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 $(A^T A)^{-1}$ 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

```c
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,

$\theta$ is a vector of length $m$ of unknown parameters,

and $\epsilon$ 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

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

$(I$ is the identity matrix)

and

$$z_i = Ax_i,$$

where $x_i$ is a vector of length $m$ containing the elements of the $i$th 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
\[ s_{ji} = \begin{cases} -\min(\max(h_{ji}/n, -BL), BL), & j > l \\ -\min(\max(\frac{1}{2}(h_{ji}/n - 1), -BD), BD), & j = l \end{cases} \]

and \( BD \) and \( BL \) are suitable bounds.

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

\texttt{nag\_robust\_m\_regsn\_wts} is based on routines in ROBETH; see Marazzi (1987a).

### 4 References


### 5 Parameters

1: \texttt{order} – \texttt{Nag\_OrderType} \hspace{2cm} \textit{Input}

\textit{On entry}: the \texttt{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 \texttt{order} = \texttt{Nag\_RowMajor}. See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

\textit{Constraint}: \texttt{order} = \texttt{Nag\_RowMajor} or \texttt{Nag\_ColMajor}.

2: \texttt{ucv} \hspace{2cm} \textit{Function}

\texttt{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:

\begin{verbatim}
double ucv (double t, Nag_Comm *comm)
1: \hspace{1cm} t \hspace{0.5cm} – \hspace{0.5cm} double \hspace{1cm} \textit{Input}
   \textit{On entry}: the argument for which \texttt{ucv} must be evaluated.

2: \hspace{1cm} comm \hspace{0.5cm} – \hspace{0.5cm} \texttt{NAG\_Comm} * \hspace{1cm} \textit{Input/Output}
   The NAG communication parameter (see the Essential Introduction).
\end{verbatim}

3: \texttt{n} – \texttt{Integer} \hspace{2cm} \textit{Input}

\textit{On entry}: the number, \( n \), of observations.

\textit{Constraint}: \( n > 1 \).

4: \texttt{m} – \texttt{Integer} \hspace{2cm} \textit{Input}

\textit{On entry}: the number, \( m \), of independent variables.

\textit{Constraint}: \( 1 \leq m \leq n \).
5: \(x[\text{dim}] \) – const double

**Note:** the dimension, \(\text{dim}\), of the array \(x\) must be at least \(\max(1, \text{pdx} \times m)\) when \(\text{order} = \text{Nag\_ColMajor}\) and at least \(\max(1, \text{pdx} \times n)\) when \(\text{order} = \text{Nag\_RowMajor}\).

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

- if \(\text{order} = \text{Nag\_ColMajor}\), \(x[(j-1) \times \text{pdx} + i - 1]\);
- if \(\text{order} = \text{Nag\_RowMajor}\), \(x[(i-1) \times \text{pdx} + j - 1]\).

**On entry:** the real matrix \(X\), i.e., the independent variables. \(X(i,j)\) must contain the \(ij\)th element of \(x\), for \(i = 1, 2, \ldots, n\), \(j = 1, 2, \ldots, m\).

6: \(\text{pdx} – \text{Integer}\)

**Input**

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

**Constraints:**

- if \(\text{order} = \text{Nag\_ColMajor}\), \(\text{pdx} \geq n\);
- if \(\text{order} = \text{Nag\_RowMajor}\), \(\text{pdx} \geq m\).

7: \(a[\text{dim}] – \text{double}\)

**Input/Output**

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

**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: \(z[n] – \text{double}\)

**Output**

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

9: \(\text{bl} – \text{double}\)

**Input**

**On entry:** the magnitude of the bound for the off-diagonal elements of \(S_k\).

**Suggested value:** \(\text{bl} = 0.9\).

**Constraint:** \(\text{bl} > 0\).

10: \(\text{bd} – \text{double}\)

**Input**

**On entry:** the magnitude of the bound for the diagonal elements of \(S_k\).

**Suggested value:** \(\text{bd} = 0.9\).

**Constraint:** \(\text{bd} > 0\).

11: \(\text{tol} – \text{double}\)

**Input**

**On entry:** the relative precision for the final value of \(A\). Iteration will stop when the maximum value of \(|s_{ij}|\) is less than \(\text{tol}\).

**Constraint:** \(\text{tol} > 0.0\).

12: \(\text{maxit} – \text{Integer}\)

**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

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_j| will be printed at the first and every nitmon iterations.

If nitmon ≤ 0 then no iteration monitoring is printed.

14: outfile – char *

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 *

On exit: the number of iterations performed.

16: comm – NAG_Comm *

The NAG communication parameter (see the Essential Introduction).

17: fail – NagError *

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 ≥ n.

On entry, pdx = ⟨value⟩, m = ⟨value⟩.

Constraint: pdx ≥ m.

On entry, n = ⟨value⟩, m = ⟨value⟩.

Constraint: n ≥ 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(⟨value⟩) = ⟨value⟩.
NE_REAL
On entry, \(bd = \text{value}\).
Constraint: \(bd > 0\).
On entry, \(bl = \text{value}\).
Constraint: \(bl > 0\).
On entry, \(tol = \text{value}\).
Constraint: \(tol > 0\).

NE_ZERO_DIAGONAL
On entry, diagonal element \(\text{value}\) of \(a\) is 0.

NE_ALLOC_FAIL
Memory allocation failed.

NE_BAD_PARAM
On entry, parameter \(\text{value}\) had an illegal value.

NE_NOT_WRITE_FILE
Cannot open file \(\text{value}\) for writing.

NE_NOT_CLOSE_FILE
Cannot close file \(\text{value}\).

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

    END:
    exit_status = -1;
    return 0;
}

#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

    END:
    exit_status = -1;
    return 0;
}
* 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 %ld 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
2.4760
1.9953
1.9953
2.5890
```

Vector of weights

```
0.4039
0.5012
0.4039
0.5012
0.3862
```