NAG C Library Function Document

nag_linf_fit (e02gcc)

1 Purpose

nag_linf_fit (e02gcc) calculates an l_∞ solution to an over-determined system of linear equations.

2 Specification

```
void nag_linf_fit (Nag_OrderType order, Integer m, Integer n, double a[],
     double b[], double tol, double *relerr, double x[], double *resmax,
     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_∞ 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_∞ norm of the residuals (the absolutely largest residual)

$$r(x) = \max_{1 \leq i \leq m} |r_i|$$

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 ith element of b and x_j the jth element of x. The matrix A need not be of full rank. The solution is not unique in this case, and may not be unique even if A is of full rank.

Alternatively, in applications where a complete minimization of the l_∞ norm is not necessary, the user may obtain an approximate solution, usually in shorter time, by giving an appropriate value to the parameter **relerr**.

Typically in applications to data fitting, data consisting of m points with co-ordinates (t_i, y_i) is to be approximated in the l_∞ 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 finding an l_∞ 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 above 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 dual formation of the l_∞ problem (see Barrodale and Phillips (1974) and Barrodale and Phillips (1975)). The modifications are designed to improve the efficiency and stability of the simplex method for this particular application.

4 References

Barrodale I and Phillips C (1974) An improved algorithm for discrete Chebyshev linear approximation *Proc. 4th Manitoba Conf. Numerical Mathematics* 177–190 University of Manitoba, Canada

Barrodale I and Phillips C (1975) Solution of an overdetermined system of linear equations in the Chebyshev norm [F4] (Algorithm 495) *ACM Trans. Math. Software* **1 (3)** 264–270

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: $\mathbf{m} \geq \mathbf{n}$.

3: **n** – Integer *Input*

On entry: the number of unknowns, n (the number of columns of the matrix A).

Constraint: $\mathbf{n} \geq 1$.

4: **a**[dim] – double *Input/Output*

Note: the dimension, dim, of the array **a** must be at least $(\mathbf{n} + 3) \times (\mathbf{m} + 1)$.

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

if **order** = **Nag_ColMajor**, $\mathbf{a}[(j - 1) \times (\mathbf{n} + 3) + i - 1]$;

if **order** = **Nag_RowMajor**, $\mathbf{a}[(i - 1) \times (\mathbf{m} + 1) + j - 1]$.

On entry: $\mathbf{A}(j, i)$ must contain a_{ij}, the element in the ith row and jth column of the matrix A for, $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$ (that is, the **transpose** of the matrix). The remaining elements need not be set. Preferably, the columns of the matrix A (rows of the parameter **a**) should be scaled before entry: see Section 7.

On exit: **a** contains the last simplex tableau.

5: **b**[**m**] – double *Input/Output*

On entry: $\mathbf{b}[i - 1]$ must contain b_i, the ith element of the vector b, for $i = 1, 2, \ldots, m$.

On exit: the ith residual r_i corresponding to the solution vector x, for $i = 1, 2, \ldots, m$. Note however that these residuals may contain few significant figures, especially when **resmax** is within one or two orders of magnitude of **tol**. Indeed if $\mathbf{resmax} \leq \mathbf{tol}$, the elements $\mathbf{b}[i - 1]$ may all be set to zero. It is therefore often advisable to compute the residuals directly.

6: **tol** – double *Input*

On entry: a threshold below which numbers are regarded as zero. The recommended threshold value is $10.0 \times \epsilon$, where ϵ is the ***machine precision***. If $\mathbf{tol} \leq 0.0$ on entry, the recommended value is used within the function. If premature termination occurs, a larger value for **tol** may result in a valid solution.

Suggested value: 0.0.

7: **relerr** – double * *Input/Output*

On entry: **relerr** must be set to a bound on the relative error acceptable in the maximum residual at the solution.

If **relerr** ≤ 0.0, then the l_∞ solution is computed, and **relerr** is set to 0.0 on exit.

If **relerr** > 0.0, then the function obtains instead an approximate solution for which the largest residual is less than $1.0 +$ **relerr** times that of the l_∞ solution; on exit, **relerr** contains a smaller value such that the above bound still applies. (The usual result of this option, say with **relerr** $= 0.1$, is a saving in the number of simplex iterations).

On exit: **relerr** is altered as described above.

8: **x**[**n**] – double *Output*

On exit: if an optimal but not necessarily unique solution is found, **x**[$j-1$] contains the jth element of the solution vector x, for $j = 1, 2, \ldots, n$. Whether this is an l_∞ solution or an approximation to one, depends on the value of **relerr** on entry.

9: **resmax** – double * *Output*

On exit: if an optimal but not necessarily unique solution is found, **resmax** contains the absolute value of the largest residual(s) for the solution vector x. (See **b** above.)

10: **rank** – Integer * *Output*

On exit: if an optimal but not necessarily unique solution is found, **rank** contains the computed rank of the matrix A.

11: **iter** – Integer * *Output*

On exit: if an optimal but not necessarily unique solution is found, **iter** contains the number of iterations taken by the simplex method.

12: **fail** – NagError * *Input/Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 1.

NE_INT_2

On entry, **m** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: **m** $\geq$ **n**.

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 **tol**: **tol** = ⟨*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 $n+1$ equations which have residuals of largest absolute value. 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 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 **tol** 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 jth 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.

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

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

nag_linf_fit (e02gcc) is used to solve these in the l_∞ sense.

9.1 Program Text

```
/* nag_linf_fit (e02gcc) Example Program.
 *
 * Copyright 2001 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 relerr, resmax, t, tol;
  Integer exit_status, i, irank, iter, m, n, 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;
  Vprintf("e02gcc Example Program Results\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");

  n = 3;
  Vscanf("%ld%*[^\n] ", &m);
  if (m > 0)
    {
      /* Allocate memory */
      if ( !(a = NAG_ALLOC((n+3)*(m+1), double)) ||
           !(b = NAG_ALLOC(m, double)) ||
           !(x = NAG_ALLOC(n, double)) )
        {
          Vprintf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }

      if (order == Nag_ColMajor)
        pda = n + 3;
      else
        pda = m + 1;

      for (i = 1; i <= m; ++i)
        {
          Vscanf("%lf%lf%*[^\n] ", &t, &b[i-1]);
          A(1, i) = exp(t);
          A(2, i) = exp(-t);
          A(3, i) = 1.0;
        }
      tol = 0.0;
      relerr = 0.0;
      e02gcc(order, m, n, a, b, tol, &relerr, x, &resmax, &irank, &iter, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from e02gcc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }
      else
        {
          Vprintf("\n");
          Vprintf("resmax = %10.2e  Rank = %5ld  Iterations = %5ld\n",
                  resmax, irank, 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

```
e02gcc Example Program Data
  5
   0.0 4.501
   0.2 4.360
   0.4 4.333
   0.6 4.418
   0.8 4.625
```

9.3 Program Results

```
e02gcc Example Program Results

resmax =   1.03e-03  Rank =     3  Iterations =     4

Solution
    1.0049    2.0149    1.4822
```