NAG Library Function Document

nag_regsn_mult_linear_newyvar (g02dgc)

1 Purpose

nag_regsn_mult_linear_newyvar (g02dgc) calculates the estimates of the arguments of a general linear regression model for a new dependent variable after a call to nag_regsn_mult_linear (g02dac).

2 Specification

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

```
void nag_regsn_mult_linear_newyvar (Integer n, const double wt[],
     double *rss, Integer ip, Integer rank, double cov[], double q[],
     Integer tdq, Nag_Boolean svd, const double p[], const double y[],
     double b[], double se[], double res[], const double com_ar[],
     NagError *fail)
```

3 Description

nag_regsn_mult_linear_newyvar (g02dgc) uses the results given by nag_regsn_mult_linear (g02dac) to fit the same set of independent variables to a new dependent variable.

nag_regsn_mult_linear (g02dac) computes a QR decomposition of the matrix of p independent variables and also, if the model is not of full rank, a singular value decomposition (SVD). These results can be used to compute estimates of the arguments for a general linear model with a new dependent variable. The QR decomposition leads to the formation of an upper triangular p by p matrix R and an n by n orthogonal matrix Q. In addition the vector $c = Q^{\mathrm{T}}y$ (or $Q^{\mathrm{T}}W^{1/2}y$) is computed. For a new dependent variable, y_{new}, nag_regsn_mult_linear_newyvar (g02dgc) computes a new value of $c = Q^{\mathrm{T}}y_{\mathrm{new}}$ or $Q^{\mathrm{T}}W^{1/2}y_{\mathrm{new}}$.

If R is of full rank, then the least squares parameter estimates, $\hat{\beta}$, are the solution to: $R\hat{\beta} = c_1$, where c_1 is the first p elements of c.

If R is not of full rank, then nag_regsn_mult_linear (g02dac) will have computed the SVD of R,

$$R = Q_* \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} P^{\mathrm{T}}$$

where D is a k by k diagonal matrix with nonzero diagonal elements, k being the rank of R, and Q_* and P are p by p orthogonal matrices. This gives the solution

$$\hat{\beta} = P_1 D^{-1} Q_{*_1}^{\mathrm{T}} c_1$$

P_1 being the first k columns of P, i.e., $P = (P_1 P_0)$ and Q_{*_1} being the first k columns of Q_*. Details of the SVD are made available by nag_regsn_mult_linear (g02dac) in the form of the matrix P^*:

$$P^* = \begin{pmatrix} D^{-1} P_1^{\mathrm{T}} \\ P_0^{\mathrm{T}} \end{pmatrix}.$$

The matrix Q_* is made available through the **com_ar** argument of nag_regsn_mult_linear (g02dac).

In addition to parameter estimates, the new residuals are computed and the variance-covariance matrix of the parameter estimates are found by scaling the variance-covariance matrix for the original regression.

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

Hammarling S (1985) The singular value decomposition in multivariate statistics *SIGNUM Newsl.* **20(3)** 2–25

Searle S R (1971) *Linear Models* Wiley

5 Arguments

1: **n** – Integer *Input*

On entry: the number of observations, n.

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

2: **wt**[**n**] – const double *Input*

On entry: optionally, the weights to be used in the weighted regression.

If $\mathbf{wt}[i-1] = 0.0$, then the ith observation is not included in the model, in which case the effective number of observations is the number of observations with nonzero weights. The values of **res** and **h** will be set to zero for observations with zero weights (see nag_regsn_mult_linear (g02dac)).

If weights are not provided then **wt** must be set to **NULL** and the effective number of observations is **n**.

Constraint: if **wt** is not **NULL**, $\mathbf{wt}[i-1] = 0.0$, for $i = 1, 2, \ldots, n$.

3: **rss** – double * *Input/Output*

On entry: the residual sum of squares for the original dependent variable.

On exit: the residual sum of squares for the new dependent variable.

4: **ip** – Integer *Input*

On entry: the number p of independent variables in the model (including the mean if fitted).

Constraint: $1 \leq \mathbf{ip} \leq \mathbf{n}$.

5: **rank** – Integer *Input*

On entry: the rank of the independent variables, as given by nag_regsn_mult_linear (g02dac).

Constraint: $\mathbf{rank} > 0$ and if $\mathbf{svd} = \text{Nag\_FALSE}$, $\mathbf{rank} = \mathbf{ip}$ otherwise $\mathbf{rank} \leq \mathbf{ip}$.

6: **cov**[$\mathbf{ip} \times (\mathbf{ip} + \mathbf{1})/\mathbf{2}$] – double *Input/Output*

On entry: the covariance matrix of the parameter estimates as given by nag_regsn_mult_linear (g02dac).

On exit: the upper triangular part of the variance-covariance matrix of the **ip** parameter estimates given in **b**. They are stored packed by column, i.e., the covariance between the parameter estimate given in $\mathbf{b}[i]$ and the parameter estimate given in $\mathbf{b}[j]$, $j \geq i$, is stored in $\mathbf{cov}[j(j+1)/2 + i]$ for $i = 0, 1, \ldots, \mathbf{ip} - 1$ and $j = i, i+1, \ldots, \mathbf{ip} - 1$.

7: **q**[$\mathbf{n} \times \mathbf{tdq}$] – double *Input/Output*

Note: the (i,j)th element of the matrix Q is stored in $\mathbf{q}[(i-1) \times \mathbf{tdq} + j - 1]$.

On entry: the results of the QR decomposition as returned by nag_regsn_mult_linear (g02dac).

On exit: the first column of **q** contains the new values of c, the remainder of **q** will be unchanged.

8: **tdq** – Integer *Input*

On entry: the stride separating matrix column elements in the array **q**.

Constraint: $\mathbf{tdq} \geq \mathbf{ip} + 1$.

9: **svd** – Nag_Boolean *Input*

On entry: indicates if a singular value decomposition was used by nag_regsn_mult_linear (g02dac).

svd = Nag_TRUE
A singular value decomposition was used by nag_regsn_mult_linear (g02dac).

svd = Nag_FALSE
A singular value decomposition was not used by nag_regsn_mult_linear (g02dac).

10: $\mathbf{p}[\mathbf{2} \times \mathbf{ip} + \mathbf{ip} \times \mathbf{ip}]$ – const double *Input*

On entry: details of the QR decomposition and SVD, if used, as returned in array **p** by nag_regsn_mult_linear (g02dac).

If **svd** = Nag_FALSE, only the first **ip** elements of **p** are used, these will contain details of the Householder vector in the QR decomposition (Sections 2.2.1 and 3.3.6 in the f08 Chapter Introduction).

If **svd** = Nag_TRUE, the first **ip** elements of **p** will contain details of the Householder vector in the QR decomposition (Sections 2.2.1 and 3.3.6 in the f08 Chapter Introduction) and the next **ip** elements of **p** contain singular values. The following **ip** by **ip** elements contain the matrix P^* stored by rows.

11: **y**[**n**] – const double *Input*

On entry: the new dependent variable y_{new}.

12: **b**[**ip**] – double *Output*

On exit: $\mathbf{b}[i]$, $i = 0, 1, \dots, \mathbf{ip} - 1$ contain the least squares estimates of the arguments of the regression model, $\hat{\beta}$.

13: **se**[**ip**] – double *Output*

On exit: $\mathbf{se}[i]$, $i = 0, 1, \dots, \mathbf{ip} - 1$ contain the standard errors of the **ip** parameter estimates given in **b**.

14: **res**[**n**] – double *Output*

On exit: the residuals for the new regression model.

15: $\mathbf{com\_ar}[\mathbf{5} \times (\mathbf{ip} - \mathbf{1}) \times \mathbf{ip}]$ – const double *Input*

On entry: if **svd** = Nag_TRUE, **com_ar** must be unaltered from the previous call to nag_regsn_mult_linear (g02dac).

16: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_2_INT_ARG_LT

On entry, $\mathbf{n} = \langle value \rangle$ while $\mathbf{ip} = \langle value \rangle$. These arguments must satisfy $\mathbf{n} \geq \mathbf{ip}$.

On entry, **tdq** $= \langle value \rangle$ while **ip** $+ 1 = \langle value \rangle$. These arguments must satisfy **tdq** $\geq$ **ip** $+ 1$.

NE_INT_ARG_LE

On entry, **rank** $= \langle value \rangle$.
Constraint: **rank** > 0.

NE_INT_ARG_LT

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

NE_REAL_ARG_LE

On entry, **rss** must not be less than or equal to 0.0: **rss** $= \langle value \rangle$.

NE_REAL_ARG_LT

On entry, **wt**$[\langle value \rangle]$ must not be less than 0.0: **wt**$[\langle value \rangle] = \langle value \rangle$.

NE_SVD_RANK_GT_IP

On entry, the Boolean variable, **svd**, is Nag_TRUE and **rank** must not be greater than **ip**: **rank** $= \langle value \rangle$, **ip** $= \langle value \rangle$.

NE_SVD_RANK_NE_IP

On entry, the Boolean variable, **svd**, is Nag_FALSE and **rank** must be equal to **ip**: **rank** $= \langle value \rangle$, **ip** $= \langle value \rangle$.

7 Accuracy

The same accuracy as nag_regsn_mult_linear (g02dac) is obtained.

8 Parallelism and Performance

nag_regsn_mult_linear_newyvar (g02dgc) is not threaded in any implementation.

9 Further Comments

The values of the leverages, h_i, are unaltered by a change in the dependent variable so a call to nag_regsn_std_resid_influence (g02fac) can be made using the value of **h** from nag_regsn_mult_linear (g02dac).

10 Example

A dataset consisting of 12 observations with four independent variables and two dependent variables is read in. A model with all four independent variables is fitted to the first dependent variable by nag_regsn_mult_linear (g02dac) and the results printed. The model is then fitted to the second dependent variable by nag_regsn_mult_linear_newyvar (g02dgc) and those results printed.

10.1 Program Text

```
/* nag_regsn_mult_linear_newyvar (g02dgc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
```

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

#define XM(I, J) xm[(I) *tdxm + J]

int main(void)
{
  Nag_Boolean svd;
  Integer exit_status = 0, i, ip, j, m, n, rank, *sx = 0, tdq, tdxm;
  NagError fail;
  Nag_IncludeMean mean;
  Nag_Boolean weight;
  char nag_enum_arg[40];
  double df, rss, tol;
  double *b = 0, *com_ar = 0, *cov = 0, *h = 0, *newy = 0, *p = 0;
  double *q = 0, *res = 0, *se = 0, *wt = 0, *wtptr, *xm = 0, *y = 0;

  INIT_FAIL(fail);

  printf("nag_regsn_mult_linear_newyvar (g02dgc) Example Program Results\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT " %" NAG_IFMT "", &n, &m);
#else
  scanf("%" NAG_IFMT " %" NAG_IFMT "", &n, &m);
#endif
#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  weight = (Nag_Boolean) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%39s%*[^\n] ", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n] ", nag_enum_arg);
#endif
  mean = (Nag_IncludeMean) nag_enum_name_to_value(nag_enum_arg);
  if (n >= 2 && m >= 1) {
    if (!(h = NAG_ALLOC(n, double)) ||
        !(newy = NAG_ALLOC(n, double)) ||
        !(res = NAG_ALLOC(n, double)) ||
        !(wt = NAG_ALLOC(n, double)) ||
        !(xm = NAG_ALLOC(n * m, double)) ||
        !(y = NAG_ALLOC(n, double)) || !(sx = NAG_ALLOC(m, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
    tdxm = m;
  }
  else {
    printf("Invalid n or m.\n");
    exit_status = 1;
    return exit_status;
  }
  if (weight) {
    wtptr = wt;
    for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++)
#ifdef _WIN32
        scanf_s("%lf", &XM(i, j));
#else
        scanf("%lf", &XM(i, j));
#endif
#ifdef _WIN32
      scanf_s("%lf%lf%lf", &y[i], &wt[i], &newy[i]);
#else
      scanf("%lf%lf%lf", &y[i], &wt[i], &newy[i]);
#endif
    }
  }
  else {
    wtptr = (double *) 0;
    for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++)
#ifdef _WIN32
        scanf_s("%lf", &XM(i, j));
#else
        scanf("%lf", &XM(i, j));
#endif
#ifdef _WIN32
      scanf_s("%lf%lf", &y[i], &newy[i]);
#else
      scanf("%lf%lf", &y[i], &newy[i]);
#endif
    }
  }
  for (j = 0; j < m; j++)
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &sx[j]);
#else
    scanf("%" NAG_IFMT "", &sx[j]);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "", &ip);
#else
  scanf("%" NAG_IFMT "", &ip);
#endif

  if (!(b = NAG_ALLOC(ip, double)) ||
      !(cov = NAG_ALLOC(ip * (ip + 1) / 2, double)) ||
      !(p = NAG_ALLOC(ip * (ip + 2), double)) ||
      !(q = NAG_ALLOC(n * (ip + 1), double)) ||
      !(se = NAG_ALLOC(ip, double)) ||
      !(com_ar = NAG_ALLOC(5 * (ip - 1) + ip * ip, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  tdq = ip + 1;

  /* Set tolerance */
  tol = 0.00001e0;
  /* Fit initial model using nag_regsn_mult_linear (g02dac) */
  /* nag_regsn_mult_linear (g02dac).
   * Fits a general (multiple) linear regression model
   */
  nag_regsn_mult_linear(mean, n, xm, tdxm, m, sx, ip,
                        y, wtptr, &rss, &df, b, se, cov, res, h, q,
                        tdq, &svd, &rank, p, tol, com_ar, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_regsn_mult_linear (g02dac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  printf("Results from g02dac\n\n");
  if (svd)
    printf("Model not of full rank\n\n");
  printf("Residual sum of squares = %13.4e\n", rss);
  printf("Degrees of freedom = %3.1f\n\n", df);
  printf("Variable   Parameter estimate   Standard error\n\n");
  for (j = 0; j < ip; j++)
    printf("%6" NAG_IFMT "%20.4e%20.4e\n", j + 1, b[j], se[j]);
  printf("\n");

  /* nag_regsn_mult_linear_newyvar (g02dgc).
   * Fits a general linear regression model to new dependent
   * variable
   */
  nag_regsn_mult_linear_newyvar(n, wtptr, &rss, ip, rank, cov, q, tdq, svd, p,
                                newy, b, se, res, com_ar, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_regsn_mult_linear_newyvar (g02dgc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  printf("\n");
  printf("Results for second y-variable using "
         "nag_regsn_mult_linear_newyvar (g02dgc)\n\n");
  printf("Residual sum of squares = %13.4e\n", rss);
  printf("Degrees of freedom = %3.1f\n\n", df);
  printf("Variable   Parameter estimate   Standard error\n\n");
  for (j = 0; j < ip; j++)
    printf("%6" NAG_IFMT "%20.4e%20.4e\n", j + 1, b[j], se[j]);
  printf("\n");

END:
  NAG_FREE(h);
  NAG_FREE(newy);
  NAG_FREE(res);
  NAG_FREE(wt);
  NAG_FREE(xm);
  NAG_FREE(y);
  NAG_FREE(sx);
  NAG_FREE(b);
  NAG_FREE(cov);
  NAG_FREE(p);
  NAG_FREE(q);
  NAG_FREE(se);
  NAG_FREE(com_ar);

  return exit_status;
}
```

10.2 Program Data

```
nag_regsn_mult_linear_newyvar (g02dgc) Example Program Data
 12 4 Nag_FALSE Nag_MeanInclude
1.0 0.0 0.0 0.0 33.63 63.0
0.0 0.0 0.0 1.0 39.62 69.0
0.0 1.0 0.0 0.0 38.18 68.0
0.0 0.0 1.0 0.0 41.46 71.0
0.0 0.0 0.0 1.0 38.02 68.0
0.0 1.0 0.0 0.0 35.83 65.0
0.0 0.0 0.0 1.0 35.99 65.0
1.0 0.0 0.0 0.0 36.58 66.0
0.0 0.0 1.0 0.0 42.92 72.0
1.0 0.0 0.0 0.0 37.80 67.0
0.0 0.0 1.0 0.0 40.43 70.0
0.0 1.0 0.0 0.0 37.89 67.0
 1   1   1   1   5
```

10.3 Program Results

```
nag_regsn_mult_linear_newyvar (g02dgc) Example Program Results
Results from g02dac

Model not of full rank

Residual sum of squares =    2.2227e+01
Degrees of freedom = 8.0

Variable   Parameter estimate   Standard error

     1         3.0557e+01         3.8494e-01
     2         5.4467e+00         8.3896e-01
     3         6.7433e+00         8.3896e-01
     4         1.1047e+01         8.3896e-01
     5         7.3200e+00         8.3896e-01


Results for second y-variable using nag_regsn_mult_linear_newyvar (g02dgc)

Residual sum of squares =    2.4000e+01
Degrees of freedom = 8.0

Variable   Parameter estimate   Standard error

     1         5.4067e+01         4.0000e-01
     2         1.1267e+01         8.7178e-01
     3         1.2600e+01         8.7178e-01
     4         1.6933e+01         8.7178e-01
     5         1.3267e+01         8.7178e-01
```