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

```
#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 ith row of X and a quantile $\tau \in (0, 1)$, nag_regsn_quant_linear (g02qgc) estimates the p-element vector β as the solution to

$$\underset{\beta \in \mathbb{R}^p}{\text{minimize}} \sum_{i=1}^{n} \rho_\tau \left(y_i - x_i^{\mathrm{T}} \beta \right) \tag{1}$$

where ρ_τ 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 τ.

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 $J_n = n^{-1} \sum_{i=1}^{n} x_i x_i^{\mathrm{T}}$ and H_n is a function of τ, as described below. Given an estimate of the covariance matrix, $\hat{\Sigma}$, 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

$$\hat{\beta}_{Li} = \hat{\beta}_i - t_{n-p,(1+\alpha)/2}\sqrt{\hat{\Sigma}_{ii}}, \hat{\beta}_{Ui} = \hat{\beta}_i + t_{n-p,(1+\alpha)/2}\sqrt{\hat{\Sigma}_{ii}}$$

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 $X^{\mathrm{T}}X$.

Four methods for estimating the covariance matrix, Σ, are available:

(i) Independent, identically distributed (IID) errors

Under an assumption of IID errors the asymptotic relationship for Σ simplifies to

$$\Sigma = \frac{\tau(1-\tau)}{n}(s(\tau))^2 \left(X^{\mathrm{T}}X\right)^{-1}$$

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

(ii) Powell Sandwich

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

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

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

$$c_n = \min(\sigma_r, (q_{r3} - q_{r1})/1.34) \times \left(\Phi^{-1}(\tau + h_n) - \Phi^{-1}(\tau - h_n)\right)$$

where h_n is a bandwidth, σ_r, q_{r1} and q_{r3} 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

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

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, β. 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

$$h_n = \left(\frac{1.5 \left(\Phi^{-1} \left(\alpha_b \right) \phi \left(\Phi^{-1} \left(\tau \right) \right) \right)^2}{n \left(2 \Phi^{-1} \left(\tau \right) + 1 \right)} \right)^{\frac{1}{3}}$$

for a user-supplied value α_b,

(ii) Bofinger

$$h_n = \left(\frac{4.5 \left(\phi \left(\Phi^{-1} \left(\tau \right) \right) \right)^4}{n \left(2 \Phi^{-1} \left(\tau \right) + 1 \right)^2} \right)^{\frac{1}{5}}$$

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

Koenker R (2005) *Quantile Regression* Econometric Society Monographs, Cambridge University Press, New York

Mehrotra S (1992) On the implementation of a primal-dual interior point method *SIAM J. Optim.* **2** 575–601

Nocedal J and Wright S J (1999) *Numerical Optimization* Springer Series in Operations Research, Springer, New York

Portnoy S and Koenker R (1997) The Gaussian hare and the Laplacian tortoise: computability of squared-error versus absolute error estimators *Statistical Science* **4** 279–300

Powell J L (1991) Estimation of monotonic regression models under quantile restrictions *Nonparametric and Semiparametric Methods in Econometrics* Cambridge University Press, Cambridge

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **intcpt** – Nag_IncludeIntercept *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.

intcpt = Nag_Intercept
An intercept will be included in the model.

intcpt = Nag_NoIntercept
An intercept will not be included in the model.

Constraint: **intcpt** = Nag_NoIntercept or Nag_Intercept.

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 $\mathbf{n} = n$. Otherwise $\mathbf{n} = n +$ the number of observations with zero weights.

Constraint: $\mathbf{n} \geq 2$.

4: **m** – Integer *Input*

On entry: m, the total number of variates in the dataset.

Constraint: $\mathbf{m} \geq 0$.

5: **dat**$[\mathit{dim}]$ – const double *Input*

Note: the dimension, *dim*, of the array **dat** must be at least

$\mathbf{pddat} \times \mathbf{m}$ when $\mathbf{order} = \text{Nag\_ColMajor}$;
$\mathbf{pddat} \times \mathbf{n}$ when $\mathbf{order} = \text{Nag\_RowMajor}$.

Where $\mathbf{DAT}(i, j)$ appears in this document, it refers to the array element

$\mathbf{dat}[(j-1) \times \mathbf{pddat} + i - 1]$ when $\mathbf{order} = \text{Nag\_ColMajor}$;
$\mathbf{dat}[(i-1) \times \mathbf{pddat} + j - 1]$ when $\mathbf{order} = \text{Nag\_RowMajor}$.

On entry: the ith value for the jth variate, for $i = 1, 2, \ldots, \mathbf{n}$ and $j = 1, 2, \ldots, \mathbf{m}$, must be supplied in $\mathbf{DAT}(i, j)$

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:

if $\mathbf{order} = \text{Nag\_ColMajor}$, $\mathbf{pddat} \geq \mathbf{n}$;
otherwise $\mathbf{pddat} \geq \mathbf{m}$.

7: **isx**$[\mathbf{m}]$ – const Integer *Input*

On entry: indicates which independent variables are to be included in the model.

$\mathbf{isx}[j-1] = 0$
The jth variate, supplied in **dat**, is not included in the regression model.

$\mathbf{isx}[j-1] = 1$
The jth variate, supplied in **dat**, is included in the regression model.

Constraints:

$\mathbf{isx}[j-1] = 0$ or 1, for $j = 1, 2, \ldots, \mathbf{m}$;
if $\mathbf{intcpt} = \text{Nag\_Intercept}$, exactly $\mathbf{ip} - 1$ values of **isx** must be set to 1;
if $\mathbf{intcpt} = \text{Nag\_NoIntercept}$, 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:

$1 \leq \mathbf{ip} < \mathbf{n}$;
if $\mathbf{intcpt} = \text{Nag\_Intercept}$, $1 \leq \mathbf{ip} \leq \mathbf{m} + 1$;
if $\mathbf{intcpt} = \text{Nag\_NoIntercept}$, $1 \leq \mathbf{ip} \leq \mathbf{m}$.

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 **NULL**.

When

Drop Zero Weights = YES
: If **wt**$[i-1] = 0.0$, the ith 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 ≥ 2;
wt$[i] = 0.0$, for all i.

11: **ntau** – Integer *Input*

On entry: the number of quantiles of interest.

Constraint: **ntau** ≥ 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 ϵ is the ***machine precision*** returned by nag_machine_precision (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 $X^{\mathrm{T}}X$.

14: **b**[**ip** × **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 **intcpt** = Nag_Intercept, **B**$(1, l)$ will contain the estimate corresponding to the intercept and **B**$(i+1, l)$ will contain the coefficient of the jth variate contained in **dat**, where **isx**$[j-1]$ is the ith nonzero value in the array **isx**.

If **intcpt** = Nag_NoIntercept, **B**(i, l) will contain the coefficient of the jth variate contained in **dat**, where **isx**$[j-1]$ is the ith 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, α, 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, α 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

if **Interval Method** $\neq$ NONE and **Matrix Returned** = COVARIANCE, **ip** $\times$ **ip** $\times$ **ntau**;
if **Interval Method** $\neq$ NONE, IID or BOOTSTRAP XY and
Matrix Returned = 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** $\times$ **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, Σ, 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**:

If **Matrix Returned** = COVARIANCE, **CH**(i, j, l) holds an estimate of the covariance between **B**(i, l) and **B**(j, l).

If **Matrix Returned** = H INVERSE, **CH**$(i, j, 1)$ holds an estimate of the (i, j)th element of nJ_n and **CH**$(i, j, l + 1)$ holds an estimate of the (i, j)th element of $n^{-1}H_n^{-1}$, for $\tau =$ **tau**$[l - 1]$.

The method used for calculating Σ 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: **iopts**[dim] – const Integer *Communication Array*

Note: the dimension, 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 **iopts** in the previous call to nag_g02_opt_set (g02zkc).

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

20: **opts**[dim] – const double *Communication Array*

Note: the dimension, 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 **opts** in the previous call to nag_g02_opt_set (g02zkc).

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

21: **state**[dim] – Integer *Communication Array*

Note: the dimension, 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 **state** in the previous call to nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc).

If **Interval Method** = BOOTSTRAP XY, **state** contains information about the selected random number generator. Otherwise **state** is not referenced and can be set to **NULL**.

22: **info**[**ntau**] – Integer *Output*

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

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 **Iteration Limit** whilst calculating the parameter estimates or relaxing the definition of convergence by increasing **Tolerance**.
2	A singular matrix was encountered during the optimization. The model was not fitted for this value of τ.
4	Some truncation occurred whilst calculating the confidence limits for this value of τ. 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 **Iteration Limit**.
16	Confidence limits for this value of τ 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 **Big**.

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

23: **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** $= \langle value \rangle$ and **m** $= \langle value \rangle$.
Constraint: **pddat** $\geq$ **m**.

On entry, **pddat** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pddat** $\geq$ **n**.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ 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** $= \langle value \rangle$.
Constraint: **m** ≥ 0.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 2.

On entry, **ntau** $= \langle value \rangle$.
Constraint: **ntau** ≥ 1.

NE_INT_2

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

NE_INT_ARRAY

On entry, **isx**$[\langle value \rangle] = \langle value \rangle$.
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 **intcpt**: **ip** $= \langle value \rangle$, expected value $= \langle value \rangle$.

NE_NEG_WEIGHT

On entry, **wt**$[\langle value \rangle] = \langle value \rangle$.
Constraint: **wt**$[i] \geq 0.0$ 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 ≥ ⟨*value*⟩.

NE_REAL_ARRAY

On entry, **tau**[⟨*value*⟩] = ⟨*value*⟩ 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

nag_regsn_quant_linear (g02qgc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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

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 **Interval Method** = BOOTSTRAP XY then a further np elements are required, and this increases by $p \times$ **ntau** × **Bootstrap Iterations** if **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 **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 τ 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("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read in the problem size */
#ifdef _WIN32
  scanf_s("%39s%*[^\n]", corder, _countof(corder));
#else
  scanf("%39s%*[^\n]", corder);
#endif
#ifdef _WIN32
  scanf_s("%39s%39s%*[^\n]", cintcpt, _countof(cintcpt), cweighted,
```

```
          _countof(cweighted));
#else
  scanf("%39s%39s%*[^\n]", cintcpt, cweighted);
#endif
#ifdef _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;

  /* Allocate memory for input arrays */
  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
#ifdef _WIN32
          scanf_s("%lf", &y[i]);
#else
          scanf("%lf", &y[i]);
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }

  /* 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++;
    }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* 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
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#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) */
  nag_g02_opt_set("INITIALIZE = G02QG", iopts, liopts, opts, lopts, &fail);
  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, " \n\t"));
      /* 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) */
      nag_g02_opt_set(optstr, iopts, liopts, opts, lopts, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_g02_opt_set (g02zkc).\n%s\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 nag_g02_opt_get (g02zlc) and calculate which
   * of the optional arrays are required and their sizes
   * ...
   */
  nag_g02_opt_get("INTERVAL METHOD", &ivalue, &rvalue, cvalue, lcvalue,
                  &optype, iopts, opts, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_g02_opt_get (g02zlc).\n%s\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).\n%s\n",
                     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).\n%s\n",
                     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).\n%s\n",
                 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 */
      nag_g02_opt_get("RETURN RESIDUALS", &ivalue, &rvalue, cvalue, lcvalue,
                      &optype, iopts, opts, &fail);
      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)) */
  nag_regsn_quant_linear(order, intcpt, n, m, dat, pddat, isx, ip, y, wt, ntau,
                         tau, &df, b, bl, bu, ch, res, iopts, opts, state,
                         info, &fail);
  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(" %3"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_NonUnitDiag, 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_NonUnitDiag, 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_NonUnitDiag, 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).\n%s\n",
                       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_NonUnitDiag, 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).\n%s\n",
                 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(dat);
  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

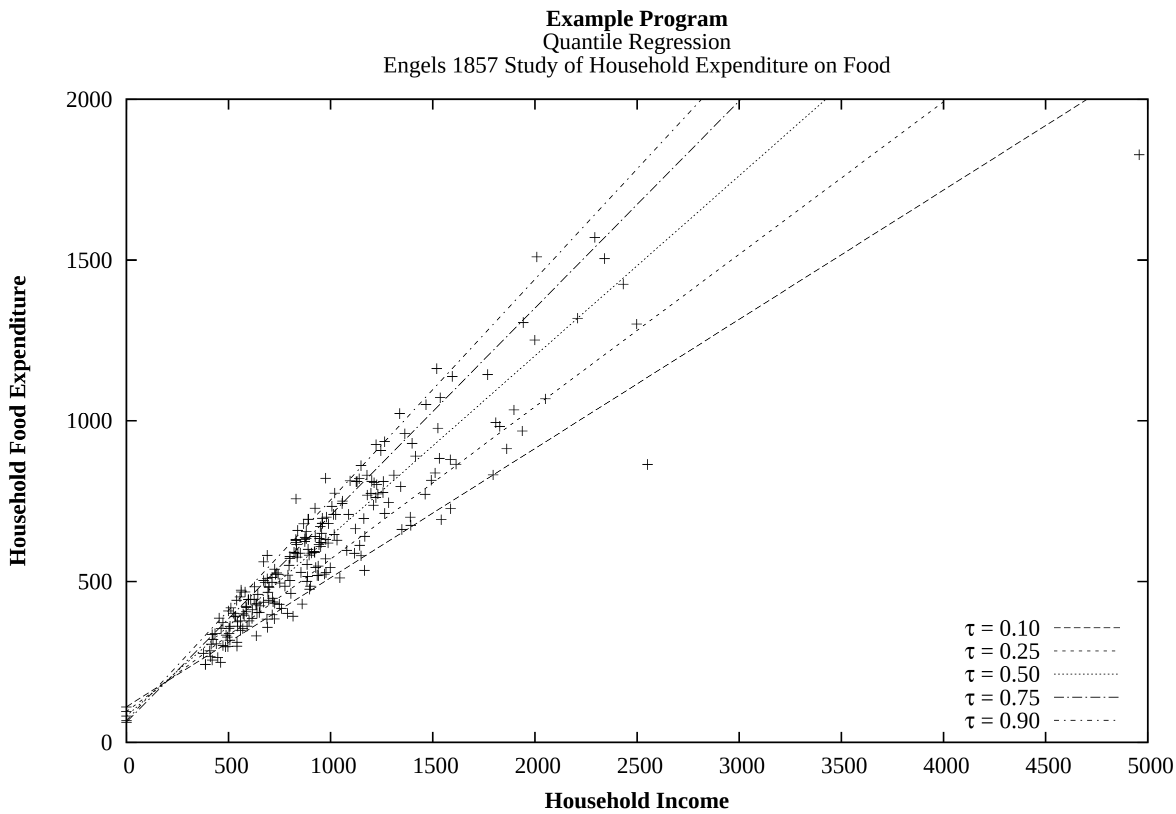
```
nag_regsn_quant_linear (g02qgc) Example Program Data
Nag_ColMajor                                                 :: sorder
Nag_Intercept    Nag_FALSE                                   :: intcpt, weighted
 235                      1                      5           :: n, m, ntau
 420.1577  255.8394      800.7990  572.0807      643.3571  459.8177
 541.4117  310.9587     1245.6964  907.3969     2551.6615  863.9199
 901.1575  485.6800     1201.0002  811.5776     1795.3226  831.4407
 639.0802  402.9974      634.4002  427.7975     1165.7734  534.7610
 750.8756  495.5608      956.2315  649.9985      815.6212  392.0502
 945.7989  633.7978     1148.6010  860.6002     1264.2066  934.9752
 829.3979  630.7566     1768.8236 1143.4211     1095.4056  813.3081
 979.1648  700.4409     2822.5330 2032.6792      447.4479  263.7100
1309.8789  830.9586      922.3548  590.6183     1178.9742  769.0838
1492.3987  815.3602     2293.1920 1570.3911      975.8023  630.5863
 502.8390  338.0014      627.4726  483.4800     1017.8522  645.9874
 616.7168  412.3613      889.9809  600.4804      423.8798  319.5584
 790.9225  520.0006     1162.2000  696.2021      558.7767  348.4518
 555.8786  452.4015     1197.0794  774.7962      943.2487  614.5068
 713.4412  512.7201      530.7972  390.5984     1348.3002  662.0096
 838.7561  658.8395     1142.1526  612.5619     2340.6174 1504.3708
 535.0766  392.5995     1088.0039  708.7622      587.1792  406.2180
 596.4408  443.5586      484.6612  296.9192     1540.9741  692.1689
 924.5619  640.1164     1536.0201 1071.4627     1115.8481  588.1371
 487.7583  333.8394      678.8974  496.5976     1044.6843  511.2609
 692.6397  466.9583      671.8802  503.3974     1389.7929  700.5600
 997.8770  543.3969      690.4683  357.6411     2497.7860 1301.1451
 506.9995  317.7198      860.6948  430.3376     1585.3809  879.0660
 654.1587  424.3209      873.3095  624.6990     1862.0438  912.8851
 933.9193  518.9617      894.4598  582.5413     2008.8546 1509.7812
 433.6813  338.0014     1148.6470  580.2215      697.3099  484.0605
 587.5962  419.6412      926.8762  543.8807      571.2517  399.6703
 896.4746  476.3200      839.0414  588.6372      598.3465  444.1001
 454.4782  386.3602      829.4974  627.9999      461.0977  248.8101
 584.9989  423.2783     1264.0043  712.1012      977.1107  527.8014
 800.7990  503.3572     1937.9771  968.3949      883.9849  500.6313
 502.4369  354.6389      698.8317  482.5816      718.3594  436.8107
 713.5197  497.3182      920.4199  593.1694      543.8971  374.7990
 906.0006  588.5195     1897.5711 1033.5658     1587.3480  726.3921
```

```
 880.5969  654.5971      891.6824  693.6795     4957.8130 1827.2000
 796.8289  550.7274      889.6784  693.6795      969.6838  523.4911
 854.8791  528.3770     1221.4818  761.2791      419.9980  334.9998
1167.3716  640.4813      544.5991  361.3981      561.9990  473.2009
 523.8000  401.3204     1031.4491  628.4522      689.5988  581.2029
 670.7792  435.9990     1462.9497  771.4486     1398.5203  929.7540
 377.0584  276.5606      830.4353  757.1187      820.8168  591.1974
 851.5430  588.3488      975.0415  821.5970      875.1716  637.5483
1121.0937  664.1978     1337.9983 1022.3202     1392.4499  674.9509
 625.5179  444.8602      867.6427  679.4407     1256.3174  776.7589
 805.5377  462.8995      725.7459  538.7491     1362.8590  959.5170
 558.5812  377.7792      989.0056  679.9981     1999.2552 1250.9643
 884.4005  553.1504     1525.0005  977.0033     1209.4730  737.8201
1257.4989  810.8962      672.1960  561.2015     1125.0356  810.6772
2051.1789 1067.9541      923.3977  728.3997     1827.4010  983.0009
1466.3330 1049.8788      472.3215  372.3186     1014.1540  708.8968
 730.0989  522.7012      590.7601  361.5210      880.3944  633.1200
2432.3910 1424.8047      831.7983  620.8006      873.7375  631.7982
 940.9218  517.9196     1139.4945  819.9964      951.4432  608.6419
1177.8547  830.9586      507.5169  360.8780      473.0022  300.9999
1222.5939  925.5795      576.1972  395.7608      601.0030  377.9984
1519.5811 1162.0024      696.5991  442.0001      713.9979  397.0015
 687.6638  383.4580      650.8180  404.0384      829.2984  588.5195
 953.1192  621.1173      949.5802  670.7993      959.7953  681.7616
 953.1192  621.1173      497.1193  297.5702     1212.9613  807.3603
 953.1192  621.1173      570.1674  353.4882      958.8743  696.8011
 939.0418  548.6002      724.7306  383.9376     1129.4431  811.1962
1283.4025  745.2353      408.3399  284.8008     1943.0419 1305.7201
1511.5789  837.8005      638.6713  431.1000      539.6388  442.0001
1342.5821  795.3402     1225.7890  801.3518      463.5990  353.6013
 511.7980  418.5976      715.3701  448.4513      562.6400  468.0008
 689.7988  508.7974      800.4708  577.9111      736.7584  526.7573
1532.3074  883.2780      975.5974  570.5210     1415.4461  890.2390
1056.0808  742.5276     1613.7565  865.3205     2208.7897 1318.8033
 387.3195  242.3202      608.5019  444.5578      636.0009  331.0005
 387.3195  242.3202      958.6634  680.4198      759.4010  416.4015
 410.9987  266.0010      835.9426  576.2779     1078.8382  596.8406
 499.7510  408.4992     1024.8177  708.4787      748.6413  429.0399
 832.7554  614.7588     1006.4353  734.2356      987.6417  619.6408
 614.9986  385.3184      726.0000  433.0010      788.0961  400.7990
 887.4658  515.6200      494.4174  327.4188     1020.0225  775.0209
1595.1611 1138.1620      776.5958  485.5198     1230.9235  772.7611
1807.9520  993.9630      415.4407  305.4390      440.5174  306.5191
 541.2006  299.1993      581.3599  468.0008      743.0772  522.6019
1057.6767  750.3202      :: (x[1..m],y)[i] for i = 0...n-1
1                        :: isx[1..m]
0.10 0.25 0.50 0.75 0.90 :: tau[1..ntau]
Return Residuals = Yes
Matrix Returned = Covariance
Interval Method = IID
```

10.3 Program Results

```
nag_regsn_quant_linear (g02qgc) Example Program Results

 Quantile:  0.100

        Lower   Parameter   Upper
        Limit   Estimate    Limit
   1    74.946   110.142   145.337
   2     0.370     0.402     0.433


 Covariance matrix
   3.19e+02 -2.54e-01
             2.59e-04

 Quantile:  0.250

        Lower   Parameter   Upper
```

```
       Limit   Estimate    Limit
  1    64.232    95.483   126.735
  2     0.446     0.474     0.502


Covariance matrix
  2.52e+02 -2.00e-01
            2.04e-04

Quantile:  0.500

       Lower   Parameter   Upper
       Limit   Estimate    Limit
  1    55.399    81.482   107.566
  2     0.537     0.560     0.584


Covariance matrix
  1.75e+02 -1.40e-01
            1.42e-04

Quantile:  0.750

       Lower   Parameter   Upper
       Limit   Estimate    Limit
  1    41.372    62.396    83.421
  2     0.625     0.644     0.663


Covariance matrix
  1.14e+02 -9.07e-02
            9.23e-05

Quantile:  0.900

       Lower   Parameter   Upper
       Limit   Estimate    Limit
  1    26.829    67.351   107.873
  2     0.650     0.686     0.723


Covariance matrix
  4.23e+02 -3.37e-01
            3.43e-04

First 10 Residuals
                                Quantile
          0.100      0.250      0.500      0.750      0.900
   1  -23.10718  -38.84219  -61.00711  -77.14462  -99.86551
   2  140.20549   96.93582   42.00636   -6.04177  -44.85812
   3   91.19725   59.31654   17.93924  -16.90993  -49.06884
   4  -16.70358  -41.20981  -73.81193 -100.11463 -127.96277
   5  296.77717  221.32470  128.09970   42.75414  -14.87476
   6 -271.39185 -441.31464 -646.95350 -841.78309 -954.63488
   7   13.48419  -37.04518 -100.61322 -157.07478 -200.13481
   8  218.91527  146.69601   57.31834  -24.28017  -80.01908
   9    0.00000 -115.21109 -255.74639 -387.16920 -468.03911
  10   36.09526    4.52393  -36.48522  -70.97584 -102.95390
```



11 Algorithmic Details

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

$$\underset{(u,v,\beta)\in\mathbb{R}^n_+\times\mathbb{R}^n_+\times\mathbb{R}^p}{\text{minimize}} \tau e^{\mathrm{T}}u + (1-\tau)e^{\mathrm{T}}v \qquad \text{subject to} \qquad y = X\beta + u - v \tag{2}$$

and its associated dual

$$\underset{d}{\text{maximize}} y^{\mathrm{T}}d \qquad \text{subject to} \qquad X^{\mathrm{T}}d = 0, d \in [\tau - 1, \tau]^n \tag{3}$$

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

$$\underset{a}{\text{maximize}} y^{\mathrm{T}}a \qquad \text{subject to} \qquad X^{\mathrm{T}}a = (1-\tau)X^{\mathrm{T}}e, a \in [0,1]^n. \tag{4}$$

The algorithm introduced by Portnoy and Koenker (1997) and used by 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 β. 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

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

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

$$L(a, s, \beta, u, \mu) = B(a, s, \mu) - \beta^{\mathrm{T}} \left(X^{\mathrm{T}} a - (1 - \tau) X^{\mathrm{T}} e \right) - u^{\mathrm{T}} (a + s - e)$$

whose central path is described by the following first order conditions

$$\begin{array}{rcl} X^{\mathrm{T}} a & = & (1 - \tau) X^{\mathrm{T}} e \\ a + s & = & e \\ X\beta + u - v & = & y \\ SUe & = & \mu e \\ AVe & = & \mu e \end{array} \qquad (5)$$

where A denotes the diagonal matrix with diagonal elements given by a, 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 $\delta_\beta, \delta_a, \delta_s, \delta_u$ and δ_v, Mehrotra substituted the steps directly into (5) giving the augmented system of equations

$$\begin{array}{rcl} X^{\mathrm{T}} (a + \delta_a) & = & (1 - \tau) X^{\mathrm{T}} e \\ (a + \delta_a) + (s + \delta_s) & = & e \\ X\left(\beta + \delta_\beta\right) + (u + \delta_u) - (v + \delta_v) & = & y \\ (S + \Delta_s)(U + \Delta_u)e & = & \mu e \\ (A + \Delta_a)(V + \Delta_v)e & = & \mu e \end{array} \qquad (6)$$

where $\Delta_a, \Delta_s, \Delta_u$ and Δ_v denote the diagonal matrices with diagonal elements given by $\delta_a, \delta_s, \delta_u$ and δ_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

$$\begin{array}{rcl} (X^{\mathrm{T}} W X) \delta_\beta & = & X^{\mathrm{T}} W (y - X\beta) + (\tau - 1) X^{\mathrm{T}} e + X^{\mathrm{T}} a \\ \delta_a & = & W\left(y - X\beta - X\delta_\beta\right) \\ \delta_s & = & -\delta_a \\ \delta_u & = & S^{-1} U \delta_a - Ue \\ \delta_v & = & A^{-1} V \delta_s - Ve \end{array} \qquad (7)$$

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

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

$$\begin{array}{l} \hat{\gamma}_P = \sigma \quad \min \left\{ \min\limits_{i, \delta_{a_i} < 0} \{a_i / \delta_{a_i}\}, \min\limits_{i, \delta_{s_i} < 0} \{s_i / \delta_{s_i}\} \right\} \\ \hat{\gamma}_D = \sigma \quad \min \left\{ \min\limits_{i, \delta_{u_i} < 0} \{u_i / \delta_{u_i}\}, \min\limits_{i, \delta_{v_i} < 0} \{v_i / \delta_{v_i}\} \right\} \end{array} \qquad (8)$$

where σ 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 μ is obtained by letting

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

and estimating μ as

$$\mu = \left(\frac{\hat{g}(\hat{\gamma}_P, \hat{\gamma}_D)}{\hat{g}(0,0)} \right)^3 \frac{\hat{g}(0,0)}{2n}.$$

This estimate, along with the nonlinear terms (Δu, Δs, Δa and Δv) from (6) are calculated using the values of $\delta_a, \delta_s, \delta_u$ and δ_v obtained from the affine scaling step.

Given an updated estimate for μ and the nonlinear terms the system of equations

$$\begin{array}{rcl} (X^{\mathrm{T}} W X)\delta_\beta & = & X^{\mathrm{T}} W(y - X\beta + \mu(S^{-1} - A^{-1})e + S^{-1}\Delta_s\Delta_u e - A^{-1}\Delta_a\Delta_v e) + (\tau - 1)X^{\mathrm{T}} e + X^{\mathrm{T}} a \\ \delta_a & = & W\big(y - X\beta - X\delta_\beta + \mu(S^{-1} - A^{-1})\big) \\ \delta_s & = & -\delta_a \\ \delta_u & = & \mu S^{-1} e + S^{-1} U \delta_a - Ue - S^{-1}\Delta_s\Delta_u e \\ \delta_v & = & \mu A^{-1} e + A^{-1} V \delta_s - Ve - A^{-1}\Delta_a\Delta_v e \end{array} \quad (9)$$

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 (β, a, s, u, v) are updated using step directions, $\big(\delta_\beta, \delta_a, \delta_s, \delta_u, \delta_v\big)$ 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

$$\begin{array}{rcl} \tau e^{\mathrm{T}} u + (1-\tau)e^{\mathrm{T}} v - d^{\mathrm{T}} y & = & \tau e^{\mathrm{T}} u + (1-\tau)e^{\mathrm{T}} v - (a - (1-\tau)e)^{\mathrm{T}} y \\ & = & s^{\mathrm{T}} u + a^{\mathrm{T}} v \\ & = & e^{\mathrm{T}} u - a^{\mathrm{T}} y + (1-\tau)e^{\mathrm{T}} X\beta \end{array}$$

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 β. If not supplied by the user, initial values for β are calculated from a least squares regression of y on X. This regression is carried out by first constructing the cross-product matrix $X^{\mathrm{T}} X$ 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 β, 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 ϵ_u is supplied in the optional argument **Epsilon**. The initial values for the a and s are always set to $1 - \tau$ and τ respectively.

The solution for δ_β 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 β. 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 $r_i = y_i - x_i^{\mathrm{T}}\hat{\beta}$.

(b) Drop any residual where $|r_i|$ is less than ϵ_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

$$c_n = \min(\sigma_r, (q_{r3} - q_{r1})/1.34) \times \left(\Phi^{-1}(\tau + h_n) - \Phi^{-1}(\tau - h_n)\right)$$

is calculated. Dependent on the value of τ 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*** (ϵ). 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 $d_i = x_i^{\mathrm{T}}\left(\hat{\beta}(\tau + h_n) - \hat{\beta}(\tau - h_n)\right)$. 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 ϵ_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

Big
Bootstrap Interval Method
Bootstrap Iterations
Bootstrap Monitoring
Calculate Initial Values
Defaults
Drop Zero Weights
Epsilon
Interval Method
Iteration Limit
Matrix Returned
Monitoring
QR Tolerance
Return Residuals
Sigma
Significance Level
Tolerance
Unit Number

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 ϵ is a generic notation for ***machine precision*** (see nag_machine_precision (X02AJC)).

Keywords and character values are case and white space insensitive.

Band Width Alpha r Default $= 1.0$

A multiplier used to construct the parameter α_b used when calculating the Sheather–Hall bandwidth (see Section 3), with $\alpha_b = (1 - \alpha) \times$ **Band Width Alpha**. Here, α is the **Significance Level**.

Constraint: **Band Width Alpha** > 0.0.

Band Width Method a Default $=$ 'SHEATHER HALL'

The method used to calculate the bandwidth used in the calculation of the asymptotic covariance matrix Σ and H^{-1} if **Interval Method** $=$ HKS, KERNEL or IID (see Section 3).

Constraint: **Band Width Method** $=$ SHEATHER HALL or BOFINGER.

Big r Default $= 10.0^{20}$

This argument should be set to something larger than the biggest value supplied in **dat** and **y**.

Constraint: **Big** > 0.0.

Bootstrap Interval Method a Default $=$ QUANTILE

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 α 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_quantiles (g01amc).

Constraint: **Bootstrap Interval Method** $=$ T or QUANTILE.

Bootstrap Iterations i Default $= 100$

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 a Default $=$ NO

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 a Default $=$ YES

If **Calculate Initial Values** $=$ YES then the initial values for the regression parameters, β, 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 a Default $=$ YES

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 r Default $= \sqrt{\epsilon}$

ϵ_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** ≥ 0.0.

Interval Method a Default $=$ IID

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.

Interval Method = HKS
The Hendricks–Koenker Sandwich is used.

Interval Method = IID
The errors are assumed to be identical, and independently distributed.

Interval Method = 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 i Default $= 100$

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

Constraint: **Iteration Limit** > 0.

Matrix Returned a Default = NONE

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

Matrix Returned = NONE
No matrices are returned and **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 **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 a 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 β are also displayed.

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

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

QR Tolerance r Default $= \epsilon^{0.9}$

The tolerance used to calculate the rank, k, of the $p \times p$ cross-product matrix, $X^{\mathrm{T}}X$. Letting Q be the orthogonal matrix obtained from a QR decomposition of $X^{\mathrm{T}}X$, then the rank is calculated by comparing Q_{ii} with $Q_{11} \times$ **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 **bl**, **bu** and **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 $= \text{NO}$

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 $= 0.95$

α, 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**.