NAG Library Function Document

nag_robust_m_regsn_user_fn (g02hdc)

1 Purpose

nag_robust_m_regsn_user_fn (g02hdc) performs bounded influence regression (M-estimates) using an iterative weighted least squares algorithm.

2 Specification

```
#include <nag.h>
#include <nagg02.h>
void nag_robust_m_regsn_user_fn (Nag_OrderType order,
     double (*chi)(double t, Nag_Comm *comm),
     double (*psi)(double t, Nag_Comm *comm),
     double psip0, double beta, Nag_RegType regtype, Nag_SigmaEst sigma_est,
     Integer n, Integer m, double x[], Integer pdx, double y[], double wgt[],
     double theta[], Integer *k, double *sigma, double rs[], double tol,
     double eps, Integer maxit, Integer nitmon, const char *outfile,
     Integer *nit, Nag_Comm *comm, NagError *fail)
```

3 Description

For the linear regression model

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

where y is a vector of length n of the dependent variable,

X is a n by m matrix of independent variables of column rank k,

θ is a vector of length m of unknown arguments,

and ϵ is a vector of length n of unknown errors with var $(\epsilon_i) = \sigma^2$,

nag_robust_m_regsn_user_fn (g02hdc) calculates the M-estimates given by the solution, $\hat{\theta}$, to the equation

$$\sum_{i=1}^{n} \psi(r_i/(\sigma w_i))w_i x_{ij} = 0, \quad j = 1, 2, \ldots, m, \tag{1}$$

where r_i is the ith residual, i.e., the ith element of the vector $r = y - X\hat{\theta}$,

ψ is a suitable weight function,

w_i are suitable weights such as those that can be calculated by using output from nag_robust_m_regsn_wts (g02hbc),

and σ may be estimated at each iteration by the median absolute deviation of the residuals $\hat{\sigma} = \text{med}_i\,[|r_i|]/\beta_1$

or as the solution to

$$\sum_{i=1}^{n} \chi(r_i/(\hat{\sigma} w_i))w_i^2 = (n-k)\beta_2$$

for a suitable weight function χ, where β_1 and β_2 are constants, chosen so that the estimator of σ is asymptotically unbiased if the errors, ϵ_i, have a Normal distribution. Alternatively σ may be held at a constant value.

The above describes the Schweppe type regression. If the w_i are assumed to equal 1 for all i, then Huber type regression is obtained. A third type, due to Mallows, replaces (1) by

$$\sum_{i=1}^{n} \psi(r_i/\sigma) w_i x_{ij} = 0, \quad j = 1, 2, \ldots, m.$$

This may be obtained by use of the transformations

$$\begin{aligned} w_i^* &\leftarrow \sqrt{w_i} \\ y_i^* &\leftarrow y_i \sqrt{w_i} \\ x_{ij}^* &\leftarrow x_{ij} \sqrt{w_i}, \quad j = 1, 2, \ldots, m \end{aligned}$$

(see Marazzi (1987)).

The calculation of the estimates of θ can be formulated as an iterative weighted least squares problem with a diagonal weight matrix G given by

$$G_{ii} = \begin{cases} \dfrac{\psi(r_i/(\sigma w_i))}{(r_i/(\sigma w_i))}, & r_i \neq 0 \\ \psi'(0), & r_i = 0. \end{cases}.$$

The value of θ at each iteration is given by the weighted least squares regression of y on X. This is carried out by first transforming the y and X by

$$\begin{aligned} \tilde{y}_i &= y_i \sqrt{G_{ii}} \\ \tilde{x}_{ij} &= x_{ij} \sqrt{G_{ii}}, \quad j = 1, 2, \ldots, m \end{aligned}$$

and then using a least squares solver. If X is of full column rank then an orthogonal-triangular (QR) decomposition is used; if not, a singular value decomposition is used.

Observations with zero or negative weights are not included in the solution.

Note: there is no explicit provision in the function for a constant term in the regression model. However, the addition of a dummy variable whose value is 1.0 for all observations will produce a value of $\hat{\theta}$ corresponding to the usual constant term.

nag_robust_m_regsn_user_fn (g02hdc) 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: **chi** – function, supplied by the user *External Function*

If **sigma_est** = Nag_SigmaChi, **chi** must return the value of the weight function χ for a given value of its argument. The value of χ must be non-negative.

The specification of **chi** is:

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

1: **t** – double *Input*

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

2: **comm** – Nag_Comm *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_robust_m_regsn_user_fn (g02hdc) you may allocate memory and initialize these pointers with various quantities for use by **chi** when called from nag_robust_m_regsn_user_fn (g02hdc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

chi is required only if **sigma_est** = Nag_SigmaConst, otherwise it can be specified as a pointer with 0 value.

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

psi must return the value of the weight function ψ 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_user_fn (g02hdc) you may allocate memory and initialize these pointers with various quantities for use by **psi** when called from nag_robust_m_regsn_user_fn (g02hdc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

4: **psip0** – double *Input*

On entry: the value of $\psi\prime(0)$.

5: **beta** – double *Input*

On entry: if **sigma_est** = Nag_SigmaRes, **beta** must specify the value of β_1.

For Huber and Schweppe type regressions, β_1 is the 75th percentile of the standard Normal distribution (see nag_deviates_normal (g01fac)). For Mallows type regression β_1 is the solution to

$$\frac{1}{n}\sum_{i=1}^{n}\Phi(\beta_1/\sqrt{w_i}) = 0.75,$$

where Φ is the standard Normal cumulative distribution function (see nag_cumul_normal (s15abc)).

If **sigma_est** $=$ Nag_SigmaChi, **beta** must specify the value of β_2.

$$\beta_2 = \int_{-\infty}^{\infty}\chi(z)\phi(z)\,dz, \qquad \text{in the Huber case;}$$

$$\beta_2 = \frac{1}{n}\sum_{i=1}^{n}w_i\int_{-\infty}^{\infty}\chi(z)\phi(z)\,dz, \qquad \text{in the Mallows case;}$$

$$\beta_2 = \frac{1}{n}\sum_{i=1}^{n}w_i^2\int_{-\infty}^{\infty}\chi(z/w_i)\phi(z)\,dz, \qquad \text{in the Schweppe case;}$$

where ϕ is the standard normal density, i.e., $\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}x^2\right)$.

If **sigma_est** $=$ Nag_SigmaConst, **beta** is not referenced.

Constraint: if **sigma_est** $\neq$ Nag_SigmaConst, **beta** > 0.0.

6: **regtype** – Nag_RegType *Input*

On entry: determines the type of regression to be performed.

regtype $=$ Nag_HuberReg
Huber type regression.

regtype $=$ Nag_MallowsReg
Mallows type regression.

regtype $=$ Nag_SchweppeReg
Schweppe type regression.

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

7: **sigma_est** – Nag_SigmaEst *Input*

On entry: determines how σ is to be estimated.

sigma_est $=$ Nag_SigmaConst
σ is held constant at its initial value.

sigma_est $=$ Nag_SigmaRes
σ is estimated by median absolute deviation of residuals.

sigma_est $=$ Nag_SigmaChi
σ is estimated using the χ function.

Constraint: **sigma_est** $=$ Nag_SigmaRes, Nag_SigmaConst or Nag_SigmaChi.

8: **n** – Integer *Input*

On entry: n, the number of observations.

Constraint: **n** > 1.

9: **m** – Integer *Input*

On entry: m, the number of independent variables.

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

10: $\mathbf{x}[dim]$ – double *Input/Output*

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 $j = 1, 2, \ldots, m$.

If **regtype** = Nag_MallowsReg, during calculations the elements of **x** will be transformed as described in Section 3. Before exit the inverse transformation will be applied. As a result there may be slight differences between the input **x** and the output **x**.

On exit: unchanged, except as described above.

11: **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, $\mathbf{pdx} \geq \mathbf{m}$.

12: $\mathbf{y}[\mathbf{n}]$ – double *Input/Output*

On entry: the data values of the dependent variable.

$\mathbf{y}[i-1]$ must contain the value of y for the ith observation, for $i = 1, 2, \ldots, n$.

If **regtype** = Nag_MallowsReg, during calculations the elements of **y** will be transformed as described in Section 3. Before exit the inverse transformation will be applied. As a result there may be slight differences between the input **y** and the output **y**.

On exit: unchanged, except as described above.

13: $\mathbf{wgt}[\mathbf{n}]$ – double *Input/Output*

On entry: the weight for the ith observation, for $i = 1, 2, \ldots, n$.

If **regtype** = Nag_MallowsReg, during calculations elements of **wgt** will be transformed as described in Section 3. Before exit the inverse transformation will be applied. As a result there may be slight differences between the input **wgt** and the output **wgt**.

If $\mathbf{wgt}[i-1] \leq 0$, the ith observation is not included in the analysis.

If **regtype** = Nag_HuberReg, **wgt** is not referenced.

On exit: unchanged, except as described above.

14: $\mathbf{theta}[\mathbf{m}]$ – double *Input/Output*

On entry: starting values of the argument vector θ. These may be obtained from least squares regression. Alternatively if **sigma_est** = Nag_SigmaRes and $\mathbf{sigma} = 1$ or if **sigma_est** = Nag_SigmaChi and **sigma** approximately equals the standard deviation of the dependent variable, y, then **theta**$[i-1] = 0.0$, for $i = 1, 2, \ldots, m$ may provide reasonable starting values.

On exit: the M-estimate of θ_i, for $i = 1, 2, \ldots, m$.

15: **k** – Integer * *Output*

On exit: the column rank of the matrix X.

16: **sigma** – double * *Input/Output*

On entry: a starting value for the estimation of σ. **sigma** should be approximately the standard deviation of the residuals from the model evaluated at the value of θ given by **theta** on entry.

Constraint: **sigma** > 0.0.

On exit: the final estimate of σ if **sigma_est** $\neq$ Nag_SigmaConst or the value assigned on entry if **sigma_est** $=$ Nag_SigmaConst.

17: **rs**[**n**] – double *Output*

On exit: the residuals from the model evaluated at final value of **theta**, i.e., **rs** contains the vector $\left(y - X\hat{\theta}\right)$.

18: **tol** – double *Input*

On entry: the relative precision for the final estimates. Convergence is assumed when both the relative change in the value of **sigma** and the relative change in the value of each element of **theta** are less than **tol**.

It is advisable for **tol** to be greater than $100 \times$ ***machine precision***.

Constraint: **tol** > 0.0.

19: **eps** – double *Input*

On entry: a relative tolerance to be used to determine the rank of X.

If **eps** $<$ ***machine precision*** or **eps** > 1.0 then ***machine precision*** will be used in place of **tol**.

A reasonable value for **eps** is 5.0×10^{-6} where this value is possible.

20: **maxit** – Integer *Input*

On entry: the maximum number of iterations that should be used during the estimation.

A value of **maxit** $= 50$ should be adequate for most uses.

Constraint: **maxit** > 0.

21: **nitmon** – Integer *Input*

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

nitmon ≤ 0
: No information is printed.

nitmon > 0
: On the first and every **nitmon** iterations the values of **sigma**, **theta** and the change in **theta** during the iteration are printed.

22: **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.

23: **nit** – Integer * *Output*

On exit: the number of iterations that were used during the estimation.

24: **comm** – Nag_Comm *

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

25: **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 $\langle value\rangle$ had an illegal value.

NE_CHI

Value given by **chi** function < 0: **chi**$(\langle value\rangle) = \langle value\rangle$.

NE_CONVERGENCE_SOL

Iterations to solve the weighted least squares equations failed to converge.

NE_CONVERGENCE_THETA

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

NE_FULL_RANK

Weighted least squares equations not of full rank: rank $= \langle value\rangle$.

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** $<$ **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 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_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, **beta** $= \langle value \rangle$.
Constraint: **beta** > 0.0.

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

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

NE_ZERO_DF

On entry, **n** $= \langle value \rangle$ and **k** $= \langle value \rangle$.
Constraint: $\mathbf{n} - \mathbf{k} > 0$.

NE_ZERO_VALUE

Estimated value of **sigma** is zero.

7 Accuracy

The accuracy of the results is controlled by **tol**.

8 Parallelism and Performance

nag_robust_m_regsn_user_fn (g02hdc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_robust_m_regsn_user_fn (g02hdc) 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

In cases when **sigma_est** $\neq$ Nag_SigmaConst it is important for the value of **sigma** to be of a reasonable magnitude. Too small a value may cause too many of the winsorized residuals, i.e., $\psi(r_i/\sigma)$, to be zero, which will lead to convergence problems and may trigger the **fail.code** = NE_FULL_RANK error.

By suitable choice of the functions **chi** and **psi** this function may be used for other applications of iterative weighted least squares.

For the variance-covariance matrix of θ see nag_robust_m_regsn_param_var (g02hfc).

10 Example

Having input X, Y and the weights, a Schweppe type regression is performed using Huber's ψ function. The function BETCAL calculates the appropriate value of β_2.

10.1 Program Text

```
/* nag_robust_m_regsn_user_fn (g02hdc) 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>
#include <nags.h>
#include <nagx01.h>
#include <nagx02.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static double NAG_CALL chi(double t, Nag_Comm *comm);
  static double NAG_CALL psi(double t, Nag_Comm *comm);
  static void NAG_CALL betcal(Integer n, double wgt[], double *beta);
#ifdef __cplusplus
}
#endif

int main(void)
{

  /* Scalars */
  double beta, eps, psip0, sigma, tol;
  Integer exit_status, i, j, k, m, maxit, n, nit, nitmon;
  Integer pdx;
  NagError fail;
  Nag_OrderType order;
  Nag_Comm comm;

  /* Arrays */
  static double ruser[2] = { -1.0, -1.0 };
  double *rs = 0, *theta = 0, *wgt = 0, *x = 0, *y = 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_user_fn (g02hdc) 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 (!(rs = NAG_ALLOC(n, double)) ||
      !(theta = NAG_ALLOC(m, double)) ||
      !(wgt = NAG_ALLOC(n, double)) ||
      !(x = NAG_ALLOC(n * m, double)) || !(y = 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, the Y values and set X(i,1) to 1 for the */
  /* constant term */
  for (i = 1; i <= n; ++i) {
    for (j = 2; j <= m; ++j)
#ifdef _WIN32
      scanf_s("%lf", &X(i, j));
#else
      scanf("%lf", &X(i, j));
#endif
#ifdef _WIN32
    scanf_s("%lf%*[^\n] ", &y[i - 1]);
#else
    scanf("%lf%*[^\n] ", &y[i - 1]);
#endif
    X(i, 1) = 1.0;
  }

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

  /* Set other parameter values */
  maxit = 50;
  tol = 5e-5;
  eps = 5e-6;
  psip0 = 1.0;

  /* Set value of isigma and initial value of sigma */
  sigma = 1.0;

  /* Set initial value of theta */
  for (j = 1; j <= m; ++j)
```

```
    theta[j - 1] = 0.0;
  /* Change nitmon to a positive value if monitoring information
   * is required
   */
  nitmon = 0;

  /* Schweppe type regression */
  /* nag_robust_m_regsn_user_fn (g02hdc).
   * Robust regression, compute regression with user-supplied
   * functions and weights
   */
  nag_robust_m_regsn_user_fn(order, chi, psi, psip0, beta, Nag_SchweppeReg,
                             Nag_SigmaChi, n, m, x, pdx, y, wgt, theta, &k,
                             &sigma, rs, tol, eps, maxit,
                             nitmon, 0, &nit, &comm, &fail);

  printf("\n");
  if (fail.code != NE_NOERROR && fail.code != NE_FULL_RANK) {
    printf("Error from nag_robust_m_regsn_user_fn (g02hdc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
  else {
    if (fail.code == NE_FULL_RANK) {
      printf("nag_robust_m_regsn_user_fn (g02hdc) returned with message "
             "%s\n", fail.message);
      printf("\n");

      printf("Some of the following results may be unreliable\n");
    }
    printf("nag_robust_m_regsn_user_fn (g02hdc) required %4" NAG_IFMT " "
           "iterations to converge\n", nit);
    printf("                   k = %4" NAG_IFMT "\n", k);
    printf("               Sigma = %9.4f\n", sigma);
    printf("    Theta\n");
    for (j = 1; j <= m; ++j)
      printf("%9.4f\n", theta[j - 1]);
    printf("\n");
    printf("  Weights  Residuals\n");
    for (i = 1; i <= n; ++i)
      printf("%9.4f%9.4f\n", wgt[i - 1], rs[i - 1]);
  }
END:
  NAG_FREE(rs);
  NAG_FREE(theta);
  NAG_FREE(wgt);
  NAG_FREE(x);
  NAG_FREE(y);

  return exit_status;
}

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 chi(double t, Nag_Comm *comm)
{
```

```
  /* Scalars */
  double ret_val;
  double ps;

  if (comm->user[1] == -1.0) {
    printf("(User-supplied callback chi, first invocation.)\n");
    comm->user[1] = 0.0;
  }
  ps = 1.5;
  if (fabs(t) < 1.5)
    ps = t;
  ret_val = ps * ps / 2.0;
  return ret_val;
}

static void NAG_CALL betcal(Integer n, double wgt[], double *beta)
{
  /* Scalars */
  double amaxex, anormc, b, d2, dc, dw, dw2, pc, w2;
  Integer i;

  /* Calculate BETA for Schweppe type regression */

  /* Function Body */
  /* nag_real_smallest_number (x02akc).
   * The smallest positive model number
   */
  amaxex = -log(nag_real_smallest_number);
  /* nag_pi (x01aac).
   * pi
   */
  anormc = sqrt(nag_pi * 2.0);
  d2 = 2.25;
  *beta = 0.0;
  for (i = 1; i <= n; ++i) {
    w2 = wgt[i - 1] * wgt[i - 1];
    dw = wgt[i - 1] * 1.5;
    /* nag_cumul_normal (s15abc).
     * Cumulative Normal distribution function P(x)
     */
    pc = nag_cumul_normal(dw);
    dw2 = dw * dw;
    dc = 0.0;
    if (dw2 < amaxex)
      dc = exp(-dw2 / 2.0) / anormc;
    b = (-dw * dc + pc - 0.5) / w2 + (1.0 - pc) * d2;
    *beta = b * w2 / (double) (n) + *beta;
  }
  return;
}
```

10.2 Program Data

```
nag_robust_m_regsn_user_fn (g02hdc) Example Program Data

    5    3            : N  M

   -1.0 -1.0 10.5     : X2  X3  Y
   -1.0  1.0 11.3
    1.0 -1.0 12.6
    1.0  1.0 13.4
    0.0  3.0 17.1     : End of X1 X2 and Y values

    0.4039            : WGT
    0.5012
    0.4039
    0.5012
    0.3862            : End of the weights
```

10.3 Program Results

```
nag_robust_m_regsn_user_fn (g02hdc) Example Program Results
(User-supplied callback chi, first invocation.)
(User-supplied callback psi, first invocation.)

nag_robust_m_regsn_user_fn (g02hdc) required    5 iterations to converge
                   k =    3
               Sigma =    2.7783
    Theta
  12.2321
   1.0500
   1.2464

  Weights  Residuals
   0.4039   0.5643
   0.5012  -1.1286
   0.4039   0.5643
   0.5012  -1.1286
   0.3862   1.1286
```