1 Purpose

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

2 Specification

```c
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

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

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 co-ordinates \((t_i, y_i)\) are to be approximated in the \( l_1 \) norm by a linear combination of known functions \( \phi_j(t) \),

\[
\alpha_1\phi_1(t) + \alpha_2\phi_2(t) + \cdots + \alpha_n\phi_n(t).
\]

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

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

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 \( \alpha_j \).

Note that the independent variable \( t \) above can, instead, be a vector of several independent variables (this includes the case where each \( \phi \) 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 (1974) Solution of an overdetermined system of equations in the $l_1$-norm
Comm. ACM 17 (6) 319–320

5 Parameters

1: order – Nag_OrderType

*Input*

*On entry:* the order parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

*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: $a[\text{dim}]$ – double

*Input/Output*

*Note:* the dimension, dim, of the array $a$ must be at least $(m + 2) \times \text{nplus2}$.

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

- if order = Nag_ColMajor, $a[(j - 1) \times (m + 2) + i - 1]$;
- if order = Nag_RowMajor, $a[(i - 1) \times \text{nplus2} + j - 1]$.

*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 $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. The remaining elements need not be set.

*On exit:* $a$ 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 $i = 1, 2, \ldots, m$.

*On exit:* the $i$th residual $r_i$ corresponding to the solution vector $x$, for $i = 1, 2, \ldots, m$.

5: nplus2 – Integer

*Input*

*On entry:* $n + 2$, where $n$ is the number of unknowns (the number of columns of the matrix $A$).

*Constraint:* $3 \leq \text{nplus2} \leq m + 2$.

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 $\epsilon$ 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: $x[\text{nplus2}]$ – double

*Output*

*On exit:* $x[j - 1]$ contains the $j$th element of the solution vector $x$, for $j = 1, 2, \ldots, n$. 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 *  
    On exit: the number of iterations taken by the simplex method.

11:  fail – NagError *  
    The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT
    On entry, nplus2 = ⟨value⟩.  
    Constraint: nplus2 ≥ 3.

NE_INT_2
    On entry, nplus2 > m + 2: nplus2 = ⟨value⟩, m = ⟨value⟩.

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 = ⟨value⟩.

NE_ALLOC_FAIL
    Memory allocation failed.

NE_BAD_PARAM
    On entry, parameter ⟨value⟩ had an illegal value.

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 consult NAG for assistance.

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 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 parameter 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 $j = 1, 2, \ldots, n$, 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 $A$. 

[NP3645/7]
9 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 \( m \)

\[ e^{t_i}k + e^{-t_i}l + m = y_i, \quad i = 1, 2, \ldots, 5. \]

\( \text{nag\_lone\_fit (e02gac)} \) is used to solve these in the \( l_1 \) sense.

9.1 Program Text

```c
/* nag_lone_fit (e02gac) Example Program. */
*
* Copyright 2001 Numerical Algorithms Group.
*
*/
#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;
    
    INIT_FAIL(fail);
    exit_status = 0;
    Vprintf("e02gac Example Program Results\n");
    /* Skip heading in data file */
    Vscanf("%\[^
\] ");
    n = 3;
    nplus2 = n + 2;
    Vscanf("%ld%\[`\n\] \", &m);
    if (m > 0)
    {
        /* Allocate memory */
        if (!a = NAG_ALLOC((m + 2) * nplus2, double)) ||
            !b = NAG_ALLOC(m, double)) ||
            !(x = NAG_ALLOC(nplus2, double)) )
        {
            Vprintf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
```
if (order == Nag_ColMajor)
   pda = m + 2;
else
   pda = nplus2;
for (i = 1; i <= m; ++i)
{
   Vscanf("%lf%lf%*[\n] ", &t, &b[i-1]);
   A(i, 1) = exp(t);
   A(i, 2) = exp(-t);
   A(i, 3) = 1.0;
}
tol = 0.0;
e02gac(order, m, a, b, nplus2, tol, x, &resid,
       &rank, &iter, &fail);
if (fail.code == NE_INT || fail.code == NE_INT_2)
{
   Vprintf("Error from e02gac.
%s
", fail.message);
   exit_status = 1;
   goto END;
}
else
{
   Vprintf("\n");
   Vprintf("resid = %10.2e Rank = %5ld Iterations = %5ld\n",
           resid, rank, iter);
   Vprintf("\n");
   Vprintf("Solution\n");
   for (i = 1; i <= n; ++i)
      Vprintf("%10.4f", x[i-1]);
   Vprintf("\n");
}
END:
   if (a) NAG_FREE(a);
   if (b) NAG_FREE(b);
   if (x) NAG_FREE(x);
   return exit_status;

9.2 Program Data

9.3 Program Results