by the weighted least-squares regression of y on X . This is carried out by first transforming the y and X by

$$\begin{aligned} \tilde{y}_i &= y_i \sqrt{G_{ii}} \\ \tilde{x}_{ij} &= x_{ij} \sqrt{G_{ii}}, \quad j = 1, 2, \dots, m \end{aligned}$$

and then obtaining the solution of the resulting least squares problem. If X is of full column rank then an orthogonal-triangular (QR) decomposition is used, if not, a singular value decomposition is used.

The following functions are available for ψ and χ in nag_robust_m_regsn_estim.

(a) **Unit Weights**

$$\psi(t) = t, \quad \chi(t) = \frac{t^2}{2}$$

this gives least-squares regression.

(b) **Huber's Function**

$$\psi(t) = \max(-c, \min(c, t)), \quad \chi(t) = \begin{cases} \frac{t^2}{2}, & |t| \leq d \\ \frac{d^2}{2}, & |t| > d \end{cases}$$

(c) **Hampel's Piecewise Linear Function**

$$\psi_{h_1, h_2, h_3}(t) = -\psi_{h_1, h_2, h_3}(-t) = \begin{cases} t, & 0 \leq t \leq h_1 \\ h_1, & h_1 \leq t \leq h_2 \\ h_1(h_3 - t)/(h_3 - h_2), & h_2 \leq t \leq h_3 \\ 0, & h_3 < t \end{cases}$$

$$\chi(t) = \begin{cases} \frac{t^2}{2}, & |t| \leq d \\ \frac{d^2}{2}, & |t| > d \end{cases}$$

(d) **Andrew's Sine Wave Function**

$$\psi(t) = \begin{cases} \sin t, & -\pi \leq t \leq \pi \\ 0, & |t| > \pi \end{cases} \quad \chi(t) = \begin{cases} \frac{t^2}{2}, & |t| \leq d \\ \frac{d^2}{2}, & |t| > d \end{cases}$$

(e) **Tukey's Bi-weight**

$$\psi(t) = \begin{cases} t(1 - t^2)^2, & |t| \leq 1 \\ 0, & |t| > 1 \end{cases} \quad \chi(t) = \begin{cases} \frac{t^2}{2}, & |t| \leq d \\ \frac{d^2}{2}, & |t| > d \end{cases}$$

where c, h_1, h_2, h_3 , and d are given constants.

Several schemes for calculating weights have been proposed, see Hampel *et al.* (1986) and Marazzi (1987a). As the different independent variables may be measured on different scales, one group of proposed weights aims to bound a standardized measure of influence. To obtain such weights the matrix A has to be found such that:

$$\frac{1}{n} \sum_{i=1}^n u(\|z_i\|_2) z_i z_i^T = I$$

and

$$z_i = Ax_i$$

where x_i is a vector of length m containing the i th row of X ,

A is a m by m lower triangular matrix,

and u is a suitable function.

The weights are then calculated as

$$w_i = f(\|z_i\|_2)$$

for a suitable function f .

nag_robust_m_regn_estim finds A using the iterative procedure

$$A_k = (S_k + I)A_{k-1}$$

where $S_k = (s_{jl})$,

$$s_{jl} = \begin{cases} -\min[\max(h_{jl}/n, -BL), BL] & j > \ell \\ -\min[\max(\frac{1}{2}(h_{jj}/n - 1), -BD), BD] & j = \ell \end{cases}$$

and

$$h_{jl} = \sum_{i=1}^n u(\|z_i\|_2) z_{ij} z_{il}$$

and BL and BD are bounds set at 0.9.

Two weights are available in nag_robust_m_regsn_estim:

- (i) Krasker-Welsch weights

$$u(t) = g_1\left(\frac{c}{t}\right)$$

where $g_1(t) = t^2 + (1 - t^2)(2\Phi(t) - 1) - 2t\phi(t)$,
 $\Phi(t)$ is the standard Normal cumulative distribution function,
 $\phi(t)$ is the standard Normal probability density function,
 and $f(t) = \frac{1}{t}$.

These are for use with Schweppe type regression.

- (ii) Maronna's proposed weights

$$u(t) = \begin{cases} c/t^2 & |t| > c \\ 1 & |t| \leq c \end{cases}$$

$$f(t) = \sqrt{u(t)}.$$

These are for use with Mallows type regression.

Finally the asymptotic variance-covariance matrix, C , of the estimates θ is calculated.

For Huber type regression

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

where

$$f_H = \frac{1}{n - m} \frac{\sum_{i=1}^n \psi^2(r_i/\hat{\sigma})}{\left(\frac{1}{n} \sum_{i=1}^n \psi'\left(\frac{r_i}{\hat{\sigma}}\right)\right)^2} \kappa^2$$

$$\kappa^2 = 1 + \frac{m}{n} \frac{\frac{1}{n} \sum_{i=1}^n \left(\psi'(r_i/\hat{\sigma}) - \frac{1}{n} \sum_{i=1}^n \psi'(r_i/\hat{\sigma})\right)^2}{\left(\frac{1}{n} \sum_{i=1}^n \psi'\left(\frac{r_i}{\hat{\sigma}}\right)\right)^2}$$

See Huber (1981) and Marazzi (1987b).

For Mallows and Schweppe type regressions C is of the form

$$\frac{\hat{\sigma}^2}{n} S_1^{-1} S_2 S_1^{-1}$$

where $S_1 = \frac{1}{n}X^TDX$ and $S_2 = \frac{1}{n}X^TPX$.

D is a diagonal matrix such that the i th element approximates $E(\psi'(r_i/(\sigma w_i)))$ in the Schweppe case and $E(\psi'(r_i/\sigma)w_i)$ in the Mallows case.

P is a diagonal matrix such that the i th element approximates $E(\psi^2(r_i/(\sigma w_i))w_i^2)$ in the Schweppe case and $E(\psi^2(r_i/\sigma)w_i^2)$ in the Mallows case.

Two approximations are available in `nag_robust_m_regsn_estim`:

- (1) Average over the r_i

$$\begin{array}{ll} \text{Schweppe} & \text{Mallows} \\ D_i = \left(\frac{1}{n} \sum_{j=1}^n \psi' \left(\frac{r_j}{\hat{\sigma} w_i} \right) \right) w_i & D_i = \left(\frac{1}{n} \sum_{j=1}^n \psi' \left(\frac{r_j}{\hat{\sigma}} \right) \right) w_i \end{array}$$

- (2) Replace expected value by observed

$$\begin{array}{ll} \text{Schweppe} & \text{Mallows} \\ D_i = \psi' \left(\frac{r_i}{\hat{\sigma} w_i} \right) w_i & D_i = \psi' \left(\frac{r_i}{\hat{\sigma}} \right) w_i \\ P_i = \psi^2 \left(\frac{r_i}{\hat{\sigma} w_i} \right) w_i^2 & P_i = \psi^2 \left(\frac{r_i}{\hat{\sigma}} \right) w_i^2 \end{array}$$

See Hampel *et al.* (1986) and Marazzi (1987b).

Note: There is no explicit provision in the routine for a constant term in the regression model. However, the addition of a dummy variable whose value is 1.0 for all observations will produce a value of $\hat{\theta}$ corresponding to the usual constant term.

`nag_robust_m_regsn_estim` is based on routines in ROBETH, see Marazzi (1987a).

4. Parameters

`regtype`

Input: specifies the type of regression to be performed.

regtype = `Nag_HuberReg`, Huber type regression.

regtype = `Nag_MallowsReg`, Mallows type regression with Maronna's proposed weights.

regtype = `Nag_SchweppeReg`, Schweppe type regression with Krasker-Welsch weights.

Constraint: **regtype** = `Nag_HuberReg`, `Nag_MallowsReg` or `Nag_SchweppeReg`.

`psifun`

Input: specifies which ψ function is to be used.

psifun = `Nag_Lsq`, $\psi(t) = t$, i.e., least-squares,

psifun = `Nag_HuberFun`, Huber's function,

psifun = `Nag_HampelFun`, Hampel's piecewise linear function,

psifun = `Nag_AndrewFun`, Andrew's sine wave,

psifun = `Nag_TukeyFun`, Tukey's bi-weight.

Constraint: **psifun** = `Nag_Lsq`, `Nag_HuberFun`, `Nag_HampelFun`, `Nag_AndrewFun` or `Nag_TukeyFun`.

`sigma_est`

Input: specifies how σ is to be estimated.

sigma_est = `Nag_SigmaRes`, σ is estimated by median absolute deviation of residuals.

sigma_est = `Nag_SigmaConst`, σ is held constant at its initial value.

sigma_est = `Nag_SigmaChi`, σ is estimated using the χ function.

Constraint: **sigma_est** = `Nag_SigmaRes`, `Nag_SigmaConst` or `Nag_SigmaChi`.

`covmat_est`

Input: if **regtype** \neq `Nag_HuberReg`, **covmat_est** specifies the approximations used in estimating the covariance matrix of $\hat{\theta}$.

covmat_est = `Nag_CovMatAve`, averaging over residuals.

covmat_est = `Nag_CovMatObs`, replacing expected by observed.

If **regtype** = `Nag_HuberReg` then **covmat_est** is not referenced.

Constraint: **covmat_est** = `Nag_CovMatAve` or `Nag_CovMatObs`.

- n**
Input: the number of observations, n .
Constraint: $n > 1$.
- m**
Input: the number m , of independent variables.
Constraint: $1 \leq m < n$.
- x[n][tdx]**
Input: the values of the X matrix, i.e., the independent variables. $x[i-1][j-1]$ must contain the ij th element of X , for $i = 1, 2, \dots, n$; $j = 1, 2, \dots, m$.
Output: If **regtype** = **Nag_MallowsReg**, then during calculations the elements of **x** will be transformed as described in Section 3. Before exit the inverse transformation will be applied. As a result there may be slight differences between the input **x** and the output **x**. Otherwise **x** is unchanged.
- tdx**
Input: the second dimension of the array **x** as declared in the function from which nag_robust_m_regsn_estim is called.
Constraint: **tdx** $\geq m$.
- y[n]**
Input: the data values of the dependent variable.
 $y[i-1]$ must contain the value of y for the i th observation, for $i = 1, 2, \dots, n$.
Output: if **regtype** = **Nag_MallowsReg**, then during calculations the elements of **y** will be transformed as described in Section 3. Before exit the inverse transformation will be applied. As a result there may be slight differences between the input **y** and the output **y**. Otherwise **y** is unchanged.
- cpfi**
Input: if **psifun** = **Nag_HuberFun**, **cpfi** must specify the parameter, c , of Huber's ψ function. Otherwise **cpfi** is not referenced.
Constraint: if **psifun** = **Nag_HuberFun** then **cpfi** > 0.0 .
- hpsi[3]**
if **psifun** = **Nag_HampelFun** then **hpsi[0]**, **hpsi[1]** and **hpsi[2]** must specify the parameters h_1 , h_2 , and h_3 , of Hampel's piecewise linear ψ function. Otherwise the elements of **hpsi** are not referenced.
Constraints: if **psifun** = **Nag_HampelFun** then
 $0.0 \leq \mathbf{hpsi}[0] \leq \mathbf{hpsi}[1] \leq \mathbf{hpsi}[2]$ and
 $\mathbf{hpsi}[2] > 0.0$.
- cucv**
Input: if **regtype** = **Nag_MallowsReg** then **cucv** must specify the value of the constant, c , of the function u for Maronna's proposed weights.
If **regtype** = **Nag_SchweppeReg** then **cucv** must specify the value of the function u for the Krasker-Welsch weights.
If **regtype** = **Nag_HuberReg** then **cucv** is not referenced.
Constraints: if **regtype** = **Nag_MallowsReg**, **cucv** $\geq m$,
if **regtype** = **Nag_SchweppeReg**, **cucv** $\geq \sqrt{m}$.
- dchi**
Input: the constant, d , of the χ function.
dchi is referenced only if **psifun** \neq **Nag_Lsq** and **sigma_est** = **Nag_SigmaChi**.
Constraint: if **psifun** \neq **Nag_Lsq** and **sigma_est** = **Nag_SigmaChi**, **dchi** > 0.0 .
- theta[m]**
Input: starting values of the parameter vector θ . These may be obtained from least-squares regression.
Alternatively if **sigma_est** = **Nag_SigmaRes** and **sigma** = 1 or if **sigma_est** = **Nag_SigmaChi** and **sigma** approximately equals the standard deviation of the dependent variable, y , then **theta[i-1]** = 0.0, for $i = 1, 2, \dots, m$ may provide reasonable starting values.
Output: **theta[i-1]** contains the M-estimate of θ_i , for $i = 1, 2, \dots, m$.

sigma

Input: a starting value for the estimation of σ .

sigma should be approximately the standard deviation of the residuals from the model evaluated at the value of θ given by **theta** on entry.

Output: **sigma** contains the final estimate of σ , unless **sigma_est** = **Nag_SigmaConst**.

Constraint: **sigma** > 0.0.

c[m][tdc]

Output: the diagonal elements of **c** contain the estimated asymptotic standard errors of the estimates of θ , i.e., **c**[$i - 1$][$i - 1$] contains the estimated asymptotic standard error of the estimate contained in **theta**[$i - 1$], for $i = 1, 2, \dots, m$.

The elements above the diagonal contain the estimated asymptotic correlation between the estimates of θ , i.e., **c**[$i - 1$][$j - 1$], $1 \leq i < j \leq m$ contains the asymptotic correlation between the estimates contained in **theta**[$i - 1$] and **theta**[$j - 1$].

The elements below the diagonal contain the estimated asymptotic covariance between the estimates of θ , i.e., **c**[$i - 1$][$j - 1$], $1 \leq j < i \leq m$ contains the estimated asymptotic covariance between the estimates contained in **theta**[$i - 1$] and **theta**[$j - 1$].

tdc

Input: the second dimension of the array **c** as declared in the function from which **nag_robust_m_regsn_estim** is called.

Constraint: **tdc** \geq **m**.

rs[n]

Output: contains the residuals from the model evaluated at final value of **theta** i.e., **rs**[$i - 1$], for $i = 1, 2, \dots, n$, contains the vector $(y - X\hat{\theta})$.

wt[n]

Output: contains the vector of weights. **wt**[$i - 1$] contains the weight for the i th observation, for $i = 1, 2, \dots, n$.

tol

Input: the relative precision for the calculation of A (if **regtype** \neq **Nag_HuberReg**), the estimates of θ and the estimate of σ (if **sigma_est** \neq **Nag_SigmaConst**). Convergence is assumed when the relative change in all elements being considered is less than **tol**.

If **regtype** = **Nag_MallowsReg** and **sigma_est** = **Nag_SigmaRes**, **tol** is also used to determine the precision of β_1 .

It is advisable for **tol** to be greater than $100 \times$ *machine precision*.

Constraint: **tol** > 0.0.

max_iter

Input: the maximum number of iterations that should be used in the calculation of A (if **regtype** \neq **Nag_HuberReg**), and of the estimates of θ and σ , and of β_1 (if **regtype** = **Nag_MallowsReg** and **sigma_est** = **Nag_SigmaRes**)

Suggested value: A value of **max_iter** = 50 should be adequate for most uses.

Constraint: **max_iter** > 0.

print_iter

Input: the amount of information that is printed on each iteration.

If **print_iter** = 0 no information is printed.

If **print_iter** \neq 0 the current estimate of θ , the change in θ during the current iteration and the current value of σ are printed on the first and every **abs(print_iter)** iterations.

Also, if **regtype** \neq **Nag_HuberReg** and **print_iter** > 0 then information on the iterations to calculate A is printed. This is the current estimate of A and the maximum value of S_{ij} (see Section 3).

outfile

Input: a null terminated character string giving the name of the file to which results should be printed. If **outfile** = **NULL** or an empty string then the **stdout** stream is used. Note that the file will be opened in the append mode.

info[4]

Output: elements of **info** contain the following values:

- info**[0] = β_1 if **sigma_est** = **Nag_SigmaRes**,
- or **info**[0] = β_2 if **sigma_est** = **Nag_SigmaChi**,
- info**[1] = number of iterations used to calculate A .
- info**[2] = number of iterations used to calculate final estimates of θ and σ .
- info**[3] = k , the rank of the weighted least-squares equations.

fail

The NAG error parameter, see Introduction to the NAG C Library.

For this function the values of output parameters may be useful even if **fail.code** \neq **NE_NOERROR** on exit. Users are therefore advised to supply the **fail** parameter and set **fail.print** = **TRUE**.

5. Error Indications and Warnings**NE_INT_ARG_LT**

- On entry, **n** must not be less than 2: **n** = $\langle value \rangle$.
- On entry, **m** must not be less than 1: **m** = $\langle value \rangle$.

NE_2_INT_ARG_GE

On entry, **m** = $\langle value \rangle$ while **n** = $\langle value \rangle$. These parameters must satisfy **m** < **n**.

NE_2_INT_ARG_LT

- On entry, **tdx** = $\langle value \rangle$ while **m** = $\langle value \rangle$. These parameters must satisfy **tdx** \geq **m**.
- On entry, **tdc** = $\langle value \rangle$ while **m** = $\langle value \rangle$. These parameters must satisfy **tdc** \geq **m**.

NE_BAD_PARAM

- On entry, parameter **regtype** had an illegal value.
- On entry, parameter **psifun** had an illegal value.
- On entry, parameter **sigma_est** had an illegal value.
- On entry, parameter **covmat_est** had an illegal value.

NE_INT_ARG_LE

On entry, **max_iter** must not be less than or equal to 0: **max_iter** = $\langle value \rangle$.

NE_REAL_ARG_LE

- On entry, **sigma** must not be less than or equal to 0.0: **sigma** = $\langle value \rangle$.
- On entry, **tol** must not be less than or equal to 0.0: **tol** = $\langle value \rangle$.

NE_INVALID_HUBER_FUN

On entry, **psifun** = **Nag_HuberFun** and **cpsi** = $\langle value \rangle$. For this value of **psifun**, **cpsi** must be > 0.0.

NE_INVALID_MALLOWS_REG_C

On entry, **regtype** = **Nag_MallowsReg**, **cuvc** = $\langle value \rangle$ and **m** = $\langle value \rangle$. For this value of **regtype**, **cuvc** must be \geq **m**.

NE_INVALID_SCHWEPPE_REG_C

On entry, **regtype** = **Nag_SchweppeReg**, **cuvc** = $\langle value \rangle$ and **m** = $\langle value \rangle$. For this value of **regtype**, **cuvc** must be $\geq \sqrt{m}$.

NE_INVALID_DCHI_FUN

On entry, **psifun** \neq **Nag_Lsq**, **sigma_est** = **Nag_SigmaChi** and **dchi** = $\langle value \rangle$. For these values of **psifun** and **sigma_est**, **dchi** must be > 0.0.

NE_BAD_HAMPEL_PSI_FUN

On entry, **psifun** = **Nag_HampelFun** and **hpsi**[0] = $\langle value \rangle$, **hpsi**[1] = $\langle value \rangle$ and **hpsi**[2] = $\langle value \rangle$. For this value of **psifun**, the elements of **hpsi** must satisfy the condition $0.0 \leq \mathbf{hpsi}[0] \leq \mathbf{hpsi}[1] \leq \mathbf{hpsi}[2]$ and **hpsi**[2] > 0.0.

NE_WT_ITER_EXCEEDED

The number of iterations required to calculate the weights exceeds **max_iter**. This is only applicable if **regtype** \neq **Nag_HuberReg**.

NE_BETA1_ITER_EXCEEDED

The number of iterations required to calculate β_1 exceeds **max_iter**.

This is only applicable if **regtype** = **Nag_MallowsReg** and **sigma_est** = **Nag_SigmaRes**.

NE_THETA_ITER_EXCEEDED

The number of iterations required to calculate θ and σ exceeds **max_iter**. In this case, **info**[2] = **max_iter** on exit.

NE_LSQ_FAIL_CONV

The iterations to solve the weighted least-squares equations failed to converge.

NE_WT_LSQ_NOT_FULL_RANK

The weighted least-squares equations are not of full rank.

NE_REG_MAT_SINGULAR

Failure to invert matrix while calculating covariance.

If **regtype** = **Nag_HuberReg**, then $(X^T X)$ is almost singular.

If **regtype** \neq **Nag_HuberReg**, then S_1 is singular or almost singular. This may be due to too many diagonal elements of the matrix being zero, see Section 6.

NE_COV_MAT_FACTOR_ZERO

In calculating the correlation factor for the asymptotic variance-covariance matrix, the factor for covariance matrix = 0.

For this error, either the value of

$$\frac{1}{n} \sum_{i=1}^n \psi'(r_i/\hat{\sigma}) = 0,$$

or $\kappa = 0$,

$$\text{or } \sum_{i=1}^n \psi^2(r_i/\hat{\sigma}) = 0.$$

See Section 6. In this case **c** is returned as $(X^T X)^{-1}$.

(This is only applicable if **regtype** = **Nag_HuberReg**).

NE_VAR_THETA_LEQ_ZERO

The estimated variance for an element of $\theta \leq 0$. In this case the diagonal element of **c** will contain the negative variance and the above diagonal elements in the row and the column corresponding to the element will be returned as zero.

This error may be caused by rounding errors or too many of the diagonal elements of **p** being zero. See Section 6.

NE_ERR_DOF_LEQ_ZERO

n = $\langle \text{value} \rangle$, rank of **x** = $\langle \text{value} \rangle$. The degrees of freedom for error, **n** – (rank of **x**) must be > 0.0 .

NE_ESTIM_SIGMA_ZERO

The estimated value of σ was 0.0 during an iteration.

NE_NOT_APPEND_FILE

Cannot open file $\langle \text{string} \rangle$ for appending.

NE_NOT_CLOSE_FILE

Cannot close file $\langle \text{string} \rangle$.

NE_ALLOC_FAIL

Memory allocation failed.

6. Further Comments

In cases when **sigma_est** \neq **Nag_SigmaRes** it is important for the value of **sigma** to be of a reasonable magnitude. Too small a value may cause too many of the winsorised residuals, i.e., $\psi(r_i/\sigma)$ to be zero or a value of $\psi'(r_i/\sigma)$, used to estimate the asymptotic covariance matrix, to be zero. This can lead to errors with **fail.code** set to one of the following values:

NE_WT_LSQ_NOT_FULL_RANK,
 NE_REG_MAT_SINGULAR (if regtype \neq Nag_HuberReg),
 NE_COV_MAT_FACTOR_ZERO (if regtype = Nag_HuberReg)
 NE_VAR_THETA_LEQ_ZERO.

6.1. Accuracy

The precision of the estimates is determined by **tol**, see Section 4. As a more stable method is used to calculate the estimates of θ than is used to calculate the covariance matrix, it is possible for the least-squares equations to be of full rank but the $(X^T X)$ matrix to be too nearly singular to be inverted.

6.2. References

Hampel F R, Ronchetti E M, Rousseeuw P J and Stahel W A (1986) *Robust Statistics. The Approach Based on Influence Functions*. Wiley.
 Huber P J (1981) *Robust Statistics*. Wiley.
 Marazzi A (1987a) Weights for Bounded Influence Regression in ROBETH *Cah Rech Doc IUMSP, No. 3 ROB 3*. Institut Universitaire de Médecine Sociale et Préventive, Lausanne.
 Marazzi A (1987b) Subroutines for Robust and Bounded Influence Regression in ROBETH *Cah Rech Doc IUMSP, No. 3 ROB 2*. Institut Universitaire de Médecine Sociale et Préventive, Lausanne.

7. See Also

None.

8. Example

The number of observations and the number of x variables are read in followed by the data. The option parameters are then read in (in this case giving: Schweppe type regression with Hampel's ψ function and Huber's χ function and then using the 'replace expected by observed' option in calculating the covariances). Finally a set of values for the constants are read in.

After a call to nag_robust_m_regsn_estim, $\hat{\theta}$, its standard error and $\hat{\sigma}$ are printed. In addition the weight and residual for each observation is printed.

8.1. Program Text

```
/* nag_robust_m_regsn_estim(g02hac) Example Program.
 *
 * Copyright 1996 Numerical Algorithms Group.
 *
 * Mark 4, 1996.
 *
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <ctype.h>
#include <nagg02.h>

#ifdef NAG_PROTO
static void set_enum(char regtypec, Nag_RegType *regtype, char psifunc,
                    Nag_PsiFun *psifun, char sigestc, Nag_SigmaEst *sigma_est);
#else
static void set_enum();
#endif

#define NMAX 8
#define MMAX 3
#define TDC MMAX
#define TDX MMAX

main()
{
```

```

char regtypec, psifunc, sigestc, covmat_estc;
Nag_RegType regtype;
Nag_PsiFun psifun;
Nag_SigmaEst sigma_est;
Nag_CovMatrixEst covmat_est;
Integer i, j, m, n;
double sigma;
double c[MMAX][MMAX], theta[MMAX], x[NMAX][MMAX], y[NMAX];
double cpsi;
double hpsi[3];
double cucv;
double dchi;
double rs[NMAX];
double wt[NMAX];
double tol;
Integer max_iter;
Integer print_iter;
double info[4];
static NagError fail;

Vprintf("g02gac Example Program Results\n\n");
/* Skip heading in data file */
Vscanf("%*[^\\n]");
Vscanf("%ld %ld", &n, &m);
if ((n > 0 && n <= NMAX) && (m > 0 && m <= MMAX))
{
    /* Read in x and y */
    for (i=0; i<n; i++)
    {
        for (j=0; j<m; j++)
            Vscanf("%lf", &x[i][j]);
        Vscanf("%lf", &y[i]);
    }
    /* Read in control parameters */
    Vscanf(" %c %c %c", &regtypec, &psifunc, &sigestc);

    /* Check control parameters */
    set_enum(regtypec, &regtype, psifunc, &psifun, sigestc, &sigma_est);

    /* Read in appropriate weight function parameters. */
    if (regtype != Nag_HuberReg)
        Vscanf(" %c %lf", &covmat_estc, &cucv);

    if (covmat_estc == 'A')
        covmat_est = Nag_CovMatAve;
    else
        covmat_est = Nag_CovMatObs;

    if (psifun != Nag_Lsq)
    {
        if (psifun == Nag_HuberFun)
            Vscanf("%lf", &cpsi);
        else
            cpsi = 0.0;
        if (psifun == Nag_HampelFun)
            for (j=0; j<3; j++)
                Vscanf("%lf", &hpsi[j]);
        if (sigma_est == Nag_SigmaChi)
            Vscanf("%lf", &dchi);
    }
    /* Set values of remaining parameters */
    tol = 5e-5;
    max_iter = 50;
    /* Change print_iter to a positive value if monitoring information
    * is required
    */
    print_iter = 1;
    sigma = 1.0e0;
    for (i = 0; i < m; ++i)
        theta[i] = 0.0e0;
}

```

```

g02hac(regtype, psifun, sigma_est, covmat_est, n, m, (double *)x,
        (Integer)TDX, y, cpsi, hpsi, cucv, dchi, theta, &sigma,
        (double *)c, (Integer)TDC, rs, wt, tol, max_iter, print_iter,
        "", info, &fail);

if ((fail.code == NE_NOERROR) || (fail.code == NE_THETA_ITER_EXCEEDED) ||
    (fail.code == NE_LSQ_FAIL_CONV) || (fail.code == NE_MAT_SINGULAR) ||
    (fail.code == NE_WT_LSQ_NOT_FULL_RANK) ||
    (fail.code == NE_REG_MAT_SINGULAR) ||
    (fail.code == NE_COV_MAT_FACTOR_ZERO) ||
    (fail.code == NE_VAR_THETA_LEQ_ZERO) ||
    (fail.code == NE_VAR_THETA_LEQ_ZERO) ||
    (fail.code == NE_ERR_DOF_LEQ_ZERO) ||
    (fail.code == NE_ESTIM_SIGMA_ZERO))
{
    if (fail.code != NE_NOERROR)
    {
        Vprintf("%s\n", fail.message);
        Vprintf("    Some of the following results may be unreliable\n");
    }
    Vprintf("Sigma = %10.4f\n\n", sigma);
    Vprintf("    Theta      Standard errors\n\n");
    for (j = 0; j < m; ++j)
        Vprintf("%12.4f %13.4f\n", theta[j], c[j][j]);
    Vprintf("\n    Weights      Residuals\n\n");
    for (i = 0; i < n; ++i)
        Vprintf("%12.4f %13.4f\n", wt[i], rs[i]);
}
else
{
    Vprintf("%s\n", fail.message);
}
}
else
{
    Vfprintf(stderr, "One or both of m and n are out of range:\n
m = %-3ld while n = %-3ld\n", m, n);
    exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}

#ifdef NAG_PROTO
static void set_enum(char regtypec, Nag_RegType *regtype, char psifunc,
                    Nag_PsiFun *psifun, char sigestc, Nag_SigmaEst *sigma_est)
#else
static void set_enum(regtypec, regtype, psifunc, psifun, sigestc,
                    sigma_est)

char regtypec;
Nag_RegType *regtype;
char psifunc;
Nag_PsiFun *psifun;
char sigestc;
Nag_SigmaEst *sigma_est;
#endif
{
    if (toupper(regtypec) == 'H' || toupper(regtypec) == 'M' ||
        toupper(regtypec) == 'S')
    {
        switch (toupper(regtypec))
        {
            case ('H'):
                *regtype = Nag_HuberReg;
                break;
            case ('M'):
                *regtype = Nag_MallowsReg;
                break;
            case ('S'):
                *regtype = Nag_SchweppesReg;
                break;
        }
    }
}

```

```

        break;
    default:
        ;
    }
}
else
{
    Vfprintf(stderr, "The parameter regtype has an invalid value:\n
regtype = %c\n", regtypec);
    exit(EXIT_FAILURE);
}

if (toupper(psifunc) == 'L' || toupper(psifunc) == 'H' ||
    toupper(psifunc) == 'P' || toupper(psifunc) == 'A'
    || toupper(psifunc) == 'T')
{
    switch (toupper(psifunc))
    {
        case ('L'):
            *psifun = Nag_Lsq;
            break;
        case ('H'):
            *psifun = Nag_HuberFun;
            break;
        case ('P'):
            *psifun = Nag_HampelFun;
            break;
        case ('A'):
            *psifun = Nag_AndrewFun;
            break;
        case ('T'):
            *psifun = Nag_TukeyFun;
            break;
        default:
            ;
    }
}
else
{
    Vfprintf(stderr, "The parameter psifun has an invalid value:\n
psifun = %c\n", psifunc);
    exit(EXIT_FAILURE);
}

if (toupper(sigestc) == 'R' || toupper(sigestc) == 'C' ||
    toupper(sigestc) == 'X')
{
    switch (toupper(sigestc))
    {
        case ('R'):
            *sigma_est = Nag_SigmaRes;
            break;
        case ('C'):
            *sigma_est = Nag_SigmaConst;
            break;
        case ('X'):
            *sigma_est = Nag_SigmaChi;
            break;
        default:
            ;
    }
}
else
{
    Vfprintf(stderr, "The parameter sigma_est has an invalid value:\n
sigma_est = %c\n", sigestc);
    exit(EXIT_FAILURE);
}
return;
}

```

8.2. Program Data

```

g02hac Example Program Data
8 3
1. -1. -1. 2.1
1. -1. 1. 3.6
1. 1. -1. 4.5
1. 1. 1. 6.1
1. -2. 0. 1.3
1. 0. -2. 1.9
1. 2. 0. 6.7
1. 0. 2. 5.5
S P X
R 3.0
1.5 3.0 4.5
1.5

```

8.3. Program Results

```

g02gac Example Program Results

```

```

** Iteration monitoring for weights **

```

```

Iteration      1  max(abs(s(i,j))) =  1.93661e-01
      A
Row
  1  1.04e+00
  2  0.00e+00  8.05e-01
  3  0.00e+00  0.00e+00  8.05e-01
Iteration      2  max(abs(s(i,j))) =  9.25129e-02
      A
Row
  1  1.08e+00
  2  0.00e+00  8.80e-01
  3  0.00e+00  0.00e+00  8.80e-01
Iteration      3  max(abs(s(i,j))) =  3.56059e-02
      A
Row
  1  1.10e+00
  2  0.00e+00  9.11e-01
  3  0.00e+00  0.00e+00  9.11e-01
Iteration      4  max(abs(s(i,j))) =  1.29404e-02
      A
Row
  1  1.11e+00
  2  0.00e+00  9.23e-01
  3  0.00e+00  0.00e+00  9.23e-01
Iteration      5  max(abs(s(i,j))) =  4.81557e-03
      A
Row
  1  1.12e+00
  2  0.00e+00  9.27e-01
  3  0.00e+00  0.00e+00  9.27e-01
Iteration      6  max(abs(s(i,j))) =  1.81167e-03
      A
Row
  1  1.12e+00
  2  0.00e+00  9.29e-01
  3  0.00e+00  0.00e+00  9.29e-01
Iteration      7  max(abs(s(i,j))) =  6.81356e-04
      A
Row
  1  1.12e+00
  2  0.00e+00  9.29e-01
  3  0.00e+00  0.00e+00  9.29e-01
Iteration      8  max(abs(s(i,j))) =  2.56005e-04
      A
Row
  1  1.12e+00
  2  0.00e+00  9.30e-01
  3  0.00e+00  0.00e+00  9.30e-01

```

```

Iteration      9  max(abs(s(i,j))) =  9.61466e-05
      A
Row
  1  1.12e+00
  2  0.00e+00  9.30e-01
  3  0.00e+00  0.00e+00  9.30e-01
Iteration      10  max(abs(s(i,j))) =  3.61034e-05
      A
Row
  1  1.12e+00
  2  0.00e+00  9.30e-01
  3  0.00e+00  0.00e+00  9.30e-01
      ** Iteration monitoring for theta **
iteration      sigma      j      theta      rs
      1      1.63136e+00      1      3.93035e+00      -3.93035e+00
      1      2      1.24942e+00      -1.24942e+00
      1      3      9.19080e-01      -9.19080e-01
      2      4.48276e-01      1      3.96250e+00      -3.21549e-02
      2      2      1.30833e+00      -5.89084e-02
      2      3      8.58333e-01      6.07465e-02
      3      3.70260e-01      1      3.97530e+00      -1.28013e-02
      3      2      1.30833e+00      -2.22045e-16
      3      3      8.41265e-01      1.70684e-02
      4      3.23188e-01      1      3.98577e+00      -1.04731e-02
      4      2      1.30833e+00      -2.22045e-16
      4      3      8.27301e-01      1.39642e-02
      5      2.91377e-01      1      3.99829e+00      -1.25129e-02
      5      2      1.30833e+00      2.22045e-16
      5      3      8.10617e-01      1.66839e-02
      6      2.62746e-01      1      4.02376e+00      -2.54714e-02
      6      2      1.30833e+00      6.66134e-16
      6      3      7.76655e-01      3.39618e-02
      7      2.26353e-01      1      4.04231e+00      -1.85490e-02
      7      2      1.30833e+00      -8.88178e-16
      7      3      7.51923e-01      2.47320e-02
      8      2.09006e-01      1      4.04231e+00      0.00000e+00
      8      2      1.30833e+00      0.00000e+00
      8      3      7.51923e-01      0.00000e+00
      9      2.04291e-01      1      4.04231e+00      0.00000e+00
      9      2      1.30833e+00      0.00000e+00
      9      3      7.51923e-01      0.00000e+00
     10      2.03057e-01      1      4.04231e+00      0.00000e+00
     10      2      1.30833e+00      0.00000e+00
     10      3      7.51923e-01      0.00000e+00
     11      2.02737e-01      1      4.04231e+00      0.00000e+00
     11      2      1.30833e+00      0.00000e+00
     11      3      7.51923e-01      0.00000e+00
     12      2.02654e-01      1      4.04231e+00      0.00000e+00
     12      2      1.30833e+00      0.00000e+00
     12      3      7.51923e-01      0.00000e+00
     13      2.02633e-01      1      4.04231e+00      0.00000e+00
     13      2      1.30833e+00      0.00000e+00
     13      3      7.51923e-01      0.00000e+00
     14      2.02627e-01      1      4.04231e+00      0.00000e+00
     14      2      1.30833e+00      0.00000e+00
     14      3      7.51923e-01      0.00000e+00
Sigma =      0.2026

```

Theta Standard errors

```

4.0423      0.0384
1.3083      0.0272
0.7519      0.0311

```

Weights Residuals

```

0.5783      0.1179
0.5783      0.1141
0.5783     -0.0987

```

0.5783	-0.0026
0.4603	-0.1256
0.4603	-0.6385
0.4603	0.0410
0.4603	-0.0462
