NAG Library Function Document

nag_dpbequ (f07hfc)

1 Purpose

nag_dpbequ (f07hfc) computes a diagonal scaling matrix S intended to equilibrate a real n by n symmetric positive definite band matrix A, with bandwidth $(2k_d + 1)$, and reduce its condition number.

2 Specification

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

void nag_dpbequ (Nag_OrderType order, Nag_UploType uplo, Integer n,
     Integer kd, const double ab[], Integer pdab, double s[], double *scond,
     double *amax, NagError *fail)
```

3 Description

nag_dpbequ (f07hfc) 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 **ab**, 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: **kd** – Integer *Input*

On entry: k_d, the number of superdiagonals of the matrix A if **uplo** = Nag_Upper, or the number of subdiagonals if **uplo** = Nag_Lower.

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

5: **ab**[*dim*] – const double *Input*

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

On entry: the upper or lower triangle of the symmetric positive definite band matrix A whose scaling factors are to be computed.

This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements of A_{ij}, depends on the **order** and **uplo** arguments as follows:

if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Upper',
A_{ij} is stored in $\mathbf{ab}[k_d + i - j + (j-1) \times \mathbf{pdab}]$, for $j = 1, \ldots, n$ and $i = \max(1, j - k_d), \ldots, j$;
if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Lower',
A_{ij} is stored in $\mathbf{ab}[i - j + (j-1) \times \mathbf{pdab}]$, for $j = 1, \ldots, n$ and $i = j, \ldots, \min(n, j + k_d)$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Upper',
A_{ij} is stored in $\mathbf{ab}[j - i + (i-1) \times \mathbf{pdab}]$, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k_d)$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Lower',
A_{ij} is stored in $\mathbf{ab}[k_d + j - i + (i-1) \times \mathbf{pdab}]$, for $i = 1, \ldots, n$ and $j = \max(1, i - k_d), \ldots, i$.

Only the elements of the array **ab** corresponding to the diagonal elements of A are referenced. (Row $(k_d + 1)$ of **ab** when **uplo** = Nag_Upper, row 1 of **ab** when **uplo** = Nag_Lower.)

6: **pdab** – Integer *Input*

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

Constraint: $\mathbf{pdab} \geq \mathbf{kd} + 1$.

7: **s**[**n**] – double *Output*

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

8: **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.

9: **amax** – double * *Output*

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

10: **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 $\langle value \rangle$ had an illegal value.

NE_INT

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

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

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

NE_INT_2

On entry, **pdab** $= \langle value \rangle$ and **kd** $= \langle value \rangle$.
Constraint: **pdab** $\geq$ **kd** $+ 1$.

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

Not applicable.

9 Further Comments

The complex analogue of this function is nag_zpbequ (f07htc).

10 Example

This example equilibrates the symmetric positive definite matrix A given by

$$A = \begin{pmatrix} 5.49 & 2.68 \times 10^{10} & 0 & 0 \\ 2.68 \times 10^{10} & 5.63 \times 10^{20} & -2.39 \times 10^{10} & 0 \\ 0 & -2.39 \times 10^{10} & 2.60 & -2.22 \\ 0 & 0 & -2.22 & 5.17 \end{pmatrix}.$$

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

10.1 Program Text

```
/* nag_dpbequ (f07hfc) 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, kd, kd1, kd2, n, pdab;

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

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

#ifdef NAG_COLUMN_MAJOR
#define AB_UPPER(I, J) ab[(J-1)*pdab + kd + I - J]
#define AB_LOWER(I, J) ab[(J-1)*pdab + I - J]
  order = Nag_ColMajor;
#else
#define AB_UPPER(I, J) ab[(I-1)*pdab + J - I]
#define AB_LOWER(I, J) ab[(I-1)*pdab + kd + J - I]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dpbequ (f07hfc) Example Program Results\n\n");
  /* Skip heading in data file */
  scanf("%*[^\n]");
  scanf("%ld%ld%*[^\n]", &n, &kd);
  if (n < 0 || kd < 0)
    {
      printf("Invalid n or kd\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);

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


  /* Read the upper or lower triangular part of the band matrix A
   * from data file.
   */
  if (uplo == Nag_Upper)
    {
      kd1 = 0;
      kd2 = kd;
      for (i = 1; i <= n; ++i)
        for (j = i; j <= MIN(n, i + kd); ++j) scanf("%lf", &AB_UPPER(i, j));
```

```
    }
  else
    {
      kd1 = kd;
      kd2 = 0;
      for (i = 1; i <= n; ++i)
        for (j = MAX(1, i - kd); j <= i; ++j) scanf("%lf", &AB_LOWER(i, j));
    }
  scanf("%*[^\n]");

  /* Print the matrix A using nag_band_real_mat_print (x04cec). */

  fflush(stdout);
  nag_band_real_mat_print(order, n, n, kd1, kd2, ab, pdab, "Matrix A", 0,
                          &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_band_real_mat_print (x04cec).\n%s\n",
             fail.message);
      exit_status = 2;
      goto END;
    }
  printf("\n");

  /* Compute diagonal scaling factors using nag_dpbequ (f07hfc). */
  nag_dpbequ(order, uplo, n, kd, ab, pdab, s, &scond, &amax, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dpbequ (f07hfc).\n%s\n", fail.message);
      exit_status = 3;
      goto END;
    }
  /* Print scond, amax and the scale factors */
  printf("scond = %10.1e, amax = %10.1e\n", scond, amax);
  printf("\nDiagonal 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 */
      if (uplo == Nag_Upper)
        for (j = 1; j <= n; ++j)
          for (i = MAX(1, j - kd); i <= j; ++i)
            AB_UPPER(i, j) *= s[i-1] * s[j-1];
      else
        for (j = 1; j <= n; ++j)
          for (i = j; i <= MIN(n, j + kd); ++i)
            AB_LOWER(i, j) *= s[i-1] * s[j-1];

      /* Print the scaled matrix using nag_band_real_mat_print (x04cec). */
      fflush(stdout);
      nag_band_real_mat_print(order, n, n, kd1, kd2, ab, pdab, "Scaled matrix",
                              0, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_band_real_mat_print (x04cec).\n%s\n",
                 fail.message);
          exit_status = 4;
          goto END;
        }
    }
 END:
  NAG_FREE(ab);
```

```
  NAG_FREE(s);

  return exit_status;
}
#undef AB_UPPER
#undef AB_LOWER
```

10.2 Program Data

```
nag_dpbequ (f07hfc) Example Program Data
  4  1                                         : n and kd
  Nag_Upper                                    : uplo
  5.49       2.68e+10
             5.63e+20  -2.39e+10
                        2.60      -2.22
                                   5.17        : matrix A
```

10.3 Program Results

```
nag_dpbequ (f07hfc) Example Program Results

 Matrix A
             1            2            3            4
 1    5.4900e+00   2.6800e+10
 2                 5.6300e+20  -2.3900e+10
 3                               2.6000e+00  -2.2200e+00
 4                                            5.1700e+00

scond =    6.8e-11, amax =    5.6e+20

Diagonal scaling factors
    4.3e-01     4.2e-11     6.2e-01     4.4e-01

 Scaled matrix
           1          2          3          4
 1      1.0000     0.4821
 2                 1.0000    -0.6247
 3                            1.0000    -0.6055
 4                                       1.0000
```