NAG Library Function Document

nag_dpoequ (f07ffc)

1 Purpose

nag_dpoequ (f07ffc) computes a diagonal scaling matrix S intended to equilibrate a real n by n symmetric positive definite matrix A and reduce its condition number.

2 Specification

```
#include <nag.h>
#include <nagf07.h>
```

```
void nag_dpoequ (Nag_OrderType order, Integer n, const double a[],
     Integer pda, double s[], double *scond, double *amax, NagError *fail)
```

3 Description

nag_dpoequ (f07ffc) computes a diagonal scaling matrix S chosen so that

$$s_j = 1/\sqrt{a_{jj}}.$$

This means that the matrix B given by

$$B = SAS,$$

has diagonal elements equal to unity. This in turn means that the condition number of B, $\kappa_2(B)$, is within a factor n of the matrix of smallest possible condition number over all possible choices of diagonal scalings (see Corollary 7.6 of Higham (2002)).

4 References

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

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: **n** – Integer *Input*

On entry: n, the order of the matrix A.

Constraint: **n** ≥ 0.

3: **a**[*dim*] – const double *Input*

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

The (i,j)th element of the matrix A is stored in

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the matrix A whose scaling factors are to be computed. Only the diagonal elements of the array **a** are referenced.

4: **pda** – Integer *Input*

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

Constraint: **pda** $\geq \max(1, \mathbf{n})$.

5: **s**[**n**] – double *Output*

On exit: if **fail.code** = NE_NOERROR, **s** contains the diagonal elements of the scaling matrix S.

6: **scond** – double * *Output*

On exit: if **fail.code** = NE_NOERROR, **scond** contains the ratio of the smallest value of **s** to the largest value of **s**. If **scond** ≥ 0.1 and **amax** is neither too large nor too small, it is not worth scaling by S.

7: **amax** – double * *Output*

On exit: $\max|a_{ij}|$. If **amax** is very close to overflow or underflow, the matrix A should be scaled.

8: **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_INT

On entry, **n** = $\langle value\rangle$.
Constraint: **n** ≥ 0.

On entry, **pda** = $\langle value\rangle$.
Constraint: **pda** > 0.

NE_INT_2

On entry, **pda** = $\langle value\rangle$ and **n** = $\langle value\rangle$.
Constraint: **pda** $\geq \max(1, \mathbf{n})$.

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_MAT_NOT_POS_DEF

The $\langle value\rangle$th diagonal element of A is not positive (and hence A cannot be positive definite).

7 Accuracy

The computed scale factors will be close to the exact scale factors.

8 Parallelism and Performance

nag_dpoequ (f07ffc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The complex analogue of this function is nag_zpoequ (f07ftc).

10 Example

This example equilibrates the symmetric positive definite matrix A given by

$$A = \begin{pmatrix} 4.16 & -3.12 \times 10^5 & 0.56 & -0.10 \\ -3.12 \times 10^5 & 5.03 \times 10^{10} & -0.83 \times 10^5 & 1.18 \times 10^5 \\ 0.56 & -0.83 \times 10^5 & 0.76 & 0.34 \\ -0.10 & 1.18 \times 10^5 & 0.34 & 1.18 \end{pmatrix}.$$

Details of the scaling factors and the scaled matrix are output.

10.1 Program Text

```
/* nag_dpoequ (f07ffc) Example Program.
 *
 * Copyright 2008 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx02.h>

int main(void)
{
  /* Scalars */
  double        amax, big, scond, small;
  Integer       i, j, n, pda;
  Integer       exit_status = 0;
  /* Arrays */
  double        *a = 0, *s = 0;

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;

#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);

  printf("nag_dpoequ (f07ffc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n]");
  scanf("%ld%*[^\n]", &n);
```

```
  if (n < 0)
    {
      printf("Invalid n\n");
      exit_status = 1;
      goto END;
    }

  pda = n;
  /* Allocate memory */
  if (!(a = NAG_ALLOC(n * n, double)) ||
      !(s = NAG_ALLOC(n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read the upper triangular part of the matrix A from data file */
  for (i = 1; i <= n; ++i)
    for (j = i; j <= n; ++j) scanf("%lf", &A(i, j));
  scanf("%*[^\n]");

  /* Print the matrix A using nag_gen_real_mat_print (x04cac). */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_UpperMatrix, Nag_NonUnitDiag, n, n, a, pda,
                         "Matrix A", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\n");
  /* Compute diagonal scaling factors using nag_dpoequ (f07ffc).*/
  nag_dpoequ(order, n, a, pda, s, &scond, &amax, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dpoequ (f07ffc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print scond, amax and the scale factors */
  printf("scond = %10.1e, amax = %10.1e\n\n", scond, amax);
  printf("Diagonal scaling factors\n");
  for (i = 0; i < n; ++i) printf("%11.1e%s", s[i], i%7 == 6?"\n":" ");
  printf("\n\n");

  /* Compute values close to underflow and overflow using
   * nag_real_safe_small_number (x02amc), nag_machine_precision (x02ajc) and
   * nag_real_base (x02bhc)
   */
  small = nag_real_safe_small_number / (nag_machine_precision * nag_real_base);
  big = 1.0 / small;
  if (scond < 0.1 || amax < small || amax > big)
    {
      /* Scale A */
      for (j = 1; j <= n; ++j)
        for (i = 1; i <= j; ++i) A(i, j) *= s[i-1] * s[j-1];

      /* Print the scaled matrix using nag_gen_real_mat_print (x04cac). */
      fflush(stdout);
      nag_gen_real_mat_print(order, Nag_UpperMatrix, Nag_NonUnitDiag, n, n, a,
                             pda, "Scaled matrix", 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
                  fail.message);
          exit_status = 1;
```

```
          goto END;
        }
    }

 END:
  NAG_FREE(a);
  NAG_FREE(s);

  return exit_status;
}
#undef A
```

10.2 Program Data

```
nag_dpoequ (f07ffc) Example Program Data
  4                                         : n
  4.16      -3.12e+05   0.56      -0.10
             5.03e+10  -0.83e+05   1.18e+05
                        0.76       0.34
                                   1.18     : matrix A
```

10.3 Program Results

```
nag_dpoequ (f07ffc) Example Program Results

 Matrix A
              1             2             3             4
 1    4.1600e+00  -3.1200e+05   5.6000e-01  -1.0000e-01
 2                 5.0300e+10  -8.3000e+04   1.1800e+05
 3                               7.6000e-01   3.4000e-01
 4                                             1.1800e+00

scond =    3.9e-06, amax =    5.0e+10

Diagonal scaling factors
    4.9e-01     4.5e-06     1.1e+00     9.2e-01

 Scaled matrix
            1          2          3          4
 1      1.0000    -0.6821     0.3149    -0.0451
 2                 1.0000    -0.4245     0.4843
 3                            1.0000     0.3590
 4                                       1.0000
```