NAG Library Function Document

nag_regsn_mult_linear_est_func (g02dnc)

1 Purpose

nag_regsn_mult_linear_est_func (g02dnc) gives the estimate of an estimable function along with its standard error.

2 Specification

```
#include <nag.h>
#include <nagg02.h>
void nag_regsn_mult_linear_est_func (Integer ip, Integer rank,
     const double b[], const double cov[], const double p[],
     const double f[], Nag_Boolean *est, double *stat, double *sestat,
     double *t, double tol, NagError *fail)
```

3 Description

This function computes the estimates of an estimable function for a general linear regression model which is not of full rank. It is intended for use after a call to nag_regsn_mult_linear (g02dac) or nag_regsn_mult_linear_upd_model (g02ddc). An estimable function is a linear combination of the arguments such that it has a unique estimate. For a full rank model all linear combinations of arguments are estimable.

In the case of a model not of full rank the functions use a singular value decomposition (SVD) to find the parameter estimates, $\hat{\beta}$, and their variance-covariance matrix. Given the upper triangular matrix R obtained from the QR decomposition of the independent variables the SVD gives:

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

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 leads to a solution:

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

P_1 being the first k columns of P, i.e., $P = (P_1 P_0)$, Q_{*_1} being the first k columns of Q_* and c_1 being the first p elements of c.

Details of the SVD are made available, in the form of the matrix P^*:

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

as given by nag_regsn_mult_linear (g02dac) and nag_regsn_mult_linear_upd_model (g02ddc).

A linear function of the arguments, $F = f^{\mathrm{T}}\beta$, can be tested to see if it is estimable by computing $\zeta = P_0^{\mathrm{T}} f$. If ζ is zero, then the function is estimable, if not, the function is not estimable. In practice $|\zeta|$ is tested against some small quantity η.

Given that F is estimable it can be estimated by $f^{\mathrm{T}}\hat{\beta}$ and its standard error calculated from the variance-covariance matrix of $\hat{\beta}$, C_β, as

$$\mathrm{se}(F) = \sqrt{f^{\mathrm{T}} C_\beta f}$$

Also a t-statistic:

$$t = \frac{f^{\mathrm{T}}\hat{\beta}}{\mathrm{se}(F)},$$

can be computed. The t-statistic will have a Student's t-distribution with degrees of freedom as given by the degrees of freedom for the residual sum of squares for the model.

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: **ip** – Integer *Input*

On entry: the number of terms in the linear model, p.

Constraint: **ip** ≥ 1.

2: **rank** – Integer *Input*

On entry: the rank of the independent variables, k.

Constraint: $1 \leq$ **rank** $\leq$ **ip**.

3: **b**[**ip**] – const double *Input*

On entry: the **ip** values of the estimates of the arguments of the model, $\hat{\beta}$.

4: **cov**[**ip** $\times$ (**ip** + **1**)/**2**] – const double *Input*

On entry: 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 **b**$[i]$ and the parameter estimate given in **b**$[j]$, $j \geq i$, is stored in **cov**$[j(j+1)/2+i]$, for $i = 0, 1, \ldots,$ **ip** $- 1$ and $j = i, \ldots,$ **ip** $- 1$.

5: **p**[**ip** $\times$ **ip** + **2** $\times$ **ip**] – const double *Input*

On entry: **p** as returned by nag_regsn_mult_linear (g02dac) or nag_regsn_mult_linear_upd_model (g02ddc).

6: **f**[**ip**] – const double *Input*

On entry: the linear function to be estimated, f.

7: **est** – Nag_Boolean * *Output*

On exit: **est** indicates if the function was estimable.

est = Nag_TRUE
The function is estimable.

est = Nag_FALSE
The function is not estimable and **stat**, **sestat** and **t** are not set.

8: **stat** – double * *Output*

On exit: if **est** = Nag_TRUE, **stat** contains the estimate of the function, $f^{\mathrm{T}}\hat{\beta}$.

9: **sestat** – double * *Output*

On exit: if **est** = Nag_TRUE, **sestat** contains the standard error of the estimate of the function, se(F).

10: **t** – double * *Output*

On exit: if **est** = Nag_TRUE, **t** contains the t-statistic for the test of the function being equal to zero.

11: **tol** – double *Input*

On entry: **tol** is the tolerance value used in the check for estimability, η. If **tol** ≤ 0.0, $\sqrt{\textbf{\textit{machine precision}}}$ is used instead.

12: **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_GT

On entry, **ip** = ⟨*value*⟩ while **rank** = ⟨*value*⟩. These arguments must satisfy **rank** $\leq$ **ip**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_INT_ARG_LT

On entry, **ip** = ⟨*value*⟩.
Constraint: **ip** ≥ 1.

On entry, **rank** = ⟨*value*⟩.
Constraint: **rank** ≥ 1.

NE_RANK_EQ_IP

On entry, **rank** = **ip**. In this case, the boolean variable **est** is returned as Nag_TRUE and all statistics are calculated.

NE_STDES_ZERO

se(F) = 0.0 probably due to rounding error or due to incorrectly specified inputs **cov** and **f**.

7 Accuracy

The computations are believed to be stable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The value of estimable functions is independent of the solution chosen from the many possible solutions. While nag_regsn_mult_linear_est_func (g02dnc) may be used to estimate functions of the arguments of the model as computed by nag_regsn_mult_linear_tran_model (g02dkc), β_c, these must be expressed in terms of the original arguments, β. The relation between the two sets of arguments may not be straightforward.

10 Example

Data from an experiment with four treatments and three observations per treatment are read in. A model, with a mean term, is fitted by nag_regsn_mult_linear (g02dac). The number of functions to be tested is read in, then the linear functions themselves are read in and tested with nag_regsn_mult_linear_est_func (g02dnc). The results of nag_regsn_mult_linear_est_func (g02dnc) are printed.

10.1 Program Text

```
/* nag_regsn_mult_linear_est_func (g02dnc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 * Mark 8 revised, 2004.
 */

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

#define X(I, J) x[(I) *tdx + J]
#define Q(I, J) q[(I) *tdq + J]
int main(void)
{
  Integer         exit_status = 0, i, ip, j, m, n, nestern, rank, *sx = 0, tdq,
                  tdx;
  double          *b = 0, *com_ar = 0, *cov = 0, df, *f = 0, *h = 0, *p = 0;
  double          *q = 0, *res = 0, rss, *se = 0, sestat, stat, t, tol;
  double          *wt = 0, *wtptr, *x = 0, *y = 0;
  char            nag_enum_arg[40];
  Nag_Boolean     est, svd, weight;
  Nag_IncludeMean mean;
  NagError        fail;

  INIT_FAIL(fail);

  printf(
          "nag_regsn_mult_linear_est_func (g02dnc) 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, _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", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf(" %39s", 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)) ||
          !(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++)
#ifdef _WIN32
            scanf_s("%lf", &X(i, j));
#else
            scanf("%lf", &X(i, j));
#endif
#ifdef _WIN32
          scanf_s("%lf%lf", &y[i], &wt[i]);
#else
          scanf("%lf%lf", &y[i], &wt[i]);
#endif
        }
    }
  else
    {
      wtptr = (double *) 0;
      for (i = 0; i < n; i++)
        {
          for (j = 0; 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]);
#else
          scanf("%lf", &y[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)) ||
      !(f = NAG_ALLOC(ip, double)) ||
      !(p = NAG_ALLOC(ip*(ip+2), double)) ||
```

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


  /* Set tolerance */
  tol = 0.00001e0;

  /*
   *    Find initial estimates using g02dac
   */
  /* 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;
    }

  printf("\n");
  printf("Estimates from g02dac\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");

#ifdef _WIN32
  scanf_s("%"NAG_IFMT"", &nestern);
#else
  scanf("%"NAG_IFMT"", &nestern);
#endif
  for (i = 1; i <= nestern; ++i)
    {
      for (j = 0; j < ip; ++j)
#ifdef _WIN32
        scanf_s("%lf", &f[j]);
#else
        scanf("%lf", &f[j]);
#endif

      /* nag_regsn_mult_linear_est_func (g02dnc).
       * Estimate of an estimable function for a general linear
       * regression model
       */
      nag_regsn_mult_linear_est_func(ip, rank, b, cov, p, f, &est, &stat,
                                     &sestat, &t, tol, &fail);

      if (fail.code == NE_NOERROR || fail.code == NE_RANK_EQ_IP)
        {
          printf("\n");
          printf("Function %"NAG_IFMT"\n\n", i);
          for (j = 0; j < ip; ++j)
            printf("%8.2f%c", f[j], (j%5 == 4 || j == ip-1)?'\n':' ');
          printf("\n");
          if (est)
            printf(" stat = %10.4f  se = %10.4f   t = %10.4f\n",
                    stat, sestat, t);
```

```
          else
            printf("Function not estimable\n");
        }
      else
        {
          printf(
                  "Error from nag_regsn_mult_linear_est_func (g02dnc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
    }
 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(f);
  NAG_FREE(p);
  NAG_FREE(q);
  NAG_FREE(se);
  NAG_FREE(com_ar);
  return exit_status;
}
```

10.2 Program Data

```
nag_regsn_mult_linear_est_func (g02dnc) Example Program Data
 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   5
 3
 1.0 1.0  0.0 0.0 0.0
 0.0 1.0 -1.0 0.0 0.0
 0.0 1.0  0.0 0.0 0.0
```

10.3 Program Results

```
nag_regsn_mult_linear_est_func (g02dnc) Example Program Results

Estimates from g02dac

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


Function 1
```

```
    1.00     1.00     0.00     0.00     0.00
 stat =    36.0033  se =     0.9623   t =    37.4119
Function 2
    0.00     1.00    -1.00     0.00     0.00
 stat =    -1.2967  se =     1.3610   t =    -0.9528
Function 3
    0.00     1.00     0.00     0.00     0.00
Function not estimable
```