

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

```
#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

Smith D M and Bremner J M (1989) All possible subset regressions using the *QR* decomposition *Comput. Statist. Data Anal.* **7** 217–236

Weisberg S (1985) *Applied Linear Regression* Wiley

5 Arguments

1: **order** – Nag_OrderType *Input*

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 *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.

3: **n** – Integer *Input*

On entry: n , the number of observations.

Constraints:

$n \geq 2$;

$n \geq m$, 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 *Input*

On entry: the number of variables contained in **x**.

Constraint: $m \geq 2$.

5: **x[dim]** – const double *Input*

Note: the dimension, dim , of the array **x** must be at least

$\max(1, \mathbf{pdx} \times \mathbf{m})$ when **order** = Nag_ColMajor;

$\max(1, \mathbf{n} \times \mathbf{pdx})$ when **order** = Nag_RowMajor.

Where $\mathbf{X}(i, j)$ appears in this document, it refers to the array element

$\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$ when **order** = Nag_ColMajor;

$\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$ when **order** = Nag_RowMajor.

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

6: **pdx** – Integer *Input*

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** \geq **n**;
if **order** = Nag_RowMajor, **pdx** \geq **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] \geq 2

The variable contained in the *j*th column of **X** is included in all regression models, i.e., is a forced variable.

sx[*j* – 1] = 1

The variable contained in the *j*th 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 *j*th column of **X** is not included in the models.

Constraints:

sx[*j* – 1] \geq 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 *i*th 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 *i*th 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 $2^k \times \mathbf{m}$, 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**)] – double *Output*
On exit: **rss**[$i - 1$] contains the residual sum of squares for the i th model, for $i = 1, 2, \dots, \mathbf{nmod}$.
- 14: **nterms**[**max**(2^k , **m**)] – Integer *Output*
On exit: **nterms**[$i - 1$] contains the number of independent variables in the i th model, not including the mean if one is fitted, for $i = 1, 2, \dots, \mathbf{nmod}$.
- 15: **mrank**[**max**(2^k , **m**)] – Integer *Output*
On exit: **mrank**[$i - 1$] contains the rank of the residual sum of squares for the i th model.
- 16: **fail** – NagError * *Input/Output*
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_FREE_VARS

There are no free **x** variables.

NE_FULL_RANK

Full model is not of full rank.

NE_INDEP_VARS_OBS

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

NE_INT

On entry, **m** = $\langle value \rangle$.

Constraint: **m** ≥ 2 .

On entry, **n** = $\langle value \rangle$.

Constraint: **n** ≥ 2 .

On entry, **pdx** = $\langle value \rangle$.

Constraint: **pdx** > 0 .

NE_INT_2

On entry, **pdx** = $\langle value \rangle$ and **m** = $\langle value \rangle$.

Constraint: **pdx** \geq **m**.

On entry, **pdx** = $\langle value \rangle$ and **n** = $\langle value \rangle$.

Constraint: **pdx** \geq **n**.

NE_INT_ARRAY_ELEM_CONS

On entry, **sx**[$\langle value \rangle$] < 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.

NE_REAL_ARRAY_ELEM_CONS

On entry, $\text{wt}[\textit{value}] < 0.0$.

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 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 $\text{rss}[\text{nmod} - 1] / (\mathbf{n} - 1 - \text{nterms}[\text{nmod} - 1])$.

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 2002 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, *mrank = 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 */
scanf("%*[\n] ");
scanf("%ld%ld%*[\n] ", &n, &m);

/* 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)
        scanf("%lf", &X(i, j));
    scanf("%lf%*[\n] ", &y[i - 1]);
}

free_vars = 1;
for (j = 1; j <= m; ++j)
{
    scanf("%ld", &sx[j - 1]);
    if (sx[j - 1] == 1)
    {
        free_vars <<= 1;
    }
}
scanf("%*[\n] ");

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("%8ld%11.4f%4ld  ", 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);
NAG_FREE(mrank);
NAG_FREE(nterms);
NAG_FREE(model);
return exit_status;
}

```

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 6960.0 0.3010
37.  990.0 202.0 5154.0 79.2 5690.0 0.3617
44.  840.0 184.0 5896.0 81.2 6932.0 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 3558.0 72.5 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

Number of parameters	Rss	Rank	Model					
0	5.0634	32						
1	5.0219	31	TKN					
1	2.5044	30	TVS					
1	2.0338	28	BOD					
1	1.5563	25	COD					
1	1.5370	24	TS					
2	2.4381	29	TKN	TVS				
2	1.7462	27	BOD	TVS				
2	1.5921	26	BOD	TKN				
2	1.4963	23	BOD	COD				
2	1.4707	22	TKN	TS				
2	1.4590	21	TS	TVS				
2	1.4397	20	BOD	TS				
2	1.4388	19	TKN	COD				
2	1.3287	15	TVS	COD				
2	1.0850	8	TS	COD				
3	1.4257	18	BOD	TKN	TVS			
3	1.3900	17	TKN	TS	TVS			
3	1.3894	16	BOD	TS	TVS			
3	1.3204	14	BOD	TVS	COD			
3	1.2764	13	BOD	TKN	COD			
3	1.2582	12	BOD	TKN	TS			
3	1.2179	10	TKN	TVS	COD			
3	1.0644	7	BOD	TS	COD			
3	1.0634	6	TS	TVS	COD			
3	0.9871	4	TKN	TS	COD			
4	1.2199	11	BOD	TKN	TS	TVS		
4	1.1565	9	BOD	TKN	TVS	COD		
4	1.0388	5	BOD	TS	TVS	COD		
4	0.9871	3	BOD	TKN	TS	COD		
4	0.9653	2	TKN	TS	TVS	COD		
5	0.9652	1	BOD	TKN	TS	TVS	COD	
