NAG C Library Function Document

nag_robust_m_regsn_param_var (g02hfc)

1 Purpose

nag_robust_m_regsn_param_var (g02hfc) calculates an estimate of the asymptotic variance-covariance matrix for the bounded influence regression estimates (M-estimates). It is intended for use with nag_robust_m_regsn_user_fn (g02hdc).

2 Specification

```
void nag_robust_m_regsn_param_var (Nag_OrderType order,
     double (*psi)(double t, Nag_Comm *comm),
     double (*psp)(double t, Nag_Comm *comm),
     Nag_RegType regtype, Nag_CovMatrixEst covmat_est, double sigma, Integer n,
     Integer m, const double x[], Integer pdx, const double rs[],
     const double wgt[], double cov[], Integer pdc, double comm_arr[],
     Nag_Comm *comm, NagError *fail)
```

3 Description

For a description of bounded influence regression see nag_robust_m_regsn_user_fn (g02hdc). Let θ be the regression parameters and let C be the asymptotic variance-covariance matrix of $\hat{\theta}$. Then for Huber type regression

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

where

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

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

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

For Mallows and Schweppe type regressions, C is of the form

$$\frac{\hat{\sigma}^2}{n}S_1^{-1}S_2S_1^{-1},$$

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

D is a diagonal matrix such that the ith 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 ith 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_param_var (g02hfc):

1. Average over the r_i

Schweppe Mallows

$$D_i = \left(\frac{1}{n}\sum_{j=1}^{n}\psi'\left(\frac{r_j}{\hat{\sigma}w_i}\right)\right)w_i \qquad D_i = \left(\frac{1}{n}\sum_{j=1}^{n}\psi'\left(\frac{r_j}{\hat{\sigma}}\right)\right)w_i$$

$$P_i = \left(\frac{1}{n}\sum_{j=1}^{n}\psi^2\left(\frac{r_j}{\hat{\sigma}w_i}\right)\right)w_i^2 \qquad P_i = \left(\frac{1}{n}\sum_{j=1}^{n}\psi^2\left(\frac{r_j}{\hat{\sigma}}\right)\right)w_i^2$$

2. Replace expected value by observed

Schweppe Mallows

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

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

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

In all cases $\hat{\sigma}$ is a robust estimate of σ.

nag_robust_m_regsn_param_var (g02hfc) is based on routines in ROBETH; see Marazzi (1987b).

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 (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

5 Parameters

1: **order** – Nag_OrderType *Input*

On entry: the **order** parameter specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = **Nag_RowMajor**. See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: **order** = **Nag_RowMajor** or **Nag_ColMajor**.

2: **psi** *Function*

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

Its specification 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 * *Input/Output*

The NAG communication parameter (see the Essential Introduction).

3: **psp** *Function*

psp must return the value of $\psi'(t) = \frac{d}{dt}\psi(t)$ for a given value of its argument.

Its specification is:

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

1: **t** – double *Input*

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

2: **comm** – NAG_Comm * *Input/Output*

The NAG communication parameter (see the Essential Introduction).

4: **regtype** – Nag_RegType *Input*

On entry: the type of regression for which the asymptotic variance-covariance matrix is to be calculated.

If **regtype** = **Nag_HuberReg**, Huber type regression.

If **regtype** = **Nag_MallowsReg**, Mallows type regression.

If **regtype** = **Nag_SchweppeReg**, Schweppe type regression.

5: **covmat_est** – Nag_CovMatrixEst *Input*

On entry: if **regtype** $\neq$ **Nag_HuberReg**, **covmat_est** must specify the approximation to be used.

If **covmat_est** = **Nag_CovMatAve**, averaging over residuals.

If **covmat_est** = **Nag_CovMatObs**, replacing expected by observed.

If **regtype** = **Nag_HuberReg**, **covmat_est** is not referenced.

6: **sigma** – double *Input*

On entry: the value of $\hat{\sigma}$, as given by nag_robust_m_regsn_user_fn (g02hdc).

Constraint: **sigma** > 0.

7: **n** – Integer *Input*

On entry: the number, n, of observations.

Constraint: **n** > 1.

8: **m** – Integer *Input*

On entry: the number, m, of independent variables.

Constraint: $1 \leq$ **m** $<$ **n**.

9: **x**[dim] – const double *Input*

Note: the dimension, dim, of the array **x** must be at least $\max(1, \mathbf{pdx} \times \mathbf{m})$ when **order** = **Nag_ColMajor** and at least $\max(1, \mathbf{pdx} \times \mathbf{n})$ when **order** = **Nag_RowMajor**.

Where $\mathbf{X}(i, j)$ appears in this document, it refers to the array element

if **order** = **Nag_ColMajor**, $\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$;

if **order** = **Nag_RowMajor**, $\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$.

On entry: the values of the X matrix, i.e., the independent variables. $\mathbf{X}(i, j)$ must contain the ijth element of X, for $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$.

10: **pdx** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in the array **x**.

Constraints:

if **order** = **Nag_ColMajor**, **pdx** $\geq$ **n**;
if **order** = **Nag_RowMajor**, **pdx** $\geq$ **m**.

11: **rs**[**n**] – const double *Input*

On entry: the residuals from the bounded influence regression. These are given by nag_robust_m_regsn_user_fn (g02hdc).

12: **wgt**[**n**] – const double *Input*

On entry: if **regtype** $\neq$ **Nag_HuberReg**, **wgt** must contain the vector of weights used by the bounded influence regression. These should be used with nag_robust_m_regsn_user_fn (g02hdc).

If **regtype** = **Nag_HuberReg**, **wgt** is not referenced.

Constraint: if **regtype** $\neq$ **Nag_HuberReg**, **wgt**[i] ≥ 0.0 for $i = 0, 1, \ldots$.

13: **cov**[dim] – double *Output*

Note: the dimension, dim, of the array **c** must be at least **pdc** $\times$ **m**.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix C is stored in **cov**[$(j-1) \times$ **pdc** $+ i - 1$] and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix C is stored in **cov**[$(i-1) \times$ **pdc** $+ j - 1$].

On exit: the estimate of the variance-covariance matrix.

14: **pdc** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in the array **cov**.

Constraint: **pdc** $\geq$ **m**.

15: **comm_arr**[dim] – double *Output*

Note: the dimension, dim, of the array **comm_arr** must be at least **m** $\times$ (**n** + **m** + 1) + 2 $\times$ **n**.

On exit: if **regtype** $\neq$ **Nag_HuberReg**, **comm_arr**[$i-1$], for $i = 1, 2, \ldots, n$, will contain the diagonal elements of the matrix D and **comm_arr**[$i-1$], for $i = n+1, n+2, \ldots, 2n$, will contain the diagonal elements of matrix P.

16: **comm** – NAG_Comm * *Input/Output*

The NAG communication parameter (see the Essential Introduction).

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

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

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

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

On entry, **m** = $\langle value \rangle$.
Constraint: **m** $\geq$ 1.

NE_INT_2

On entry, **m** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: $1 \leq$ **m** $<$ **n**.

On entry, **pdx** $=$ ⟨*value*⟩, **n** $=$ ⟨*value*⟩.
Constraint: **pdx** $\geq$ **n**.

On entry, **pdx** $=$ ⟨*value*⟩, **m** $=$ ⟨*value*⟩.
Constraint: **pdx** $\geq$ **m**.

On entry, **pdc** $=$ ⟨*value*⟩, **m** $=$ ⟨*value*⟩.
Constraint: **pdc** $\geq$ **m**.

On entry, **m** $=$ ⟨*value*⟩, **pdc** $=$ ⟨*value*⟩.
Constraint: **pdc** $\geq$ **m**.

On entry, **n** $\leq$ **m**: **n** $=$ ⟨*value*⟩, **m** $=$ ⟨*value*⟩.

NE_ENUM_INT

On entry, **regtype** $=$ ⟨*value*⟩, **wgt** $=$ ⟨*value*⟩.
Constraint: if **regtype** $\neq$ **Nag_HuberReg**, **wgt**$[i] \geq 0.0$ for $i = 0, \ldots,$.

NE_CORRECTION_FACTOR

Correction factor = 0 (Huber type regression).

NE_POS_DEF

$X^T X$ matrix not positive definite.

NE_REAL

On entry, **sigma** $=$ ⟨*value*⟩.
Constraint: **sigma** ≥ 0.

NE_REAL_ARRAY_ELEM_CONS

On entry, an element of **wgt** < 0.

NE_SINGULAR

$S1$ matrix is singular or almost singular.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

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

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 consult NAG for assistance.

7 Accuracy

In general, the accuracy of the variance-covariance matrix will depend primarily on the accuracy of the results from nag_robust_m_regsn_user_fn (g02hdc).

8 Further Comments

This routine is only for situations in which X has full column rank.

Care has to be taken in the choice of the ψ function since if $\psi'(t) = 0$ for too wide a range then either the value of f_H will not exist or too many values of D_i will be zero and it will not be possible to calculate C.

9 Example

The asymptotic variance-covariance matrix is calculated for a Schweppe type regression. The values of X, $\hat{\sigma}$ and the residuals and weights are read in. The averaging over residuals approximation is used.

9.1 Program Text

```
/* nag_robust_m_regsn_param_var (g02hfc) Example Program.
 *
 * Copyright 2002 Numerical Algorithms Group.
 *
 * Mark 7, 2002.
 */

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

static double psi(double t, Nag_Comm *comm);
static double psp(double t, Nag_Comm *comm);
int main(void)
{

/* Scalars */
  double sigma;
  Integer exit_status, i, ic, ix, j, k, m, n;
  Integer pdc, pdx;
  NagError fail;
  Nag_OrderType order;
  Nag_Comm comm;

  /* Arrays */
  double *cov=0, *rs=0, *wgt=0, *comm_arr=0, *x=0;

#ifdef NAG_COLUMN_MAJOR
#define COV(I,J) cov[(J-1)*pdc + I - 1]
#define X(I,J) x[(J-1)*pdx + I - 1]
  order = Nag_ColMajor;
#else
#define COV(I,J) cov[(I-1)*pdc + J - 1]
#define X(I,J) x[(I-1)*pdx + J - 1]
  order = Nag_RowMajor;
#endif
  exit_status = 0;
  INIT_FAIL(fail);

  Vprintf("g02hfc Example Program Results\n");

/* Skip heading in data file */
  Vscanf("%*[^\n] ");

  /* Read in the dimensions of X */
  Vscanf("%ld%ld%*[^\n] ",   &n,  &m);

  /* Allocate memory */
  if ( !(cov = NAG_ALLOC(m * m, double)) ||
       !(rs = NAG_ALLOC(n, double)) ||
       !(wgt = NAG_ALLOC(n, double)) ||
       !(comm_arr = NAG_ALLOC(m*(n+m+1)+2*n, double)) ||
```

```
        !(x = NAG_ALLOC(n * m, double)) )
     {
       Vprintf("Allocation failure\n");
       exit_status = -1;
       goto END;
     }

#ifdef NAG_COLUMN_MAJOR
  pdc = m;
  pdx = n;
#else
  pdc = m;
  pdx = m;
#endif

  Vprintf("\n");

/* Read in the X matrix */
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= m; ++j)
        {
          Vscanf("%lf", &X(i,j));
        }
      Vscanf("%*[^\n] ");
    }

  /* Read in sigma */
  Vscanf("%lf%*[^\n] ",   &sigma);

  /* Read in weights and residuals */
  for (i = 1; i <= n; ++i)
    {
      Vscanf("%lf%lf%*[^\n] ",   &wgt[i - 1],  &rs[i - 1]);
    }

  /* Set other parameter values */
  ix = 5;
  ic = 3;
  /* Set parameters for Schweppe type regression */
  g02hfc(order, psi, psp, Nag_SchweppeReg, Nag_CovMatAve, sigma, n, m, x, pdx,
         rs, wgt, cov, pdc, comm_arr, &comm, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from g02hfc.\n%s\n",   fail.message);
      exit_status = 1;
      goto END;

    }

  Vprintf("Covariance matrix\n");
  for (j = 1; j <= m; ++j)
    {
      for (k = 1; k <= m; ++k)
        {
          Vprintf("%10.4f%s",   COV(j,k),  k%6 == 0 || k == m ?"\n":" ");
        }
    }

 END:
  if (cov) NAG_FREE(cov);
  if (rs) NAG_FREE(rs);
  if (wgt) NAG_FREE(wgt);
  if (comm_arr) NAG_FREE(comm_arr);
  if (x) NAG_FREE(x);

  return exit_status;
}

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

```
  double ret_val;

  if (t <= -1.5)
    {
      ret_val = -1.5;
    }
  else if (fabs(t) < 1.5)
    {
      ret_val = t;
    }
  else
    {
      ret_val = 1.5;
    }
  return ret_val;
}

static double psp(double t, Nag_Comm *comm)
{
  double ret_val;

  ret_val = 0.0;
  if (fabs(t) < 1.5)
    {
      ret_val = 1.0;
    }
  return ret_val;
}
```

9.2 Program Data

```
g02hfc Example Program Data

    5    3             : N  M

  1.0 -1.0 -1.0        : X1  X2  X3
  1.0 -1.0  1.0
  1.0  1.0 -1.0
  1.0  1.0  1.0
  1.0  0.0  3.0        : End of X1 X2 and X3 values

    20.7783            : SIGMA

    0.4039    0.5643   : Weights and residuals, WGT and RS
    0.5012   -1.1286
    0.4039    0.5643
    0.5012   -1.1286
    0.3862    1.1286   : End of weights and residuals
```

9.3 Program Results

```
g02hfc Example Program Results

Covariance matrix
    0.2070     0.0000    -0.0478
    0.0000     0.2229    -0.0000
   -0.0478    -0.0000     0.0796
```