

NAG Library Function Document

nag_robust_m_corr_user_fn_no_derr (g02hmc)

1 Purpose

nag_robust_m_corr_user_fn_no_derr (g02hmc) computes a robust estimate of the covariance matrix for user-supplied weight functions. The derivatives of the weight functions are not required.

2 Specification

```
#include <nag.h>
#include <nagg02.h>

void nag_robust_m_corr_user_fn_no_derr (Nag_OrderType order,
    void (*ucv)(double t, double *u, double *w, Nag_Comm *comm),
    Integer indm, Integer n, Integer m, const double x[], Integer pdx,
    double cov[], double a[], double wt[], double theta[], double bl, double bd,
    Integer maxit, Integer nitmon, const char *outfile, double tol, Integer *nit,
    Nag_Comm *comm, NagError *fail)
```

3 Description

For a set of n observations on m variables in a matrix X , a robust estimate of the covariance matrix, C , and a robust estimate of location, θ , are given by

$$C = \tau^2 (A^T A)^{-1},$$

where τ^2 is a correction factor and A is a lower triangular matrix found as the solution to the following equations.

$$z_i = A(x_i - \theta)$$

$$\frac{1}{n} \sum_{i=1}^n w(\|z_i\|_2) z_i = 0$$

and

$$\frac{1}{n} \sum_{i=1}^n u(\|z_i\|_2) z_i z_i^T - v(\|z_i\|_2) I = 0,$$

where x_i is a vector of length m containing the elements of the i th row of X ,

z_i is a vector of length m ,

I is the identity matrix and 0 is the zero matrix.

and w and u are suitable functions.

nag_robust_m_corr_user_fn_no_derr (g02hmc) covers two situations:

- (i) $v(t) = 1$ for all t ,
- (ii) $v(t) = u(t)$.

The robust covariance matrix may be calculated from a weighted sum of squares and cross-products matrix about θ using weights $wt_i = u(\|z_i\|)$. In case (i) a divisor of n is used and in case (ii) a divisor of $\sum_{i=1}^n wt_i$ is used. If $w(\cdot) = \sqrt{u(\cdot)}$, then the robust covariance matrix can be calculated by scaling each row of X by $\sqrt{wt_i}$ and calculating an unweighted covariance matrix about θ .

In order to make the estimate asymptotically unbiased under a Normal model a correction factor, τ^2 , is needed. The value of the correction factor will depend on the functions employed (see Huber (1981) and Marazzi (1987)).

nag_robust_m_corr_user_fn_no_derr (g02hmc) finds A using the iterative procedure as given by Huber; see Huber (1981).

$$A_k = (S_k + I)A_{k-1}$$

and

$$\theta_{j_k} = \frac{b_j}{D_1} + \theta_{j_{k-1}},$$

where $S_k = (s_{jl})$, for $j, l = 1, 2, \dots, m$ is a lower triangular matrix such that

$$s_{jl} = \begin{cases} -\min[\max(h_{jl}/D_2, -BL), BL], & j > l \\ -\min[\max(\frac{1}{2}(h_{jj}/D_2 - 1), -BD), BD], & j = l \end{cases},$$

where

$$D_1 = \sum_{i=1}^n w(\|z_i\|_2)$$

$$D_2 = \sum_{i=1}^n u(\|z_i\|_2)$$

$$h_{jl} = \sum_{i=1}^n u(\|z_i\|_2) z_{ij} z_{il}, \text{ for } j \geq l$$

$$b_j = \sum_{i=1}^n w(\|z_i\|_2) (x_{ij} - b_j)$$

and BD and BL are suitable bounds.

The value of τ may be chosen so that C is unbiased if the observations are from a given distribution.

nag_robust_m_corr_user_fn_no_derr (g02hmc) is based on routines in ROBETH; see Marazzi (1987).

4 References

Huber P J (1981) *Robust Statistics* Wiley

Marazzi A (1987) Weights for bounded influence regression in ROBETH *Cah. Rech. Doc. IUMSP, No. 3 ROB 3* 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 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.
Constraint: **order** = Nag_RowMajor or Nag_ColMajor.
- 2: **ucv** – function, supplied by the user *External Function*
ucv must return the values of the functions u and w for a given value of its argument.

The specification of **ucv** is:

```
void ucv (double t, double *u, double *w, Nag_Comm *comm)
```

1:	t – double	<i>Input</i>
	<i>On entry:</i> the argument for which the functions <i>u</i> and <i>w</i> must be evaluated.	
2:	u – double *	<i>Output</i>
	<i>On exit:</i> the value of the <i>u</i> function at the point t .	
	<i>Constraint:</i> u ≥ 0.0.	
3:	w – double *	<i>Output</i>
	<i>On exit:</i> the value of the <i>w</i> function at the point t .	
	<i>Constraint:</i> w ≥ 0.0.	
4:	comm – Nag_Comm *	<i>Communication Structure</i>
	Pointer to structure of type Nag_Comm; the following members are relevant to ucv .	
	user – double *	
	iuser – Integer *	
	p – Pointer	
	<p>The type Pointer will be void *. Before calling nag_robust_m_corr_user_fn_no_derr (g02hmc) you may allocate memory and initialize these pointers with various quantities for use by ucv when called from nag_robust_m_corr_user_fn_no_derr (g02hmc).</p>	

- 3: **indm** – Integer *Input*
On entry: indicates which form of the function *v* will be used.
- indm** = 1
 $v = 1.$
- indm** ≠ 1
 $v = u.$
- 4: **n** – Integer *Input*
On entry: *n*, the number of observations.
Constraint: **n** > 1.
- 5: **m** – Integer *Input*
On entry: *m*, the number of columns of the matrix *X*, i.e., number of independent variables.
Constraint: $1 \leq \mathbf{m} \leq \mathbf{n}.$
- 6: **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 **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: $\mathbf{X}(i, j)$ must contain the i th observation on the j th variable, for $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$.

- 7: **pdx** – Integer *Input*
On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **x**.
Constraints:
 if **order** = Nag_ColMajor, **pdx** \geq **n**;
 if **order** = Nag_RowMajor, **pdx** \geq **m**.
- 8: **cov**[$\mathbf{m} \times (\mathbf{m} + 1)/2$] – double *Output*
On exit: a robust estimate of the covariance matrix, C . The upper triangular part of the matrix C is stored packed by columns (lower triangular stored by rows), that is C_{ij} is returned in **cov**[$j \times (j - 1)/2 + i - 1$], $i \leq j$.
- 9: **a**[$\mathbf{m} \times (\mathbf{m} + 1)/2$] – double *Input/Output*
On entry: an initial estimate of the lower triangular real matrix A . Only the lower triangular elements must be given and these should be stored row-wise in the array.
 The diagonal elements must be $\neq 0$, and in practice will usually be > 0 . If the magnitudes of the columns of X are of the same order, the identity matrix will often provide a suitable initial value for A . If the columns of X are of different magnitudes, the diagonal elements of the initial value of A should be approximately inversely proportional to the magnitude of the columns of X .
Constraint: **a**[$j \times (j - 1)/2 + j$] $\neq 0.0$, for $j = 0, 1, \dots, m - 1$.
On exit: the lower triangular elements of the inverse of the matrix A , stored row-wise.
- 10: **wt**[**n**] – double *Output*
On exit: **wt**[$i - 1$] contains the weights, $wt_i = u(\|z_i\|_2)$, for $i = 1, 2, \dots, n$.
- 11: **theta**[**m**] – double *Input/Output*
On entry: an initial estimate of the location argument, θ_j , for $j = 1, 2, \dots, m$.
 In many cases an initial estimate of $\theta_j = 0$, for $j = 1, 2, \dots, m$, will be adequate. Alternatively medians may be used as given by nag_median_1var (g07dac).
On exit: contains the robust estimate of the location argument, θ_j , for $j = 1, 2, \dots, m$.
- 12: **bl** – double *Input*
On entry: the magnitude of the bound for the off-diagonal elements of S_k , BL .
Suggested value: **bl** = 0.9.
Constraint: **bl** $>$ 0.0.
- 13: **bd** – double *Input*
On entry: the magnitude of the bound for the diagonal elements of S_k , BD .
Suggested value: **bd** = 0.9.
Constraint: **bd** $>$ 0.0.
- 14: **maxit** – Integer *Input*
On entry: the maximum number of iterations that will be used during the calculation of A .
Suggested value: **maxit** = 150.
Constraint: **maxit** $>$ 0.

- 15: **nitmon** – Integer *Input*
On entry: indicates the amount of information on the iteration that is printed.
nitmon > 0
 The value of A , θ and δ (see Section 7) will be printed at the first and every **nitmon** iterations.
nitmon ≤ 0
 No iteration monitoring is printed.
- 16: **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** = **NULL** or an empty string then the `stdout` stream is used. Note that the file will be opened in the append mode.
- 17: **tol** – double *Input*
On entry: the relative precision for the final estimate of the covariance matrix. Iteration will stop when maximum δ (see Section 7) is less than **tol**.
Constraint: **tol** > 0.0.
- 18: **nit** – Integer * *Output*
On exit: the number of iterations performed.
- 19: **comm** – Nag_Comm * *Communication Structure*
- 20: **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.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_CONST_COL

Column $\langle value \rangle$ of \mathbf{x} has constant value.

NE_CONVERGENCE

Iterations to calculate weights failed to converge.

NE_FUN_RET_VAL

u value returned by **ucv** < 0.0: $u(\langle value \rangle) = \langle value \rangle$.

w value returned by **ucv** < 0.0: $w(\langle value \rangle) = \langle value \rangle$.

NE_INT

On entry, **m** = $\langle value \rangle$.

Constraint: **m** ≥ 1.

On entry, **maxit** = $\langle value \rangle$.

Constraint: **maxit** > 0.

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

On entry, **n** = $\langle value \rangle$.
 Constraint: **n** \geq 2.

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

NE_INT_2

On entry, **m** = $\langle value \rangle$, **n** = $\langle value \rangle$.
 Constraint: $1 \leq \mathbf{m} \leq \mathbf{n}$.

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

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

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

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_NOT_CLOSE_FILE

Cannot close file $\langle value \rangle$.

NE_NOT_WRITE_FILE

Cannot open file $\langle value \rangle$ for writing.

NE_REAL

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

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

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

NE_ZERO_DIAGONAL

On entry, diagonal element $\langle value \rangle$ of **a** is 0.0.

NE_ZERO_SUM

Sum of *u*'s (*D2*) is zero.

Sum of *w*'s (*D1*) is zero.

7 Accuracy

On successful exit the accuracy of the results is related to the value of **tol**; see Section 5. At an iteration let

- (i) $d1 =$ the maximum value of $|s_{jl}|$
- (ii) $d2 =$ the maximum absolute change in $wt(i)$
- (iii) $d3 =$ the maximum absolute relative change in θ_j

and let $\delta = \max(d1, d2, d3)$. Then the iterative procedure is assumed to have converged when $\delta < \text{tol}$.

8 Further Comments

The existence of A will depend upon the function u (see Marazzi (1987)); also if X is not of full rank a value of A will not be found. If the columns of X are almost linearly related, then convergence will be slow.

If derivatives of the u and w functions are available then the method used in `nag_robust_m_corr_user_fn` (g02hlc) will usually give much faster convergence.

9 Example

A sample of 10 observations on three variables is read in along with initial values for A and θ and argument values for the u and w functions, c_u and c_w . The covariance matrix computed by `nag_robust_m_corr_user_fn_no_derr` (g02hmc) is printed along with the robust estimate of θ .

`ucv` computes the Huber's weight functions:

$$u(t) = 1, \quad \text{if} \quad t \leq c_u^2$$

$$u(t) = \frac{c_u}{t^2}, \quad \text{if} \quad t > c_u^2$$

and

$$w(t) = 1, \quad \text{if} \quad t \leq c_w$$

$$w(t) = \frac{c_w}{t}, \quad \text{if} \quad t > c_w.$$

9.1 Program Text

```

/* nag_robust_m_corr_user_fn_no_derr (g02hmc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagg02.h>

static void NAG_CALL ucv(double t, double *u, double *w, Nag_Comm *comm);

int main(int argc, char *argv[])
{
    FILE          *fpin, *fpout;

    /* Scalars */
    double        bd, bl, tol;
    Integer        exit_status, i, indm, j, k, l1, l2, m, maxit, mm, n, nit,
                  nitmon;
    Integer        pdx;
    NagError       fail;
    Nag_OrderType order;
    Nag_Comm       comm;

    /* Arrays */
    double        *a = 0, *cov = 0, *theta = 0, *userp = 0, *wt = 0, *x = 0;

#ifdef NAG_COLUMN_MAJOR
#define X(I, J) x[(J-1)*pdx + I - 1]
    order = Nag_ColMajor;

```

```

#else
#define X(I, J) x[(I-1)*pdx + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);

    /* Check for command-line IO options */
    fpin = nag_example_file_io(argc, argv, "-data", NULL);
    fpout = nag_example_file_io(argc, argv, "-results", NULL);
    exit_status = 0;
    fprintf(fpout,
           "nag_robust_m_corr_user_fn_no_derr (g02hmc) Example Program Results"
           "\n");

    /* Skip heading in data file */
    fscanf(fpin, "%*[\n] ");

    /* Read in the dimensions of x */
    fscanf(fpin, "%ld%ld%*[\n] ", &n, &m);

    /* Allocate memory */
    if (!(a = NAG_ALLOC(m*(m+1)/2, double)) ||
        !(cov = NAG_ALLOC(m*(m+1)/2, double)) ||
        !(theta = NAG_ALLOC(m, double)) ||
        !(userp = NAG_ALLOC(2, double)) ||
        !(wt = NAG_ALLOC(n, double)) ||
        !(x = NAG_ALLOC(n * m, double)))
    {
        fprintf(fpout, "Allocation failure\n");
        exit_status = -1;
        goto END;
    }
#endif
#ifdef NAG_COLUMN_MAJOR
    pdx = n;
#else
    pdx = m;
#endif

    /* Read in the X matrix */
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= m; ++j)
            fscanf(fpin, "%lf", &X(i, j));
        fscanf(fpin, "%*[\n] ");
    }
    /* Read in the initial value of A */
    mm = (m + 1) * m / 2;
    for (j = 1; j <= mm; ++j)
        fscanf(fpin, "%lf", &a[j - 1]);
    fscanf(fpin, "%*[\n] ");

    /* Read in the initial value of theta */
    for (j = 1; j <= m; ++j)
        fscanf(fpin, "%lf", &theta[j - 1]);
    fscanf(fpin, "%*[\n] ");

    /* Read in the values of the parameters of the ucv functions */
    fscanf(fpin, "%lf%lf%*[\n] ", &userp[0], &userp[1]);

    /* Set the values remaining parameters */
    indm = 1;
    bl = 0.9;
    bd = 0.9;
    maxit = 50;
    tol = 5e-5;
    /* Change nitmon to a positive value if monitoring information
     * is required
     */
    nitmon = 0;

```

```

comm.p = (void *) userp;
/* nag_robust_m_corr_user_fn_no_derr (g02hmc).
 * Calculates a robust estimation of a correlation matrix,
 * user-supplied weight function
 */
nag_robust_m_corr_user_fn_no_derr(order, ucv, indm, n, m, x, pdx, cov, a, wt,
                                theta, bl, bd, maxit, nitmon, 0, tol, &nit,
                                &comm, &fail);

if (fail.code != NE_NOERROR)
{
    fprintf(fpout,
            "Error from nag_robust_m_corr_user_fn_no_derr (g02hmc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}

fprintf(fpout, "\n");
fprintf(fpout,
        "nag_robust_m_corr_user_fn_no_derr (g02hmc) required %4ld "
        "iterations to converge\n\n", nit);
fprintf(fpout, "Robust covariance matrix\n");
l2 = 0;
for (j = 1; j <= m; ++j)
{
    l1 = l2 + 1;
    l2 += j;

    for (k = l1; k <= l2; ++k)
    {
        fprintf(fpout, "%10.3f", cov[k - 1]);
        fprintf(fpout, "%s", k%6 == 0 || k == l2?"\n":" ");
    }
}
fprintf(fpout, "\n");

fprintf(fpout, "Robust estimates of Theta\n");
for (j = 1; j <= m; ++j)
    fprintf(fpout, " %10.3f\n", theta[j - 1]);

END:
if (fpin != stdin) fclose(fpin);
if (fpout != stdout) fclose(fpout);
if (a) NAG_FREE(a);
if (cov) NAG_FREE(cov);
if (theta) NAG_FREE(theta);
if (userp) NAG_FREE(userp);
if (wt) NAG_FREE(wt);
if (x) NAG_FREE(x);

return exit_status;
}

void NAG_CALL ucv(double t, double *u, double *w, Nag_Comm *comm)
{
    double t2, cu, cw;

    /* Function Body */
    double *userp = (double *) comm->p;

    cu = userp[0];
    *u = 1.0;
    if (t != 0.0)
    {
        {
            t2 = t * t;
            if (t2 > cu)
                *u = cu / t2;
        }
    }
    /* w function */
    cw = userp[1];
    if (t > cw)

```

```

    *w = cw / t;
else
    *w = 1.0;
return;
}

```

9.2 Program Data

nag_robust_m_corr_user_fn_no_derr (g02hmc) Example Program Data

```

10      3      : N M
3.4  6.9  12.2      : X1 X2 X3
6.4  2.5  15.1
4.9  5.5  14.2
7.3  1.9  18.2
8.8  3.6  11.7
8.4  1.3  17.9
5.3  3.1  15.0
2.7  8.1   7.7
6.1  3.0  21.9
5.3  2.2  13.9      : End of X1 X2 and X3 values
1.0  0.0  1.0  0.0  0.0  1.0      : A
0.0  0.0  0.0      : THETA
4.0  2.0      : CU CW

```

9.3 Program Results

nag_robust_m_corr_user_fn_no_derr (g02hmc) Example Program Results

nag_robust_m_corr_user_fn_no_derr (g02hmc) required 34 iterations to converge

Robust covariance matrix

```

3.278
-3.692      5.284
4.739      -6.409      11.837

```

Robust estimates of Theta

```

5.700
3.864
14.704

```
