NAG Library Function Document  
nag_regsn_quant_linear (g02qgc)

Note: this function uses optional arguments to define choices in the problem specification and in the  
details of the algorithm. If you wish to use default settings for all of the optional arguments, you need only  
read Sections 1 to 10 of this document. If, however, you wish to reset some or all of the settings please  
refer to Section 11 for a detailed description of the algorithm, to Section 12 for a detailed description of  
the specification of the optional arguments and to Section 13 for a detailed description of the monitoring  
information produced by the function.

1 Purpose

nag_regsn_quant_linear (g02qgc) performs a multiple linear quantile regression. Parameter estimates  
and, if required, confidence limits, covariance matrices and residuals are calculated.  
nag_regsn_quant_linear (g02qgc) may be used to perform a weighted quantile regression. A simplified  
interface for nag_regsn_quant_linear (g02qgc) is provided by nag_regsn_quant_linear_iid (g02qfc).

2 Specification

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

void nag_regsn_quant_linear (Nag_OrderType order,  
Nag_IncludeIntercept intcpt, Integer n, Integer m, const double dat[],  
Integer pddat, const Integer isx[], Integer ip, const double y[],  
const double wt[], Integer ntau, const double tau[], double *df,  
double b[], double bl[], double bu[], double ch[], double res[],  
const Integer iopts[], const double opts[], Integer state[],  
Integer info[], NagError *fail)
```

3 Description

Given a vector of \(n\) observed values, \(y = \{y_i : i = 1, 2, \ldots, n\}\), an \(n \times p\) design matrix \(X\), a column  
vector, \(x\), of length \(p\) holding the \(i\)th row of \(X\) and a quantile \(\tau \in (0, 1)\), nag_regsn_quant_linear  
(g02qgc) estimates the \(p\)-element vector \(\beta\) as the solution to

\[
\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{n} \rho_{\tau}(y_i - x_i^T \beta)
\]

where \(\rho_{\tau}\) is the piecewise linear loss function \(\rho_{\tau}(z) = z(\tau - I(z < 0))\), and \(I(z < 0)\) is an indicator  
function taking the value 1 if \(z < 0\) and 0 otherwise. Weights can be incorporated by replacing \(X\) and \(y\)  
with \(WX\) and \(Wy\) respectively, where \(W\) is an \(n \times n\) diagonal matrix. Observations with zero weights  
can either be included or excluded from the analysis; this is in contrast to least squares regression where  
such observations do not contribute to the objective function and are therefore always dropped.

nag_regsn_quant_linear (g02qgc) uses the interior point algorithm of Portnoy and Koenker (1997),  
described briefly in Section 11, to obtain the parameter estimates \(\hat{\beta}\), for a given value of \(\tau\).  
Under the assumption of Normally distributed errors, Koenker (2005) shows that the limiting covariance  
matrix of \(\hat{\beta} - \beta\) has the form

\[
\Sigma = \frac{\tau(1 - \tau)}{n} H_n^{-1} J_n H_n^{-1}
\]

where \(J_n = n^{-1} \sum_{i=1}^{n} x_i x_i^T\) and \(H_n\) is a function of \(\tau\), as described below. Given an estimate of the  
covariance matrix, \(\Sigma\), lower (\(\hat{\beta}_L\)) and upper (\(\hat{\beta}_U\)) limits for an \((100 \times \alpha)\)% confidence interval can be  
calculated for each of the \(p\) parameters, via

Mark 25  
g02qgc.1
\[ \hat{\beta}_L = \hat{\beta}_i - t_{n-p,(1+\alpha)/2} \sqrt{\hat{\Sigma}_{ii}}, \hat{\beta}_U = \hat{\beta}_i + t_{n-p,(1+\alpha)/2} \sqrt{\hat{\Sigma}_{ii}} \]

where \( t_{n-p,0.975} \) is the 97.5 percentile of the Student’s t distribution with \( n - k \) degrees of freedom, where \( k \) is the rank of the cross-product matrix \( X^T X \).

Four methods for estimating the covariance matrix, \( \Sigma \), are available:

(i) Independent, identically distributed (IID) errors

Under an assumption of IID errors the asymptotic relationship for \( \Sigma \) simplifies to

\[ \Sigma = \frac{\tau(1-\tau)}{n} (s(\tau))^2 (X^T X)^{-1} \]

where \( s \) is the sparsity function. nag_regsn_quant_linear (g02qgc) estimates \( s(\tau) \) from the residuals, \( r_i = y_i - x_i^T \hat{\beta} \) and a bandwidth \( h_n \).

(ii) Powell Sandwich

Powell (1991) suggested estimating the matrix \( H_n \) by a kernel estimator of the form

\[ \hat{H}_n = (nc_n)^{-1} \sum_{i=1}^{n} K \left( \frac{r_i}{c_n} \right) x_i x_i^T \]

where \( K \) is a kernel function and \( c_n \) satisfies \( \lim_{n \to \infty} c_n = 0 \) and \( \lim_{n \to \infty} \sqrt{n} c_n = \infty \). When the Powell method is chosen, nag_regsn_quant_linear (g02qgc) uses a Gaussian kernel (i.e., \( K = \phi \)) and sets

\[ c_n = \min(\sigma_r, (q_{23} - q_{11})/1.34) \times \left( \Phi^{-1}(\tau + h_n) - \Phi^{-1}(\tau - h_n) \right) \]

where \( h_n \) is a bandwidth, \( \sigma_r, q_{11} \) and \( q_{23} \) are, respectively, the standard deviation and the 25\% and 75\% quantiles for the residuals, \( r_i \).

(iii) Hendricks–Koenker Sandwich

Koenker (2005) suggested estimating the matrix \( H_n \) using

\[ \hat{H}_n = n^{-1} \sum_{i=1}^{n} \left[ \frac{2h_n}{x_i^T \left( \hat{\beta}(\tau + h_n) - \hat{\beta}(\tau - h_n) \right)} \right] x_i x_i^T \]

where \( h_n \) is a bandwidth and \( \hat{\beta}(\tau + h_n) \) denotes the parameter estimates obtained from a quantile regression using the \((\tau + h_n)\)th quantile. Similarly with \( \hat{\beta}(\tau - h_n) \).

(iv) Bootstrap

The last method uses bootstrapping to either estimate a covariance matrix or obtain confidence intervals for the parameter estimates directly. This method therefore does not assume Normally distributed errors. Samples of size \( n \) are taken from the paired data \( \{y_i, x_i\} \) (i.e., the independent and dependent variables are sampled together). A quantile regression is then fitted to each sample resulting in a series of bootstrap estimates for the model parameters, \( \hat{\beta} \). A covariance matrix can then be calculated directly from this series of values. Alternatively, confidence limits, \( \hat{\beta}_L \) and \( \hat{\beta}_U \), can be obtained directly from the \((1 - \alpha)/2\) and \((1 + \alpha)/2\) sample quantiles of the bootstrap estimates.

Further details of the algorithms used to calculate the covariance matrices can be found in Section 11.

All three asymptotic estimates of the covariance matrix require a bandwidth, \( h_n \). Two alternative methods for determining this are provided:
(i) Sheather–Hall

\[ h_n = \left( \frac{1.5(\phi^{-1}(\alpha_n)\phi^{-1}(\tau)))^2}{n(2\phi^{-1}(\tau) + 1)} \right)^{\frac{1}{2}} \]

for a user-supplied value \( \alpha_n \),

(ii) Bofinger

\[ h_n = \left( \frac{4.5(\phi^{-1}(\tau)))^4}{n(2\phi^{-1}(\tau) + 1)^2} \right)^{\frac{1}{2}} \]

nag_regsn_quant_linear (g02qgc) allows optional arguments to be supplied via the iopts and opts arrays (see Section 12 for details of the available options). If the default values for these optional arguments are sufficient then iopts and opts can be set to NULL, otherwise prior to calling nag_regsn_quant_linear (g02qgc) the optional argument arrays, must be initialized by calling nag_g02_opt_set (g02zkc) with optstr set to Initialize = g02qgc. If bootstrap confidence limits are required (Interval Method = BOOTSTRAP XY) then one of the random number initialization functions nag_rand_init_repeatable (g05kfc) (for a repeatable analysis) or nag_rand_init_nonrepeatable (g05kgc) (for an unrepeatable analysis) must also have been previously called.

4 References

5 Arguments
1:  \( \text{order} \) – Nag_OrderType

\( \text{Input} \)

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:  \( \text{intcpt} \) – Nag_IncludeIntercept

\( \text{Input} \)

On entry: indicates whether an intercept will be included in the model. The intercept is included by adding a column of ones as the first column in the design matrix, \( X \).

\( \text{intcpt} = \) Nag_IncludeIntercept

An intercept will be included in the model.

\( \text{intcpt} = \) Nag_RemoveIntercept

An intercept will not be included in the model.

Constraint: intcpt = Nag_IncludeIntercept or Nag_RemoveIntercept.
3: n – Integer  
   *Input*  
   *On entry*: the total number of observations in the dataset. If no weights are supplied, or no zero weights are supplied or observations with zero weights are included in the model then \( n = n \). Otherwise \( n = n + \) the number of observations with zero weights.  
   *Constraint*: \( n \geq 2 \).

4: m – Integer  
   *Input*  
   *On entry*: \( m \), the total number of variates in the dataset.  
   *Constraint*: \( m \geq 0 \).

5: dat[dim] – const double  
   *Input*  
   *Note*: the dimension, \( dim \), of the array \( dat \) must be at least  
   \[ pddat \times m \text{ when } order = \text{Nag\_ColMajor}; \]
   \[ pddat \times n \text{ when } order = \text{Nag\_RowMajor}. \]
   Where \( DAT(i,j) \) appears in this document, it refers to the array element  
   \[ dat[(j - 1) \times pddat + i - 1] \text{ when } order = \text{Nag\_ColMajor}; \]
   \[ dat[(i - 1) \times pddat + j - 1] \text{ when } order = \text{Nag\_RowMajor}. \]
   *On entry*: the \( i \)th value for the \( j \)th variate, for \( i = 1, 2, \ldots, n \) and \( j = 1, 2, \ldots, m \), must be supplied in \( DAT(i,j) \).
   The design matrix \( X \) is constructed from \( dat, isx \) and \( intcpt \).

6: pddat – Integer  
   *Input*  
   *On entry*: the stride separating row or column elements (depending on the value of \( order \)) in the array \( dat \).
   *Constraints*:  
   \[ \text{if } order = \text{Nag\_ColMajor}, \text{ pddat } \geq n; \]
   \[ \text{otherwise } pddat \geq m. \]

7: isx[m] – const Integer  
   *Input*  
   *On entry*: indicates which independent variables are to be included in the model.  
   \[ isx[j - 1] = 0 \]
   The \( j \)th variate, supplied in \( dat \), is not included in the regression model.  
   \[ isx[j - 1] = 1 \]
   The \( j \)th variate, supplied in \( dat \), is included in the regression model.  
   *Constraints*:  
   \[ isx[j - 1] = 0 \text{ or } 1, \text{ for } j = 1, 2, \ldots, m; \]
   if \( intcpt = \text{Nag\_Intercept} \), exactly \( ip - 1 \) values of \( isx \) must be set to 1;  
   if \( intcpt = \text{Nag\_NoIntercept} \), exactly \( ip \) values of \( isx \) must be set to 1.

8: ip – Integer  
   *Input*  
   *On entry*: \( p \), the number of independent variables in the model, including the intercept, see \( intcpt \), if present.  
   *Constraints*:  
   \[ 1 \leq ip < n; \]
   if \( intcpt = \text{Nag\_Intercept} \), \( 1 \leq ip \leq m + 1; \]
   if \( intcpt = \text{Nag\_NoIntercept} \), \( 1 \leq ip \leq m. \]
9:  \( y[n] \) – const double  
*Input*

*On entry:* \( y \), the observations on the dependent variable.

10:  \( wt[n] \) – const double  
*Input*

*On entry:* optionally, the diagonal elements of the weight matrix \( W \). 

If weights are not provided then \( wt \) must be set to \text{NULL}.

When

**Drop Zero Weights = YES**

If \( wt[i - 1] = 0.0 \), the \( i \)th observation is not included in the model, in which case the effective number of observations, \( n \), is the number of observations with nonzero weights. If **Return Residuals = YES**, the values of \( res \) will be set to zero for observations with zero weights.

**Drop Zero Weights = NO**

All observations are included in the model and the effective number of observations is \( n \), i.e., \( n = n \).

**Constraints:**

- the effective number of observations \( \geq 2 \);
- \( wt[i] = 0.0 \), for all \( i \).

11:  \( ntau \) – Integer  
*Input*

*On entry:* the number of quantiles of interest.

**Constraint:** \( ntau \geq 1 \).

12:  \( tau[ntau] \) – const double  
*Input*

*On entry:* the vector of quantiles of interest. A separate model is fitted to each quantile.

**Constraint:** \( \sqrt{\epsilon} < tau[j - 1] < 1 - \sqrt{\epsilon} \) where \( \epsilon \) is the *machine precision* returned by \text{nag_machinePrecision (X02AJC)}, for \( j = 1, 2, \ldots, ntau \).

13:  \( df \) – double *  
*Output*

*On exit:* the degrees of freedom given by \( n - k \), where \( n \) is the effective number of observations and \( k \) is the rank of the cross-product matrix \( X^T X \).

14:  \( b[ip \times ntau] \) – double  
*Input/Output*

**Note:** where \( B(i, l) \) appears in this document, it refers to the array element \( b[(l - 1) \times ip + i - 1] \).

*On entry:* if **Calculate Initial Values = NO**, \( B(i, l) \) must hold an initial estimates for \( \hat{\beta}_i \), for \( i = 1, 2, \ldots, ip \) and \( l = 1, 2, \ldots, ntau \). If **Calculate Initial Values = YES**, \( b \) need not be set.

*On exit:* \( B(i, l) \), for \( i = 1, 2, \ldots, ip \), contains the estimates of the parameters of the regression model, \( \hat{\beta} \), estimated for \( \tau = tau[l - 1] \).

If **intcept = Nag_Intercept**, \( B(1, l) \) will contain the estimate corresponding to the intercept and \( B(i + 1, l) \) will contain the coefficient of the \( j \)th variate contained in \( dat \), where \( isx[j - 1] \) is the \( i \)th nonzero value in the array \( isx \).

If **intcept = Nag_NoIntercept**, \( B(i, l) \) will contain the coefficient of the \( j \)th variate contained in \( dat \), where \( isx[j - 1] \) is the \( i \)th nonzero value in the array \( isx \).

15:  \( bl[dim] \) – double  
*Output*

**Note:** the dimension, \( dim \), of the array \( bl \) must be at least \( ntau \) when **Interval Method \( \neq NONE**.

Where \( BL(i, l) \) appears in this document, it refers to the array element \( bl[(l - 1) \times ip + i - 1] \).
On exit: if Interval Method \( \neq \) NONE, \( BL(i,l) \) contains the lower limit of an \( (100 \times \alpha)\% \) confidence interval for \( B(i,l), \) for \( i = 1, 2, \ldots, ip \) and \( l = 1, 2, \ldots, ntau. \)

If Interval Method = NONE, \( bl \) is not referenced and can be set to NULL.

The method used for calculating the interval is controlled by the optional arguments Interval Method and Bootstrap Interval Method. The size of the interval, \( \alpha \), is controlled by the optional argument Significance Level.

16: \( bu[dim] \) – double

Output

Note: the dimension, \( dim \), of the array \( bu \) must be at least \( ntau \) when Interval Method \( \neq \) NONE.

Where \( BU(i,l) \) appears in this document, it refers to the array element \( bu[(l-1) \times ip + i - 1] \).

On exit: if Interval Method \( \neq \) NONE, \( BU(i,l) \) contains the upper limit of an \( (100 \times \alpha)\% \) confidence interval for \( B(i,l), \) for \( i = 1, 2, \ldots, ip \) and \( l = 1, 2, \ldots, ntau. \)

If Interval Method = NONE, \( bu \) is not referenced and can be set to NULL.

The method used for calculating the interval is controlled by the optional arguments Interval Method and Bootstrap Interval Method. The size of the interval, \( \alpha \), is controlled by the optional argument Significance Level.

17: \( ch[dim] \) – double

Output

Note: the dimension, \( dim \), of the array \( ch \) must be at least

\[
\text{if Interval Method } \neq \text{ NONE and Matrix Returned } = \text{ COVARIANCE, } ip \times ip \times ntau; \\
\text{if Interval Method } \neq \text{ NONE, IID or Bootstrap XY and Matrix Returned } = \text{ H INVERSE, } ip \times ip \times (ntau + 1).
\]

Where \( CH(i,j,l) \) appears in this document, it refers to the array element \( ch[(l-1) \times ip + (j-1) \times ip + i - 1] \).

On exit: depending on the supplied optional arguments, \( ch \) will either not be referenced, hold an estimate of the upper triangular part of the covariance matrix, \( \Sigma \), or an estimate of the upper triangular parts of \( nJ_n \) and \( n^{-1}H_n^{-1} \).

If Interval Method = NONE or Matrix Returned = NONE, \( ch \) is not referenced.

If Interval Method = Bootstrap XY or IID and Matrix Returned = H INVERSE, \( ch \) is not referenced.

Otherwise, for \( i, j = 1, 2, \ldots, ip, j \geq i \) and \( l = 1, 2, \ldots, ntau \):

\[
\text{if Matrix Returned } = \text{ COVARIANCE, } CH(i,j,l) \text{ holds an estimate of the covariance between } B(i,l) \text{ and } B(j,l). \\
\text{if Matrix Returned } = \text{ H INVERSE, } CH(i,j,1) \text{ holds an estimate of the } (i,j)\text{th element of } nJ_n \text{ and } CH(i,j,l+1) \text{ holds an estimate of the } (i,j)\text{th element of } n^{-1}H_n^{-1}, \text{ for } \tau = tau[l-1].
\]

The method used for calculating \( \Sigma \) and \( H_n^{-1} \) is controlled by the optional argument Interval Method.

In cases where \( ch \) is not going to be referenced it can be set to NULL.

18: \( res[n \times ntau] \) – double

Output

Note: the \( (i,j)\)th element of the matrix is stored in \( res[(j-1) \times n + i - 1] \).

On exit: if Return Residuals = YES, \( res[(l-1) \times n + i - 1] \) holds the (weighted) residuals, \( r_i \), for \( \tau = tau[l-1], \) for \( i = 1, 2, \ldots, n \) and \( l = 1, 2, \ldots, ntau. \)

If wt is not NULL and Drop Zero Weights = YES, the value of \( res \) will be set to zero for observations with zero weights.

If Return Residuals = NO, \( res \) is not referenced and can be set to NULL.
19: \( \text{iopts}[\text{dim}] \) – const Integer

**Communication Array**

**Note:** the dimension, \( \text{dim} \), of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument \( \text{iopts} \) in the previous call to \text{nag_g02_opt_set} (g02zkc).

**On entry:** if the default values of the optional arguments are sufficient, then \( \text{iopts} \) can be set to \text{NULL}, otherwise the optional argument array, as initialized by a call to \text{nag_g02_opt_set} (g02zkc) must be supplied.

20: \( \text{opts}[\text{dim}] \) – const double

**Communication Array**

**Note:** the dimension, \( \text{dim} \), of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument \( \text{opts} \) in the previous call to \text{nag_g02_opt_set} (g02zkc).

**On entry:** if the default values of the optional arguments are sufficient, then \( \text{opts} \) can be set to \text{NULL}, otherwise the optional argument array, as initialized by a call to \text{nag_g02_opt_set} (g02zkc) must be supplied.

21: \( \text{state}[\text{dim}] \) – Integer

**Communication Array**

**Note:** the dimension, \( \text{dim} \), of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument \( \text{state} \) in the previous call to \text{nag_rand_init_repeatable} (g05kfc) or \text{nag_rand_init_nonrepeatable} (g05kgc).

If \text{Interval Method} = \text{BOOTSTRAP XY}, \( \text{state} \) contains information about the selected random number generator. Otherwise \( \text{state} \) is not referenced and can be set to \text{NULL}.

22: \( \text{info}[\text{ntau}] \) – Integer

**Output**

**On exit:** \( \text{info}[i] \) holds additional information concerning the model fitting and confidence limit calculations when \( \tau = \text{tau}[i] \).

**Code**

**Warning**

0 Model fitted and confidence limits (if requested) calculated successfully
1 The function did not converge. The returned values are based on the estimate at the last iteration. Try increasing \text{Iteration Limit} whilst calculating the parameter estimates or relaxing the definition of convergence by increasing \text{Tolerance}.
2 A singular matrix was encountered during the optimization. The model was not fitted for this value of \( \tau \).
4 Some truncation occurred whilst calculating the confidence limits for this value of \( \tau \). See Section 11 for details. The returned upper and lower limits may be narrower than specified.
8 The function did not converge whilst calculating the confidence limits. The returned limits are based on the estimate at the last iteration. Try increasing \text{Iteration Limit}.
16 Confidence limits for this value of \( \tau \) could not be calculated. The returned upper and lower limits are set to a large positive and large negative value respectively as defined by the optional argument \text{Big}.

It is possible for multiple warnings to be applicable to a single model. In these cases the value returned in \( \text{info} \) is the sum of the corresponding individual nonzero warning codes.

23: \( \text{fail} \) – 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_ARRAY_SIZE
    On entry, pddat = ⟨value⟩ and m = ⟨value⟩.
    Constraint: pddat ≥ m.
    On entry, pddat = ⟨value⟩ and n = ⟨value⟩.
    Constraint: pddat ≥ n.

NE_BAD_PARAM
    On entry, argument ⟨value⟩ had an illegal value.

NE_INITIALIZATION
    On entry, either the option arrays have not been initialized or they have been corrupted.

NE_INT
    On entry, m = ⟨value⟩.
    Constraint: m ≥ 0.
    On entry, n = ⟨value⟩.
    Constraint: n ≥ 2.
    On entry, ntau = ⟨value⟩.
    Constraint: ntau ≥ 1.

NE_INT_2
    On entry, ip = ⟨value⟩ and n = ⟨value⟩.
    Constraint: 1 ≤ ip < n.

NE_INT_ARRAY
    On entry, isx[⟨value⟩] = ⟨value⟩.
    Constraint: isx[i] = 0 or 1 for all i.

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_INVALID_STATE
    On entry, state vector has been corrupted or not initialized.

NE_IP_INCOMP_SX
    On entry, ip is not consistent with isx or interpt: ip = ⟨value⟩, expected value = ⟨value⟩.

NE_NEG_WEIGHT
    On entry, wt[⟨value⟩] = ⟨value⟩.
    Constraint: wt[i] ≥ 0.0 for all i.
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_OBSERVATIONS

On entry, effective number of observations = \(value\).
Constraint: effective number of observations \(\geq value\).

NE_REAL_ARRAY

On entry, \(tau[\langle value\rangle] = \langle value\rangle\) is invalid.

NW_POTENTIAL_PROBLEM

A potential problem occurred whilst fitting the model(s).
Additional information has been returned in info.

7 Accuracy

Not applicable.

8 Parallelism and Performance

\texttt{nag\_regsn\_quant\_linear (g02qgc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\texttt{nag\_regsn\_quant\_linear (g02qgc)} 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

\texttt{nag\_regsn\_quant\_linear (g02qgc)} allocates internally approximately the following elements of double storage: \(13n + np + 3p^2 + 6p + 3(p + 1) \times ntau\). If \texttt{Interval Method = Bootstrap XY} then a further \(np\) elements are required, and this increases by \(p \times ntau \times Bootstrap\ Iterations\) if \texttt{Bootstrap Interval Method = Quantile}. Where possible, any user-supplied output arrays are used as workspace and so the amount actually allocated may be less. If \texttt{order = Nag\_RowMajor, wt is NULL, intcpt = Nag\_NoIntercept and ip = m\) an internal copy of the input data is avoided and the amount of locally allocated memory is reduced by \(np\).

10 Example

A quantile regression model is fitted to Engels 1857 study of household expenditure on food. The model regresses the dependent variable, household food expenditure, against two explanatory variables, a column of ones and household income. The model is fit for five different values of \(\tau\) and the covariance matrix is estimated assuming Normal IID errors. Both the covariance matrix and the residuals are returned.
10.1 Program Text

/* nag_regsn_quant_linear (g02qgc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 23, 2011.
*/
/* Pre-processor includes */
#include <stdio.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>
#include <nagg05.h>
#include <nagx04.h>
#define DAT(i,j) dat[(order==Nag_RowMajor) ? (i*pddat+j) : (j*pddat+i)]
#define CH(i, j, k) ch[k*ip*ip + j*ip + i]
#define LOPTSTR 80

int main(void)
{
    /* Integer scalar and array declarations */
    Integer lseed = 1, liopts = 100, lopts = 100, lcvalue = LOPTSTR;
    Integer exit_status = 0;
    Integer genid, i, ip, ivalue, j, l, lc, lstate, loptstr,
            m, n, ntau, subid, tdch, pddat;
    Integer *info = 0, *iopts = 0, *isx = 0, *state = 0;
    Integer seed[1];
    /* NAG structures */
    NagError fail;
    Nag_OrderType order;
    Nag_IncludeIntercept intcpt;
    Nag_Boolean weighted;
    Nag_VariableType optype;
    /* Double scalar and array declarations */
    double df, rvalue;
    double *b = 0, *bl = 0, *bu = 0, *ch = 0, *dat = 0,
            *opts = 0, *res = 0, *tau = 0, *wt = 0, *y = 0;
    /* Character scalar and array declarations */
    char semeth[30], *poptstr, *cvalue = 0;
    char optstr[LOPTSTR], corder[40], cintcpt[40],
            cweighted[40], cgenid[40];
    char *clabs = 0, **clabsc = 0;
    /* Initialise the error structure to print out any error messages */
    INIT_FAIL(fail);
    printf("nag_regsn_quant_linear (g02qgc) Example Program Results\n\n");
    fflush(stdout);
    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\r\n] ");
    #else
    scanf("%*[\r\n] ");
    #endif
    /* Read in the problem size */
    #ifdef _WIN32
    scanf_s("%39s%*[\r\n]", corder, _countof(corder));
    #else
    scanf("%39s%*[\r\n]", corder);
    #endif
    #ifdef _WIN32
    scanf_s("%39s%39s*%*[\r\n]", cintcpt, _countof(cintcpt), cweighted,
#ifdef _WIN32
    scanf_s("%39s%39s%*[\n"] , cintcpt , cweighted);
#else
    scanf("%39s%39s%*[\n"] , cintcpt , cweighted);
#endif

#define _countof(x) sizeof(x)/sizeof(x[0])

define _WIN32
    scanf_s("%NAG_IFMT"%NAG_IFMT"%NAG_IFMT"%*[\n"] , &n , &m , &ntau);
#else
    scanf("%NAG_IFMT"%NAG_IFMT"%NAG_IFMT"%*[\n"] , &n , &m , &ntau);
#endif

order = (Nag_OrderType) nag_enum_name_to_value(corder);
intcpt = (Nag_IncludeIntercept) nag_enum_name_to_value(cintcpt);
/* weighted is a Nag_Boolean flag used in this example program to indicate
 * whether weights are being supplied (weighted=Nag_TRUE)
 * or not (weighted=Nag_FALSE)
 */
weighted = (Nag_Boolean) nag_enum_name_to_value(cweighted);

pddat = (order == Nag_RowMajor) ? m : n;

#define NAG_ALLOC(n, T) malloc((n) * (sizeof(T)) )

if (!(y = NAG_ALLOC(n, double))
    || !(tau = NAG_ALLOC(ntau, double))
    || !(isx = NAG_ALLOC(m, Integer))
    || !(dat = NAG_ALLOC(m*n, double))
    || !(cvalue = NAG_ALLOC(lcvalue, char))
    || !(clabs = NAG_ALLOC(10*10, char))
    || !(clabsc = NAG_ALLOC(10, char *))
}{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

if (weighted)
{
    /* Data includes a weight */
    if (!(wt = NAG_ALLOC(n, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    for (i = 0; i < n; i++)
    {
        #ifdef _WIN32
            for (j = 0; j < m; j++) scanf_s("%lf", &DAT(i, j));
        #else
            for (j = 0; j < m; j++) scanf("%lf", &DAT(i, j));
        #endif
        #ifdef _WIN32
            scanf_s("%lf%lf", &y[i], &wt[i]);
        #else
            scanf("%lf%lf", &y[i], &wt[i]);
        #endif
    }
    #ifdef _WIN32
        scanf_s("%*[\n"]
    #else
        scanf("%*[\n"]
    #endif
    }
else
{
    /* No weights supplied */
    for (i = 0; i < n; i++)
    {
        #ifdef _WIN32
            for (j = 0; j < m; j++) scanf_s("%lf", &DAT(i, j));
        #else
            for (j = 0; j < m; j++) scanf("%lf", &DAT(i, j));
        #endif
    }

Mark 25

---
g02qgc
/* Read in variable inclusion flags and calculate IP */
ip = (intcpt == Nag_Intercept) ? 1 : 0;
for (j = 0; j < m; j++)
{
    #ifdef _WIN32
    scanf_s("%" NAG_IFMT, &isx[j]);
    #else
    scanf("%" NAG_IFMT, &isx[j]);
    #endif
    if (isx[j] == 1) ip++;
}
/* Read in the quantiles required */
#ifdef _WIN32
    for (l = 0; l < ntau; l++) scanf_s("%lf", &tau[l]);
#else
    for (l = 0; l < ntau; l++) scanf("%lf", &tau[l]);
#endif
    /* Allocate memory for option arrays */
    if (!(opts = NAG_ALLOC(lopts, double)) ||
        !(iopts = NAG_ALLOC(liopts, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Initialize the optional argument array with nag_g02_opt_set (g02zkc) */
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_g02_opt_set (g02zkc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Read in any optional arguments. Reads in to the end of the input data, or until a blank line is reached */
for (;;)
{
    if (!fgets(optstr, LOPTSTR, stdin)) break;
    /* Left justify the option */
    poptstr = (optstr+strspn(optstr, " 
	"));
    /* Get the string length */
loptstr = strlen(poptstr);
if (poptstr[loptstr-1] == '\n')
{
    /* Remove any trailing line breaks */
    poptstr[loptstr] = '\setminus 0';
} else
{
    /* Clear the rest of the line */
#ifdef _WIN32
    scanf_s("%*[\n"];
#else
    scanf("%*[\n"];
#endif
}
/* Break if read in a blank line */
if (!*(poptstr)) break;
/* Set the supplied option (g02zkc) */
naq_g02_opt_set(optstr, iopts, liopts, opts, lopts, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from naq_g02_opt_set (g02zkc).\n",
            fail.message);
    exit_status = 1;
    goto END;
}
/* Allocate memory for the output arrays */
if (!(b = NAG_ALLOC(ip*ntau, double)) ||
    !(info = NAG_ALLOC(ntau, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Query optional arguments via naq_g02_opt_get (g02zlc) and calculate which
 * of the optional arrays are required and their sizes
 * ...
 * naq_g02_opt_get("INTERVAL METHOD", &ivalue, &rvalue, cvalue, lcvalue,
 *                  &optype, iopts, opts, &fail);
 if (fail.code != NE_NOERROR)
{
    printf("Error from naq_g02_opt_get (g02zlc).\n",
            fail.message);
    exit_status = 1;
    goto END;
}
#ifdef _WIN32
    strcpy_s(semeth, _countof(semeth), cvalue);
#else
    strcpy(semeth, cvalue);
#endif
if (strcmp(semeth, "NONE") != 0)
{
    /* Require the intervals to be output */
    if (!(bl = NAG_ALLOC(ip*ntau, double)) ||
        !(bu = NAG_ALLOC(ip*ntau, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Decide whether the state array is required, and initialise if it is */
    if (strcmp(semeth, "BOOTSTRAP XY") == 0)
/* Read in the generator ID and a seed */

#ifdef _WIN32
    scanf_s("%39s"NAG_IFMT"%"NAG_IFMT"%[\n] ", cgenid,
         _countof(cgenid), &subid, &seed[0]);
#else
    scanf("%39s"NAG_IFMT"%"NAG_IFMT"%[\n] ", cgenid, &subid, &seed[0]);
#endif

genid = (Nag_BaseRNG) nag_enum_name_to_value(cgenid);

/* Query the length of the state array (g05kfc) */
lstate = 0;
    nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate,
                     &fail);
if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_rand_init_repeatable (g05kfc).
               ", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Allocate memory to state */
    if (!(state = NAG_ALLOC(lstate, Integer)))
        {
            printf("Allocation failure\n")
            exit_status = -1;
            goto END;
        }

    /* Initialise the RNG (g05kfc) */
    nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate,
                     &fail);
    if (fail.code != NE_NOERROR)
        {
            printf("Error from nag_rand_init_repeatable (g05kfc).
                   ", fail.message);
            exit_status = 1;
            goto END;
        }

    /* Calculate the size of the covariance matrix, ch. */
    tdch = 0;
    nag_g02_opt_get("MATRIX RETURNED", &ivalue, &rvalue, cvalue, lcvalue,
                     &optype, iopts, opts, &fail);
    if (fail.code != NE_NOERROR)
        {
            printf("Error from nag_g02_opt_get (g02zlc).
                   ", fail.message);
            exit_status = 1;
            goto END;
        }

    if (strcmp(cvalue, "COVARIANCE") == 0)
        { 
            tdch = ntau;
        } else if (strcmp(cvalue, "H INVERSE") == 0)
        { 
            /* NB: If we are using bootstrap or IID errors then any request for 
            H INVERSE is ignored */
            if (strcmp(semeth, "BOOTSTRAP XY") != 0 && strcmp(semeth, "IID") != 0)
                tdch = ntau + 1;
        }
    if (tdch > 0)
        { 
            /* Need to allocate ch */
            if (!((ch = NAG_ALLOC(ip*ip*tdch, double)))
                { 
                    printf("Allocation failure\n");
                }
exit_status = -1;
goto END;
}

/* Calculate the size of the residual array, res */
exit_status = -1;
goto END;
}

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_g02_opt_get (g02zlc).\n%s\n", fail.message);
    exit_status = 1;
goto END;
}

if (strcmp(cvalue, "YES") == 0)
{
    /* Need to allocate res */
    if (!(res = NAG_ALLOC(n*ntau, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
goto END;
    }
}

/* ... 
* end of handling the optional arguments, and allocating optional arrays 
*/

/* Call the model fitting routine (nag_regsn_quant_linear (g02qgc)) */
exit_status = -1;
goto END;
}

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_regsn_quant_linear (g02qgc).\n%s\n", fail.message);
    if (fail.code == NW_POTENTIAL_PROBLEM)
    {
        printf("Additional error information: ");
        for (i = 0; i < ntau; i++)
            printf("%"NAG_IFMT", info[i]);
        printf("\n");
    }
    else
    {
        printf("Error from nag_regsn_quant_linear (g02qgc).\n%s\n", fail.message);
        exit_status = -1;
goto END;
    }
}

/* Display the parameter estimates */
for (l = 0; l < ntau; l++)
{
    printf(" Quantile: %6.3f\n\n", tau[l]);
    if (bl && bu)
    {
        printf(" Lower Parameter Upper\n");
        printf(" Limit Estimate Limit\n");
        for (j = 0; j < ip; j++)
            printf(" %"NAG_IFMT"%10.3f%10.3f%10.3f\n", j+1, bl[l*ip+j], b[l*ip+j], bu[l*ip+j]);
    }
    else
    {
        printf(" Parameter\n");
        printf(" Estimate\n");
        for (j = 0; j < ip; j++)
printf(" %3"NAG_IFMT "%10.3f\n", j+1, b[l*ip+j]);
}
printf("\n\n");
fflush(stdout);
if (ch)
{
  lc = l*ip*ip;
  if (tdch == ntau)
  {
    /* nag_gen_real_mat_print_comp (x04cbc).
     * Print real general matrix (comprehensive).
     */
    nag_gen_real_mat_print_comp(Nag_ColMajor, Nag_UpperMatrix,
      Nag_NoUnitDiag, ip, ip, &ch[lc], ip,
      "%9.2e", "Covariance matrix",
      Nag_NoLabels, 0, Nag_NoLabels, 0, 80,
      0, 0, &fail);
  }
  else
  {
    if (l == 0)
    {
      nag_gen_real_mat_print_comp(Nag_ColMajor, Nag_UpperMatrix,
        Nag_NoUnitDiag, ip, ip, ch, ip,
        "%9.2e", "J", Nag_NoLabels, 0,
        Nag_NoLabels, 0, 80, 0, 0, &fail);
      printf("\n");
    }
    lc = lc + ip*ip;
    nag_gen_real_mat_print_comp(Nag_ColMajor, Nag_UpperMatrix,
      Nag_NoUnitDiag, ip, ip, &ch[lc], ip,
      "%9.2e", "H inverse",
      Nag_NoLabels, 0, Nag_NoLabels, 0, 80,
      0, 0, &fail);
  }
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_gen_real_mat_print_comp (x04cbc).
%s
", fail.message);
    exit_status = 1;
    goto END;
  }
  printf("\n");
}
if (res)
{
  printf(" First 10 Residuals\n");
  fflush(stdout);
  /* set up column labels for matrix printer */
  #ifdef _WIN32
  for (l = 0; l < ntau; l++) sprintf_s(&clabs[10*l], 10, "%6.3f", tau[l]);
  #else
  for (l = 0; l < ntau; l++) sprintf(&clabs[10*l], "%6.3f", tau[l]);
  #endif
  for (l = 0; l < ntau; l++) clabsc[l] = &clabs[l*10];
  /* nag_gen_real_mat_print_comp (x04cbc).
   * Print real general matrix (comprehensive).
   */
  nag_gen_real_mat_print_comp(Nag_ColMajor, Nag_GeneralMatrix,
    Nag_NoUnitDiag, MIN(10, n), ntau, res, n,
    "%10.5f", "Quantile",
    Nag_IntegerLabels, NULL, Nag_CharacterLabels,
    (const char **) clabsc, 80, 2, NULL, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_gen_real_mat_print_comp (x04cbc).
%s
", fail.message);
    exit_status = 1;
    goto END;
  }
else
{
    printf(" Residuals not returned\n");
}

END:

NAG_FREE(info);
NAG_FREE(iopts);
NAG_FREE(isx);
NAG_FREE(state);
NAG_FREE(b);
NAG_FREE(bl);
NAG_FREE(bu);
NAG_FREE(ch);
NAG_FREE(opts);
NAG_FREE(res);
NAG_FREE(tau);
NAG_FREE(wt);
NAG_FREE(y);
NAG_FREE(cvalue);
NAG_FREE(clabs);
NAG_FREE(clabsc);

return(exit_status);

10.2 Program Data

nag_regsn_quant_linear (g02qgc) Example Program Data

Nag_ColMajor:: sorder
Nag_Intercept Nag_FALSE :: intcpt, weighted

235 1 5 :: n, m, ntau

420.1577 255.8394 800.7990 572.0807
541.4117 310.9587 1245.6964 907.3996
901.1575 1285.0486 1245.6964 255.8394
639.0802 363.4002 427.7975
750.8756 495.5608 649.9985
945.7989 633.7978 860.6002
829.3979 630.7566 1143.4211
979.1648 700.4409 2032.6792
1309.8789 830.9586 922.3548
502.8390 412.3613 889.9809
616.7168 495.5608 483.4800
790.9225 520.0006 696.2021
555.8786 452.4015 774.7962
713.4412 512.7201 390.5984
838.7561 658.8395 612.5619
535.0766 392.5995 708.7622
596.4408 443.5586 296.192
924.5619 640.1164 1536.0201
487.7583 333.8394 496.5976
692.6397 466.9587 503.3974
997.8770 543.3969 357.641
506.9995 317.7198 430.3376
654.1577 423.2783 357.6411
933.9193 518.9617 582.5413
433.6813 386.3602 580.2215
587.5962 419.6412 543.8807
896.4746 476.3209 588.6372
454.4782 386.3602 627.9999
584.9989 423.2783 1264.043
800.7990 503.3572 968.3949
502.4369 354.6389 482.5816
713.5197 497.3182 593.1694
906.0006 588.5195 1033.5658

Mark 25
Return Residuals = Yes
Matrix Returned = Covariance
Interval Method = IID

10.3 Program Results

nag_regrs_quant_linear (g02qgc) Example Program Results

Quantile: 0.100

<table>
<thead>
<tr>
<th>Lower Parameter</th>
<th>Upper Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit</td>
<td>Estimate</td>
</tr>
<tr>
<td>1 74.946</td>
<td>110.142</td>
</tr>
<tr>
<td>2 0.370</td>
<td>0.402</td>
</tr>
</tbody>
</table>

Covariance matrix

```
      3.19e+02  -2.54e-01
2.59e-04
```

Quantile: 0.250

<table>
<thead>
<tr>
<th>Lower Parameter</th>
<th>Upper Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10 0.25 0.50 0.75 0.90 : tau[1..ntau]</td>
<td>Return Residuals = Yes</td>
</tr>
<tr>
<td>Interval Method = IID</td>
<td></td>
</tr>
</tbody>
</table>

g02qgc.18 Mark 25
## g02 – Correlation and Regression Analysis

### Limit Estimate Limit

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64.232</td>
<td>95.483</td>
</tr>
<tr>
<td>2</td>
<td>0.446</td>
<td>0.474</td>
</tr>
</tbody>
</table>

### Covariance matrix

\[ 2.52e+02 -2.00e-01 \]
\[ 2.04e-04 \]

### Quantile: 0.500

<table>
<thead>
<tr>
<th>Lower Parameter Upper Limit</th>
<th>Estimate</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>55.399</td>
<td>81.482</td>
</tr>
<tr>
<td>2</td>
<td>0.537</td>
<td>0.560</td>
</tr>
</tbody>
</table>

### Covariance matrix

\[ 1.75e+02 -1.40e-01 \]
\[ 1.42e-04 \]

### Quantile: 0.750

<table>
<thead>
<tr>
<th>Lower Parameter Upper Limit</th>
<th>Estimate</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>41.372</td>
<td>62.396</td>
</tr>
<tr>
<td>2</td>
<td>0.625</td>
<td>0.644</td>
</tr>
</tbody>
</table>

### Covariance matrix

\[ 1.14e+02 -9.07e-02 \]
\[ 9.23e-05 \]

### Quantile: 0.900

<table>
<thead>
<tr>
<th>Lower Parameter Upper Limit</th>
<th>Estimate</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>26.829</td>
<td>67.351</td>
</tr>
<tr>
<td>2</td>
<td>0.650</td>
<td>0.686</td>
</tr>
</tbody>
</table>

### Covariance matrix

\[ 4.23e+02 -3.37e-01 \]
\[ 3.43e-04 \]

### First 10 Residuals

<table>
<thead>
<tr>
<th>Quantile</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.100</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

---

**Mark 25**
11 Algorithmic Details

By the addition of slack variables the minimization (1) can be reformulated into the linear programming problem

\[
\min_{(u,v) \in \mathbb{R}_+^{n}\times\mathbb{R}^d} \quad \tau e^T u + (1 - \tau)e^T v \\
\text{subject to} \quad y = X\beta + u - v
\]  

(2)

and its associated dual

\[
\max_d y^T d \quad \text{subject to} \quad X^T d = 0, \quad d \in [\tau - 1, \tau]^n
\]  

(3)

where \( e \) is a vector of \( n \) 1s. Setting \( a = d + (1 - \tau)e \) gives the equivalent formulation

\[
\max_a y^T a \quad \text{subject to} \quad X^T a = (1 - \tau)X^Te, \quad a \in [0, 1]^n.
\]  

(4)

The algorithm introduced by Portnoy and Koenker (1997) and used by \texttt{nag_regsn_quant_linear (g02qgc)}, uses the primal-dual formulation expressed in equations (2) and (4) along with a logarithmic barrier function to obtain estimates for \( \beta \). The algorithm is based on the predictor-corrector algorithm of Mehrotra (1992) and further details can be obtained from Portnoy and Koenker (1997) and Koenker (2005). A good description of linear programming, interior point algorithms, barrier functions and Mehrotra’s predictor-corrector algorithm can be found in Nocedal and Wright (1999).

11.1 Interior Point Algorithm

In this section a brief description of the interior point algorithm used to estimate the model parameters is presented. It should be noted that there are some differences in the equations given here – particularly (7) and (9) – compared to those given in Koenker (2005) and Portnoy and Koenker (1997).
11.1.1 Central path

Rather than optimize (4) directly, an additional slack variable \( s \) is added and the constraint \( a \in [0, 1]^n \) is replaced with \( a + s = e, a_i \geq 0, s_i \geq 0 \), for \( i = 1, 2, \ldots, n \).

The positivity constraint on \( a \) and \( s \) is handled using the logarithmic barrier function

\[
B(a, s, \mu) = y^T a + \mu \sum_{i=1}^n (\log a_i + \log s_i).
\]

The primal-dual form of the problem is used giving the Lagrangian

\[
L(a, s, \beta, u, \mu) = B(a, s, \mu) - \beta^T (X^T a - (1 - \tau)X^T e) - u^T (a + s - e)
\]

whose central path is described by the following first order conditions

\[
\begin{align*}
X^T a &= (1 - \tau)X^T e \\
a + s &= e \\
X\beta + u - v &= y \\
SUe &= \mu e \\
AV e &= \mu e 
\end{align*}
\]

where \( A \) denotes the diagonal matrix with diagonal elements given by \( a_i \), similarly with \( S, U \) and \( V \). By enforcing the inequalities on \( s \) and \( a \) strictly, i.e., \( a_i > 0 \) and \( s_i > 0 \) for all \( i \) we ensure that \( A \) and \( S \) are positive definite diagonal matrices and hence \( A^{-1} \) and \( S^{-1} \) exist.

Rather than applying Newton’s method to the system of equations given in (5) to obtain the step directions \( \hat{\gamma}_P, \hat{\gamma}_D, \hat{\gamma}_u, \hat{\gamma}_s, \hat{\gamma}_v \) and \( \hat{\gamma}_u \), Mehrotra substituted the steps directly into (5) giving the augmented system of equations

\[
\begin{align*}
X^T (a + \delta_a) &= (1 - \tau)X^T e \\
(a + \delta_u) + (s + \delta_s) &= e \\
X(\beta + \delta_\beta) + (u + \delta_u) - (v + \delta_v) &= y \\
(S + \Delta_s)(U + \Delta_u)e &= \mu e \\
(A + \Delta_a)(V + \Delta_v)e &= \mu e 
\end{align*}
\]

where \( \Delta_s, \Delta_a, \Delta_u \) and \( \Delta_v \) denote the diagonal matrices with diagonal elements given by \( \delta_u, \delta_s, \delta_u \) and \( \delta_v \), respectively.

11.1.2 Affine scaling step

The affine scaling step is constructed by setting \( \mu = 0 \) in (5) and applying Newton’s method to obtain an intermediate set of step directions

\[
\begin{align*}
(X^T WX)\delta_\beta &= X^T W(y - X\beta) + (\tau - 1)X^T e + X^T a \\
\delta_u &= W(y - X\beta - X\delta_\beta) \\
\delta_s &= -\delta_u \\
\delta_u &= S^{-1}U\delta_u - U e \\
\delta_v &= A^{-1}V\delta_s - V e 
\end{align*}
\]

where \( W = (S^{-1}U + A^{-1}V)^{-1} \).

Initial step sizes for the primal (\( \hat{\gamma}_P \)) and dual (\( \hat{\gamma}_D \)) parameters are constructed as

\[
\begin{align*}
\hat{\gamma}_P &= \sigma \min \left\{ \min_{i, \delta_u < 0} \{a_i/\delta_u\}, \min_{i, \delta_s < 0} \{s_i/\delta_s\} \right\} \\
\hat{\gamma}_D &= \sigma \min \left\{ \min_{i, \delta_u < 0} \{u_i/\delta_u\}, \min_{i, \delta_v < 0} \{v_i/\delta_v\} \right\}
\end{align*}
\]

where \( \sigma \) is a user-supplied scaling factor. If \( \hat{\gamma}_P \times \hat{\gamma}_D \geq 1 \) then the nonlinearity adjustment, described in Section 11.1.3, is not made and the model parameters are updated using the current step size and directions.
11.1.3 Nonlinearity Adjustment

In the nonlinearity adjustment step a new estimate of $\mu$ is obtained by letting

$$\hat{g}(\hat{\gamma}_p, \hat{\gamma}_D) = (s + \hat{\gamma}_p \delta_s) \underline{T}(u + \hat{\gamma}_D \delta_u) + (a + \hat{\gamma}_p \delta_a) \underline{T}(v + \hat{\gamma}_D \delta_v)$$

and estimating $\mu$ as

$$\mu = \left(\frac{\hat{g}(\hat{\gamma}_p, \hat{\gamma}_D)}{\hat{g}(0, 0)}\right)^2 \frac{\hat{g}(0, 0)}{2u}.$$  

This estimate, along with the nonlinear terms ($\Delta u$, $\Delta s$, $\Delta a$ and $\Delta v$) from (6) are calculated using the values of $\delta_a, \delta_s, \delta_u$ and $\delta_v$ obtained from the affine scaling step.

Given an updated estimate for $\mu$ and the nonlinear terms the system of equations

\[
\begin{align*}
(X^T W_XX)\delta_\beta &= X^T W(y - X\beta + \mu(S^{-1} - A^{-1})e + S^{-1} \Delta_a \underline{\Delta}_a e - A^{-1} \underline{\Delta}_a \underline{\Delta}_a e) + (\tau - 1)X^Te + X^Ta \\
\delta_a &= W(y - X\beta - X\delta_\beta + \mu(S^{-1} - A^{-1})) \\
\delta_s &= -\delta_\beta \\
\delta_u &= \mu S^{-1} e + S^{-1} U \delta_\beta - U e - S^{-1} \underline{\Delta}_a \underline{\Delta}_a e \\
\delta_v &= \mu A^{-1} e + A^{-1} V \delta_\beta - V e - A^{-1} \underline{\Delta}_a \underline{\Delta}_a e
\end{align*}
\]

are solved and updated values for $\delta_\beta, \delta_a, \delta_s, \delta_u, \delta_v, \hat{\gamma}_p$ and $\hat{\gamma}_D$ calculated.

11.1.4 Update and convergence

At each iteration the model parameters ($\beta, a, s, u, v$) are updated using step directions, ($\delta_\beta, \delta_a, \delta_s, \delta_u, \delta_v$) and step lengths ($\hat{\gamma}_p, \hat{\gamma}_D$).

Convergence is assessed using the duality gap, that is, the differences between the objective function in the primal and dual formulations. For any feasible point $(u, v, s, a)$ the duality gap can be calculated from equations (2) and (3) as

$$\tau e^Tu + (1 - \tau)e^Tv - d^Ty = \tau e^Tu + (1 - \tau)e^Tv - (a - (1 - \tau)e)^T y$$

and the optimization terminates if the duality gap is smaller than the tolerance supplied in the optional argument Tolerance.

11.1.5 Additional information

Initial values are required for the parameters $a, s, u, v$ and $\beta$. If not supplied by the user, initial values for $\beta$ are calculated from a least squares regression of $y$ on $X$. This regression is carried out by first constructing the cross-product matrix $X^T X$ and then using a pivoted QR decomposition as performed by nag_dgeqp3 (f08bfc). In addition, if the cross-product matrix is not of full rank, a rank reduction is carried out and, rather than using the full design matrix, $X$, a matrix formed from the first $p$-rank columns of $XP$ is used instead, where $P$ is the pivot matrix used during the QR decomposition. Parameter estimates, confidence intervals and the rows and columns of the matrices returned in the argument ch (if any) are set to zero for variables dropped during the rank-reduction. The rank reduction step is performed irrespective of whether initial values are supplied by the user.

Once initial values have been obtained for $\beta$, the initial values for $u$ and $v$ are calculated from the residuals. If $|r_i| < \epsilon_u$ then a value of $\pm \epsilon_u$ is used instead, where $\epsilon_u$ is supplied in the optional argument Epsilon. The initial values for the $a$ and $s$ are always set to $1 - \tau$ and $\tau$ respectively.

The solution for $\delta_\beta$ in both (7) and (9) is obtained using a Bunch–Kaufman decomposition, as implemented in nag_dsytrf (f07mdc).
11.2 Calculation of Covariance Matrix

nag_regsn_quant_linear (g02qgc) supplies four methods to calculate the covariance matrices associated with the parameter estimates for $\beta$. This section gives some additional detail on three of the algorithms, the fourth, (which uses bootstrapping), is described in Section 3.

(i) Independent, identically distributed (IID) errors

When assuming IID errors, the covariance matrices depend on the sparsity, $s(\tau)$, which nag_regsn_quant_linear (g02qgc) estimates as follows:

(a) Let $r_i$ denote the residuals from the original quantile regression, that is $r_i = y_i - \hat{x}_i^T \hat{\beta}$.
(b) Drop any residual where $|r_i| < \epsilon_u$, supplied in the optional argument Epsilon.
(c) Sort and relabel the remaining residuals in ascending order, by absolute value, so that $\epsilon_u < r_1 < r_2 < \ldots$.
(d) Select the first $l$ values where $l = h_n n$, for some bandwidth $h_n$.
(e) Sort and relabel these $l$ residuals again, so that $r_1 < r_2 < \ldots < r_l$ and regress them against a design matrix with two columns ($p = 2$) and rows given by $x_i = \{1, i/(n-p)\}$ using quantile regression with $\tau = 0.5$.
(f) Use the resulting estimate of the slope as an estimate of the sparsity.

(ii) Powell Sandwich

When using the Powell Sandwich to estimate the matrix $H_n$, the quantity

$$c_n = \min(\sigma_r, (q_{0.95} - q_{0.05})/1.34) \times \left(\Phi^{-1}(\tau + h_n) - \Phi^{-1}(\tau - h_n)\right)$$

is calculated. Dependent on the value of $\tau$ and the method used to calculate the bandwidth ($h_n$), it is possible for the quantities $\tau \pm h_n$ to be too large or small, compared to machine precision ($\epsilon$). More specifically, when $\tau - h_n \leq \sqrt{\epsilon}$, or $\tau + h_n \geq 1 - \sqrt{\epsilon}$, a warning flag is raised in info, the value is truncated to $\sqrt{\epsilon}$ or $1 - \sqrt{\epsilon}$ respectively and the covariance matrix calculated as usual.

(iii) Hendricks–Koenker Sandwich

The Hendricks–Koenker Sandwich requires the calculation of the quantity $d_i = x_i^T \left(\hat{\beta}(\tau + h_n) - \hat{\beta}(\tau - h_n)\right)$. As with the Powell Sandwich, in cases where $\tau - h_n \leq \sqrt{\epsilon}$, or $\tau + h_n \geq 1 - \sqrt{\epsilon}$, a warning flag is raised in info, the value truncated to $\sqrt{\epsilon}$ or $1 - \sqrt{\epsilon}$ respectively and the covariance matrix calculated as usual.

In addition, it is required that $d_i > 0$, in this method. Hence, instead of using $2h_n/d_i$ in the calculation of $H_n$, $\max(2h_n/(d_i + \epsilon_u), 0)$ is used instead, where $\epsilon_u$ is supplied in the optional argument Epsilon.

12 Optional Arguments

Several optional arguments in nag_regsn_quant_linear (g02qgc) control aspects of the optimization algorithm, methodology used, logic or output. Their values are contained in the arrays iopts and opts; these must be initialized before calling nag_regsn_quant_linear (g02qgc) by first calling nag_g02_opt_set (g02zkc) with optstr set to Initialize = g02qgc.

Each optional argument has an associated default value; to set any of them to a non-default value, use nag_g02_opt_set (g02zkc). The current value of an optional argument can be queried using nag_g02_opt_get (g02zlc).

The remainder of this section can be skipped if you wish to use the default values for all optional arguments.

The following is a list of the optional arguments available. A full description of each optional argument is provided in Section 12.1.

**Band Width Alpha**

**Band Width Method**
12.1 Description of the Optional Arguments

For each option, we give a summary line, a description of the optional argument and details of constraints.

The summary line contains:

- the keywords, where the minimum abbreviation of each keyword is underlined (if no characters of an optional qualifier are underlined, the qualifier may be omitted);
- a parameter value, where the letters \(a\), \(i\) and \(r\) denote options that take character, integer and real values respectively;
- the default value, where the symbol \(\varepsilon\) is a generic notation for machine precision (see \texttt{nag\_machine\_precision (X02AJC)}).

Keywords and character values are case and white space insensitive.

\begin{itemize}
  \item \textbf{Band Width Alpha} \( r \) \hspace{1cm} \text{Default } = 1.0 \\
  A multiplier used to construct the parameter \( \alpha_b \) used when calculating the Sheather–Hall bandwidth (see Section 3), with \( \alpha_b = (1 - \alpha) \times \text{Band Width Alpha} \). Here, \( \alpha \) is the \textbf{Significance Level}.
  \textbf{Constraint}: Band Width Alpha \( > 0.0 \).

  \item \textbf{Band Width Method} \( a \) \hspace{1cm} \text{Default } = '\text{SHEATHER HALL}'
  The method used to calculate the bandwidth used in the calculation of the asymptotic covariance matrix \( \Sigma \) and \( H^{-1} \) if \textbf{Interval Method} = HKS, KERNEL or IID (see Section 3).
  \textbf{Constraint}: Band Width Method = \text{SHEATHER HALL} or \text{BOFINGER}.

  \item \textbf{Big} \hspace{1cm} \text{Default } = 10.0^{20}
  This argument should be set to something larger than the biggest value supplied in \texttt{dat} and \texttt{y}.
  \textbf{Constraint}: Big \( > 0.0 \).
\end{itemize}
Bootstrap Interval Method

If Interval Method = BOOTSTRAP XY, Bootstrap Interval Method controls how the confidence intervals are calculated from the bootstrap estimates.

Bootstrap Interval Method = T

t intervals are calculated. That is, the covariance matrix, \( \Sigma = \{\sigma_{ij} : i, j = 1, 2, \ldots, p\} \) is calculated from the bootstrap estimates and the limits calculated as \( \beta_i \pm t_{(n-p, (1+\alpha)/2)} \sigma_{ii} \) where \( t_{(n-p, (1+\alpha)/2)} \) is the \((1 + \alpha)/2 \) percentage point from a Student’s t distribution on \( n - p \) degrees of freedom, \( n \) is the effective number of observations and \( \alpha \) is given by the optional argument Significance Level.

Bootstrap Interval Method = QUANTILE

Quantile intervals are calculated. That is, the upper and lower limits are taken as the \( (1+\alpha)/2 \) and \( (1-\alpha)/2 \) quantiles of the bootstrap estimates, as calculated using nag_double_quants (g01amc).

Constraint: Bootstrap Interval Method = T or QUANTILE.

Bootstrap Iterations

The number of bootstrap samples used to calculate the confidence limits and covariance matrix (if requested) when Interval Method = BOOTSTRAP XY.

Constraint: Bootstrap Iterations > 1.

Bootstrap Monitoring

If Bootstrap Monitoring = YES and Interval Method = BOOTSTRAP XY, then the parameter estimates for each of the bootstrap samples are displayed. This information is sent to the unit number specified by Unit Number.

Constraint: Bootstrap Monitoring = YES or NO.

Calculate Initial Values

If Calculate Initial Values = YES then the initial values for the regression parameters, \( \beta \), are calculated from the data. Otherwise they must be supplied in \( b \).

Constraint: Calculate Initial Values = YES or NO.

Defaults

This special keyword is used to reset all optional arguments to their default values.

Drop Zero Weights

If a weighted regression is being performed and Drop Zero Weights = YES then observations with zero weight are dropped from the analysis. Otherwise such observations are included.

Constraint: Drop Zero Weights = YES or NO.

Epsilon

\( \epsilon_{\text{u}} \), the tolerance used when calculating the covariance matrix and the initial values for \( u \) and \( v \). For additional details see Section 11.2 and Section 11.1.5 respectively.

Constraint: Epsilon \( \geq 0.0 \).

Interval Method

The value of Interval Method controls whether confidence limits are returned in \( bl \) and \( bu \) and how these limits are calculated. This argument also controls how the matrices returned in \( ch \) are calculated.

Interval Method = NONE

No limits are calculated and \( bl \), \( bu \) and \( ch \) are not referenced.
**Interval Method**

- **KERNEL**
  
  The Powell Sandwich method with a Gaussian kernel is used.

- **HKS**
  
  The Hendricks–Koenker Sandwich is used.

- **IID**
  
  The errors are assumed to be identical, and independently distributed.

- **BOOTSTRAP XY**
  
  A bootstrap method is used, where sampling is done on the pair \((y_i, x_i)\). The number of bootstrap samples is controlled by the argument **Bootstrap Iterations** and the type of interval constructed from the bootstrap samples is controlled by **Bootstrap Interval Method**.

**Constraint:** **Interval Method** = NONE, KERNEL, HKS, IID or BOOTSTRAP XY.

**Iteration Limit**

- **Default** = 100

The maximum number of iterations to be performed by the interior point optimization algorithm.

**Constraint:** **Iteration Limit** > 0.

**Matrix Returned**

- **Default** = NONE

The value of **Matrix Returned** controls the type of matrices returned in \( \text{ch} \). If **Interval Method** = NONE, this argument is ignored and \( \text{ch} \) is not referenced. Otherwise:

- **Matrix Returned** = NONE
  
  No matrices are returned and \( \text{ch} \) is not referenced.

- **Matrix Returned** = COVARIANCE
  
  The covariance matrices are returned.

- **Matrix Returned** = H INVERSE
  
  If **Interval Method** = KERNEL or HKS, the matrices \( J \) and \( H^{-1} \) are returned. Otherwise no matrices are returned and \( \text{ch} \) is not referenced.

The matrices returned are calculated as described in Section 3, with the algorithm used specified by **Interval Method**. In the case of **Interval Method** = BOOTSTRAP XY the covariance matrix is calculated directly from the bootstrap estimates.

**Constraint:** **Matrix Returned** = NONE, COVARIANCE or H INVERSE.

**Monitoring**

- **Default** = NO

If **Monitoring** = YES then the duality gap is displayed at each iteration of the interior point optimization algorithm. In addition, the final estimates for \( \beta \) are also displayed.

The monitoring information is sent to the unit number specified by **Unit Number**.

**Constraint:** **Monitoring** = YES or NO.

**QR Tolerance**

- **Default** = \( \varepsilon^{0.9} \)

The tolerance used to calculate the rank, \( k \), of the \( p \times p \) cross-product matrix, \( X^T X \). Letting \( Q \) be the orthogonal matrix obtained from a QR decomposition of \( X^T X \), then the rank is calculated by comparing \( Q_{ii} \) with \( Q_{11} \times \text{QR Tolerance} \).

If the cross-product matrix is rank deficient, then the parameter estimates for the \( p - k \) columns with the smallest values of \( Q_{ii} \) are set to zero, along with the corresponding entries in \( \text{bl}, \text{bu} \) and \( \text{ch} \), if returned. This is equivalent to dropping these variables from the model. Details on the QR decomposition used can be found in nag_dgeqp3 (f08bfc).

**Constraint:** **QR Tolerance** > 0.0.
Return Residuals $a$ Default = NO

If Return Residuals = YES, the residuals are returned in res. Otherwise res is not referenced.

Constraint: Return Residuals = YES or NO.

Sigma $r$ Default = 0.99995

The scaling factor used when calculating the affine scaling step size (see equation (8)).

Constraint: 0.0 < Sigma < 1.0.

Significance Level $r$ Default = 0.95

$\alpha$, the size of the confidence interval whose limits are returned in bl and bu.

Constraint: 0.0 < Significance Level < 1.0.

Tolerance $r$ Default = $\sqrt{\epsilon}$

Convergence tolerance. The optimization is deemed to have converged if the duality gap is less than Tolerance (see Section 11.1.4).

Constraint: Tolerance > 0.0.

Unit Number $i$ Output sent to stdout

The unit number to which any monitoring information is sent. See nag_open_file (x04acc) for details on how to assign a file to a unit number. If no unit number is specified then any monitoring information will be sent to standard output (stdout).

Constraint: Unit Number > 1.

13 Description of Monitoring Information

See the description of the optional argument Monitoring.