NAG Library Function Document

nag_regsn_mult_linear (g02dac)

1 Purpose

nag_regsn_mult_linear (g02dac) performs a general multiple linear regression when the independent variables may be linearly dependent. Parameter estimates, standard errors, residuals and influence statistics are computed. nag_regsn_mult_linear (g02dac) may be used to perform a weighted regression.

2 Specification

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

void nag_regsn_mult_linear (Nag_IncludeMean mean, Integer n,
     const double x[], Integer tdx, Integer m, const Integer sx[],
     Integer ip, const double y[], const double wt[], double *rss,
     double *df, double b[], double se[], double cov[], double res[],
     double h[], double q[], Integer tdq, Nag_Boolean *svd, Integer *rank,
     double p[], double tol, double com_ar[], NagError *fail)
```

3 Description

The general linear regression model is defined by

$$y = X\beta + \epsilon$$

where

y is a vector of n observations on the dependent variable,

X is an n by p matrix of the independent variables of column rank k,

β is a vector of length p of unknown arguments, and

ϵ is a vector of length n of unknown random errors such that $\mathrm{var}\,\epsilon = V\sigma^2$, where V is a known diagonal matrix.

Note: the p independent variables may be selected from a set of m potential independent variables.

If $V = I$, the identity matrix, then least squares estimation is used.

If $V \neq I$, then for a given weight matrix $W \propto V^{-1}$, weighted least squares estimation is used.

The least squares estimates $\hat{\beta}$ of the arguments β minimize $(y - X\beta)^{\mathrm{T}}(y - X\beta)$ while the weighted least squares estimates minimize $(y - X\beta)^{\mathrm{T}}W(y - X\beta)$.

nag_regsn_mult_linear (g02dac) finds a QR decomposition of X (or $W^{1/2}X$ in the weighted case), i.e.,

$$X = QR^* \quad \left(\text{or } W^{1/2}X = QR^*\right)$$

where $R^* = \begin{pmatrix} R \\ 0 \end{pmatrix}$ and R is a p by p upper triangular matrix and Q is an n by n orthogonal matrix.

If R is of full rank, then $\hat{\beta}$ is the solution to

$$R\hat{\beta} = c_1$$

where $c = Q^{\mathrm{T}}y$ (or $Q^{\mathrm{T}}W^{1/2}y$) and c_1 is the first p elements of c.

If R is not of full rank a solution is obtained by means of a singular value decomposition (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, in the form of the matrix P^*:

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

This will be only one of the possible solutions. Other estimates may be obtained by applying constraints to the arguments. These solutions can be obtained by using nag_regsn_mult_linear_tran_model (g02dkc) after using nag_regsn_mult_linear (g02dac). Only certain linear combinations of the arguments will have unique estimates; these are known as estimable functions.

The fit of the model can be examined by considering the residuals, $r_i = y_i - \hat{y}$, where $\hat{y} = X\hat{\beta}$ are the fitted values. The fitted values can be written as Hy for an n by n matrix H. The ith diagonal element of H, h_i, gives a measure of the influence of the ith value of the independent variables on the fitted regression model. The values h_i are sometimes known as leverages. Both r_i and h_i are provided by nag_regsn_mult_linear (g02dac).

The output of nag_regsn_mult_linear (g02dac) also includes $\hat{\beta}$, the residual sum of squares and associated degrees of freedom, $(n - k)$, the standard errors of the parameter estimates and the variance-covariance matrix of the parameter estimates.

In many linear regression models the first term is taken as a mean term or an intercept, i.e., $X_{i,1} = 1$, for $i = 1, 2, \ldots, n$. This is provided as an option. Also note that not all the potential independent variables need to be included in a model; a facility to select variables to be included in the model is provided.

Details of the QR decomposition and, if used, the SVD, are made available. These allow the regression to be updated by adding or deleting an observation using nag_regsn_mult_linear_addrem_obs (g02dcc), adding or deleting a variable using nag_regsn_mult_linear_add_var (g02dec) and nag_regsn_mult_linear_delete_var (g02dfc) or estimating and testing an estimable function using nag_regsn_mult_linear_est_func (g02dnc).

4 References

Cook R D and Weisberg S (1982) *Residuals and Influence in Regression* Chapman and Hall

Draper N R and Smith H (1985) *Applied Regression Analysis* (2nd Edition) Wiley

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

McCullagh P and Nelder J A (1983) *Generalized Linear Models* Chapman and Hall

Searle S R (1971) *Linear Models* Wiley

5 Arguments

1: **mean** – Nag_IncludeMean *Input*

On entry: indicates if a mean term is to be included.

mean = Nag_MeanInclude
A mean term, (intercept), will be included in the model.

mean = Nag_MeanZero
The model will pass through the origin, zero point.

Constraint: **mean** = Nag_MeanInclude or Nag_MeanZero.

2: **n** – Integer *Input*

On entry: the number of observations, n.

Constraint: **n** ≥ 2.

3: **x**[**n** × **tdx**] – const double *Input*

On entry: **x**$[(i) \times$ **tdx** $+ j]$ must contain the ith observation for the jth potential independent variable, for $i = 0, 1, \ldots, n - 1$ and $j = 0, 1, \ldots, m - 1$.

4: **tdx** – Integer *Input*

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

Constraint: **tdx** $\geq$ **m**.

5: **m** – Integer *Input*

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

Constraint: **m** ≥ 1.

6: **sx**[**m**] – const Integer *Input*

On entry: indicates which of the potential independent variables are to be included in the model. If **sx**$[j] > 0$, then the variable contained in the corresponding column of **x** is included in the regression model.

Constraints:

sx$[j] \geq 0$, for $j = 0, 1, \ldots, m - 1$;
if **mean** = Nag_MeanInclude, then exactly **ip** $- 1$ values of **sx** must be > 0;
if **mean** = Nag_MeanZero, then exactly **ip** values of **sx** must be > 0.

7: **ip** – Integer *Input*

On entry: the number p of independent variables in the model, including the mean or intercept if present.

Constraints:

if **mean** = Nag_MeanInclude, $1 \leq$ **ip** $\leq$ **m** $+ 1$;
if **mean** = Nag_MeanZero, $1 \leq$ **ip** $\leq$ **m**.

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

On entry: observations on the dependent variable, y.

9: **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.

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 $i = 1, 2, \ldots, n$.

10: **rss** – double * *Output*

On exit: the residual sum of squares for the regression.

11: **df** – double * *Output*

On exit: the degrees of freedom associated with the residual sum of squares.

12: **b**[**ip**] – double *Output*

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

If **mean** = Nag_MeanInclude, then $\mathbf{b}[0]$ will contain the estimate of the mean argument and $\mathbf{b}[i]$ will contain the coefficient of the variable contained in column j of **x**, where $\mathbf{sx}[j]$ is the ith positive value in the array **sx**.

If **mean** = Nag_MeanZero, then $\mathbf{b}[i-1]$ will contain the coefficient of the variable contained in column j of **x**, where $\mathbf{sx}[j]$ is the ith positive value in the array **sx**.

13: **se**[**ip**] – double *Output*

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

14: **cov**[**ip** × (**ip** + **1**)/**2**] – double *Output*

On exit: the first $\mathbf{ip} \times (\mathbf{ip} + 1)/2$ elements of **cov** contain 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 $j = i, \ldots, \mathbf{ip} - 1$.

15: **res**[**n**] – double *Output*

On exit: the (weighted) residuals, r_i.

16: **h**[**n**] – double *Output*

On exit: the diagonal elements of H, h_i, the leverages.

17: **q**[**n** × **tdq**] – double *Output*

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

On exit: the results of the QR decomposition: the first column of **q** contains c, the upper triangular part of columns 2 to $\mathbf{ip} + 1$ contain the R matrix, the strictly lower triangular part of columns 2 to $\mathbf{ip} + 1$ contain details of the Q matrix.

18: **tdq** – Integer *Input*

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

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

19: **svd** – Nag_Boolean * *Output*

On exit: if a singular value decomposition has been performed then **svd** will be Nag_TRUE, otherwise **svd** will be Nag_FALSE.

20: **rank** – Integer * *Output*

On exit: the rank of the independent variables.

If **svd** = Nag_FALSE, **rank** = **ip**.

If **svd** = Nag_TRUE, **rank** is an estimate of the rank of the independent variables. **rank** is calculated as the number of singular values greater than **tol** (largest singular value). It is possible for the SVD to be carried out but **rank** to be returned as **ip**.

21: **p**[**2** × **ip** + **ip** × **ip**] – double *Output*

On exit: details of the QR decomposition and SVD if used.

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 (see 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 and the next **ip** elements of **p** contain singular values. The following **ip** by **ip** elements contain the matrix P^* stored by rows.

22: **tol** – double *Input*

On entry: the value of **tol** is used to decide what is the rank of the independent variables. The smaller the value of **tol** the stricter the criterion for selecting the singular value decomposition. If **tol** = 0.0, then the singular value decomposition will never be used, this may cause run time errors or inaccurate results if the independent variables are not of full rank.

Suggested value: **tol** = 0.000001.

Constraint: **tol** ≥ 0.0.

23: **com_ar**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **com_ar** must be at least 5 × (**ip** − 1) × **ip** × **ip**.

On exit: if **svd** = Nag_TRUE, **com_ar** contains information which is needed by nag_regsn_mult_linear_newyvar (g02dgc).

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_2_INT_ARG_LT

On entry, **n** = ⟨*value*⟩ while **ip** = ⟨*value*⟩. These arguments must satisfy **n** ≥ **ip**.

On entry, **tdq** = ⟨*value*⟩ while **ip** + 1 = ⟨*value*⟩. These arguments must satisfy **tdq** ≥ **ip** + 1.

On entry, **tdx** = ⟨*value*⟩ while **m** = ⟨*value*⟩. These arguments must satisfy **tdx** ≥ **m**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument **mean** had an illegal value.

NE_BAD_SX_OR_IP

Either a value of **sx** is < 0, or **ip** is incompatible with **mean** and **sx**, or **ip** $>$ the effective number of observations.

NE_INT_ARG_LT

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

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

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

On entry, **sx**[$\langle value \rangle$] must not be less than 0: **sx**[$\langle value \rangle$] $= \langle value \rangle$.

NE_REAL_ARG_LT

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

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

NE_SVD_NOT_CONV

The singular value decomposition has failed to converge.

NE_ZERO_DOF_RESID

The degrees of freedom for the residuals are zero, i.e., the designated number of arguments $=$ the effective number of observations. In this case the parameter estimates will be returned along with the diagonal elements of H, but neither standard errors nor the variance-covariance matrix will be calculated.

7 Accuracy

The accuracy of this function is closely related to the accuracy of the QR decomposition.

8 Parallelism and Performance

Not applicable.

9 Further Comments

Function nag_regsn_std_resid_influence (g02fac) can be used to compute standardized residuals and further measures of influence. nag_regsn_mult_linear (g02dac) requires, in particular, the results stored in **res** and **h**.

10 Example

For this function two examples are presented. There is a single example program for nag_regsn_mult_linear (g02dac), with a main program and the code to solve the two example problems is given in the functions ex1 and ex2.

Example 1 (ex1)

Data from an experiment with four treatments and three observations per treatment are read in. The treatments are represented by dummy $(0 - 1)$ variables. An unweighted model is fitted with a mean included in the model.

Example 2 (ex2)

This example program uses nag_regsn_mult_linear (g02dac) to find the coefficient of the n degree polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots a_1 x + a_o$$

that fits the data, $p(x(i))$ to $y(i)$, in a least squares sense.

In this example nag_regsn_mult_linear (g02dac) is called with both **mean** = Nag_MeanInclude and **mean** = Nag_MeanZero. The polynomial degree, the number of data points and the tolerance can be modified using the example data file.

10.1 Program Text

```
/* nag_regsn_mult_linear (g02dac) Example Program.
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5 revised, 1998.
 * Mark 6 revised, 2000.
 * Mark 8 revised, 2004.
 */

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

static int ex1(void);
static int ex2(void);

int main(void)
{
  Integer  exit_status_ex1 = 0;
  Integer  exit_status_ex2 = 0;

  printf("nag_regsn_mult_linear (g02dac) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");

  exit_status_ex1 = ex1();
  exit_status_ex2 = ex2();

  return (exit_status_ex1 == 0 && exit_status_ex2 == 0) ? 0 : 1;
}

#define X(I, J) x[(I) *tdx + J]
#define Q(I, J) q[(I) *tdq + J]

static int ex1(void)
{
  Integer         exit_status = 0, i, ip, j, m, n, rank, *sx = 0, tdq, tdx;
  char            nag_enum_arg[40];
  double          *b = 0, *com_ar = 0, *cov = 0, df, *h = 0, *p = 0, *q = 0;
  double          *res = 0, rss, *se = 0, tol, *wt = 0, *wtptr, *x = 0, *y = 0;
  Nag_Boolean     svd, weight;
  Nag_IncludeMean mean;
  NagError        fail;

  INIT_FAIL(fail);

  printf("Example 1\n");
  /* Skip heading in data file */
  scanf("%*[^\n]");
  scanf("%ld %ld", &n, &m);
  scanf(" %39s", nag_enum_arg);
  /* nag_enum_name_to_value (x04nac).
```

```
 * Converts NAG enum member name to value
 */
weight = (Nag_Boolean) nag_enum_name_to_value(nag_enum_arg);
scanf(" %39s", nag_enum_arg);
mean = (Nag_IncludeMean) nag_enum_name_to_value(nag_enum_arg);
if (n >= 2 && m >= 1)
  {
    if (!(h = NAG_ALLOC(n, double)) ||
        !(res = NAG_ALLOC(n, double)) ||
        !(wt = NAG_ALLOC(n, double)) ||
        !(x = NAG_ALLOC(n*m, double)) ||
        !(y = NAG_ALLOC(n, double)) ||
        !(sx = NAG_ALLOC(m, Integer)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
    tdx = 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++)
          scanf("%lf", &X(i, j));
        scanf("%lf%lf", &y[i], &wt[i]);
      }
  }
else
  {
    wtptr = (double *) 0;
    for (i = 0; i < n; i++)
      {
        for (j = 0; j < m; j++)
          scanf("%lf", &X(i, j));
        scanf("%lf", &y[i]);
      }
  }
for (j = 0; j < m; j++)
  scanf("%ld", &sx[j]);
/* Calculate ip */
ip = 0;
if (mean == Nag_MeanInclude)
  ip += 1;
for (i = 0; i < m; i++)
  if (sx[i] > 0) ip += 1;

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

/* Set tolerance */
tol = 0.00001e0;
/* nag_regsn_mult_linear (g02dac).
```

```
   * Fits a general (multiple) linear regression model
   */
  nag_regsn_mult_linear(mean, n, x, tdx, 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;
    }

  if (svd)
    printf("Model not of full rank, rank = %4ld\n\n", rank);
  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("%6ld%20.4e%20.4e\n", j+1, b[j], se[j]);
  printf("\n");
  printf("    Obs         Residuals              h\n\n");
  for (i = 0; i < n; i++)
    printf("%6ld%20.4e%20.4e\n", i+1, res[i], h[i]);

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

  return exit_status;
}
#undef x
#undef q

#define X(I, J) x[(I) *tdx + J]
#define Q(I, J) q[(I) *tdq + J]
static int ex2(void)
{
  Integer         exit_status = 0;
  double          rss, tol;
  Integer         i, ip, rank, j, m, mmax, n, degree, digits, tdx, tdq;
  double          df;
  Nag_Boolean     svd;
  Nag_IncludeMean mean;
  double          *h = 0, *res = 0, *wt = 0, *x = 0, *y = 0;
  double          *b = 0, *cov = 0, *p = 0, *q = 0, *com_ar = 0, *se = 0;
  double          *wtptr = (double *) 0; /* don't use weights */
  Integer         *sx = 0;
  NagError        fail;

  INIT_FAIL(fail);

  printf(
          "\n\n\nExample 2\n");
  /* Skip heading in data file */
  scanf(" %*[^\n]");

  /* Use mean = Nag_MeanInclude */

  mean = Nag_MeanInclude;
  scanf("%ld%ld%ld", &degree, &n, &digits);
```

```
mmax = degree+1;
if (n >= 1)
  {
    if (!(h = NAG_ALLOC(n, double)) ||
        !(res = NAG_ALLOC(n, double)) ||
        !(wt = NAG_ALLOC(n, double)) ||
        !(x = NAG_ALLOC(n*mmax, double)) ||
        !(y = NAG_ALLOC(n, double)) ||
        !(sx = NAG_ALLOC(mmax, Integer)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
    tdx = mmax;
  }
else
  {
    printf("Invalid n.\n");
    exit_status = 1;
    return exit_status;
  }

/* Set tolerance */
tol = pow(10.0, -(double) digits);
m = degree;
ip = degree + 1;
if (!(b = NAG_ALLOC(ip, double)) ||
    !(cov = NAG_ALLOC((ip*ip+ip)/2, double)) ||
    !(p = NAG_ALLOC(ip*(ip+2), double)) ||
    !(q = NAG_ALLOC(n*(ip+1), double)) ||
    !(com_ar = NAG_ALLOC(ip*ip+5*(ip-1), double)) ||
    !(se = NAG_ALLOC(ip, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
tdq = ip+1;

for (i = 0; i < ip-1; ++i)
  sx[i] = 1;

for (i = 0; i < n; i++)
  {
    scanf("%lf%lf", &X(i, degree-1), &y[i]);
    for (j = 0; j < degree; ++j)
      X(i, j) = pow(X(i, degree-1), (double)(degree-j));
  }

/* nag_regsn_mult_linear (g02dac), see above. */
nag_regsn_mult_linear(mean, n, x, tdx, 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("Regression estimates  (mean = Nag_MeanInclude) \n\n");
printf("Coefficient    Estimate         Standard error\n\n");
for (j = 1; j < ip; j++)
  printf("a(%ld)%20.4e%20.4e\n", degree+1-j, b[j], se[j]);
printf("a(0)%20.4e%20.4e\n", b[0], se[0]);
printf("\n\n");

/* Use mean = Nag_MeanZero */
```

```
  mean = Nag_MeanZero;

  m = degree + 1;
  for (i = 0; i < ip; ++i)
    sx[i] = 1;

  for (i = 0; i < n; i++)
    X(i, m-1) = 1.0;

  /* nag_regsn_mult_linear (g02dac), see above. */
  nag_regsn_mult_linear(mean, n, x, tdx, 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("Regression estimates  (mean = Nag_MeanZero) \n\n");
  printf("Coefficient    Estimate         Standard error\n\n");
  for (j = 0; j < ip; j++)
    printf("a(%ld)%20.4e%20.4e\n", degree-j, b[j], se[j]);
  printf("\n\n");

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

  return exit_status;
}
```

10.2 Program Data

```
nag_regsn_mult_linear (g02dac) Example Program Data
Example 1
 12 4  Nag_FALSE  Nag_MeanInclude
1.0 0.0 0.0 0.0 33.63
0.0 0.0 0.0 1.0 39.62
0.0 1.0 0.0 0.0 38.18
0.0 0.0 1.0 0.0 41.46
0.0 0.0 0.0 1.0 38.02
0.0 1.0 0.0 0.0 35.83
0.0 0.0 0.0 1.0 35.99
1.0 0.0 0.0 0.0 36.58
0.0 0.0 1.0 0.0 42.92
1.0 0.0 0.0 0.0 37.80
0.0 0.0 1.0 0.0 40.43
0.0 1.0 0.0 0.0 37.89
 1   1   1   1
Example 2
3 11 15
 31.80  -1.23
 50.20   -1.08
120.00  -0.83
188.84  -0.53
250.20  -0.28
270.66  -0.15
```

```
360.20   0.26
392.97   0.53
444.54   0.93
530.50   1.08
550.02   1.35
```

10.3 Program Results

```
nag_regsn_mult_linear (g02dac) Example Program Results

Example 1
Model not of full rank, rank =    4

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

    Obs          Residuals              h

     1         -2.3733e+00          3.3333e-01
     2          1.7433e+00          3.3333e-01
     3          8.8000e-01          3.3333e-01
     4         -1.4333e-01          3.3333e-01
     5          1.4333e-01          3.3333e-01
     6         -1.4700e+00          3.3333e-01
     7         -1.8867e+00          3.3333e-01
     8          5.7667e-01          3.3333e-01
     9          1.3167e+00          3.3333e-01
    10          1.7967e+00          3.3333e-01
    11         -1.1733e+00          3.3333e-01
    12          5.9000e-01          3.3333e-01


Example 2
Regression estimates  (mean = Nag_MeanInclude)

Coefficient    Estimate         Standard error

a(3)         -8.8628e-09          7.9470e-09
a(2)          9.0059e-06          7.0244e-06
a(1)          2.3641e-03          1.7199e-03
a(0)         -1.2614e+00          1.0568e-01


Regression estimates  (mean = Nag_MeanZero)

Coefficient    Estimate         Standard error

a(3)         -8.8628e-09          7.9470e-09
a(2)          9.0059e-06          7.0244e-06
a(1)          2.3641e-03          1.7199e-03
a(0)         -1.2614e+00          1.0568e-01
```
