NAG Library Function Document

nag_robust_m_estim_1var_usr (g07dcc)

1 Purpose

nag_robust_m_estim_1var_usr (g07dcc) computes an M-estimate of location with (optional) simultaneous estimation of scale, where you provide the weight functions.

2 Specification

```
#include <nag.h>
#include <nagg07.h>
void nag_robust_m_estim_1var_usr (
     double (*chi)(double t, Nag_Comm *comm),
     double (*psi)(double t, Nag_Comm *comm),
     Integer isigma, Integer n, const double x[], double beta, double *theta,
     double *sigma, Integer maxit, double tol, double rs[], Integer *nit,
     Nag_Comm *comm, 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 θ is a location argument, and σ is a scale argument. M-estimators of θ and σ are given by the solution to the following system of equations;

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

$$\sum_{i=1}^{n} \chi\Big(\Big(x_i - \hat{\theta}\Big)/\hat{\sigma}\Big) \quad = \quad (n-1)\beta$$

where ψ and χ are user-supplied weight functions, and β is a constant. 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 constant β should be chosen so that $\hat{\sigma}$ is an unbiased estimator when x_i, for $i = 1, 2, \ldots, n$ has a Normal distribution. To achieve this the value of β is calculated as:

$$\beta = E(\chi) = \int_{-\infty}^{\infty} \chi(z) \frac{1}{\sqrt{2\pi}} \exp\left\{\frac{-z^2}{2}\right\} dz$$

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

The equations 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

$$\hat{\sigma}_k = \sigma_c$$

if σ is fixed.

The initial values for $\hat{\theta}$ and $\hat{\sigma}$ may be user-supplied or calculated within nag_robust_m_estim_1var (g07dbc) as the sample median and an estimate of σ based on the median absolute deviation respectively.

nag_robust_m_estim_1var_usr (g07dcc) is based upon function LYHALG within the ROBETH library, see Marazzi (1987).

4 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 (1987) Subroutines for robust estimation of location and scale in ROBETH *Cah. Rech. Doc. IUMSP, No. 3 ROB 1* Institut Universitaire de Médecine Sociale et Préventive, Lausanne

5 Arguments

1: **chi** – function, supplied by the user *External Function*

chi must return the value of the weight function χ for a given value of its argument. The value of χ must be non-negative.

The specification of **chi** is:

```
double chi (double t, Nag_Comm *comm)
```

1: **t** – double *Input*

On entry: the argument for which **chi** must be evaluated.

2: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **chi**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_robust_m_estim_1var_usr (g07dcc) you may allocate memory and initialize these pointers with various quantities for use by **chi** when called from nag_robust_m_estim_1var_usr (g07dcc) (see Section 3.2.1.1 in the Essential Introduction).

2: **psi** – function, supplied by the user *External Function*

psi must return the value of the weight function ψ for a given value of its argument.

The specification of **psi** is:

```
double psi (double t, Nag_Comm *comm)
```

1: **t** – double *Input*

On entry: the argument for which **psi** must be evaluated.

2: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **psi**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_robust_m_estim_1var_usr (g07dcc) you may allocate memory and initialize these pointers with various quantities for use by **psi** when called from nag_robust_m_estim_1var_usr (g07dcc) (see Section 3.2.1.1 in the Essential Introduction).

3: **isigma** – Integer *Input*

On entry: the value assigned to **isigma** determines whether $\hat{\sigma}$ is to be simultaneously estimated.

isigma $= 0$
The estimation of $\hat{\sigma}$ is bypassed and **sigma** is set equal to σ_c.

isigma $= 1$
$\hat{\sigma}$ is estimated simultaneously.

4: **n** – Integer *Input*

On entry: n, the number of observations.

Constraint: **n** > 1.

5: **x**[**n**] – const double *Input*

On entry: the vector of observations, $x_1, x_2, \ldots, x_n$.

6: **beta** – double *Input*

On entry: the value of the constant β of the chosen **chi** function.

Constraint: **beta** > 0.0.

7: **theta** – double * *Input/Output*

On entry: if **sigma** > 0, then **theta** must be set to the required starting value of the estimate of the location argument $\hat{\theta}$. A reasonable initial value for $\hat{\theta}$ will often be the sample mean or median.

On exit: the M-estimate of the location argument $\hat{\theta}$.

8: **sigma** – double * *Input/Output*

On entry: the role of **sigma** depends on the value assigned to **isigma** as follows.

If **isigma** $= 1$, **sigma** must be assigned a value which determines the values of the starting points for the calculation of $\hat{\theta}$ and $\hat{\sigma}$. If **sigma** ≤ 0.0, then nag_robust_m_estim_1var_usr (g07dcc) will determine the starting points of $\hat{\theta}$ and $\hat{\sigma}$. Otherwise, the value assigned to **sigma** will be taken as the starting point for $\hat{\sigma}$, and **theta** must be assigned a relevant value before entry, see above.

If **isigma** $= 0$, **sigma** must be assigned a value which determines the values of σ_c, which is held fixed during the iterations, and the starting value for the calculation of $\hat{\theta}$. If **sigma** ≤ 0, then nag_robust_m_estim_1var_usr (g07dcc) will determine the value of σ_c as the median absolute deviation adjusted to reduce bias (see nag_median_1var (g07dac)) and the starting point for θ. Otherwise, the value assigned to **sigma** will be taken as the value of σ_c and **theta** must be assigned a relevant value before entry, see above.

On exit: the M-estimate of the scale argument $\hat{\sigma}$, if **isigma** was assigned the value 1 on entry, otherwise **sigma** will contain the initial fixed value σ_c.

9: **maxit** – Integer *Input*

On entry: the maximum number of iterations that should be used during the estimation.

Suggested value: **maxit** $= 50$.

Constraint: **maxit** > 0.

10: **tol** – double *Input*

On entry: the relative precision for the final estimates. Convergence is assumed when the increments for **theta**, and **sigma** are less than **tol** $\times \max(1.0, \sigma_{k-1})$.

Constraint: **tol** > 0.0.

11: **rs**[**n**] – double *Output*

On exit: the Winsorized residuals.

12: **nit** – Integer * *Output*

On exit: the number of iterations that were used during the estimation.

13: **comm** – Nag_Comm *

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

14: **fail** – NagError * *Input/Output*

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_FUN_RET_VAL

The **chi** function returned a negative value: **chi** $= \langle value \rangle$.

NE_INT

On entry, **isigma** $= \langle value \rangle$.
Constraint: **isigma** $= 0$ or 1.

On entry, **maxit** $= \langle value \rangle$.
Constraint: **maxit** > 0.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** > 1.

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.

An unexpected error has been triggered by this function. Please contact NAG. See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly. See Section 3.6.5 in the Essential Introduction for further information.

NE_REAL

On entry, **beta** $= \langle value \rangle$.
Constraint: **beta** > 0.0.

On entry, **tol** $= \langle value \rangle$.
Constraint: **tol** > 0.0.

NE_REAL_ARRAY_ELEM_CONS

All elements of **x** are equal.

NE_SIGMA_NEGATIVE

Current estimate of **sigma** is zero or negative: **sigma** $= \langle value \rangle$.

NE_TOO_MANY_ITER

Number of iterations required exceeds **maxit**: **maxit** $= \langle value \rangle$.

NE_ZERO_RESID

All winsorized residuals are zero.

7 Accuracy

On successful exit the accuracy of the results is related to the value of **tol**, see Section 5.

8 Parallelism and Performance

nag_robust_m_estim_1var_usr (g07dcc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_robust_m_estim_1var_usr (g07dcc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

Standard forms of the functions ψ and χ are given in Hampel *et al.* (1986), Huber (1981) and Marazzi (1987). nag_robust_m_estim_1var (g07dbc) calculates M-estimates using some standard forms for ψ and χ.

When you supply the initial values, care has to be taken over the choice of the initial value of σ. If too small a value is chosen then initial values of the standardized residuals $\dfrac{x_i - \hat{\theta}_k}{\sigma}$ will be large. If the redescending ψ functions are used, i.e., $\psi = 0$ if $|t| > \tau$, for some positive constant τ, then these large values are Winsorized as zero. If a sufficient number of the residuals fall into this category then a false solution may be returned, see page 152 of Hampel *et al.* (1986).

10 Example

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

The **psi** and **chi** functions used are Hampel's Piecewise Linear Function and Hubers **chi** function respectively.

Using the following starting values various estimates of θ and σ are calculated and printed along with the number of iterations used:

(a) nag_robust_m_estim_1var_usr (g07dcc) determined the starting values, σ is estimated simultaneously.

(b) You must supply the starting values, σ is estimated simultaneously.

(c) nag_robust_m_estim_1var_usr (g07dcc) determined the starting values, σ is fixed.

(d) You must supply the starting values, σ is fixed.

10.1 Program Text

```
/* nag_robust_m_estim_1var_usr (g07dcc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 */

#include <math.h>
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg07.h>

#ifdef __cplusplus
extern "C" {
#endif
static double NAG_CALL chi(double t, Nag_Comm *comm);
static double NAG_CALL psi(double t, Nag_Comm *comm);

#ifdef __cplusplus
}
#endif

int main(void)
{

  /* Scalars */
  double   beta, sigma, sigsav, thesav, theta, tol;
  Integer  exit_status, i, isigma, maxit, n, nit;
  NagError fail;
  Nag_Comm comm;

  /* Arrays */
  static double ruser[2] = {-1.0, -1.0};
  double   *rs = 0, *x = 0;

  INIT_FAIL(fail);

  exit_status = 0;
  printf(
          "nag_robust_m_estim_1var_usr (g07dcc) Example Program Results\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
```

```
#else
  scanf("%*[^\n] ");
#endif

#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n] ", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n] ", &n);
#endif
  /* Allocate memory */
  if (!(rs = NAG_ALLOC(n, double)) ||
      !(x = NAG_ALLOC(n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  printf("\n");

  for (i = 1; i <= n; ++i)
    {
#ifdef _WIN32
      scanf_s("%lf", &x[i - 1]);
#else
      scanf("%lf", &x[i - 1]);
#endif
    }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

#ifdef _WIN32
  scanf_s("%lf%"NAG_IFMT"%*[^\n] ", &beta, &maxit);
#else
  scanf("%lf%"NAG_IFMT"%*[^\n] ", &beta, &maxit);
#endif
  printf("          Input parameters     Output parameters\n");
  printf("isigma   sigma   theta   tol    sigma  theta\n");
#ifdef _WIN32
  while (scanf_s("%"NAG_IFMT"%lf%lf%lf%*[^\n] ", &isigma, &sigma,
                 &theta, &tol) != EOF)
    {
#else
  while (scanf("%"NAG_IFMT"%lf%lf%lf%*[^\n] ", &isigma, &sigma,
                 &theta, &tol) != EOF)
    {
#endif
      sigsav = sigma;
      thesav = theta;

      /* nag_robust_m_estim_1var_usr (g07dcc).
       * Robust estimation, M-estimates for location and scale
       * parameters, user-defined weight functions
       */
      nag_robust_m_estim_1var_usr(chi, psi, isigma, n, x, beta, &theta,
                                  &sigma, maxit, tol, rs, &nit, &comm,
                                  &fail);
      if (fail.code != NE_NOERROR)
        {
          printf(
                  "Error from nag_robust_m_estim_1var_usr (g07dcc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }


      printf("%3"NAG_IFMT"%3s%8.4f%8.4f%7.4f", isigma, "", sigsav,
              thesav, tol);
```

```
      printf("%8.4f%8.4f\n", sigma, theta);
    }
 END:
  NAG_FREE(rs);
  NAG_FREE(x);
  return exit_status;
}

static double NAG_CALL psi(double t, Nag_Comm *comm)
{
  /* Scalars */
  double abst;
  double ret_val;


  /* Hampel's Piecewise Linear Function. */
  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback psi, first invocation.)\n");
      comm->user[0] = 0.0;
    }
  abst = fabs(t);
  if (abst < 4.5)
    {
      if (abst <= 3.0)
        {
          ret_val = MIN(1.5, abst);
        }
      else
        {
          ret_val = (4.5 - abst) * 1.5 / 1.5;
        }
      if (t < 0.0)
        {
          ret_val = -ret_val;
        }
    }
  else
    {
      ret_val = 0.0;
    }
  return ret_val;
} /* psi */


double NAG_CALL chi(double t, Nag_Comm *comm)
{
  /* Scalars */
  double abst, ps;
  double ret_val;

  /* Huber's chi function. */
  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback chi, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  abst = fabs(t);
  ps = MIN(1.5, abst);
  ret_val = ps * ps / 2;
  return ret_val;
}
```

10.2 Program Data

```
nag_robust_m_estim_1var_usr (g07dcc) Example Program Data
11                                  : n, 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  : x, observations
0.3892326     50                    : beta    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
```

10.3 Program Results

```
nag_robust_m_estim_1var_usr (g07dcc) Example Program Results

          Input parameters     Output parameters
isigma   sigma   theta   tol    sigma  theta
(User-supplied callback chi, first invocation.)
(User-supplied callback psi, first invocation.)
  1    -1.0000  0.0000 0.0001  6.3247 10.5487
  1     7.0000  2.0000 0.0001  6.3249 10.5487
  0    -1.0000  0.0000 0.0001  5.9304 10.4896
  0     7.0000  2.0000 0.0001  7.0000 10.6500
```