nag_robust_m_estim_1var (g07dbc)

1. Purpose

nag_robust_m_estim_1var (g07dbc) computes an $M$-estimate of location with (optional) simultaneous estimation of the scale using Huber’s algorithm.

2. Specification

```c
#include <nag.h>
#include <nagg07.h>

void nag_robust_m_estim_1var(Nag_SigmaSimulEst sigma_est, Integer n,
  double x[], Nag_PsiFun psifun, double c, double h1,
  double h2, double h3, double dchi, double *theta,
  double *sigma, Integer maxit, double tol, double rs[],
  Integer *nit, double sorted_x[], NagError *fail)
```

3. Description

The data consists of a sample of size $n$, denoted by $x_1, x_2, \ldots, x_n$, drawn from a random variable $X$.

The $x_i$ are assumed to be independent with an unknown distribution function of the form

$$F((x_i - \theta)/\sigma)$$

where $\theta$ is a location parameter, and $\sigma$ is a scale parameter. $M$-estimators of $\theta$ and $\sigma$ are given by the solution to the following system of equations:

$$\sum_{i=1}^{n} \psi\left(\frac{x_i - \hat{\theta}}{\hat{\sigma}}\right) = 0 \quad (1)$$

$$\sum_{i=1}^{n} \chi\left(\frac{x_i - \hat{\theta}}{\hat{\sigma}}\right) = (n - 1)\beta \quad (2)$$

where $\psi$ and $\chi$ are given functions, and $\beta$ is a constant, such that $\hat{\sigma}$ is an unbiased estimator when $x_i$, for $i = 1, 2, \ldots, n$ has a normal distribution. Optionally, the second equation can be omitted and the first equation is solved for $\hat{\theta}$ using an assigned value of $\sigma = \sigma_c$.

The values of $\psi\left(\frac{x_i - \hat{\theta}}{\sigma}\right)\hat{\sigma}$ are known as the Winsorized residuals.

The following functions are available for $\psi$ and $\chi$ in nag_robust_m_estim_1var:

(a) Null Weights

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

Use of these null functions leads to the mean and standard deviation of the data.

(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**

\[
\psi_{h_1, h_2, h_3}(t) = -\psi_{h_1, h_2, h_3}(-t)
\]

\[
= t \quad 0 \leq t \leq h_1 \\
= h_1 \quad h_1 \leq t \leq h_2 \\
= h_1(h_3 - t)/(h_3 - h_2) \quad h_2 \leq t \leq h_3 \\
= 0 \quad t > h_3
\]

\[
\chi(t) = \begin{cases} 
|t|/2 & |t| \leq d \\
0 & |t| > d 
\end{cases}
\]

(d) **Andrew’s Sine Wave Function**

\[
\psi(t) = \sin t \quad -\pi \leq t \leq \pi
\]

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

(e) **Tukey’s Bi-weight**

\[
\psi(t) = t(1 - t^2)^2 \quad |t| \leq 1
\]

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

where \(c, h_1, h_2, h_3\) and \(d\) are constants.

Equations (1) and (2) are solved by a simple iterative procedure suggested by Huber:

\[
\hat{\sigma}_k = \sqrt{\frac{1}{\beta(n-1)} \left( \sum_{i=1}^{n} \chi \left( \frac{x_i - \hat{\theta}_{k-1}}{\hat{\sigma}_{k-1}} \right) \right)} \hat{\sigma}_{k-1}^2
\]

and

\[
\hat{\theta}_k = \hat{\theta}_{k-1} + \frac{1}{n} \sum_{i=1}^{n} \psi \left( \frac{x_i - \hat{\theta}_{k-1}}{\hat{\sigma}_k} \right) \hat{\sigma}_k
\]

or

\[
\hat{\sigma}_k = \sigma_c, \quad \text{if} \quad \sigma \text{ is fixed.}
\]

The initial values for \(\hat{\theta}\) and \(\hat{\sigma}\) may either be user-supplied or calculated within `nag_robust_m_estim_1var` as the sample median and an estimate of \(\sigma\) based on the median absolute deviation respectively.

`nag_robust_m_estim_1var` is based upon subroutine LYHALG within the ROBETH library, see Marazzi (1987).

**4. Parameters**

**sigma_est**

Input: the value assigned to `sigma_est` determines whether \(\hat{\sigma}\) is to be simultaneously estimated.

- `sigma_est = Nag_SigmaBypas`
  The estimation of \(\hat{\sigma}\) is bypassed and `sigma` is set equal to \(\sigma_c\);
- `sigma_est = Nag_SigmaSimul`
  \(\hat{\sigma}\) is estimated simultaneously.

**n**

Input: the number of observations, \(n\).

Constraint: \(n > 1\).
\(x[n]\)
Input: the vector of observations, \(x_1, x_2, \ldots, x_n\).

\textbf{psifun}
Input: which \(\psi\) function is to be used.

- \(psifun = \text{Nag\_Lsq}\), \(\psi(t) = t\)
- \(psifun = \text{Nag\_HuberFun}\), Huber's function,
- \(psifun = \text{Nag\_HampelFun}\), Hampel's piecewise linear function,
- \(psifun = \text{Nag\_AndrewFun}\), Andrew's sine wave,
- \(psifun = \text{Nag\_TukeyFun}\), Tukey's bi-weight.

Constraint: \(psifun = \text{Nag\_Lsq}, \text{Nag\_HuberFun}, \text{Nag\_HampelFun}, \text{Nag\_AndrewFun} \) or \(\text{Nag\_TukeyFun}\).

\(c\)
If \(psifun = \text{Nag\_HuberFun}\) on entry, \(c\) must specify the parameter, \(c\), of Huber's \(\psi\) function. \(c\) is not referenced if \(psifun \neq \text{Nag\_HuberFun}\).

Constraint: \(c > 0\) if \(psifun = \text{Nag\_HuberFun}\).

\(h1\)
\(h2\)
\(h3\)
If \(psifun = \text{Nag\_HampelFun}\) on entry, \(h1\), \(h2\), and \(h3\) must specify the parameters \(h_1\), \(h_2\), and \(h_3\), of Hampel's piecewise linear \(\psi\) function. \(h1\), \(h2\), and \(h3\) are not referenced if \(psifun \neq \text{Nag\_HampelFun}\).

Constraint: \(0 \leq h1 \leq h2 \leq h3\) and \(h3 > 0\) if \(psifun = \text{Nag\_HampelFun}\).

\(dchi\)
Input: the parameter, \(d\), of the \(\chi\) function. \(dchi\) is not referenced if \(psifun = \text{Nag\_Lsq}\).

Constraint: \(dchi > 0\) if \(psifun \neq \text{Nag\_Lsq}\).

\textbf{theta}
Input: if \(\sigma > 0\) then \(\text{theta}\) must be set to the required starting value of the estimation of the location parameter \(\hat{\theta}\). A reasonable initial value for \(\hat{\theta}\) will often be the sample mean or median.

Output: the \(M\)-estimate of the location parameter, \(\hat{\theta}\).

\textbf{sigma}
The role of \(sigma\) depends on the value assigned to \textbf{sigma\_est} (see above) as follows:

- \textbf{sigma\_est = Nag\_SigmaSimul}
Input: \(sigma\) must be assigned a value which determines the values of the starting points for the calculations of \(\hat{\theta}\) and \(\hat{\sigma}\). If \(sigma \leq 0\) then \text{robust_m_estim} will determine the starting points of \(\hat{\theta}\) and \(\hat{\sigma}\). Otherwise the value assigned to \textbf{sigma} will be taken as the starting point for \(\hat{\sigma}\), and \textbf{theta} must be assigned a value before entry, see above.

- \textbf{sigma\_est = Nag\_SigmaBypas}
Input: \(sigma\) must be assigned a value which determines the value of \(\sigma_c\), which is held fixed during the iterations, and the starting value for the calculation of \(\hat{\theta}\). If \(sigma \leq 0\), then \text{robust_m_estim} will determine the value of \(\sigma_c\) as the median absolute deviation adjusted to reduce bias and the starting point for \(\hat{\theta}\). Otherwise, the value assigned to \textbf{sigma} will be taken as the value of \(\sigma_c\), and \textbf{theta} must be assigned a relevant value before entry, see above.

Output: \(sigma\) contains the \(M\) - estimate of the scale parameter, \(\hat{\sigma}\), if \textbf{sigma\_est} was assigned the value \textbf{Nag\_SigmaSimul} on entry, otherwise \(sigma\) will contain the initial fixed value \(\sigma_c\).
5. Error Indications and Warnings

**NE_INT_ARG_LE**
On entry, `n` must not be less than or equal to 1: `n = ⟨value⟩`.
On entry, `maxit` must not be less than or equal to 0: `maxit = ⟨value⟩`.

**NE_REAL_ARG_LE**
On entry, `tol` must not be less than or equal to 0.0: `tol = ⟨value⟩`.

**NE_BAD_PARAM**
On entry, parameter `sigma_est` had an illegal value.
On entry, parameter `psifun` had an illegal value.

**NE_REAL_ENUM_ARG_CONS**
On entry, `c = ⟨value⟩`, `psifun = ⟨value⟩`.
These parameters must satisfy `c > 0`, `psifun = Nag_HuberFun`.
On entry, `h1 = ⟨value⟩`, `psifun = ⟨value⟩`.
These parameters must satisfy `h1 ≥ 0`, `psifun = Nag_HampelFun`.
On entry, `dchi = ⟨value⟩`, `psifun = ⟨value⟩`.
These parameters must satisfy `dchi > 0`, `psifun ≠ Nag_Lsq`.

**NE_REAL2_ENUM_ARG_CONS**
On entry, `h1 = ⟨value⟩`, `h2 = ⟨value⟩` and `psifun = ⟨value⟩`.
These parameters must satisfy `h1 ≤ h2`, `psifun = Nag_HampelFun`.
On entry, `h1 = ⟨value⟩`, `h3 = ⟨value⟩` and `psifun = ⟨value⟩`.
These parameters must satisfy `h1 ≤ h3`, `psifun = Nag_HampelFun`.
On entry, `h2 = ⟨value⟩`, `h3 = ⟨value⟩` and `psifun = ⟨value⟩`.
These parameters must satisfy `h2 ≤ h3`, `psifun = Nag_HampelFun`.

**NE_REAL3_ENUM_ARG_CONS**
On entry, `h1 = ⟨value⟩`, `h2 = ⟨value⟩`, `h3 = ⟨value⟩`, `psifun = ⟨value⟩`.
These parameters must satisfy `h1 = h2 = h3 ≠ 0.0`, `psifun = Nag_HampelFun`.

**NE_ALL_ELEMENTS_EQUAL**
On entry, all the values in the array `x` must not be equal.

**NE_ESTIM_SIGMA_ZERO**
The estimated value of `sigma` was ≤ 0.0 during an iteration.

**NE_TOO_MANY**
Too many iterations (`⟨value⟩`).
The Winsorized residuals are zero.

On completion of the iterations, the Winsorized residuals were all zero. This may occur when using the `sigma_est = Nag_SigmaBypass` option with a redescending $\psi$ function, i.e., Hampel’s piecewise linear function, Andrew’s sine wave, and Tukey’s biweight.

If the given value of $\sigma$ is too small, then the standardised residuals $\frac{x_i - \hat{\theta}_k}{\sigma_c}$ will be large and all the residuals may fall into the region for which $\psi(t) = 0$. This may incorrectly terminate the iterations thus making `theta` and `sigma` invalid.

Re-enter the routine with a larger value of $\sigma_c$ or with `sigma_est = Nag_SigmaSimul`.

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

6. Further Comments
When the user supplies the initial values, care has to be taken over the choice of the initial value of $\sigma$. If too small a value of $\sigma$ is chosen then initial values of the standardized residuals $\frac{x_i - \hat{\theta}_k}{\sigma}$ will be large. If the redescending $\psi$ functions are used, i.e., Hampel’s piecewise linear function, Andrew’s sine wave, or Tukey’s bi-weight, then these large values of the standardised residuals are Winsorized as zero. If a sufficient number of the residuals fall into this category then a false solution may be returned, see Hampel (1986) page 152.

6.1. Accuracy
On successful exit the accuracy of the results is related to the value of TOL, see Section 4.

6.2. References

7. See Also
None.

8. Example
The following program reads in a set of data consisting of eleven observations of a variable $X$.

For this example, Hampel’s Piecewise Linear Function is used (`psifun = Nag_HampelFun`), values for $h_1$, $h_2$ and $h_3$ along with $d$ for the $\chi$ function, being read from the data file.

Using the following starting values various estimates of $\theta$ and $\sigma$ are calculated and printed along with the number of iterations used:

(a) `nag_robust_m_estim_1var` determines the starting values, $\sigma$ is estimated simultaneously.

(b) The user supplies the starting values, $\sigma$ is estimated simultaneously.

(c) `nag_robust_m_estim_1var` determines the starting values, $\sigma$ is fixed.

(d) The user supplies the starting values, $\sigma$ is fixed.
/* nag_robust_m_estim_1var(g07dbc) Example Program.
 * Mark 4, 1996.
 */

#include <nag.h>
#include <nag_stdbb.h>
#include <nag_string.h>
#include <stdio.h>
#include <nagg07.h>

#define NMAX 25

main()
{
    double dchi;
    double c;
    double x[NMAX], sigma, theta;
    double h1, h2, h3, rs[NMAX];
    double thesav, sigsav;
    double tol, wrk[NMAX];
    Integer ipsi;
    Integer i;
    Integer n;
    Integer maxit;
    Integer isigma;
    Integer nit;

    char sigma_enum_str[20];
    Nag_SigmaSimulEst sigma_enum;

    Vprintf("g07dbc Example Program Results
\n");
    /* Skip heading in data file */
    Vscanf("%*[\n]\n");
    Vscanf("%ld %*[\n]\n", &n);
    if (n <= NMAX)
    {
        for (i = 1; i <= n; ++i)
            Vscanf("%lf", &x[i - 1]);
        Vscanf("%*[\n]\n");
        Vscanf("%lf %lf %lf %lf %ld %*[\n]\n", &h1, &h2, &h3, &dchi, &maxit);
        Vprintf("%25sInput parameters Output parameters
\n\n","");
        while (scanf("%ld %lf %lf %lf %*[\n]\n", &isigma, &sigma, &theta, &tol)) != EOF)
        {
            if (isigma == 1)
            {
                sigma_enum = Nag_SigmaSimul;
                strcpy(sigma_enum_str, "Nag_SigmaSimul");
            }
            else if (isigma == 0)
            {
                sigma_enum = Nag_SigmaBypas;
                strcpy(sigma_enum_str, "Nag_SigmaBypas");
            }
            sigsav = sigma;
            thesav = theta;
            c = 0.0;
            g07dbc(sigma_enum, n, x, Nag_HampelFun, c, h1, h2, h3, dchi, &theta,
                   &sigma, maxit, tol, rs, &nit, wrk, NAGERR_DEFAULT);
        }
    }
}
8.2. Program Data

g07dbc Example Program Data
11 : Number of observations
13.0 11.0 16.0 5.0 3.0 18.0 9.0 8.0 6.0 27.0 7.0 : Observations
1.5 3.0 4.5 1.5 50 : h1 h2 h3 dchi maxit
1 -1.0 0.0 0.0001 :isigma sigma theta tol
1 7.0 2.0 0.0001
0 -1.0 0.0 0.0001
0 7.0 2.0 0.0001

8.3. Program Results

g07dbc Example Program Results

<table>
<thead>
<tr>
<th>sigma_est</th>
<th>Input parameters</th>
<th>Output parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigma_est</td>
<td>sigma theta tol</td>
<td>sigma theta</td>
</tr>
<tr>
<td>Nag_SigmaSimul</td>
<td>-1.0000 0.0000 0.0001</td>
<td>6.3247 10.5487</td>
</tr>
<tr>
<td>Nag_SigmaSimul</td>
<td>7.0000 2.0000 0.0001</td>
<td>6.3249 10.5487</td>
</tr>
<tr>
<td>Nag_SigmaBypas</td>
<td>-1.0000 0.0000 0.0001</td>
<td>5.9304 10.4896</td>
</tr>
<tr>
<td>Nag_SigmaBypas</td>
<td>7.0000 2.0000 0.0001</td>
<td>7.0000 10.6500</td>
</tr>
</tbody>
</table>