NAG 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 $(A^{\mathrm{T}}A)^{-1}$ 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

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

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 (1987)). As the different independent variables may be measured on different scales one group of proposed weights aims to bound a standardized measure of influence. To obtain such weights the matrix A has to be found such that

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

and

$$z_i = A x_i,$$

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-supplied 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 $j = 1, 2, \ldots, m$ 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 (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) 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 value of the function u for a given value of its argument. The value of u must be non-negative.

The specification of **ucv** 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 *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_robust_m_regsn_wts (g02hbc) you may allocate memory and initialize these pointers with various quantities for use by **ucv** when called from nag_robust_m_regsn_wts (g02hbc) (see Section 3.2.1.1 in the Essential Introduction).

3: **n** – Integer *Input*

On entry: n, the number of observations.

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

4: **m** – Integer *Input*

On entry: m, the number 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;
$\max(1, \mathbf{n} \times \mathbf{pdx})$ 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 real matrix X, i.e., the independent variables. $\mathbf{X}(i,j)$ must contain the ijth element of **x**, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$.

6: **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, $\mathbf{pdx} \geq \mathbf{n}$;
if **order** = Nag_RowMajor, $\mathbf{pdx} \geq \mathbf{m}$.

7: $\mathbf{a}[\mathbf{m} \times (\mathbf{m}+\mathbf{1})/\mathbf{2}]$ – 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 $\neq 0$, 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** > 0.0.

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** > 0.0.

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.

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

nitmon ≤ 0
: No iteration monitoring is printed.

14: **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.

15: **nit** – Integer * *Output*

On exit: the number of iterations performed.

16: **comm** – Nag_Comm *

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

17: **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_CONVERGENCE

Iterations to calculate weights failed to converge in **maxit** iterations: **maxit** $= \langle value \rangle$.

NE_FUN_RET_VAL

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

NE_INT

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

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

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

NE_INT_2

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

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 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_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** > 0.0.

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

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

NE_ZERO_DIAGONAL

On entry, diagonal element $\langle value \rangle$ of **a** is 0.

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_regsn_wts (g02hbc) is not threaded by NAG in any implementation.

nag_robust_m_regsn_wts (g02hbc) 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

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

10 Example

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

10.1 Program Text

```
/* nag_robust_m_regsn_wts (g02hbc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2002.
 * Mark 7b revised, 2004.
 */

#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>

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

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 */
  static double ruser[1] = {-1.0};
  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;
  printf("nag_robust_m_regsn_wts (g02hbc) 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 (!(a = NAG_ALLOC(m*(m+1)/2, double)) ||
      !(x = NAG_ALLOC(n * m, double)) ||
      !(z = NAG_ALLOC(n, double)))
    {
      printf("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)
#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 the initial value of A */
  mm = (m + 1) * m / 2;
  for (j = 1; j <= mm; ++j)
#ifdef _WIN32
    scanf_s("%lf", &a[j - 1]);
#else
    scanf("%lf", &a[j - 1]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
```

```
#endif

  /* 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;
  /* nag_robust_m_regsn_wts (g02hbc).
   * Robust regression, compute weights for use with
   * nag_robust_m_regsn_user_fn (g02hdc)
   */
  nag_robust_m_regsn_wts(order, ucv, n, m, x, pdx, a, z, bl, bd, tol, maxit,
                         nitmon, 0, &nit, &comm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_robust_m_regsn_wts (g02hbc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

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

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

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

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

  return exit_status;
}

static double NAG_CALL ucv(double t, Nag_Comm *comm)
{
  /* Scalars */
  double pc, pd, q, q2;
  double ret_val;

  /* ucv function for Krasker-Welsch weights */
  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback ucv, first invocation.)\n");
      comm->user[0] = 0.0;
```

```
    }
  ret_val = 1.0;
  if (t != 0.0)
    {
      q = 2.5 / t;
      q2 = q * q;
      /* nag_cumul_normal (s15abc).
       * Cumulative Normal distribution function P(x)
       */
      pc = nag_cumul_normal(q);
      /* nag_real_smallest_number (x02akc).
       * The smallest positive model number
       */
      if (q2 < -log(nag_real_smallest_number))
        /* nag_pi (x01aac).
         * pi
         */
        pd = exp(-q2 / 2.0) / sqrt(nag_pi * 2.0);
      else
        pd = 0.0;
      ret_val = (pc * 2.0 - 1.0) * (1.0 - q2) + q2 - q * 2.0 * pd;
    }
  return ret_val;
}
```

10.2 Program Data

```
nag_robust_m_regsn_wts (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
```

10.3 Program Results

```
nag_robust_m_regsn_wts (g02hbc) Example Program Results
(User-supplied callback ucv, first invocation.)
nag_robust_m_regsn_wts (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
```