NAG Library Function Document

nag_regsn_ridge (g02kbc)

1 Purpose
nag_regsn_ridge (g02kbc) calculates a ridge regression, with ridge parameters supplied by you.

2 Specification

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

void nag_regsn_ridge (Nag_OrderType order, Integer n, Integer m,
const double x[], Integer pdx, const Integer isx[], Integer ip,
const double y[], Integer lh, const double h[], double nep[],
Nag_ParaOption wantb, double b[], Integer pdb, Nag_VIFOption wantvf,
double vf[], Integer pdvf, Integer lpec, const Nag_PredictError pec[],
double pe[], Integer pdpe, NagError *fail)
```

3 Description

A linear model has the form:

\[ y = c + X\beta + \epsilon, \]

where

- \( y \) is an \( n \) by 1 matrix of values of a dependent variable;
- \( c \) is a scalar intercept term;
- \( X \) is an \( n \) by \( m \) matrix of values of independent variables;
- \( \beta \) is a \( m \) by 1 matrix of unknown values of parameters;
- \( \epsilon \) is an \( n \) by 1 matrix of unknown random errors such that variance of \( \epsilon = \sigma^2 I \).

Let \( \tilde{X} \) be the mean-centred \( X \) and \( \tilde{y} \) the mean-centred \( y \). Furthermore, \( \tilde{X} \) is scaled such that the diagonal elements of the cross product matrix \( \tilde{X}^T \tilde{X} \) are one. The linear model now takes the form:

\[ \tilde{y} = \tilde{X}\tilde{\beta} + \epsilon. \]

Ridge regression estimates the parameters \( \tilde{\beta} \) in a penalised least squares sense by finding the \( \tilde{b} \) that minimizes

\[ \| \tilde{X}\tilde{b} - \tilde{y} \|^2 + h\| \tilde{b} \|^2, \quad h > 0, \]

where \( \| \cdot \| \) denotes the \( \ell_2 \)-norm and \( h \) is a scalar regularization or ridge parameter. For a given value of \( h \), the parameters estimates \( \tilde{b} \) are found by evaluating

\[ \tilde{b} = (\tilde{X}^T \tilde{X} + hI)^{-1} \tilde{X}^T \tilde{y}. \]

Note that if \( h = 0 \) the ridge regression solution is equivalent to the ordinary least squares solution.

Rather than calculate the inverse of \((\tilde{X}^T \tilde{X} + hI)\) directly, nag_regsn_ridge (g02kbc) uses the singular value decomposition (SVD) of \( \tilde{X} \). After decomposing \( \tilde{X} \) into \( UDV^T \) where \( U \) and \( V \) are orthogonal matrices and \( D \) is a diagonal matrix, the parameter estimates become

\[ \tilde{b} = V(D^T D + hI)^{-1} D^T \tilde{y}. \]

A consequence of introducing the ridge parameter is that the effective number of parameters, \( \gamma \), in the model is given by the sum of diagonal elements of
\[ D^T D(D^T D + hI)^{-1}, \]

see Moody (1992) for details.

Any multi-collinearity in the design matrix \( X \) may be highlighted by calculating the variance inflation factors for the fitted model. The \( j \)th variance inflation factor, \( v_j \), is a scaled version of the multiple correlation coefficient between independent variable \( j \) and the other independent variables, \( R_j \), and is given by

\[ v_j = \frac{1}{1 - R_j^2}, \quad j = 1, 2, \ldots, m. \]

The \( m \) variance inflation factors are calculated as the diagonal elements of the matrix:

\[ (\hat{X}^T \hat{X} + hI)^{-1} \hat{X}^T \hat{X} (\hat{X}^T \hat{X} + hI)^{-1}, \]

which, using the SVD of \( \hat{X} \), is equivalent to the diagonal elements of the matrix:

\[ V(D^T D + hI)^{-1} D^T D(D^T D + hI)^{-1} V^T. \]

Given a value of \( h \), any or all of the following prediction criteria are available:

(a) Generalized cross-validation (GCV):

\[ \frac{ns}{(n - \gamma)^2}; \]

(b) Unbiased estimate of variance (UEV):

\[ \frac{s}{n - \gamma}; \]

(c) Future prediction error (FPE):

\[ \frac{1}{n} \left( s + \frac{2\gamma s}{n - \gamma} \right); \]

(d) Bayesian information criterion (BIC):

\[ \frac{1}{n} \left( s + \frac{\log(n)\gamma}{n - \gamma} \right); \]

(e) Leave-one-out cross-validation (LOOCV),

where \( s \) is the sum of squares of residuals.

Although parameter estimates \( \hat{b} \) are calculated by using \( \hat{X} \), it is usual to report the parameter estimates \( b \) associated with \( X \). These are calculated from \( \hat{b} \), and the means and scalings of \( X \). Optionally, either \( \hat{b} \) or \( b \) may be calculated.

4 References


5 Arguments

1: \texttt{order} – Nag_OrderType

\textit{Input}

On entry: the \texttt{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: n – Integer

On entry: n, the number of observations.

Constraint: n ≥ 1.

3: m – Integer

On entry: the number of independent variables available in the data matrix X.

Constraint: m ≤ n.

4: x[\text{dim}] – const double

Note: the dimension, \text{dim}, of the array x must be at least

\text{max}(1, \text{pdx} \times m) \text{ when order = Nag_ColMajor;}
\text{max}(1, n \times \text{pdx}) \text{ when order = Nag_RowMajor.}

The \( (i, j) \)th element of the matrix \( X \) is stored in

\[ x[(j - 1) \times \text{pdx} + i - 1] \text{ when order = Nag_ColMajor;} \]
\[ x[(i - 1) \times \text{pdx} + j - 1] \text{ when order = Nag_RowMajor.} \]

On entry: the values of independent variables in the data matrix X.

5: pdx – Integer

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

Constraints:

- if order = Nag_ColMajor, pdx ≥ n;
- if order = Nag_RowMajor, pdx ≥ m.

6: isx[m] – const Integer

On entry: indicates which \( m \) independent variables are included in the model.

\[ \text{isx}[j - 1] = 1 \]

The \( j \)th variable in \( x \) will be included in the model.

\[ \text{isx}[j - 1] = 0 \]

Variable \( j \) is excluded.

Constraint: \text{isx}[j - 1] = 0 or 1, for \( j = 1, 2, \ldots, m \).

7: ip – Integer

On entry: \( m \), the number of independent variables in the model.

Constraints:

- \( 1 \leq \text{ip} \leq m; \)
  - Exactly \( \text{ip} \) elements of \text{isx} must be equal to 1.

8: y[n] – const double

On entry: the \( n \) values of the dependent variable \( y \).
9: lh – Integer

*Input*

*On entry:* the number of supplied ridge parameters.

*Constraint:* \( lh > 0 \).

10: \( h[lh] \) – const double

*Input*

*On entry:* \( h[j - 1] \) is the value of the \( j \)th ridge parameter \( h \).

*Constraint:* \( h[j - 1] \geq 0.0 \), for \( j = 1, 2, \ldots, lh \).

11: nep[lh] – double

*Output*

*On exit:* \( nep[j - 1] \) is the number of effective parameters, \( \gamma \), in the \( j \)th model, for \( j = 1, 2, \ldots, lh \).

12: wantb – Nag_ParaOption

*Input*

*On entry:* defines the options for parameter estimates.

wantb = Nag_NoPara

Parameter estimates are not calculated and \( b \) is not referenced.

wantb = Nag_OrigPara

Parameter estimates \( b \) are calculated for the original data.

wantb = Nag_StandPara

Parameter estimates \( \tilde{b} \) are calculated for the standardized data.

*Constraint:* wantb = Nag_NoPara, Nag_OrigPara or Nag_StandPara.

13: \( b[dim] \) – double

*Output*

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

\[
\begin{align*}
pdb \times lh & \quad \text{when wantb} \neq \text{Nag_NoPara and order} = \text{Nag_ColMajor}; \\
\max(1, (ip + 1) \times pdb) & \quad \text{when wantb} \neq \text{Nag_NoPara and order} = \text{Nag_RowMajor}; \\
1 & \quad \text{otherwise}.
\end{align*}
\]

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

\[
\begin{align*}
b[(j - 1) \times pdb + i - 1] & \quad \text{when order} = \text{Nag_ColMajor}; \\
b[(i - 1) \times pdb + j - 1] & \quad \text{when order} = \text{Nag_RowMajor}.
\end{align*}
\]

*On exit:* if wantb \neq Nag_NoPara, \( b \) contains the intercept and parameter estimates for the fitted ridge regression model in the order indicated by isx. \( B(1, j) \), for \( j = 1, 2, \ldots, lh \), contains the estimate for the intercept; \( B(i + 1, j) \) contains the parameter estimate for the \( i \)th independent variable in the model fitted with ridge parameter \( h[j - 1] \), for \( i = 1, 2, \ldots, ip \).

14: pdb – Integer

*Input*

*On entry:* the stride separating row or column elements (depending on the value of order) in the array \( b \).

*Constraints:*

if order = Nag_ColMajor,

if wantb \neq Nag_NoPara, pdb \geq ip + 1;

otherwise pdb \geq 1.

if order = Nag_RowMajor,

if wantb \neq Nag_NoPara, pdb \geq lh;

otherwise pdb \geq 1.
15: **wantvf** – Nag_VIFOption

*On entry:* defines the options for variance inflation factors.

**wantvf** = Nag_NoVIF

Variance inflation factors are not calculated and the array *vf* is not referenced.

**wantvf** = Nag_WantVIF

Variance inflation factors are calculated.

**Constraints:**

**wantvf** = Nag_NoVIF or Nag_WantVIF;

if **wantb** = Nag_NoPara, **wantvf** = Nag_WantVIF.

16: **vf**[**dim**] – double

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

\[ \text{pdvf} \times \text{lh} \]

when **wantvf** ≠ Nag_NoVIF and **order** = Nag_ColMajor;

\[ \max(1, \text{ip} \times \text{pdvf}) \]

when **wantvf** ≠ Nag_NoVIF and **order** = Nag_RowMajor;

1 otherwise.

Where \( \text{VF}(i, j) \) appears in this document, it refers to the array element

\[ \text{vf}[(j - 1) \times \text{pdvf} + i - 1] \]

when **order** = Nag_ColMajor;

\[ \text{vf}[(i - 1) \times \text{pdvf} + j - 1] \]

when **order** = Nag_RowMajor.

*On exit:* if **wantvf** = Nag_WantVIF, the variance inflation factors. For the \( i \)th independent variable in a model fitted with ridge parameter \( h[j - 1] \), \( \text{VF}(i, j) \) is the value of \( v_i \), for \( i = 1, 2, \ldots, \text{ip} \).

17: **pdvf** – Integer

*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array *vf*.

**Constraints:**

if **order** = Nag_ColMajor,

if **wantvf** ≠ Nag_NoVIF, **pdvf** ≥ **ip**;

otherwise **pdvf** ≥ 1.;

if **order** = Nag_RowMajor,

if **wantvf** ≠ Nag_NoVIF, **pdvf** ≥ **lh**;

otherwise **pdvf** ≥ 1..

18: **lpec** – Integer

*On entry:* the number of prediction error statistics to return; set **lpec** ≤ 0 for no prediction error estimates.

19: **pec[**lpec]**] – const Nag_PredictError

*On entry:* if **lpec** > 0, **pec[j - 1]** defines the \( j \)th prediction error, for \( j = 1, 2, \ldots, \text{lpec} \); otherwise **pec** is not referenced.

**pec[j - 1]** = Nag_BIC

Bayesian information criterion (BIC).

**pec[j - 1]** = Nag_FPE

Future prediction error (FPE).

**pec[j - 1]** = Nag_GCV

Generalized cross-validation (GCV).

**pec[j - 1]** = Nag_LOOCV

Leave-one-out cross-validation (LOOCV).
pec[j − 1] = Nag_EUV
Unbiased estimate of variance (UEV).

Constraint: if lpec > 0, pec[j − 1] = Nag_BIC, Nag_FPE, Nag_GCV, Nag_LOOCV or Nag_EUV, for j = 1, 2, . . . , lpec.

Output

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

pdpe × lh when lpec > 0 and order = Nag_ColMajor;
max(1, lpec × pdpe) when lpec > 0 and order = Nag_RowMajor;
1 otherwise.

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

pe[(j − 1) × pdpe + i − 1] when order = Nag_ColMajor;
pe[(i − 1) × pdpe + j − 1] when order = Nag_RowMajor.

On exit: if lpec ≤ 0, pe is not referenced; otherwise PE(i, j) contains the prediction error of criterion pec[i − 1] for the model fitted with ridge parameter h[j − 1], for i = 1, 2, . . . , lpec and j = 1, 2, . . . , lh.

Input

pdpe – Integer
On entry: the stride separating row or column elements (depending on the value of order) in the array pe.

Constraints:

if order = Nag_ColMajor,
   if lpec > 0, pdpe ≥ lpec;
   otherwise pdpe ≥ 1;
if order = Nag_RowMajor,
   if lpec > 0, pdpe ≥ lh;
   otherwise pdpe ≥ 1.

Input/Output

fail – NagError *
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

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

NE_CONSTRAINT
On entry, wantb = Nag_NoPara and wantvf = Nag_NoVIF.

NE_ENUM_INT_2
On entry, pdb = ⟨value⟩ and ip = ⟨value⟩.
Constraint: if wantb ≠ Nag_NoPara, pdb ≥ ip + 1.
On entry, pdvf = ⟨value⟩ and ip = ⟨value⟩.
Constraint: if wantvf ≠ Nag_NoVIF, pdvf ≥ ip.
On entry, \( \text{wantb} = \langle \text{value} \rangle \), \( \text{pdb} = \langle \text{value} \rangle \), \( \text{lh} = \langle \text{value} \rangle \).
Constraint: if \( \text{wantb} \neq \text{Nag}_\text{NoPara} \), \( \text{pdb} \geq \text{lh} \); otherwise \( \text{pdb} \geq 1 \).

On entry, \( \text{wantvf} = \langle \text{value} \rangle \), \( \text{pdvf} = \langle \text{value} \rangle \), \( \text{lh} = \langle \text{value} \rangle \).
Constraint: if \( \text{wantvf} \neq \text{Nag}_\text{NoVIF} \), \( \text{pdvf} \geq \text{lh} \); otherwise \( \text{pdvf} \geq 1 \).

\text{NE_INT}

On entry, \( \text{lh} = \langle \text{value} \rangle \).
Constraint: \( \text{lh} > 0 \).

On entry, \( \text{n} = \langle \text{value} \rangle \).
Constraint: \( \text{n} \geq 1 \).

\text{NE_INT_2}

On entry, \( \text{m} = \langle \text{value} \rangle \) and \( \text{n} = \langle \text{value} \rangle \).
Constraint: \( \text{m} \leq \text{n} \).

On entry, \( \text{pdpe} = \langle \text{value} \rangle \) and \( \text{lpec} = \langle \text{value} \rangle \).
Constraint: \( \text{pdpe} \geq \text{lpec} \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \) and \( \text{m} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} \geq \text{m} \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \) and \( \text{n} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} \geq \text{n} \).

\text{NE_INT_3}

On entry, \( \text{pdpe} = \langle \text{value} \rangle \), \( \text{lpec} = \langle \text{value} \rangle \) and \( \text{lh} = \langle \text{value} \rangle \).
Constraint: if \( \text{lpec} > 0 \), \( \text{pdpe} \geq \text{lh} \); otherwise \( \text{pdpe} \geq 1 \).

\text{NE_INT_ARG_CONS}

ip does not equal the sum of elements in isx.

\text{NE_INT_ARRAY_VAL_1_OR_2}

On entry, \( \text{isx}[i - 1] \neq 0 \) or 1 for at least one \( i \).

\text{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.

\text{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.

\text{NE_REAL_ARRAY_CONS}

On entry, \( h[i - 1] < 0 \) for at least one \( i \).

7 Accuracy

The accuracy of nag_regsn_ridge (g02kbc) is closely related to that of the singular value decomposition.
8 Parallelism and Performance

nag_regsn_ridge (g02kbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_regsn_ridge (g02kbc) 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_ridge (g02kbc) allocates internally \( \max(5 \times (n - 1), 2 \times ip \times ip) + (n + 3) \times ip + n \) elements of double precision storage.

10 Example

This example reads in data from an experiment to model body fat, and a selection of ridge regression models are calculated.

10.1 Program Text

/* nag_regsn_ridge (g02kbc) Example Program. *
 * Copyright 2014 Numerical Algorithms Group.
 * * Mark 9, 2009. */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>

int main(void)
{
  /*Integer scalar and array declarations */
  Integer exit_status = 0;
  Integer i, ip, ip1, j, lh, lpec, m, n, pl;
  Integer pdb, pdpe, pdvf, pdx;
  Integer *isx = 0;
  /*Double scalar and array declarations */
  double *b = 0, *h = 0, *nep = 0, *pe = 0, *vf = 0, *x = 0, *y = 0;
  /*Character scalar and array declarations */
  char spec[40], swantb[40];
  /*NAG Types */
  Nag_OrderType order;
  Nag_ParaOption wantb;
  Nag_VIFOption wantvf;
  Nag_PredictError *pec = 0;
  NagError fail;

  INIT_FAIL(fail);
  printf("%s\n", "nag_regsn_ridge (g02kbc) Example Program Results");
  /* Skip heading in data file*/
  #ifdef _WIN32
    scanf_s("%*[\n \] ");
  #else
    scanf("%*[\n \] ");
  #endif
  /* Read in the problem size information*/
```c
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%39s%*\n", &n, &m, &lh, &lpec, swantb, _countof(swantb));
#else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%39s%*\n", &n, &m, &lh, &lpec, swantb);
#endif

wantb = (Nag_ParaOption) nag_enum_name_to_value(swantb);
if (!(isx = NAG_ALLOC(m, Integer))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read in the ISX flags*/
for (i = 0; i < m; i++)
#ifdef _WIN32
    scanf_s("%"NAG_IFMT", &isx[i]);
#else
    scanf("%"NAG_IFMT", &isx[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

/* Total number of variables*/
ip = 0;
for (j = 0; j < m; j++)
    if (isx[j] == 1)
        ip = ip+1;
#ifdef NAG_COLUMN_MAJOR
    pdb = ip+1;
#define B(I, J) b[(J-1)*pdb + I-1]
    pdpe = lpec;
#define PE(I, J) pe[(J-1)*pdpe + I-1]
    pdvf = ip;
#define VF(I, J) vf[(J-1)*pdvf + I-1]
    pdx = n;
#define X(I, J) x[(J-1)*pdx + I-1]
    order = Nag_ColMajor;
#else
    pdb = lh;
#define B(I, J) b[(I-1)*pdb + J-1]
    pdpe = lh;
#define PE(I, J) pe[(I-1)*pdpe + J-1]
    pdvf = lh;
#define VF(I, J) vf[(I-1)*pdvf + J-1]
    pdx = m;
#define X(I, J) x[(I-1)*pdx + J-1]
    order = Nag_RowMajor;
#endif

if (!(b = NAG_ALLOC(pdb*(order == Nag_RowMajor?(ip+1):lh), double)) ||
    !(h = NAG_ALLOC(lh, double)) ||
    !(nep = NAG_ALLOC(lh, double)) ||
    !(pe = NAG_ALLOC(pdpe*(order == Nag_RowMajor?lpec:lh), double)) ||
    !(vf = NAG_ALLOC(pdvf*(order == Nag_RowMajor?ip:lh), double)) ||
    !(x = NAG_ALLOC(pdx*(order == Nag_RowMajor?n:m, double)) ||
    !(y = NAG_ALLOC(n, double)) ||
    !(pec = NAG_ALLOC(lpec, Nag_PredictError))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read in the data*/
if (lpec > 0) {
    for (i = 0; i < lpec; i++)
```
{
    #ifdef _WIN32
        scanf_s("%39s ", spec, _countof(spec));
    #else
        scanf("%39s ", spec);
    #endif
    pec[i] = (Nag_PredictError) nag_enum_name_to_value(spec);
}
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    } for (i = 1; i <= n; i++)
    { for (j = 1; j <= m; j++)
#ifdef _WIN32
        scanf_s("%lf ", &X(i, j));
#else
        scanf("%lf ", &X(i, j));
#endif
#ifdef _WIN32
        scanf_s("%lf ", &y[i-1]);
#else
        scanf("%lf ", &y[i-1]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    /* Read in the ridge coefficients*/
    for (i = 0; i < lh; i++)
#ifdef _WIN32
    scanf_s("%lf ", &h[i]);
#else
    scanf("%lf ", &h[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    /* Output the variance inflation factors and parameter estimates*/
    wantvf = Nag_WantVIF;
    /* Run the analysis*/
    /* nag_regsn_ridge (g02kbc) 
    * Ridge regression 
    */
    nag_regsn_ridge(order, n, m, x, pdx, isx, ip, y, lh, h, nep, wantb, b, pdb,
    wantvf, vf, pdvf, lpec, pec, pe, pdpe, &fail);
    if (fail.code != NE_NOERROR)
    { printf("Error from nag_regsn_ridge (g02kbc).\n\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Output results*/
    ip1 = ip-1;
    /* Summaries*/
    printf("%s\%s\n", "Number of parameters used = ", ip+1);
        printf("%s\", "Effective number of parameters (NEP): ");
        printf("%s\", " Ridge ");
        printf("%s\", " Coeff. ", "NEP");
        for (i = 1; i <= lh; i++)
            printf(" %10.4f %10.4f\n", h[i-1], nep[i-1]);
    /* Parameter estimates*/
}
if (wantb != Nag_NoPara)
{
    printf("\n");
    if (wantb == Nag_OrigPara)
    {
        printf("%s\n", "Parameter Estimates (Original scalings)");
    }
    else
    {
        printf("%s\n", "Parameter Estimates (Standarised)");
    }
    pl = MIN(ip, 4);
    printf("%s\n", " Ridge ");
    printf("%s ", " Coeff. ");
    printf("%s ", "Intercept ");
    for (i = 1; i <= pl; i++)
        printf("%10\%s", i, i%4? " ":"\n");
    printf("\n");
    if (pl < ip1)
    {
        for (i = pl+1; i <= ip1; i++)
            printf("%10\%s", i, i%5? " ":"\n");
    }
    pl = MIN(ip+1, 5);
    for (i = 1; i <= lh; i++)
    {
        printf("%10.4f", h[i-1]);
        for (j = 1; j <= pl; j++)
            printf("%10.4f%s", B(j, i), j%5? " ":"\n");
        printf("\n");
        if (pl < ip)
        {
            for (j = pl+1; j <= ip; j++)
                printf("%10.4f%s", VF(j, i), j%5? " ":"\n");
        }
    }
}
/* Variance inflation factors*/
if (wantvf != Nag_NoVIF)
{
    printf("\n");
    printf("%s\n", "Variance Inflation Factors");
    pl = MIN(ip, 5);
    printf("%s\n", " Ridge ");
    printf("%s", " Coeff. ");
    for (i = 1; i <= pl; i++)
        printf("%10\%s", i, i%5? " ":"\n");
    printf("\n");
    if (pl < ip)
    {
        for (i = pl+1; i <= ip; i++)
            printf("%10\%s", i, i%5? " ":"\n");
    }
    for (i = 1; i <= lh; i++)
    {
        printf("%10.4f", h[i-1]);
        for (j = 1; j <= pl; j++)
            printf("%10.4f%s", VF(j, i), j%5? " ":"\n");
        printf("\n");
        if (pl < ip)
        {
            for (j = pl+1; j <= ip; j++)
                printf("%10.4f%s", VF(j, i), j%5? " ":"\n");
        }
    }
}
if (lpec > 0)
{
    printf("\n");
    printf("%s\n", "Prediction error criterion");
    pl = MIN(lpec, 5);
    printf("%s\n", " Ridge ");
    printf("%s", " Coeff. ");
    for (i = 1; i <= pl; i++)
        printf("%10"NAG_IFMT", i, i%5?":\n");
    printf("\n");
    if (pl < lpec)
    {
        for (i = pl+1; i <= lpec; i++)
            printf("%10"NAG_IFMT", i, i%5?":\n");
        printf("\n");
    }
    for (i = 1; i <= lh; i++)
    {
        printf("%10.4f", h[i-1]);
        for (j = 1; j <= pl; j++)
            printf("%10.4f", PE(j, i), j%5?":\n");
        if (pl < ip)
        {
            for (j = pl+1; j <= ip; j++)
                printf("%10.4f", PE(j, i), j%5?":\n");
        }
    }
    printf("\n");
    printf("%s\n", "Key:");
    for (i = 1; i <= lpec; i++)
    {
        if (pec[i-1] == Nag_LOOCV)
        {
            printf("%5"NAG_IFMT" Leave one out cross-validation\n", i);
        }
        else if (pec[i-1] == Nag_GCV)
        {
            printf("%5"NAG_IFMT" Generalised cross-validation\n", i);
        }
        else if (pec[i-1] == Nag_EUV)
        {
            printf("%5"NAG_IFMT" Unbiased estimate of variance\n", i);
        }
        else if (pec[i-1] == Nag_FPE)
        {
            printf("%5"NAG_IFMT" Final prediction error\n", i);
        }
        else if (pec[i-1] == Nag_BIC)
        {
            printf("%5"NAG_IFMT" Bayesian information criterion\n", i);
        }
    }
}

END:
NAG_FREE(b);
NAG_FREE(h);
NAG_FREE(nep);
NAG_FREE(pe);
NAG_FREE(vf);
NAG_FREE(x);
NAG_FREE(y);
NAG_FREE(isx);
NAG_FREE(pec);
return exit_status;
10.2 Program Data

nag_regsn_ridge (g02kbc) Example Program Data
20 3 16 5 Nag_OrigPara : n, m, lh, lpec, wantb
1 1 l : isx
Nag_LOOCV Nag_GCV Nag_EUV Nag_FPE Nag_BIC : pec
19.5 43.1 29.1 11.9
24.7 49.8 28.2 22.8
19.1 42.2 30.9 12.9
25.6 53.9 23.7 21.7
31.4 58.5 27.6 27.1
27.9 52.1 30.6 25.4
22.1 49.9 23.2 21.3
25.5 53.5 24.8 19.3
31.1 56.6 30.0 25.4
30.4 56.7 28.3 27.2
18.7 46.5 23.0 11.7
19.7 44.2 28.6 17.8
14.6 42.7 21.3 12.8
29.5 54.4 30.1 23.9
27.7 55.3 25.7 22.6
30.2 58.6 24.6 25.4
22.7 48.2 27.1 14.8
25.2 51.0 27.5 21.1 : End of observations
0.0 0.002 0.004 0.006 0.008 0.010 0.012 0.014 0.016 0.018 0.020 0.022 0.024 0.026 0.028 0.030 : Ridge co-efficients

10.3 Program Results

nag_regsn_ridge (g02kbc) Example Program Results
Number of parameters used = 4
Effective number of parameters (NEP): Ridge
Coeff. NEP
0.0000 4.0000
0.0020 3.2634
0.0040 3.1475
0.0060 3.0987
0.0080 3.0709
0.0100 3.0523
0.0120 3.0386
0.0140 3.0278
0.0160 3.0189
0.0180 3.0112
0.0200 3.0045
0.0220 2.9984
0.0240 2.9928
0.0260 2.9876
0.0280 2.9828
0.0300 2.9782

Parameter Estimates (Original scalings)
Coeff. Intercept 1 2 3
0.0000 117.0847 4.3341 -2.8568 -2.1861
0.0020 22.2748 1.4644 -0.4012 -0.6738
0.0040 7.7209 1.0229 -0.1282 -0.3460
0.0060 1.8363 0.8437 0.1282 -0.3460
0.0080 -1.3396 0.7465 0.2105 -0.2944
0.0100 -3.3219 0.6853 0.2618 -0.2619
0.0120 -4.6734 0.6432 0.2968 -0.2393
0.0140 -5.6511 0.6125 0.3222 -0.2228
0.0160 -6.3891 0.5890 0.3413 -0.2100
0.0180 -6.9642 0.5704 0.3562 -0.1999
0.0200 -7.4236 0.5554 0.3681 -0.1916
0.0220 -7.7978 0.5429 0.3779 -0.1847
0.0240 -8.1075 0.5323 0.3859 -0.1788
0.0260 -8.3673 0.5233 0.3926 -0.1737
### Variance Inflation Factors

**Ridge Coeff.**

<table>
<thead>
<tr>
<th>Coeff.</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>708.8429</td>
<td>564.3434</td>
<td>104.6060</td>
</tr>
<tr>
<td>0.0020</td>
<td>50.5592</td>
<td>40.4483</td>
<td>8.2797</td>
</tr>
<tr>
<td>0.0040</td>
<td>16.9816</td>
<td>13.7247</td>
<td>3.3628</td>
</tr>
<tr>
<td>0.0060</td>
<td>8.5033</td>
<td>6.9764</td>
<td>2.1185</td>
</tr>
<tr>
<td>0.0080</td>
<td>5.1472</td>
<td>4.3046</td>
<td>1.6238</td>
</tr>
<tr>
<td>0.0100</td>
<td>3.4855</td>
<td>2.9813</td>
<td>1.3770</td>
</tr>
<tr>
<td>0.0120</td>
<td>2.5434</td>
<td>2.2306</td>
<td>1.2356</td>
</tr>
<tr>
<td>0.0140</td>
<td>1.9581</td>
<td>1.7640</td>
<td>1.1463</td>
</tr>
<tr>
<td>0.0160</td>
<td>1.5698</td>
<td>1.4541</td>
<td>1.0859</td>
</tr>
<tr>
<td>0.0180</td>
<td>1.2990</td>
<td>1.2377</td>
<td>1.0428</td>
</tr>
<tr>
<td>0.0200</td>
<td>1.1026</td>
<td>1.0805</td>
<td>1.0105</td>
</tr>
<tr>
<td>0.0220</td>
<td>0.9356</td>
<td>0.9627</td>
<td>0.9855</td>
</tr>
<tr>
<td>0.0240</td>
<td>0.8427</td>
<td>0.8721</td>
<td>0.9655</td>
</tr>
<tr>
<td>0.0260</td>
<td>0.7541</td>
<td>0.8007</td>
<td>0.9491</td>
</tr>
<tr>
<td>0.0280</td>
<td>0.6832</td>
<td>0.7435</td>
<td>0.9353</td>
</tr>
<tr>
<td>0.0300</td>
<td>0.6257</td>
<td>0.6969</td>
<td>0.9235</td>
</tr>
</tbody>
</table>

### Prediction error criterion

**Ridge Coeff.**

<table>
<thead>
<tr>
<th>Coeff.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>8.0368</td>
<td>7.6879</td>
<td>6.1503</td>
<td>7.3804</td>
<td>8.6052</td>
</tr>
<tr>
<td>0.0020</td>
<td>7.5464</td>
<td>7.4238</td>
<td>6.2124</td>
<td>7.2261</td>
<td>8.2355</td>
</tr>
<tr>
<td>0.0040</td>
<td>7.5575</td>
<td>7.4520</td>
<td>6.2793</td>
<td>7.2675</td>
<td>8.2515</td>
</tr>
<tr>
<td>0.0060</td>
<td>7.5656</td>
<td>7.4668</td>
<td>6.3100</td>
<td>7.2876</td>
<td>8.2611</td>
</tr>
<tr>
<td>0.0080</td>
<td>7.5701</td>
<td>7.4749</td>
<td>6.3272</td>
<td>7.2987</td>
<td>8.2661</td>
</tr>
<tr>
<td>0.0100</td>
<td>7.5723</td>
<td>7.4796</td>
<td>6.3381</td>
<td>7.3053</td>
<td>8.2685</td>
</tr>
<tr>
<td>0.0120</td>
<td>7.5732</td>
<td>7.4823</td>
<td>6.3455</td>
<td>7.3095</td>
<td>8.2695</td>
</tr>
<tr>
<td>0.0140</td>
<td>7.5734</td>
<td>7.4838</td>
<td>6.3508</td>
<td>7.3122</td>
<td>8.2696</td>
</tr>
<tr>
<td>0.0160</td>
<td>7.5731</td>
<td>7.4846</td>
<td>6.3548</td>
<td>7.3140</td>
<td>8.2691</td>
</tr>
<tr>
<td>0.0180</td>
<td>7.5724</td>
<td>7.4848</td>
<td>6.3578</td>
<td>7.3151</td>
<td>8.2683</td>
</tr>
<tr>
<td>0.0200</td>
<td>7.5715</td>
<td>7.4847</td>
<td>6.3603</td>
<td>7.3158</td>
<td>8.2671</td>
</tr>
<tr>
<td>0.0220</td>
<td>7.5705</td>
<td>7.4843</td>
<td>6.3623</td>
<td>7.3161</td>
<td>8.2659</td>
</tr>
<tr>
<td>0.0240</td>
<td>7.5694</td>
<td>7.4838</td>
<td>6.3639</td>
<td>7.3162</td>
<td>8.2645</td>
</tr>
<tr>
<td>0.0260</td>
<td>7.5682</td>
<td>7.4832</td>
<td>6.3654</td>
<td>7.3162</td>
<td>8.2630</td>
</tr>
<tr>
<td>0.0280</td>
<td>7.5669</td>
<td>7.4825</td>
<td>6.3666</td>
<td>7.3161</td>
<td>8.2615</td>
</tr>
<tr>
<td>0.0300</td>
<td>7.5657</td>
<td>7.4818</td>
<td>6.3677</td>
<td>7.3159</td>
<td>8.2600</td>
</tr>
</tbody>
</table>

**Key:**

1. Leave one out cross-validation
2. Generalised cross-validation
3. Unbiased estimate of variance
4. Final prediction error
5. Bayesian information criterion