NAG Library Function Document

nag_all_regsn (g02eac)

1 Purpose

nag_all_regsn (g02eac) calculates the residual sums of squares for all possible linear regressions for a given set of independent variables.

2 Specification

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

void nag_all_regsn (Nag_OrderType order, Nag_IncludeMean mean, Integer n,
                   Integer m, const double x[], Integer pdx, const char *var_names[],
                   const Integer sx[], const double y[], const double wt[], Integer *nmod,
                   const char *model[], double rss[], Integer nterms[], Integer mrank[],
                   NagError *fail)
```

3 Description

For a set of \( k \) possible independent variables there are \( 2^k \) linear regression models with from zero to \( k \) independent variables in each model. For example if \( k = 3 \) and the variables are \( A, B \) and \( C \) then the possible models are:

(i) null model
(ii) \( A \)
(iii) \( B \)
(iv) \( C \)
(v) \( A \) and \( B \)
(vi) \( A \) and \( C \)
(vii) \( B \) and \( C \)
(viii) \( A, B \) and \( C \).

nag_all_regsn (g02eac) calculates the residual sums of squares from each of the \( 2^k \) possible models. The method used involves a QR decomposition of the matrix of possible independent variables. Independent variables are then moved into and out of the model by a series of Givens rotations and the residual sums of squares computed for each model; see Clark (1981) and Smith and Bremner (1989).

The computed residual sums of squares are then ordered first by increasing number of terms in the model, then by decreasing size of residual sums of squares. So the first model will always have the largest residual sum of squares and the \( 2^k \)th will always have the smallest. This aids you in selecting the best possible model from the given set of independent variables.

nag_all_regsn (g02eac) allows you to specify some independent variables that must be in the model, the forced variables. The other independent variables from which the possible models are to be formed are the free variables.
4 References

Clark M R B (1981) A Givens algorithm for moving from one linear model to another without going back to the data Appl. Statist. 30 198–203


5 Arguments

1: order – Nag_OrderType

On entry: the 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: mean – Nag_IncludeMean

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.

3: n – Integer

On entry: n, the number of observations.

Constraints:

n ≥ 2;
n ≥ m, is the number of independent variables to be considered (forced plus free plus mean if included), as specified by mean and sx.

4: m – Integer

On entry: the number of variables contained in x.

Constraint: m ≥ 2.

5: x[dim] – const double

Note: the dimension, dim, of the array x must be at least
max(1,pdx × m) when order = Nag_ColMajor;
max(1,n × pdx) when order = Nag_RowMajor.

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

x[(j − 1) × pdx + i − 1] when order = Nag_ColMajor;

x[(i − 1) × pdx + j − 1] when order = Nag_RowMajor.

On entry: X(i,j) must contain the i th observation for the j th independent variable, for i = 1, 2, . . . , n and j = 1, 2, . . . , m.

6: pdx – Integer

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

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

7: var_names[m] – const char *
   Input
   On entry: var_names[i − 1] must contain the name of the independent variable in row i of x, for i = 1, 2, . . . , m.

8: sx[m] – const Integer
   Input
   On entry: indicates which independent variables are to be considered in the model.
   sx[j − 1] ≥ 2
   The variable contained in the jth column of X is included in all regression models, i.e., is a forced variable.
   sx[j − 1] = 1
   The variable contained in the jth column of X is included in the set from which the regression models are chosen, i.e., is a free variable.
   sx[j − 1] = 0
   The variable contained in the jth column of X is not included in the models.

Constraints:
   sx[j − 1] ≥ 0, for j = 1, 2, . . . , m;
   at least one value of sx = 1.

9: y[n] – const double
   Input
   On entry: y[i − 1] must contain the ith observation on the dependent variable, y_i, for i = 1, 2, . . . , n.

10: wt[n] – const double
    Input
    On entry: optionally, the weights to be used in the weighted regression.
    If 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.
    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, wt[i − 1] = 0.0, for i = 1, 2, . . . , n.

11: nmod – Integer *
    Output
    On exit: the total number of models for which residual sums of squares have been calculated.

12: model[dim] – const char *
    Output
    Note: the dimension, dim, of the array model must be at least big enough to hold the names of all the free independent variables which appear in all the models. This will never exceed 2^k × m, where k is the number of free variables in the model.
    On exit: the names of the independent variables in each model, represented as pointers to the names provided by you in var_names. The model names are stored as follows:
    if the first model has three names, i.e., nterms[0] = 3; then model[0], model[1] and model[2] will contain these three names;
    if the second model has two names, i.e., nterms[1] = 2; then model[3], model[4] will contain these two names.
13: rss[\max(2^k, m)] \quad \text{double} \quad \text{Output}

On exit: \( \text{rss}[i - 1] \) contains the residual sum of squares for the \( i \)th model, for \( i = 1, 2, \ldots, \text{nmod} \).

14: nterms[\max(2^k, m)] \quad \text{Integer} \quad \text{Output}

On exit: \( \text{nterms}[i - 1] \) contains the number of independent variables in the \( i \)th model, not including the mean if one is fitted, for \( i = 1, 2, \ldots, \text{nmod} \).

15: mrank[\max(2^k, m)] \quad \text{Integer} \quad \text{Output}

On exit: \( \text{mrank}[i - 1] \) contains the rank of the residual sum of squares for the \( i \)th model.

16: fail \quad \text{NagError} * \quad \text{Input/Output}

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

6 Error Indicators and Warnings

\textbf{NE_ALLOC_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_BAD_PARAM}

On entry, argument \( \langle\text{value}\rangle \) had an illegal value.

\textbf{NE_FREE_VARS}

There are no free x variables.

\textbf{NE_FULL_RANK}

Full model is not of full rank.

\textbf{NE_INDEP_VARS_OBS}

Number of requested x-variables \( \geq \) number of observations.

\textbf{NE_INT}

On entry, \( m = \langle\text{value}\rangle \).

Constraint: \( m \geq 2 \).

On entry, \( n = \langle\text{value}\rangle \).

Constraint: \( n \geq 2 \).

On entry, \( \text{pdx} = \langle\text{value}\rangle \).

Constraint: \( \text{pdx} > 0 \).

\textbf{NE_INT_2}

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

Constraint: \( \text{pdx} \geq m \).

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

Constraint: \( \text{pdx} \geq n \).

\textbf{NE_INT_ARRAY_ELEM_CONS}

On entry, \( \text{sx}[\langle\text{value}\rangle] < 0 \).
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.

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.

NE_REAL_ARRAY_ELEM_CONS
On entry, wt[<value>] < 0.0.

7 Accuracy
For a discussion of the improved accuracy obtained by using a method based on the QR decomposition see Smith and Bremner (1989).

8 Parallelism and Performance
nag_all_regsn (g02eac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.
nag_all_regsn (g02eac) 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_cp_stat (g02ecc) may be used to compute $R^2$ and $C_p$-values from the results of nag_all_regsn (g02eac).
If a mean has been included in the model and no variables are forced in then rss[0] contains the total sum of squares and in many situations a reasonable estimate of the variance of the errors is given by rss[nmod – 1]/(n – 1 – nterms[nmod – 1]).

10 Example
The data for this example is given in Weisberg (1985). The independent variables and the dependent variable are read, as are the names of the variables. These names are as given in Weisberg (1985). The residual sums of squares computed and printed with the names of the variables in the model.

10.1 Program Text
/* nag_all_regsn (g02eac) Example Program. * 
 * Copyright 2014 Numerical Algorithms Group. * 
 * Mark 7, 2002. */
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>

int main(void)
{
    /* Scalars */
    Integer exit_status, free_vars, i, ii, j, m, n, nmod, pdx;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    char **model = 0;
    const char *var_names[] = { "DAY", "BOD", "TKN", "TS", "TVS", "COD" };
    double *rss = 0, *x = 0, *y = 0, *wtptr = 0;
    Integer *sx = 0, *mrk = 0, *nterms = 0;

    /* For iteration over model */
    Integer model_index = 0;

    #ifdef NAG_COLUMN_MAJOR
    #define X(I, J) x[(J-1)*pdx + I - 1]
    order = Nag_ColMajor;
    #else
    #define X(I, J) x[(I-1)*pdx + J - 1]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    exit_status = 0;
    printf("nag_all_regsn (g02eac) Example Program Results\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%n[\n] ");
    #else
    scanf("%n[\n] ");
    #endif

    #ifdef _WIN32
    scanf_s("%"NAG_IFMT"%"NAG_IFMT"%n", &n, &m);
    #else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%n", &n, &m);
    #endif

    /* Allocate memory */
    if (!(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;
    }

    #ifdef NAG_COLUMN_MAJOR
    pdx = n;
    order = Nag_ColMajor;
    #else
    pdx = m;
    order = Nag_RowMajor;
    #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
    }
```c
#define _WIN32
    scanf_s("%f*['\n"]", &y[i - 1]);
#else
    scanf("%f*['\n"]", &y[i - 1]);
#endif
}

free_vars = 1;
for (j = 1; j <= m; ++j)
{
    #ifdef _WIN32
    scanf_s("%"NAG_IFMT"", &sx[j - 1]);
#else
    scanf("%"NAG_IFMT"", &sx[j - 1]);
#endif
    if (sx[j - 1] == 1)
    {
        free_vars <<= 1;
    }
}

#define _WIN32
scans_s("%*[\n"]");
#else
scans("%*[\n"]");
#endif

if (!(model = NAG_ALLOC(free_vars*m, char *)) ||
    !(rss = NAG_ALLOC(free_vars, double)) ||
    !(mrank = NAG_ALLOC(free_vars, Integer)) ||
    !(nterms = NAG_ALLOC(free_vars, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* nag_all_regsn (g02eac).
* Computes residual sums of squares for all possible linear
* regressions for a set of independent variables
*/
nag_all_regsn(order, Nag_MeanInclude, n, m, x, pdx, var_names, sx, y, wtptr,
    &nmod, (const char **)model, rss, nterms, mrank, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_all_regsn (g02eac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

printf("\n");
printf("Number of Rss Rank Model\n");
printf("parameters\n");
for (i = 1; i <= nmod; ++i)
{
    ii = nterms[i - 1];
    printf("%8"NAG_IFMT"%11.4f"NAG_IFMT" ", ii, rss[i-1],
        mrank[i-1]);
    for (j = 1; j <= ii; ++j)
        printf("-%3.3s %s", model[model_index++],
            j%5 == 0 || j == 5?"\n": "");
    printf("\n");
}
END:
NAG_FREE(rss);
NAG_FREE(x);
NAG_FREE(y);
NAG_FREE(sx);
```
10.2 Program Data

nag_all_regsn (g02eac) Example Program Data
20 6
  0. 1125.0 232.0 7160.0 85.9 8905.0 1.5563
  7. 920.0 268.0 8804.0 86.5 7388.0 0.8976
15. 835.0 271.0 8108.0 85.2 5348.0 0.7482
22. 1000.0 237.0 6370.0 83.8 8056.0 0.7160
29. 1150.0 192.0 6441.0 82.1 6932.0 0.3010
37. 990.0 202.0 5154.0 79.2 5690.0 0.3617
44. 840.0 184.0 5896.0 86.5 7388.0 0.1139
58. 650.0 200.0 5336.0 80.6 5400.0 0.1139
65. 640.0 180.0 5041.0 78.4 3177.0 -0.2218
72. 583.0 165.0 5012.0 79.3 4461.0 -0.1549
80. 570.0 151.0 4825.0 78.7 3901.0 0.0000
86. 570.0 171.0 4391.0 78.0 5002.0 0.0000
93. 510.0 243.0 4320.0 72.3 4665.0 -0.0969
100. 555.0 147.0 3709.0 74.9 4642.0 -0.2218
107. 460.0 286.0 3969.0 74.4 4840.0 -0.3979
122. 275.0 198.0 3969.0 74.4 4479.0 -0.1549
129. 510.0 196.0 4361.0 57.7 4200.0 -0.2218
151. 165.0 210.0 3301.0 71.8 3410.0 -0.3979
171. 244.0 327.0 2964.0 72.5 3360.0 -0.5229
220. 79.0 334.0 2777.0 71.9 2599.0 -0.0458
  0 1 1 1 1 1

10.3 Program Results

nag_all_regsn (g02eac) Example Program Results

<table>
<thead>
<tr>
<th>Number of parameters</th>
<th>Rss</th>
<th>Rank</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.0634</td>
<td>32</td>
<td>TKN</td>
</tr>
<tr>
<td>1</td>
<td>5.0219</td>
<td>31</td>
<td>TVS</td>
</tr>
<tr>
<td>2</td>
<td>2.0338</td>
<td>28</td>
<td>BOD</td>
</tr>
<tr>
<td>1</td>
<td>1.5563</td>
<td>25</td>
<td>COD</td>
</tr>
<tr>
<td>1</td>
<td>1.5370</td>
<td>24</td>
<td>TS</td>
</tr>
<tr>
<td>2</td>
<td>2.4381</td>
<td>29</td>
<td>TKN</td>
</tr>
<tr>
<td>2</td>
<td>1.7462</td>
<td>27</td>
<td>TVS</td>
</tr>
<tr>
<td>2</td>
<td>1.5921</td>
<td>26</td>
<td>BOD</td>
</tr>
<tr>
<td>2</td>
<td>1.4963</td>
<td>23</td>
<td>TKN</td>
</tr>
<tr>
<td>2</td>
<td>1.4707</td>
<td>22</td>
<td>TS</td>
</tr>
<tr>
<td>2</td>
<td>1.4590</td>
<td>21</td>
<td>TVS</td>
</tr>
<tr>
<td>2</td>
<td>1.4397</td>
<td>20</td>
<td>BOD</td>
</tr>
<tr>
<td>2</td>
<td>1.4388</td>
<td>19</td>
<td>COD</td>
</tr>
<tr>
<td>2</td>
<td>1.3287</td>
<td>15</td>
<td>TVS</td>
</tr>
<tr>
<td>2</td>
<td>1.0850</td>
<td>8</td>
<td>COD</td>
</tr>
<tr>
<td>3</td>
<td>1.4257</td>
<td>18</td>
<td>BOD</td>
</tr>
<tr>
<td>3</td>
<td>1.3900</td>
<td>17</td>
<td>TKN</td>
</tr>
<tr>
<td>3</td>
<td>1.3894</td>
<td>16</td>
<td>BOD</td>
</tr>
<tr>
<td>3</td>
<td>1.3204</td>
<td>14</td>
<td>TVS</td>
</tr>
<tr>
<td>3</td>
<td>1.2764</td>
<td>13</td>
<td>BOD</td>
</tr>
<tr>
<td>3</td>
<td>1.2582</td>
<td>12</td>
<td>TKN</td>
</tr>
<tr>
<td>3</td>
<td>1.2179</td>
<td>10</td>
<td>TVS</td>
</tr>
<tr>
<td>3</td>
<td>1.0644</td>
<td>7</td>
<td>BOD</td>
</tr>
<tr>
<td>3</td>
<td>1.0634</td>
<td>6</td>
<td>TS</td>
</tr>
<tr>
<td>3</td>
<td>0.9871</td>
<td>4</td>
<td>TKN</td>
</tr>
<tr>
<td>4</td>
<td>1.2199</td>
<td>11</td>
<td>BOD</td>
</tr>
<tr>
<td>4</td>
<td>1.1565</td>
<td>9</td>
<td>BOD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>4</td>
<td>1.0388</td>
<td>5</td>
<td>BOD TS TVS COD</td>
</tr>
<tr>
<td>4</td>
<td>0.9871</td>
<td>3</td>
<td>BOD TKN TS COD</td>
</tr>
<tr>
<td>4</td>
<td>0.9653</td>
<td>2</td>
<td>TKN TS TVS COD</td>
</tr>
<tr>
<td>5</td>
<td>0.9652</td>
<td>1</td>
<td>BOD TKN TS TVS COD</td>
</tr>
</tbody>
</table>