NAG C Library Function Document

nag_robust_m_regsn_wts (g02hbc)

1 Purpose

nag_robust_m_regsn_wts (g02hbc) finds, for a real matrix X of full column rank, a lower triangular matrix A such that $(A^TA)^{-1}$ is proportional to a robust estimate of the covariance of the variables. nag_robust_m_regsn_wts (g02hbc) is intended for the calculation of weights of bounded influence regression using nag_robust_m_regsn_user_fn (g02hdc).

2 Specification

```
void nag_robust_m_regsn_wts (Nag_OrderType order,
     double (*ucv)(double t, Nag_Comm *comm),
     Integer n, Integer m, const double x[], Integer pdx, double a[], double z[],
     double bl, double bd, double tol, Integer maxit, Integer nitmon,
     const char *outfile, Integer *nit, Nag_Comm *comm, NagError *fail)
```

3 Description

In fitting the linear regression model

$$y = X\theta + \epsilon,$$

where y is a vector of length n of the dependent variable,
X is an n by m matrix of independent variables,
θ is a vector of length m of unknown parameters,
and ϵ is a vector of length n of unknown errors,

it may be desirable to bound the influence of rows of the X matrix. This can be achieved by calculating a weight for each observation. Several schemes for calculating weights have been proposed (see Hampel *et al.* (1986) and Marazzi (1987a)). As the different independent variables may be measured on different scales one group of proposed weights aims to bound a standardised measure of influence. To obtain such weights the matrix A has to be found such that

$$\frac{1}{n}\sum_{i=1}^{n} u(\|z_i\|_2) z_i z_i^T = I, \quad (I \text{ is the identity matrix})$$

and

$$z_i = Ax_i,$$

where x_i is a vector of length m containing the elements of the ith row of X,
A is an m by m lower triangular matrix,
z_i is a vector of length m,
and u is a suitable function.

The weights for use with nag_robust_m_regsn_user_fn (g02hdc) may then be computed using

$$w_i = f(\|z_i\|_2)$$

for a suitable user function f.

nag_robust_m_regsn_wts (g02hbc) finds A using the iterative procedure

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

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

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

$h_{jl} = \sum_{i=1}^{n} u(\|z_i\|_2) z_{ij} z_{il}$

and BD and BL are suitable bounds.

In addition the values of $\|z_i\|_2$, for $i = 1, 2, \ldots, n$, are calculated.

nag_robust_m_regsn_wts (g02hbc) is based on routines in ROBETH; see Marazzi (1987a).

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 (1987a) Weights for bounded influence regression in ROBETH *Cah. Rech. Doc. IUMSP, No. 3 ROB 3* 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: **ucv** *Function*

ucv must return the value of the function u for a given value of its argument. The value of u must be non-negative.

Its specification is:

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

1: **t** – double *Input*

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

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

The NAG communication parameter (see the Essential Introduction).

3: **n** – Integer *Input*

On entry: the number, n, of observations.

Constraint: $\mathbf{n} > 1$.

4: **m** – Integer *Input*

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

Constraint: $1 \leq \mathbf{m} \leq \mathbf{n}$.

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

6: **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**.

7: **a**[dim] – double *Input/Output*

Note: the dimension, dim, of the array **a** must be at least $\mathbf{m} \times (\mathbf{m} + 1)/2$.

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 $\neq 0$, although 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.

On exit: the lower triangular elements of the matrix A, stored row-wise.

8: **z**[**n**] – double *Output*

On exit: the value $\|z_i\|_2$, for $i = 1, 2, \ldots, n$.

9: **bl** – double *Input*

On entry: the magnitude of the bound for the off-diagonal elements of S_k.

Suggested value: **bl** = 0.9.

Constraint: **bl** > 0.

10: **bd** – double *Input*

On entry: the magnitude of the bound for the diagonal elements of S_k.

Suggested value: **bd** = 0.9.

Constraint: **bd** > 0.

11: **tol** – double *Input*

On entry: the relative precision for the final value of A. Iteration will stop when the maximum value of $|s_{jl}|$ is less than **tol**.

Constraint: **tol** > 0.0.

12: **maxit** – Integer *Input*

On entry: the maximum number of iterations that will be used during the calculation of A.

A value of **maxit** $= 50$ will often be adequate.

Constraint: **maxit** > 0.

13: **nitmon** – Integer *Input*

On entry: determines the amount of information that is printed on each iteration.

If **nitmon** > 0 then the value of A and the maximum value of $|s_{jl}|$ will be printed at the first and every **nitmon** iterations.

If **nitmon** ≤ 0 then no iteration monitoring is printed.

14: **outfile** – 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.

15: **nit** – Integer * *Output*

On exit: the number of iterations performed.

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** = ⟨*value*⟩.
Constraint: **n** > 1.

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

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

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

NE_INT_2

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

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

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

NE_CONVERGENCE

Iterations to calculate weights failed to converge in **maxit** iterations: **maxit** = ⟨*value*⟩.

NE_FUN_RET_VAL

Value returned by **ucv** function < 0: $u(\langle value \rangle) = \langle value \rangle$.

NE_REAL

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

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

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

NE_ZERO_DIAGONAL

On entry, diagonal element ⟨*value*⟩ of **a** is 0.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

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

NE_NOT_WRITE_FILE

Cannot open file ⟨*value*⟩ for writing.

NE_NOT_CLOSE_FILE

Cannot close file ⟨*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

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

8 Further Comments

The existence of A will depend upon the function u; (see Hampel *et al.* (1986) and Marazzi (1987a)), 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.

9 Example

The example program reads in a matrix of real numbers and computes the Krasker–Welsch weights (see Marazzi (1987a)). The matrix A and the weights are then printed.

9.1 Program Text

```
/* nag_robust_m_regsn_wts (g02hbc) 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>
```

```
#include <nags.h>
#include <nagx01.h>
#include <nagx02.h>

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

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

  /* Arrays */
  double *a=0, *x=0, *z=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);
  exit_status = 0;
  Vprintf("g02hbc 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 ( !(a = NAG_ALLOC(m*(m+1)/2, double)) ||
       !(x = NAG_ALLOC(n * m, double)) ||
       !(z = NAG_ALLOC(n, double)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

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

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

```
   * is required
   */
  nitmon = 0;
  g02hbc(order, ucv, n, m, x, pdx, a, z, bl, bd, tol, maxit,
         nitmon, 0, &nit, &comm, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from g02hbc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  Vprintf("g02hbc required %4ld iterations to converge\n\n", nit);
  Vprintf("Matrix A\n");
  l2 = 0;
  for (j = 1; j <= m; ++j)
    {
      l1 = l2 + 1;
      l2 += j;
      for (k = l1; k <= l2; ++k)
        Vprintf("%9.4f%s", a[k - 1], k%6 == 0 || k == l2 ?"\n":" ");
    }

  Vprintf("\n");
  Vprintf("Vector Z\n");
  for (i = 1; i <= n; ++i)
    Vprintf("%9.4f\n", z[i - 1]);

  /* Calculate Krasker-Welsch weights */
  Vprintf("\n");
  Vprintf("Vector of weights\n");
  for (i = 1; i <= n; ++i)
    {
      z[i - 1] = 1.0 / z[i - 1];
      Vprintf("%9.4f\n", z[i - 1]);
    }

 END:
  if (a) NAG_FREE(a);
  if (x) NAG_FREE(x);
  if (z) NAG_FREE(z);

  return exit_status;
}

static double ucv(double t, Nag_Comm *comm)
{

  /* Scalars */
  double pc, pd, q, q2;
  double ret_val;

/* ucv function for Krasker-Welsch weights */
  ret_val = 1.0;
  if (t != 0.0)
    {
      q = 2.5 / t;
      q2 = q * q;
      pc = s15abc(q);
      if (q2 < -log(X02AKC))
        pd = exp(-q2 / 2.0) / sqrt(X01AAC * 2.0);
      else
        pd = 0.0;
      ret_val = (pc * 2.0 - 1.0) * (1.0 - q2) + q2 - q * 2.0 * pd;
    }
  return ret_val;
}
```

9.2 Program Data

```
g02hbc 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

  1.0 0.0 1.0 0.0 0.0 1.0    : A
```

9.3 Program Results

```
g02hbc Example Program Results
g02hbc required   16 iterations to converge

Matrix A
   1.3208
  -0.0000    1.4518
  -0.5753    0.0000    0.9340

Vector Z
   2.4760
   1.9953
   2.4760
   1.9953
   2.5890

Vector of weights
   0.4039
   0.5012
   0.4039
   0.5012
   0.3862
```