NAG Library Function Document

nag_dspcon (f07pgc)

1 Purpose

nag_dspcon (f07pgc) estimates the condition number of a real symmetric indefinite matrix A, where A has been factorized by nag_dsptrf (f07pdc), using packed storage.

2 Specification

```
#include <nag.h>
#include <nagf07.h>
void nag_dspcon (Nag_OrderType order, Nag_UploType uplo, Integer n,
     const double ap[], const Integer ipiv[], double anorm, double *rcond,
     NagError *fail)
```

3 Description

nag_dspcon (f07pgc) estimates the condition number (in the 1-norm) of a real symmetric indefinite matrix A:

$$\kappa_1(A) = \|A\|_1 \left\|A^{-1}\right\|_1.$$

Since A is symmetric, $\kappa_1(A) = \kappa_\infty(A) = \|A\|_\infty \left\|A^{-1}\right\|_\infty$.

Because $\kappa_1(A)$ is infinite if A is singular, the function actually returns an estimate of the **reciprocal** of $\kappa_1(A)$.

The function should be preceded by a call to nag_dsp_norm (f16rdc) to compute $\|A\|_1$ and a call to nag_dsptrf (f07pdc) to compute the Bunch–Kaufman factorization of A. The function then uses Higham's implementation of Hager's method (see Higham (1988)) to estimate $\left\|A^{-1}\right\|_1$.

4 References

Higham N J (1988) FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation *ACM Trans. Math. Software* **14** 381–396

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: specifies how A has been factorized.

uplo = Nag_Upper
$A = PUDU^{\mathrm{T}}P^{\mathrm{T}}$, where U is upper triangular.

uplo = Nag_Lower
$A = PLDL^{\mathrm{T}}P^{\mathrm{T}}$, where L is lower triangular.

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 factorization of A stored in packed form, as returned by nag_dsptrf (f07pdc).

5: **ipiv**[dim] – const Integer *Input*

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

On entry: details of the interchanges and the block structure of D, as returned by nag_dsptrf (f07pdc).

6: **anorm** – double *Input*

On entry: the 1-norm of the **original** matrix A, which may be computed by calling nag_dsp_norm (f16rdc) with its argument **norm** = Nag_OneNorm. **anorm** must be computed either **before** calling nag_dsptrf (f07pdc) or else from a **copy** of the original matrix A.

Constraint: **anorm** ≥ 0.0.

7: **rcond** – double * *Output*

On exit: an estimate of the reciprocal of the condition number of A. **rcond** is set to zero if exact singularity is detected or the estimate underflows. If **rcond** is less than ***machine precision***, A is singular to working precision.

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.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

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

NE_INT

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

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_REAL

On entry, **anorm** $= \langle value \rangle$.
Constraint: **anorm** ≥ 0.0.

7 Accuracy

The computed estimate **rcond** is never less than the true value ρ, and in practice is nearly always less than 10ρ, although examples can be constructed where **rcond** is much larger.

8 Parallelism and Performance

nag_dspcon (f07pgc) is not threaded by NAG in any implementation.

nag_dspcon (f07pgc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

A call to nag_dspcon (f07pgc) involves solving a number of systems of linear equations of the form $Ax = b$; the number is usually 4 or 5 and never more than 11. Each solution involves approximately $2n^2$ floating-point operations but takes considerably longer than a call to nag_dsptrs (f07pec) with one right-hand side, because extra care is taken to avoid overflow when A is approximately singular.

The complex analogues of this function are nag_zhpcon (f07puc) for Hermitian matrices and nag_zspcon (f07quc) for symmetric matrices.

10 Example

This example estimates the condition number in the 1-norm (or ∞-norm) of the matrix A, where

$$A = \begin{pmatrix} 2.07 & 3.87 & 4.20 & -1.15 \\ 3.87 & -0.21 & 1.87 & 0.63 \\ 4.20 & 1.87 & 1.15 & 2.06 \\ -1.15 & 0.63 & 2.06 & -1.81 \end{pmatrix}.$$

Here A is symmetric indefinite, stored in packed form, and must first be factorized by nag_dsptrf (f07pdc). The true condition number in the 1-norm is 75.68.

10.1 Program Text

```
/* nag_dspcon (f07pgc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagf16.h>
```

```
#include <nagx02.h>

int main(void)
{
  /* Scalars */
  double        anorm, rcond;
  Integer       ap_len, i, j, n;
  Integer       exit_status = 0;
  NagError      fail;
  Nag_UploType  uplo;
  Nag_OrderType order;
  /* Arrays */
  Integer       *ipiv = 0;
  char          nag_enum_arg[40];
  double        *ap = 0;

#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_dspcon (f07pgc) Example Program Results\n\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n] ", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n] ", &n);
#endif
  ap_len = n * (n + 1)/2;

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

  /* Read A from data file */
#ifdef _WIN32
  scanf_s(" %39s%*[^\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf(" %39s%*[^\n] ", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

  if (uplo == Nag_Upper)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
#ifdef _WIN32
            scanf_s("%lf", &A_UPPER(i, j));
#else
            scanf("%lf", &A_UPPER(i, j));
```

```
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
#ifdef _WIN32
            scanf_s("%lf", &A_LOWER(i, j));
#else
            scanf("%lf", &A_LOWER(i, j));
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }

  /* Compute norm of A */
  /* nag_dsp_norm (f16rdc).
   * 1-norm, infinity-norm, Frobenius norm, largest absolute
   * element, real symmetric matrix, packed storage
   */
  nag_dsp_norm(order, Nag_OneNorm, uplo, n, ap, &anorm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dsp_norm (f16rdc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Factorize A */
  /* nag_dsptrf (f07pdc).
   * Bunch-Kaufman factorization of real symmetric indefinite
   * matrix, packed storage
   */
  nag_dsptrf(order, uplo, n, ap, ipiv, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dsptrf (f07pdc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Estimate condition number */
  /* nag_dspcon (f07pgc).
   * Estimate condition number of real symmetric indefinite
   * matrix, matrix already factorized by nag_dsptrf (f07pdc),
   * packed storage
   */
  nag_dspcon(order, uplo, n, ap, ipiv, anorm, &rcond, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dspcon (f07pgc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* nag_machine_precision (x02ajc).
   * The machine precision
   */
  if (rcond >= nag_machine_precision)
    printf("Estimate of condition number =%11.2e\n\n", 1.0/rcond);
  else
    printf("A is singular to working precision\n");
```

```
 END:
  NAG_FREE(ipiv);
  NAG_FREE(ap);
  return exit_status;
}
```

10.2 Program Data

```
nag_dspcon (f07pgc) Example Program Data
  4                           :Value of n
  Nag_Lower                   :Value of uplo
  2.07
  3.87  -0.21
  4.20   1.87   1.15
 -1.15   0.63   2.06  -1.81   :End of matrix A
```

10.3 Program Results

```
nag_dspcon (f07pgc) Example Program Results

Estimate of condition number =   7.57e+01
```