NAG Library Function Document

nag_regsn_mult_linear_tran_model (g02dkc)

1 Purpose

nag_regsn_mult_linear_tran_model (g02dkc) calculates the estimates of the arguments of a general linear regression model for given constraints from the singular value decomposition results.

2 Specification

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

void nag_regsn_mult_linear_tran_model (Integer ip, Integer iconst,
     const double p[], const double c[], Integer tdc, double b[],
     double rss, double df, double se[], double cov[],
     NagError *fail)
```

3 Description

nag_regsn_mult_linear_tran_model (g02dkc) computes the estimates given a set of linear constraints 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).

In the case of a model not of full rank the functions use a singular value decomposition (SVD) to find the parameter estimates, $\hat{\beta}_{svd}$, and their variance-covariance matrix. Details of the SVD are made available, in the form of the matrix $P^*:

$$P^* = \begin{pmatrix} D^{-1} P^T_1 \\ P^T_0 \end{pmatrix}$$

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

Alternative solutions can be formed by imposing constraints on the arguments. If there are $p$ arguments and the rank of the model is $k$, then $n_c = p - k$ constraints will have to be imposed to obtain a unique solution.

Let $C$ be a $p$ by $n_c$ matrix of constraints, such that

$$C^T \beta = 0,$$

then the new parameter estimates $\hat{\beta}_c$ are given by:

$$\hat{\beta}_c = \left( I - P_0 (C^T P_0)^{-1} \right) \hat{\beta}_{svd},$$

where $I$ is the identity matrix, and the variance-covariance matrix is given by:

$$AP_1 D^{-2} P_1^T A^T$$

provided $(C^T P_0)^{-1}$ exists.
4 References


Searle S R (1971) Linear Models Wiley

5 Arguments

1: ip – Integer
   
   On entry: the number of terms in the linear model, \( p \).
   
   Constraint: \( ip \geq 1 \).

2: iconst – Integer
   
   On entry: the number of constraints to be imposed on the arguments, \( n_c \).
   
   Constraint: \( 0 < iconst < ip \).

3: \( p[ip \times ip + 2 \times ip] \) – const double
   
   On entry: \( p \) as returned by nag_regsn_mult_linear (g02dac) and nag_regsn_mult_linear_upd_model (g02ddc).

4: \( c[ip \times tdc] \) – const double
   
   Note: the \((i, j)\)th element of the matrix \( C \) is stored in \( c[(i-1) \times tdc + j - 1] \).
   
   On entry: the \( iconst \) constraints stored by column, i.e., the \( i \)th constraint is stored in the \( i \)th column of \( c \).

5: tdc – Integer
   
   On entry: the stride separating matrix column elements in the array \( c \).
   
   Constraint: \( tdc \geq iconst \).

6: \( b[ip] \) – double
   
   On entry: the parameter estimates computed by using the singular value decomposition, \( \hat{\beta}_{svd} \).
   
   On exit: the parameter estimates of the arguments with the constraints imposed, \( \hat{\beta}_c \).

7: rss – double
   
   On entry: the residual sum of squares as returned by nag_regsn_mult_linear (g02dac) or nag_regsn_mult_linear_upd_model (g02ddc).
   
   Constraint: \( rss > 0.0 \).

8: df – double
   
   On entry: the degrees of freedom associated with the residual sum of squares as returned by nag_regsn_mult_linear (g02dac) or nag_regsn_mult_linear_upd_model (g02ddc).
   
   Constraint: \( df > 0.0 \).

9: se[ip] – double
   
   On exit: the standard error of the parameter estimates in \( b \).
10: \textbf{covip} \times (\textbf{ip} + 1)/2 \quad \text{double} \\
\textit{Output} \\
\textit{On exit:} the upper triangular part of the variance-covariance matrix of the \textbf{ip} parameter estimates given in \textbf{b}. They are stored packed by column, i.e., the covariance between the parameter estimate given in \textbf{b}[i] and the parameter estimate given in \textbf{b}[j], \( j \geq i \), is stored in \textbf{cov}[j(j + 1)/2 + i], for \( i = 0, 1, \ldots, \text{ip} - 1 \) and \( j = i, \ldots, \text{ip} - 1 \).

11: \textbf{fail} \quad \text{NagError} * \\
\textit{Input/Output} \\
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

\textbf{NE_2_INT_ARG_GE} \\
On entry, \textbf{iconst} = \langle \text{value} \rangle while \textbf{ip} = \langle \text{value} \rangle. These arguments must satisfy \textbf{iconst} < \textbf{ip}.

\textbf{NE_2_INT_ARG_LT} \\
On entry, \textbf{tdc} = \langle \text{value} \rangle while \textbf{iconst} = \langle \text{value} \rangle. These arguments must satisfy \textbf{tdc} \geq \textbf{iconst}.

\textbf{NE_ALLOC_FAIL} \\
Dynamic memory allocation failed.

\textbf{NE_INT_ARG_LE} \\
On entry, \textbf{iconst} = \langle \text{value} \rangle. \\
Constraint: \textbf{iconst} > 0.

\textbf{NE_INT_ARG_LT} \\
On entry, \textbf{ip} = \langle \text{value} \rangle. \\
Constraint: \textbf{ip} \geq 1.

\textbf{NE_MAT_NOT_FULL_RANK} \\
Matrix \textbf{c} does not give a model of full rank.

\textbf{NE_REAL_ARG_LE} \\
On entry, \textbf{df} must not be less than or equal to 0.0: \textbf{df} = \langle \text{value} \rangle. \\
On entry, \textbf{rss} must not be less than or equal to 0.0: \textbf{rss} = \langle \text{value} \rangle.

7 Accuracy

It should be noted that due to rounding errors an argument that should be zero when the constraints have been imposed may be returned as a value of order \textit{machine precision}.

8 Parallelism and Performance

Not applicable.

9 Further Comments

\texttt{nag\_regr\_mult\_linear\_tran\_model} (g02dkc) is intended for use in situations in which dummy (0-1) variables have been used such as in the analysis of designed experiments when you do not wish to change the arguments of the model to give a full rank model. The function is not intended for situations in which the relationships between the independent variables are only approximate.
10 Example

Data from an experiment with four treatments and three observations per treatment are read in. A model, including the mean term, is fitted by nag_regsn_mult_linear (g02dac) and the results printed. The constraint that the sum of treatment effects is zero is then read in and the parameter estimates with this constraint imposed are computed by nag_regsn_mult_linear_tran_model (g02dkc) and printed.

10.1 Program Text

/* nag_regsn_mult_linear_tran_model (g02dkc) 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 C(I, J) c[(I) *tdc + J]
#define Q(I, J) q[(I) *tdq + J]

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

    INIT_FAIL(fail);

    printf("nag_regsn_mult_linear_tran_model (g02dkc) Example Program 
"Results\n");
    /* Skip heading in data file */
    #ifdef _WIN32
        scanf_s("%*[^
]");
    #else
        scanf("%*[^
]");
    #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*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.\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("%NAG_IFMT", &sx[j]);
}
if (!(b = NAG_ALLOC(ip, double)) ||
!(c = NAG_ALLOC((ip)*(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(4*ip*ip+5*(ip-1), double)))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

tdq = ip+1;
tdc = ip;

/* Set tolerance */
tol = 0.00001e0;
/* Find initial estimates using nag_regsn_mult_linear (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).
%s
", fail.message);
  exit_status = 1;
  goto END;
}

printf("Estimates from g02dac\n");
printf("Residual sum of squares = %13.4e\n", rss);
printf("Degrees of freedom = %3.1f\n\n", df);
for (j = 0; j < ip; j++)
  printf("%6"NAG_IFMT"%20.4e%20.4e\n", j+1, b[j], se[j]);
printf("\n");

/* Input constraints and call nag_regsn_mult_linear_tran_model (g02dkc)
*/
iconst = ip - rank;
for (i = 0; i < ip; ++i)
  for (j = 0; j < iconst; ++j)
    #ifdef _WIN32
      scanf_s("%lf", &C(i, j));
    #else
      scanf("%lf", &C(i, j));
    #endif

/* nag_regsn_mult_linear_tran_model (g02dkc).
* Estimates of parameters of a general linear regression
* model for given constraints */
nag_regsn_mult_linear_tran_model(ip, iconst, p, c, tdc, b, rss, df, se, cov,
&fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_regsn_mult_linear_tran_model (g02dkc).
%s\n", fail.message);
  exit_status = 1;
  goto END;
}

printf("\n");
printf("Estimates from nag_regsn_mult_linear_tran_model (g02dkc) using 
constraints\n");
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_FREE(h);
NAG_FREE(res);
NAG_FREE(wt);
NAG_FREE(x);
NAG_FREE(y);
NAG_FREE(sx);
NAG_FREE(b);
NAG_FREE(c);
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_tran_model (g02dkc) 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
0.0
1.0
1.0
1.0
1.0

10.3 Program Results

nag_regsn_mult_linear_tran_model (g02dkc) 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

Estimates from nag_regsn_mult_linear_tran_model (g02dkc) using constraints
Variable Parameter estimate Standard error
1 3.8196e+01 4.8117e-01
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(-2.1925e+00)</td>
<td>(8.3342e-01)</td>
</tr>
<tr>
<td>3</td>
<td>(-8.9583e-01)</td>
<td>(8.3342e-01)</td>
</tr>
<tr>
<td>4</td>
<td>(3.4075e+00)</td>
<td>(8.3342e-01)</td>
</tr>
<tr>
<td>5</td>
<td>(-3.1917e-01)</td>
<td>(8.3342e-01)</td>
</tr>
</tbody>
</table>