NAG 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

```
#include <nag.h>
#include <nagg02.h>
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 arguments and let C be the asymptotic variance-covariance matrix of $\hat{\theta}$. Then for Huber type regression

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

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 (1987).

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

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

where $S_1 = \frac{1}{n}X^{\mathrm{T}}DX$ and $S_2 = \frac{1}{n}X^{\mathrm{T}}PX$.

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 $E\left(\psi^2(r_i/(\sigma w_i))w_i^2\right)$ in the Schweppe case and $E\left(\psi^2(r_i/\sigma)w_i^2\right)$ in the Mallows case.

Two approximations are available in nag_robust_m_regsn_param_var (g02hfc):

1. Average over the r_i

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

2. Replace expected value by observed

	Schweppe	Mallows
	$D_i = \psi'\left(\frac{r_i}{\hat{\sigma}w_i}\right)w_i$	$D_i = \psi'\left(\frac{r_i}{\hat{\sigma}}\right)w_i$
	$P_i = \psi^2\left(\frac{r_i}{\hat{\sigma}w_i}\right)w_i^2$	$P_i = \psi^2\left(\frac{r_i}{\hat{\sigma}}\right)w_i^2$

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

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 (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 and bounded influence regression in ROBETH *Cah. Rech. Doc. IUMSP, No. 3 ROB 2* Institut Universitaire de Médecine Sociale et Préventive, Lausanne

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument 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 2.3.1.3 in How to Use the NAG Library and its Documentation for a more detailed explanation of the use of this argument.

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

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

psi must return the value of the ψ function 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_regsn_param_var (g02hfc) you may allocate memory and initialize these pointers with various quantities for use by **psi** when called from nag_robust_m_regsn_param_var (g02hfc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

3: **psp** – function, supplied by the user *External Function*

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

The specification of **psp** 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 *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_robust_m_regsn_param_var (g02hfc) you may allocate memory and initialize these pointers with various quantities for use by **psp** when called from nag_robust_m_regsn_param_var (g02hfc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

4: **regtype** – Nag_RegType *Input*

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

regtype = Nag_MallowsReg
Mallows type regression.

regtype = Nag_HuberReg
Huber type regression.

regtype = Nag_SchweppeReg
Schweppe type regression.

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

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.

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

6: **sigma** – double *Input*

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

Constraint: **sigma** > 0.0.

7: **n** – Integer *Input*

On entry: n, the number of observations.

Constraint: **n** > 1.

8: **m** – Integer *Input*

On entry: m, the number 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;
$\max(1, \mathbf{n} \times \mathbf{pdx})$ when **order** $=$ Nag_RowMajor.

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

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

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 $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$.

10: **pdx** – Integer *Input*

On entry: the stride separating 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**$[\mathbf{n}]$ – 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**$[\mathbf{n}]$ – 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.

13: **cov**$[dim]$ – double *Output*

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

The (i,j)th element of the matrix is stored in

$\mathbf{cov}[(j-1) \times \mathbf{pdc} + i - 1]$ when **order** $=$ Nag_ColMajor;
$\mathbf{cov}[(i-1) \times \mathbf{pdc} + j - 1]$ when **order** $=$ Nag_RowMajor.

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

14: **pdc** – Integer *Input*

On entry: the stride separating 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 $i = n+1, \ldots, 2n$, will contain the diagonal elements of matrix P.

16: **comm** – Nag_Comm *

The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_CORRECTION_FACTOR

Correction factor = 0 (Huber type regression).

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 1.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** > 1.

On entry, **pdc** = ⟨*value*⟩.
Constraint: **pdc** > 0.

On entry, **pdx** = ⟨*value*⟩.
Constraint: **pdx** > 0.

NE_INT_2

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

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

On entry, **n** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **n** $>$ **m**.

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

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

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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_POS_DEF

$X^{\mathrm{T}}X$ matrix not positive definite.

NE_REAL

On entry, **sigma** $= \langle value \rangle$.
Constraint: **sigma** ≥ 0.0.

NE_SINGULAR

S_1 matrix is singular or almost singular.

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 Parallelism and Performance

nag_robust_m_regsn_param_var (g02hfc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_robust_m_regsn_param_var (g02hfc) 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

nag_robust_m_regsn_param_var (g02hfc) 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.

10 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.

10.1 Program Text

```
/* nag_robust_m_regsn_param_var (g02hfc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

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

int main(void)
{

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

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

  printf("nag_robust_m_regsn_param_var (g02hfc) 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

  /* Read in the dimensions of X */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &n, &m);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &n, &m);
```

```
#endif

  /* 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)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

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

  printf("\n");

  /* Read in the X matrix */
  for (i = 1; i <= n; ++i) {
    for (j = 1; j <= m; ++j) {
#ifdef _WIN32
      scanf_s("%lf", &X(i, j));
#else
      scanf("%lf", &X(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif
  }

  /* Read in sigma */
#ifdef _WIN32
  scanf_s("%lf%*[^\n] ", &sigma);
#else
  scanf("%lf%*[^\n] ", &sigma);
#endif

  /* Read in weights and residuals */
  for (i = 1; i <= n; ++i) {
#ifdef _WIN32
    scanf_s("%lf%lf%*[^\n] ", &wgt[i - 1], &rs[i - 1]);
#else
    scanf("%lf%lf%*[^\n] ", &wgt[i - 1], &rs[i - 1]);
#endif
  }

  /* Set parameters for Schweppe type regression */
  /* nag_robust_m_regsn_param_var (g02hfc).
   * Robust regression, variance-covariance matrix following
   * nag_robust_m_regsn_user_fn (g02hdc)
   */
  nag_robust_m_regsn_param_var(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) {
    printf("Error from nag_robust_m_regsn_param_var (g02hfc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
```

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

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

  return exit_status;
}

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

  if (comm->user[0] == -1.0) {
    printf("(User-supplied callback psi, first invocation.)\n");
    comm->user[0] = 0.0;
  }

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

  if (comm->user[1] == -1.0) {
    printf("(User-supplied callback psp, first invocation.)\n");
    comm->user[1] = 0.0;
  }

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

10.2 Program Data

```
nag_robust_m_regsn_param_var (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
```

10.3 Program Results

```
nag_robust_m_regsn_param_var (g02hfc) Example Program Results

(User-supplied callback psp, first invocation.)
(User-supplied callback psi, first invocation.)
Covariance matrix
    0.2070     0.0000    -0.0478
    0.0000     0.2229     0.0000
   -0.0478     0.0000     0.0796
```