NAG Library Function Document

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

```c
#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, 
   const 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, const 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 \( k \),
\( \theta \) is a vector of length \( m \) of unknown arguments,
and \( \epsilon \) is a vector of length \( n \) of unknown errors with \( \text{var}(\epsilon_i) = \sigma^2 \):

nag_robust_m_regsn_estim (g02hac) calculates the M-estimates given by the solution, \( \hat{\theta} \), to the equation

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

where \( r_i \) is the \( i \)th residual, i.e., the \( i \)th element of \( r = y - X\hat{\theta} \),
\( \psi \) is a suitable weight function,
\( w_i \) are suitable weights,
and \( \sigma \) may be estimated at each iteration by the median absolute deviation of the residuals:
\[ \hat{\sigma} = \text{med}[|r_i|]/\beta_1 \]

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 \( \chi \), where \( \beta_1 \) and \( \beta_2 \) are constants, chosen so that the estimator of \( \sigma \) is asymptotically unbiased if the errors, \( \epsilon_i \), have a Normal distribution. Alternatively \( \sigma \) 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
This may be obtained by use of the transformations
\[ w_i^* \leftarrow \sqrt{\omega_i}, \]
\[ y_i^* \leftarrow y_i \sqrt{\omega_i}, \]
\[ x_{ij}^* \leftarrow x_{ij} \sqrt{\omega_i}, \quad j = 1, 2, \ldots, m. \]

(see Marazzi (1987a)).

For Huber and Schweppe type regressions, \( \beta_1 \) is the 75th percentile of the standard Normal distribution. For Mallows type regression \( \beta_1 \) is the solution to
\[ \frac{1}{n} \sum_{i=1}^{n} \Phi(\beta_1 / \sqrt{\omega_i}) = .75 \]
where \( \Phi \) is the standard Normal cumulative distribution function.

\( \beta_2 \) is given by:
\[
\begin{align*}
\beta_2 &= \int_{-\infty}^{\infty} \chi(z) \phi(z) dz, \quad \text{in Huber case;} \\
\beta_2 &= \frac{1}{n} \sum_{i=1}^{n} w_i \int_{-\infty}^{\infty} \chi(z) \phi(z) dz, \quad \text{in Mallows case;} \\
\beta_2 &= \frac{1}{n} \sum_{i=1}^{n} w_i^2 \int_{-\infty}^{\infty} \chi(z/w_i) \phi(z) dz, \quad \text{in Schweppe case;}
\end{align*}
\]
where \( \phi \) is the standard Normal density, i.e.,
\[
\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}t^2\right).
\]

The calculation of the estimates of \( \theta \) can be formulated as an iterative weighted least squares problem with a diagonal weight matrix \( G \) given by
\[
G_{ii} = \begin{cases} \\
\psi(r_i/(\sigma w_i)) & r_i \neq 0 \\
\psi'(0) & r_i = 0
\end{cases}
\]
where \( \psi'(t) \) is the derivative of \( \psi \) at the point \( t \).

The value of \( \theta \) 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
\[
\begin{align*}
\bar{y}_i &= y_i \sqrt{G_{ii}} \\
\bar{x}_{ij} &= x_{ij} \sqrt{G_{ii}}, \quad j = 1, 2, \ldots, m
\end{align*}
\]
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 \( \psi \) and \( \chi \) in nag_robust_m_regsn_estim (g02hac).

(a) **Unit Weights**

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

this gives least squares regression.

(b) **Huber’s Function**

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

\[ g02hac.2 \quad \text{Mark 25} \]
(c) **Hampel’s Piecewise Linear Function**

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

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

(d) **Andrew’s Sine Wave Function**

\[ \psi(t) = \begin{cases} 
  \sin t, & -\pi \leq t \leq \pi \\
  0, & |t| > \pi 
\end{cases} \]

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

(e) **Tukey’s Bi-weight**

\[ \psi(t) = \begin{cases} 
  t(1-t^2)^2, & |t| \leq 1 \\
  0, & |t| > 1 
\end{cases} \]

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

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_regsn_estim (g02hac)` finds \( A \) using the iterative procedure

\[ A_k = (S_k + I)A_{k-1} \]

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

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

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 (g02hac)`:
(i) Krasker–Welsch weights

\[ u(t) = g_1 \left( \frac{t}{\Phi(t)} \right) \]

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

\[ u(t) = \begin{cases} \frac{c}{t^2} & |t| > c \\ 1 & |t| \leq c \end{cases} \]

\[ f(t) = \sqrt{u(t)}. \]

These are for use with Mallows type regression.

Finally the asymptotic variance-covariance matrix, \( C \), of the estimates \( \theta \) is calculated. For Huber type regression

\[ C = f_H (X^TX)^{-1} \hat{\sigma}^2 \]

where

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

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


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 \( S_1 = \frac{1}{n} X^T DX \) and \( S_2 = \frac{1}{n} X^T PX \).

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

Two approximations are available in nag_robust_m_regrn_estim (g02hac):
1. Average over the $r_i$

$$D_i = \left( \frac{1}{n} \sum_{j=1}^{n} \psi' \left( \frac{w_j}{w_i} \right) \right) w_i$$

Mallows

$$D_i = \left( \frac{1}{n} \sum_{j=1}^{n} \psi' \left( \frac{w_j}{w_i} \right) \right) w_i$$

2. Replace expected value by observed

$$D_i = \psi' \left( \frac{r_i}{w_i} \right) w_i$$

Mallows

$$D_i = \psi' \left( \frac{r_i}{w_i} \right) w_i$$

$$P_i = \psi^2 \left( \frac{r_i}{w_i} \right) w_i^2$$

$$P_i = \psi^2 \left( \frac{r_i}{w_i} \right) w_i^2$$

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

Note: There is no explicit provision in the function 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 (g02hac) is based on routines in ROBETH, see Marazzi (1987a).

4 References


5 Arguments

1: regtype – Nag_RegType

Input

On entry: 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_SchwepppeReg

Schwepppe type regression with Krasker–Welsch weights.

Constraint: regtype = Nag_HuberReg, Nag_MallowsReg or Nag_SchwepppeReg.

2: psifun – Nag_PsiFun

Input

On entry: specifies which $\psi$ 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.
\texttt{psifun} = \texttt{Nag\_TukeyFun}

Tukey’s bi-weight.

\textit{Constraint:} \texttt{psifun} = \texttt{Nag\_Lsq}, \texttt{Nag\_HuberFun}, \texttt{Nag\_HampelFun}, \texttt{Nag\_AndrewFun} or \texttt{Nag\_TukeyFun}.

3: \texttt{sigma\_est} – \texttt{Nag\_SigmaEst} \hspace{1cm} \textit{Input}

\textit{On entry:} specifies how \( \sigma \) is to be estimated.

\texttt{sigma\_est} = \texttt{Nag\_SigmaRes}

\( \sigma \) is estimated by median absolute deviation of residuals.

\texttt{sigma\_est} = \texttt{Nag\_SigmaConst}

\( \sigma \) is held constant at its initial value.

\texttt{sigma\_est} = \texttt{Nag\_SigmaChi}

\( \sigma \) is estimated using the \( \chi \) function.

\textit{Constraint:} \texttt{sigma\_est} = \texttt{Nag\_SigmaRes}, \texttt{Nag\_SigmaConst} or \texttt{Nag\_SigmaChi}.

4: \texttt{covmat\_est} – \texttt{Nag\_CovMatrixEst} \hspace{1cm} \textit{Input}

\textit{On entry:} if \texttt{regtype} \neq \texttt{Nag\_HuberReg}, \texttt{covmat\_est} specifies the approximations used in estimating the covariance matrix of \( \hat{\theta} \). \texttt{covmat\_est} = \texttt{Nag\_CovMatAve}, averaging over residuals. \texttt{covmat\_est} = \texttt{Nag\_CovMatObs}, replacing expected by observed.

If \texttt{regtype} = \texttt{Nag\_HuberReg} then \texttt{covmat\_est} is not referenced.

\textit{Constraint:} \texttt{covmat\_est} = \texttt{Nag\_CovMatAve} or \texttt{Nag\_CovMatObs}.

5: \texttt{n} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} the number of observations, \( n \).

\textit{Constraint:} \( n > 1 \).

6: \texttt{m} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} the number \( m \), of independent variables.

\textit{Constraint:} \( 1 \leq m < n \).

7: \texttt{x[n \times tdx]} – double \hspace{1cm} \textit{Input/Output}

\textit{Note:} the \((i, j)\)th element of the matrix \( X \) is stored in \texttt{x[(i - 1) \times tdx + j - 1]}.

\textit{On entry:} the values of the \( X \) matrix, i.e., the independent variables. \texttt{x[i - 1][j - 1]} must contain the \( ij \)th element of \( X \), for \( i = 1, 2, \ldots, n \) and \( j = 1, 2, \ldots, m \).

\textit{On exit:} if \texttt{regtype} = \texttt{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.

8: \texttt{tdx} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} the stride separating matrix column elements in the array \( x \).

\textit{Constraint:} \( tdx \geq m \).

9: \texttt{y[n]} – double \hspace{1cm} \textit{Input/Output}

\textit{On entry:} the data values of the dependent variable. \texttt{y[i - 1]} must contain the value of \( y \) for the \( i \)th observation, for \( i = 1, 2, \ldots, n \).

\textit{On exit:} if \texttt{regtype} = \texttt{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.

10: $cpsi$ – double

*Input*

*On entry:* if $psifun = \text{Nag\_HuberFun}$, $cpsi$ must specify the argument, $c$, of Huber’s $\psi$ function. Otherwise $cpsi$ is not referenced.

*Constraint:* if $psifun = \text{Nag\_HuberFun}$ then $cpsi > 0.0$.


*Input*

*On entry:* if $psifun = \text{Nag\_HampelFun}$ then $hpsi[0]$, $hpsi[1]$ and $hpsi[2]$ must specify the arguments $h_1$, $h_2$, and $h_3$, of Hampel’s piecewise linear $\psi$ function. Otherwise the elements of $hpsi$ are not referenced.

*Constraint:* if $psifun = \text{Nag\_HampelFun}$, $0 \leq hpsi[0] \leq hpsi[1] \leq hpsi[2]$ and $hpsi[2] > 0.0$.

12: $cucv$ – double

*Input*

*On entry:* if $regtype = \text{Nag\_MallowsReg}$ then $cucv$ must specify the value of the constant, $c$, of the function $u$ for Maronna’s proposed weights.

If $regtype = \text{Nag\_SchweppeReg}$ then $cucv$ must specify the value of the function $u$ for the Krasker–Welsch weights.

If $regtype = \text{Nag\_HuberReg}$ then $cucv$ is not referenced.

*Constraints:*

- if $regtype = \text{Nag\_MallowsReg}$, $cucv \geq m$;
- if $regtype = \text{Nag\_SchweppeReg}$, $cucv \geq \sqrt{m}$.

13: $dchi$ – double

*Input*

*On entry:* the constant, $d$, of the $\chi$ function.

$dchi$ is referenced only if $psifun \neq \text{Nag\_Lsq}$ and $sigma\_est = \text{Nag\_SigmaChi}$.

*Constraint:* if $psifun \neq \text{Nag\_Lsq}$ and $sigma\_est = \text{Nag\_SigmaChi}$, $dchi > 0.0$.

14: $theta[m]$ – double

*Input/Output*

*On entry:* starting values of the argument vector $\theta$. These may be obtained from least squares regression.

Alternatively if $sigma\_est = \text{Nag\_SigmaRes}$ and $sigma = 1$ or if $sigma\_est = \text{Nag\_SigmaChi}$ and $sigma$ approximately equals the standard deviation of the dependent variable, $y$, then $theta[i - 1] = 0.0$, for $i = 1, 2, \ldots, m$ may provide reasonable starting values.

*On exit:* $theta[i - 1]$ contains the M-estimate of $\theta_i$, for $i = 1, 2, \ldots, m$.

15: $sigma$ – double *

*Input/Output*

*On entry:* a starting value for the estimation of $\sigma$.

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

*On exit:* $sigma$ contains the final estimate of $\sigma$, unless $sigma\_est = \text{Nag\_SigmaConst}$.

*Constraint:* $sigma > 0.0$.

16: $c[m \times tdc]$ – double

*Output*

*On exit:* the diagonal elements of $c$ contain the estimated asymptotic standard errors of the estimates of $\theta_i$, i.e., $c(i - 1) \times tdc + i - 1$ contains the estimated asymptotic standard error of the estimate contained in $theta[i - 1]$, for $i = 1, 2, \ldots, m$. 
The elements above the diagonal contain the estimated asymptotic correlation between the estimates of \( \theta \), i.e., \([c(i-1) \times tdc + j - 1], 1 \leq i < j \leq m \) contains the asymptotic correlation between the estimates contained in \( \text{theta}[i-1] \) and \( \text{theta}[j-1] \).

The elements below the diagonal contain the estimated asymptotic covariance between the estimates of \( \theta \), i.e., \([c(i-1) \times tdc + j - 1], 1 \leq j < i \leq m \) contains the estimated asymptotic covariance between the estimates contained in \( \text{theta}[i-1] \) and \( \text{theta}[j-1] \).

17: \textbf{tdc} – Integer
\textit{Input}

\textit{On entry:} the stride separating matrix column elements in the array \( c \).

\textit{Constraint:} \( tdc \geq m \).

18: \textbf{rs[n]} – double
\textit{Output}

\textit{On exit:} contains the residuals from the model evaluated at final value of \( \text{theta} \), i.e., \( rs[i-1] \), for \( i = 1, 2, \ldots, n \), contains the vector \( (y - X\hat{\theta}) \).

19: \textbf{wt[n]} – double
\textit{Output}

\textit{On exit:} contains the vector of weights. \( wt[i-1] \) contains the weight for the \( i \)th observation, for \( i = 1, 2, \ldots, n \).

20: \textbf{tol} – double
\textit{Input}

\textit{On entry:} the relative precision for the calculation of \( A \) (if \textbf{regtype} \( \neq \) Nag_HuberReg), the estimates of \( \theta \) and the estimate of \( \sigma \) (if \textbf{sigma_est} \( \neq \) Nag_SigmaConst). Convergence is assumed when the relative change in all elements being considered is less than \( \text{tol} \).

If \textbf{regtype} = Nag_MallowsReg and \textbf{sigma_est} = Nag_SigmaRes, \( \text{tol} \) is also used to determine the precision of \( \beta_1 \).

It is advisable for \( \text{tol} \) to be greater than 100\( \times \) machine precision.

\textit{Constraint:} \( \text{tol} > 0.0 \).

21: \textbf{max_iter} – Integer
\textit{Input}

\textit{On entry:} the maximum number of iterations that should be used in the calculation of \( A \) (if \textbf{regtype} \( \neq \) Nag_HuberReg), and of the estimates of \( \theta \) and \( \sigma \), and of \( \beta_1 \) (if \textbf{regtype} = Nag_MallowsReg and \textbf{sigma_est} = Nag_SigmaRes)

\textit{Suggested value:} A value of \( \text{max_iter} = 50 \) should be adequate for most uses.

\textit{Constraint:} \( \text{max_iter} > 0.0 \).

22: \textbf{print_iter} – Integer
\textit{Input}

\textit{On entry:} the amount of information that is printed on each iteration.

\textbf{print_iter} = 0
No information is printed.

\textbf{print_iter} \neq 0
The current estimate of \( \theta \), the change in \( \theta \) during the current iteration and the current value of \( \sigma \) are printed on the first and every abs(\textbf{print_iter}) iterations.

Also, if \textbf{regtype} \( \neq \) Nag_HuberReg and \textbf{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).
23: **outfile** – const char *

*Input*

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

24: **info[4]** – double

*Output*

On exit: elements of info contain the following values:

- **info[0]** = $\beta_1$ if **sigma_est** = Nag_SigmaRes,
- or **info[0]** = $\beta_2$ 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 $\theta$ and $\sigma$,
- **info[3]** = $k$, the rank of the weighted least squares equations.

25: **fail** – NagError *

*Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

**NE_2_INT_ARG_GE**

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

**NE_2_INT_ARG_LT**

On entry, $tdc = \langle value \rangle$ while $m = \langle value \rangle$. These arguments must satisfy $tdc \geq m$.

On entry, $tdx = \langle value \rangle$ while $m = \langle value \rangle$. These arguments must satisfy $tdx \geq m$.

**NE_ALLOC_FAIL**

Dynamic memory allocation failed.

**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 $0.0 \leq hpsi[0] \leq hpsi[1] \leq hpsi[2]$ and $hpsi[2] > 0.0$.

**NE_BAD_PARAM**

On entry, argument **covmat_est** had an illegal value.

On entry, argument **psifun** had an illegal value.

On entry, argument **regtype** had an illegal value.

On entry, argument **sigma_est** had an illegal value.

**NE_BETA1_ITER_EXCEEDED**

The number of iterations required to calculate $\beta_1$ exceeds max_iter. This is only applicable if **regtype** = Nag_MallowsReg and **sigma_est** = Nag_SigmaRes.

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

See Section 9. In this case \( c \) is returned as \((X^TX)^{-1}\).
(This is only applicable if \( \text{regtype} = \text{Nag_HuberReg} \)).

**NE_ERR_DOF_LEQ_ZERO**

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

**NE_ESTIM_SIGMA_ZERO**

The estimated value of \( \sigma \) was 0.0 during an iteration.

**NE_INT_ARG_LE**

On entry, \( \text{max_iter} \) must not be less than or equal to 0: \( \text{max_iter} = \langle \text{value} \rangle \).

**NE_INT_ARG_LT**

On entry, \( m = \langle \text{value} \rangle \).
Constraint: \( m \geq 1 \).

On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 2 \).

**NE_INTERNAL_ERROR**

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

**NE_INVALID_DCHI_FUN**

On entry, \( \text{psifun} \neq \text{Nag_Lsq}, \text{sigma_est} = \text{Nag_SigmaChi} \) and \( \text{dchi} = \langle \text{value} \rangle \). For these values of \( \text{psifun} \) and \( \text{sigma_est} \), \( \text{dchi} \) must be > 0.0.

**NE_INVALID_HUBER_FUN**

On entry, \( \text{psifun} = \text{Nag_HuberFun} \) and \( \text{cpsi} = \langle \text{value} \rangle \). For this value of \( \text{psifun} \), \( \text{cpsi} \) must be > 0.0.

**NE_INVALID_MALLOWS_REG_C**

On entry, \( \text{regtype} = \text{Nag_MallowsReg}, \text{cucv} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \). For this value of \( \text{regtype} \), \( \text{cucv} \) must be \( \geq m \).

**NE_INVALID_SCHWEPPE_REG_C**

On entry, \( \text{regtype} = \text{Nag_SchweppesReg}, \text{cucv} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \). For this value of \( \text{regtype} \), \( \text{cucv} \) must be \( \geq \sqrt{m} \).

**NE_LSQ_FAIL_CONV**

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

**NE_NOT_APPEND_FILE**

Cannot open file \( \langle \text{string} \rangle \) for appending.
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_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 9.

NE_THETA_ITER_EXCEEDED
The number of iterations required to calculate \(\theta\) and \(\sigma\) exceeds max_iter. In this case, info[2] = max_iter on exit.

NE_VAR_THETA_LEQ_ZERO
The estimated variance for an element of \(\theta \leq 0\). In this case the diagonal element of \(\epsilon\) 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 9.

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_WT LSQ NOT FULL_RANK
The weighted least squares equations are not of full rank.

7 Accuracy
The precision of the estimates is determined by tol, see Section 5. As a more stable method is used to calculate the estimates of \(\theta\) 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.

8 Parallelism and Performance
Not applicable.

9 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 winsorized 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 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.
10  Example

The number of observations and the number of $x$ variables are read in followed by the data. The option arguments are then read in (in this case giving: Schweppe type regression with Hampel’s $\psi$ function and Huber’s $\chi$ 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 (g02hac), $\hat{\theta}$, its standard error and $\hat{\sigma}$ are printed. In addition the weight and residual for each observation is printed.

10.1  Program Text

/* nag_robust_m_regsn_estim (g02hac) Example Program. */
* Copyright 2014 Numerical Algorithms Group. *
* Mark 4, 1996. *
* Mark 8 revised, 2004. *
*/
#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <ctype.h>
#include <nagg02.h>
define C(I, J) c[(I) *tdc + J]
define X(I, J) x[(I) *tdx + J]

int main(void)
{
    Integer exit_status = 0, i, j, m, max_iter, n, print_iter, tdc, tdx;
double *c = 0, cpsi, cucv, dchi, *hpsi = 0, *info = 0, *rs = 0,
sigma, *theta = 0;
char nag_enum_arg[40];
Nag_CovMatrixEst covmat_est;
Nag_PsiFun psifun;
Nag_RegType regtype;
Nag_SigmaEst sigma_est;
NagError fail;

INIT_FAIL(fail);

printf("nag_robust_m_regsn_estim (g02hac) Example Program Results\n\n");
/* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
#ifdef _WIN32
    scanf_s("%NAG_IFMT%NAG_IFMT", &n, &m);
#else
    scanf("%NAG_IFMT%NAG_IFMT", &n, &m);
#endif
if (n > 1 && (m >= 1 && m <= n))
{
    if (!((c = NAG_ALLOC(m*m, double)) ||
        !(theta = NAG_ALLOC(m, double)) ||
        !(x = NAG_ALLOC(n*m, double)) ||
        !(y = NAG_ALLOC(n, double)) ||
        !(rs = NAG_ALLOC(n, double)) ||
        !(wt = NAG_ALLOC(n, double)) ||
        !(info = NAG_ALLOC(4, double)) ||
        !(hpsi = NAG_ALLOC(3, double)))
    {

printf("Allocation failure\n");
exit_status = -1;
goto END;
}
tdc = m;
tdx = m;
else
{
  printf("Invalid n or m.\n");
  exit_status = 1;
  return exit_status;
}
/* Read in x and y */
for (i = 0; i < n; i++)
{
  for (j = 0; j < m; j++)
  {
    #ifdef _WIN32
      scanf_s("%lf", &X(i, j));
    #else
      scanf("%lf", &X(i, j));
    #endif
    #ifdef _WIN32
      scanf_s("%lf", &y[i]);
    #else
      scanf("%lf", &y[i]);
    #endif
    scanf("%lf", &y[i]);
  }
  /* Read in control parameters */
  #ifdef _WIN32
    scanf_s(" %39s", nag_enum_arg, _countof(nag_enum_arg));
  #else
    scanf(" %39s", nag_enum_arg);
  #endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  regtype = (Nag_RegType) nag_enum_name_to_value(nag_enum_arg);
  #ifdef _WIN32
    scanf_s(" %39s", nag_enum_arg, _countof(nag_enum_arg));
  #else
    scanf(" %39s", nag_enum_arg);
  #endif
  psifun = (Nag_PsiFun) nag_enum_name_to_value(nag_enum_arg);
  #ifdef _WIN32
    scanf_s(" %39s", nag_enum_arg, _countof(nag_enum_arg));
  #else
    scanf(" %39s", nag_enum_arg);
  #endif
  sigma_est = (Nag_SigmaEst) nag_enum_name_to_value(nag_enum_arg);
  if (regtype != Nag_HuberReg)
  {
    #ifdef _WIN32
      scanf_s(" %39s %lf", nag_enum_arg, _countof(nag_enum_arg), &cucv);
    #else
      scanf(" %39s %lf", nag_enum_arg, &cucv);
    #endif
    covmat_est = (Nag_CovMatrixEst) nag_enum_name_to_value(nag_enum_arg);
  }
  if (psifun != Nag_Lsq)
  {
    if (psifun == Nag_HuberFun)
    {
      #ifdef _WIN32
        scanf_s("%lf", &cpsi);
      #else
        scanf("%lf", &cpsi);
      #endif
    }
  }
cpsi = 0.0;
if (psifun == Nag_HampelFun)
    for (j = 0; j < 3; j++)
        ifdef _WIN32
            scanf_s("%lf", &hpsi[j]);
        #else
            scanf("%lf", &hpsi[j]);
        #endif
    if (sigma_est == Nag_SigmaChi)
        ifdef _WIN32
            scanf_s("%lf", &dchi);
        #else
            scanf("%lf", &dchi);
        #endif
/* Set values of remaining parameters */
tol = 5e-5;
max_iter = 50;
/* Change print_iter to a positive value if monitoring information */
    print_iter = 1;
sigma = 1.0e0;
for (i = 0; i < m; ++i)
    theta[i] = 0.0e0;
/* nag_robust_m_regsn_estim (g02hac). */
    robust regression, standard M-estimates */
    fflush(stdout);
    nag_robust_m_regsn_estim(regtype, psifun, sigma_est, covmat_est, n, m, x,
tdx, y, cpsi, hpsi, cucv, dchi, theta, &sigma,
c, tdc, rs, wt, tol, max_iter, print_iter,
0, 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_REG_MAT_SINGULAR) ||
    (fail.code == NE_COV_MAT_FACTOR_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)
        printf("Error from nag_robust_m_regsn_estim (g02hac).\n%sn", fail.message);
    printf(" Some of the following results may be unreliable\n");
    printf("Sigma = %10.4f\n\n", sigma);
    printf(" Theta Standard errors\n";
    for (j = 0; j < m; ++j)
        printf("%12.4f %13.4f\n", theta[j], C(j, j));
    printf("\n Weights Residuals\n";
    for (i = 0; i < n; ++i)
        printf("%12.4f %13.4f\n", wt[i], rs[i]);
    else
        printf("Error from nag_robust_m_regsn_estim (g02hac).\n%sn", fail.message);
        exit_status = 1;
goto END;
}
END:
NAG_FREE(c);
NAG_FREE(theta);
NAG_FREE(x);
NAG_FREE(y);
NAG_FREE(rs);
NAG_FREE(wt);
NAG_FREE(info);
NAG_FREE(hpsi);

return exit_status;
}

10.2 Program Data

nag_robust_m_regsn_estim (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
Nag_SchweppeReg Nag_HampelFun Nag_SigmaChi Nag_CovMatObs 3.0
1.5 3.0 4.5
1.5

10.3 Program Results

nag_robust_m_regsn_estim (g02hac) Example Program Results

** Iteration monitoring for weights **

Iteration 1 max(abs(s(i,j))) = 1.93661e-01

Row
1 1.04e+00
2 8.73e-18 8.05e-01
3 8.73e-18 5.26e-20 8.05e-01

Iteration 2 max(abs(s(i,j))) = 9.25129e-02

Row
1 1.08e+00
2 -7.92e-18 8.80e-01
3 -6.96e-18 1.97e-18 8.80e-01

Iteration 3 max(abs(s(i,j))) = 3.56059e-02

Row
1 1.10e+00
2 2.34e-18 9.11e-01
3 2.46e-18 -4.84e-19 9.11e-01

Iteration 4 max(abs(s(i,j))) = 1.29404e-02

Row
1 1.11e+00
2 2.56e-18 9.23e-01
3 2.94e-18 7.72e-18 9.23e-01

Iteration 5 max(abs(s(i,j))) = 4.81557e-03

Row
1 1.12e+00
2 2.37e-18 9.27e-01
3 3.04e-18 -9.20e-20 9.27e-01

Iteration 6 max(abs(s(i,j))) = 1.81167e-03

Row
1 1.12e+00
2 -1.54e-17 9.29e-01
3 -1.44e-17 -8.96e-18 9.29e-01

Iteration 7 max(abs(s(i,j))) = 6.81356e-04
<table>
<thead>
<tr>
<th>Iteration</th>
<th>max(abs(s(i,j)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2.56005e-04</td>
</tr>
<tr>
<td>9</td>
<td>9.61466e-05</td>
</tr>
<tr>
<td>10</td>
<td>3.61034e-05</td>
</tr>
</tbody>
</table>

** Iteration monitoring for theta **

<table>
<thead>
<tr>
<th>iteration</th>
<th>sigma</th>
<th>j</th>
<th>theta</th>
<th>rs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.63136e+00</td>
<td>1</td>
<td>3.93035e+00</td>
<td>-3.93035e+00</td>
</tr>
<tr>
<td></td>
<td>2.48276e-01</td>
<td>1</td>
<td>3.96250e+00</td>
<td>-3.21549e-02</td>
</tr>
<tr>
<td></td>
<td>3.70260e-01</td>
<td>1</td>
<td>3.97530e+00</td>
<td>-1.28013e-02</td>
</tr>
<tr>
<td></td>
<td>3.23188e-01</td>
<td>1</td>
<td>3.98577e+00</td>
<td>-1.04731e-02</td>
</tr>
<tr>
<td></td>
<td>2.91377e-01</td>
<td>1</td>
<td>3.99829e+00</td>
<td>-2.22045e-16</td>
</tr>
<tr>
<td></td>
<td>2.62746e-01</td>
<td>1</td>
<td>4.02376e+00</td>
<td>-4.44089e-16</td>
</tr>
<tr>
<td></td>
<td>2.26353e-01</td>
<td>1</td>
<td>4.04231e+00</td>
<td>-1.85490e-02</td>
</tr>
<tr>
<td></td>
<td>2.09006e-01</td>
<td>1</td>
<td>4.04231e+00</td>
<td>0.00000e+00</td>
</tr>
<tr>
<td></td>
<td>2.04291e-01</td>
<td>1</td>
<td>4.04231e+00</td>
<td>0.00000e+00</td>
</tr>
<tr>
<td></td>
<td>2.03057e-01</td>
<td>1</td>
<td>4.04231e+00</td>
<td>0.00000e+00</td>
</tr>
<tr>
<td></td>
<td>2.02737e-01</td>
<td>1</td>
<td>4.04231e+00</td>
<td>0.00000e+00</td>
</tr>
<tr>
<td></td>
<td>2.02654e-01</td>
<td>1</td>
<td>4.04231e+00</td>
<td>0.00000e+00</td>
</tr>
<tr>
<td></td>
<td>2.02633e-01</td>
<td>1</td>
<td>4.04231e+00</td>
<td>0.00000e+00</td>
</tr>
<tr>
<td></td>
<td>2.02627e-01</td>
<td>1</td>
<td>4.04231e+00</td>
<td>0.00000e+00</td>
</tr>
</tbody>
</table>

Sigma = 0.2026

Theta Standard errors
<table>
<thead>
<tr>
<th>Weights</th>
<th>Residuals</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0423</td>
<td>0.0384</td>
</tr>
<tr>
<td>1.3083</td>
<td>0.0272</td>
</tr>
<tr>
<td>0.7519</td>
<td>0.0311</td>
</tr>
<tr>
<td>0.5783</td>
<td>0.1179</td>
</tr>
<tr>
<td>0.5783</td>
<td>0.1141</td>
</tr>
<tr>
<td>0.5783</td>
<td>-0.0987</td>
</tr>
<tr>
<td>0.5783</td>
<td>-0.0026</td>
</tr>
<tr>
<td>0.4603</td>
<td>-0.1256</td>
</tr>
<tr>
<td>0.4603</td>
<td>-0.6385</td>
</tr>
<tr>
<td>0.4603</td>
<td>0.0410</td>
</tr>
<tr>
<td>0.4603</td>
<td>-0.0462</td>
</tr>
</tbody>
</table>