

NAG Library Function Document

nag_lone_fit (e02gac)

1 Purpose

nag_lone_fit (e02gac) calculates an l_1 solution to an over-determined system of linear equations.

2 Specification

```
#include <nag.h>
#include <nage02.h>

void nag_lone_fit (Nag_OrderType order, Integer m, double a[], double b[],
                  Integer nplus2, double toler, double x[], double *resid, Integer *rank,
                  Integer *iter, NagError *fail)
```

3 Description

Given a matrix A with m rows and n columns ($m \geq n$) and a vector b with m elements, the function calculates an l_1 solution to the over-determined system of equations

$$Ax = b.$$

That is to say, it calculates a vector x , with n elements, which minimizes the l_1 norm (the sum of the absolute values) of the residuals

$$r(x) = \sum_{i=1}^m |r_i|,$$

where the residuals r_i are given by

$$r_i = b_i - \sum_{j=1}^n a_{ij}x_j, \quad i = 1, 2, \dots, m.$$

Here a_{ij} is the element in row i and column j of A , b_i is the i th element of b and x_j the j th element of x . The matrix A need not be of full rank.

Typically in applications to data fitting, data consisting of m points with coordinates (t_i, y_i) are to be approximated in the l_1 norm by a linear combination of known functions $\phi_j(t)$,

$$\alpha_1\phi_1(t) + \alpha_2\phi_2(t) + \dots + \alpha_n\phi_n(t).$$

This is equivalent to fitting an l_1 solution to the over-determined system of equations

$$\sum_{j=1}^n \phi_j(t_i)\alpha_j = y_i, \quad i = 1, 2, \dots, m.$$

Thus if, for each value of i and j , the element a_{ij} of the matrix A in the previous paragraph is set equal to the value of $\phi_j(t_i)$ and b_i is set equal to y_i , the solution vector x will contain the required values of the α_j . Note that the independent variable t above can, instead, be a vector of several independent variables (this includes the case where each ϕ_i is a function of a different variable, or set of variables).

The algorithm is a modification of the simplex method of linear programming applied to the primal formulation of the l_1 problem (see Barrodale and Roberts (1973) and Barrodale and Roberts (1974)). The modification allows several neighbouring simplex vertices to be passed through in a single iteration, providing a substantial improvement in efficiency.

4 References

Barrodale I and Roberts F D K (1973) An improved algorithm for discrete l_1 linear approximation *SIAM J. Numer. Anal.* **10** 839–848

Barrodale I and Roberts F D K (1974) Solution of an overdetermined system of equations in the l_1 -norm *Comm. ACM* **17(6)** 319–320

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: **m** – Integer *Input*
On entry: the number of equations, m (the number of rows of the matrix A).
Constraint: $m \geq n \geq 1$.
- 3: **a**[(**m** + 2) × **nplus2**] – double *Input/Output*
Note: where $A(i, j)$ appears in this document, it refers to the array element
 $\mathbf{a}[(j - 1) \times ((\mathbf{m} + 2)) + i - 1]$ when **order** = Nag_ColMajor;
 $\mathbf{a}[(i - 1) \times \mathbf{nplus2} + j - 1]$ when **order** = Nag_RowMajor.
On entry: $A(i, j)$ must contain a_{ij} , the element in the i th row and j th column of the matrix A , for $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$. The remaining elements need not be set.
On exit: contains the last simplex tableau generated by the simplex method.
- 4: **b**[**m**] – double *Input/Output*
On entry: **b**[$i - 1$] must contain b_i , the i th element of the vector b , for $i = 1, 2, \dots, m$.
On exit: the i th residual r_i corresponding to the solution vector x , for $i = 1, 2, \dots, m$.
- 5: **nplus2** – Integer *Input*
On entry: $n + 2$, where n is the number of unknowns (the number of columns of the matrix A).
Constraint: $3 \leq \mathbf{nplus2} \leq \mathbf{m} + 2$.
- 6: **toler** – double *Input*
On entry: a non-negative value. In general **toler** specifies a threshold below which numbers are regarded as zero. The recommended threshold value is $\epsilon^{2/3}$ where ϵ is the *machine precision*. The recommended value can be computed within the function by setting **toler** to zero. If premature termination occurs a larger value for **toler** may result in a valid solution.
Suggested value: 0.0.
- 7: **x**[**nplus2**] – double *Output*
On exit: **x**[$j - 1$] contains the j th element of the solution vector x , for $j = 1, 2, \dots, n$. The elements **x**[n] and **x**[$n + 1$] are unused.
- 8: **resid** – double * *Output*
On exit: the sum of the absolute values of the residuals for the solution vector x .

- 9: **rank** – Integer * Output
On exit: the computed rank of the matrix A .
- 10: **iter** – Integer * Output
On exit: the number of iterations taken by the simplex method.
- 11: **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.
 See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

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

NE_INT

On entry, **nplus2** = $\langle value \rangle$.
 Constraint: **nplus2** \geq 3.

NE_INT_2

On entry, **nplus2** = $\langle value \rangle$ and **m** = $\langle value \rangle$.
 Constraint: $3 \leq \mathbf{nplus2} \leq \mathbf{m} + 2$.

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_NON_UNIQUE

An optimal solution has been obtained, but may not be unique.

NE_TERMINATION_FAILURE

Premature termination due to rounding errors. Try using larger value of **toler**: **toler** = $\langle value \rangle$.

7 Accuracy

Experience suggests that the computational accuracy of the solution x is comparable with the accuracy that could be obtained by applying Gaussian elimination with partial pivoting to the n equations satisfied by this algorithm (i.e., those equations with zero residuals). The accuracy therefore varies with the conditioning of the problem, but has been found generally very satisfactory in practice.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The effects of m and n on the time and on the number of iterations in the Simplex Method vary from problem to problem, but typically the number of iterations is a small multiple of n and the total time taken is approximately proportional to mn^2 .

It is recommended that, before the function is entered, the columns of the matrix A are scaled so that the largest element in each column is of the order of unity. This should improve the conditioning of the matrix, and also enable the argument **toler** to perform its correct function. The solution x obtained will then, of course, relate to the scaled form of the matrix. Thus if the scaling is such that, for each $j = 1, 2, \dots, n$, the elements of the j th column are multiplied by the constant k_j , the element x_j of the solution vector x must be multiplied by k_j if it is desired to recover the solution corresponding to the original matrix A .

10 Example

Suppose we wish to approximate a set of data by a curve of the form

$$y = Ke^t + Le^{-t} + M$$

where K , L and M are unknown. Given values y_i at 5 points t_i we may form the over-determined set of equations for K , L and M

$$e^{x_i}K + e^{-x_i}L + M = y_i, \quad i = 1, 2, \dots, 5.$$

nag_lone_fit (e02gac) is used to solve these in the l_1 sense.

10.1 Program Text

```

/* nag_lone_fit (e02gac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage02.h>

int main(void)
{
    /* Scalars */
    double      resid, t, tol;
    Integer     exit_status, i, iter, m, rank, n, nplus2, pda;
    NagError    fail;
    Nag_OrderType order;

    /* Arrays */
    double      *a = 0, *b = 0, *x = 0;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
    order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);

```

```

    exit_status = 0;
    printf("nag_lone_fit (e02gac) Example Program Results\n");

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

    n = 3;
    nplus2 = n + 2;

#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n] ", &m);
#else
    scanf("%"NAG_IFMT"%*[\n] ", &m);
#endif
    if (m > 0)
    {
        /* Allocate memory */
        if (!(a = NAG_ALLOC((m + 2) * nplus2, double)) ||
            !(b = NAG_ALLOC(m, double)) ||
            !(x = NAG_ALLOC(nplus2, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }

        if (order == Nag_ColMajor)
            pda = m + 2;
        else
            pda = nplus2;

        for (i = 1; i <= m; ++i)
        {
#ifdef _WIN32
            scanf_s("%lf%lf%*[\n] ", &t, &b[i-1]);
#else
            scanf("%lf%lf%*[\n] ", &t, &b[i-1]);
#endif
            A(i, 1) = exp(t);
            A(i, 2) = exp(-t);
            A(i, 3) = 1.0;
        }
        tol = 0.0;
        /* nag_lone_fit (e02gac).
        * L_1-approximation by general linear function
        */
        nag_lone_fit(order, m, a, b, nplus2, tol, x, &resid,
                    &rank, &iter, &fail);
        if (fail.code == NE_INT || fail.code == NE_INT_2 ||
            fail.code == NE_NO_LICENCE)
        {
            printf("Error from nag_lone_fit (e02gac).\n%s\n",
                    fail.message);
            exit_status = 1;
            goto END;
        }
        else
        {
            printf("\n");
            printf("resid = %11.2e Rank = %5"NAG_IFMT" Iterations = "
                    " %5"NAG_IFMT"\n", resid, rank, iter);

            printf("\n");
            printf("Solution\n");

            for (i = 1; i <= n; ++i)

```

```
        printf("%10.4f", x[i-1]);  
        printf("\n");  
    }  
}  
  
END:  
NAG_FREE(a);  
NAG_FREE(b);  
NAG_FREE(x);  
  
return exit_status;  
}
```

10.2 Program Data

```
nag_lone_fit (e02gac) Example Program Data  
5  
0.0 4.501  
0.2 4.360  
0.4 4.333  
0.6 4.418  
0.8 4.625
```

10.3 Program Results

```
nag_lone_fit (e02gac) Example Program Results  
  
resid = 2.78e-03 Rank = 3 Iterations = 5  
  
Solution  
1.0014 2.0035 1.4960
```
