NAG Library Function Document

nag_dppequ (f07gfc)

1 Purpose

nag_dppequ (f07gfc) computes a diagonal scaling matrix S intended to equilibrate a real n by n symmetric positive definite matrix A, stored in packed format, and reduce its condition number.

2 Specification

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

```
void nag_dppequ (Nag_OrderType order, Nag_UploType uplo, Integer n,
     const double ap[], double s[], double *scond, double *amax,
     NagError *fail)
```

3 Description

nag_dppequ (f07gfc) 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: **uplo** – Nag_UploType *Input*

On entry: indicates whether the upper or lower triangular part of A is stored in the array **ap**, as follows:

uplo = Nag_Upper
The upper triangle of A is stored.

uplo = Nag_Lower
The lower triangle of A is stored.

Constraint: **uplo** = Nag_Upper or Nag_Lower.

3: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 0$.

4: **ap**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **ap** must be at least $\max(1, \mathbf{n} \times (\mathbf{n} + 1)/2)$.

On entry: the n by n symmetric matrix A, packed by rows or columns.

The storage of elements A_{ij} depends on the **order** and **uplo** arguments as follows:

if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Upper',
A_{ij} is stored in $\mathbf{ap}[(j-1) \times j/2 + i - 1]$, for $i \leq j$;
if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Lower',
A_{ij} is stored in $\mathbf{ap}[(2n-j) \times (j-1)/2 + i - 1]$, for $i \geq j$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Upper',
A_{ij} is stored in $\mathbf{ap}[(2n-i) \times (i-1)/2 + j - 1]$, for $i \leq j$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Lower',
A_{ij} is stored in $\mathbf{ap}[(i-1) \times i/2 + j - 1]$, for $i \geq j$.

Only the elements of **ap** corresponding to the diagonal elements A are referenced.

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 $\mathbf{scond} \geq 0.1$ 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_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 0$.

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_MAT_NOT_POS_DEF

The ⟨*value*⟩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_dppequ (f07gfc) 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_zppequ (f07gtc).

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_dppequ (f07gfc) Example Program.
 *
 * Copyright 2004 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       exit_status = 0, i, j, n;


  /* Arrays */
  double        *ap = 0, *s = 0;
  char          nag_enum_arg[40];

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

#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I, J) ap[J*(J-1)/2 + I - 1]
#define A_LOWER(I, J) ap[(2*n-J)*(J-1)/2 + I - 1]
  order = Nag_ColMajor;
#else
#define A_LOWER(I, J) ap[I*(I-1)/2 + J - 1]
```

```
#define A_UPPER(I, J) ap[(2*n-I)*(I-1)/2 + J - 1]
  order = Nag_RowMajor;
#endif
  INIT_FAIL(fail);

  printf("nag_dppequ (f07gfc) 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;
    }
  scanf(" %39s%*[^\n]", nag_enum_arg);
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

  /* Allocate memory */
  if (!(ap = NAG_ALLOC(n*(n+1)/2, double)) ||
      !(s =  NAG_ALLOC(n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read the upper or lower triangular part of the matrix A from data file */
  if (uplo == Nag_Upper)
    for (i = 1; i <= n; ++i)
      for (j = i; j <= n; ++j) scanf("%lf", &A_UPPER(i, j));
  else if (uplo == Nag_Lower)
    for (i = 1; i <= n; ++i)
      for (j = 1; j <= i; ++j) scanf("%lf", &A_LOWER(i, j));
  scanf("%*[^\n]");

  /* Print the matrix A using nag_pack_real_mat_print (x04ccc). */
  fflush(stdout);
  nag_pack_real_mat_print(order, uplo, Nag_NonUnitDiag, n, ap,
                          "Matrix A", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_pack_real_mat_print (x04ccc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  printf("\n");

  /* Compute diagonal scaling factors using nag_dppequ (f07gfc). */
  nag_dppequ(order, uplo, n, ap, s, &scond, &amax, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dppequ (f07gfc).\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. / small;
  if (scond < 0.1 || amax < small || amax > big)
    {
      /* Scale A */
      if (uplo == Nag_Upper)
        for (j = 1; j <= n; ++j)
          for (i = 1; i <= j; ++i) A_UPPER(i, j) *= s[i-1] * s[j-1];
      else
        for (j = 1; j <= n; ++j)
          for (i = j; i <= n; ++i) A_LOWER(i, j) *= s[i-1] * s[j-1];

      /* Print the scaled matrix using nag_pack_real_mat_print (x04ccc). */
      fflush(stdout);
      nag_pack_real_mat_print(order, uplo, Nag_NonUnitDiag, n, ap,
                              "Scaled matrix", 0, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_pack_real_mat_print (x04ccc).\n%s\n",
                 fail.message);
          exit_status = 1;
          goto END;
        }
    }
 END:
  NAG_FREE(ap);
  NAG_FREE(s);

  return exit_status;
}
#undef A_UPPER
#undef A_LOWER
```

10.2 Program Data

```
nag_dppequ (f07gfc) Example Program Data
  4                                         : n
  Nag_Upper                                 : uplo
  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_dppequ (f07gfc) 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
```