NAG Library Function Document

nag_dsptrf (f07pdc)

1 Purpose

nag_dsptrf (f07pdc) computes the Bunch–Kaufman factorization of a real symmetric indefinite matrix, using packed storage.

2 Specification

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

```
void nag_dsptrf (Nag_OrderType order, Nag_UploType uplo, Integer n,
     double ap[], Integer ipiv[], NagError *fail)
```

3 Description

nag_dsptrf (f07pdc) factorizes a real symmetric matrix A, using the Bunch–Kaufman diagonal pivoting method and packed storage. A is factorized as either $A = PUDU^{\mathrm{T}}P^{\mathrm{T}}$ if **uplo** $=$ Nag_Upper or $A = PLDL^{\mathrm{T}}P^{\mathrm{T}}$ if **uplo** $=$ Nag_Lower, where P is a permutation matrix, U (or L) is a unit upper (or lower) triangular matrix and D is a symmetric block diagonal matrix with 1 by 1 and 2 by 2 diagonal blocks; U (or L) has 2 by 2 unit diagonal blocks corresponding to the 2 by 2 blocks of D. Row and column interchanges are performed to ensure numerical stability while preserving symmetry.

This method is suitable for symmetric matrices which are not known to be positive definite. If A is in fact positive definite, no interchanges are performed and no 2 by 2 blocks occur in D.

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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 2.3.1.3 in How to Use the NAG Library and its Documentation 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 whether the upper or lower triangular part of A is stored and how A is to be factorized.

uplo $=$ Nag_Upper
The upper triangular part of A is stored and A is factorized as $PUDU^{\mathrm{T}}P^{\mathrm{T}}$, where U is upper triangular.

uplo $=$ Nag_Lower
The lower triangular part of A is stored and A is factorized as $PLDL^{\mathrm{T}}P^{\mathrm{T}}$, 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*] – double *Input/Output*

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 $i \geq j$.

On exit: A is overwritten by details of the block diagonal matrix D and the multipliers used to obtain the factor U or L as specified by **uplo**.

5: **ipiv**[**n**] – Integer *Output*

On exit: details of the interchanges and the block structure of D. More precisely,

if $\mathbf{ipiv}[i-1] = k > 0$, d_{ii} is a 1 by 1 pivot block and the ith row and column of A were interchanged with the kth row and column;

if **uplo** = Nag_Upper and $\mathbf{ipiv}[i-2] = \mathbf{ipiv}[i-1] = -l < 0$, $\begin{pmatrix} d_{i-1,i-1} & \bar{d}_{i,i-1} \\ \bar{d}_{i,i-1} & d_{ii} \end{pmatrix}$ is a 2 by 2 pivot block and the $(i-1)$th row and column of A were interchanged with the lth row and column;

if **uplo** = Nag_Lower and $\mathbf{ipiv}[i-1] = \mathbf{ipiv}[i] = -m < 0$, $\begin{pmatrix} d_{ii} & d_{i+1,i} \\ d_{i+1,i} & d_{i+1,i+1} \end{pmatrix}$ is a 2 by 2 pivot block and the $(i+1)$th row and column of A were interchanged with the mth row and column.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_SINGULAR

Element $\langle value \rangle$ of the diagonal is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

7 Accuracy

If **uplo** $=$ Nag_Upper, the computed factors U and D are the exact factors of a perturbed matrix $A + E$, where

$$|E| \leq c(n)\epsilon P|U||D||U^{\mathrm{T}}|P^{\mathrm{T}},$$

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***.

If **uplo** $=$ Nag_Lower, a similar statement holds for the computed factors L and D.

8 Parallelism and Performance

nag_dsptrf (f07pdc) 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

The elements of D overwrite the corresponding elements of A; if D has 2 by 2 blocks, only the upper or lower triangle is stored, as specified by **uplo**.

The unit diagonal elements of U or L and the 2 by 2 unit diagonal blocks are not stored. The remaining elements of U or L overwrite elements in the corresponding columns of A, but additional row interchanges must be applied to recover U or L explicitly (this is seldom necessary). If **ipiv**$[i-1] = i$, for $i = 1, 2, \ldots, n$ (as is the case when A is positive definite), then U or L are stored explicitly in packed form (except for their unit diagonal elements which are equal to 1).

The total number of floating-point operations is approximately $\frac{1}{3}n^3$.

A call to nag_dsptrf (f07pdc) may be followed by calls to the functions:

nag_dsptrs (f07pec) to solve $AX = B$;

nag_dspcon (f07pgc) to estimate the condition number of A;

nag_dsptri (f07pjc) to compute the inverse of A.

The complex analogues of this function are nag_zhptrf (f07prc) for Hermitian matrices and nag_zsptrf (f07qrc) for symmetric matrices.

10 Example

This example computes the Bunch–Kaufman factorization of the matrix A, where

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

using packed storage.

10.1 Program Text

```
/* nag_dsptrf (f07pdc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

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

#ifdef NAG_LOAD_FP
  /* The following line is needed to force the Microsoft linker
     to load floating point support */
  float force_loading_of_ms_float_support = 0;
#endif /* NAG_LOAD_FP */

#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]
#define B(I, J)       b[(J-1)*pdb + 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]
#define B(I, J)       b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dsptrf (f07pdc) 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 "%" NAG_IFMT "%*[^\n] ", &n, &nrhs);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &n, &nrhs);
#endif
  ap_len = n * (n + 1) / 2;
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
#else
  pdb = nrhs;
#endif

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

  /* Read A and B from data file */
#ifdef _WIN32
  scanf_s(" %39s%*[^\n] ", nag_enum_arg, (unsigned)_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
  }
  for (i = 1; i <= n; ++i) {
    for (j = 1; j <= nrhs; ++j)
#ifdef _WIN32
      scanf_s("%lf", &B(i, j));
#else
      scanf("%lf", &B(i, j));
#endif
  }
#ifdef _WIN32
```

```
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* 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;
  }
  /* Compute solution */
  /* nag_dsptrs (f07pec).
   * Solution of real symmetric indefinite system of linear
   * equations, multiple right-hand sides, matrix already
   * factorized by nag_dsptrf (f07pdc), packed storage
   */
  nag_dsptrs(order, uplo, n, nrhs, ap, ipiv, b, pdb, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dsptrs (f07pec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Print solution */
  /* nag_gen_real_mat_print (x04cac).
   * Print real general matrix (easy-to-use)
   */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs,
                         b, pdb, "Solution(s)", 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(ap);
  NAG_FREE(ipiv);
  NAG_FREE(b);
  return exit_status;
}
```

10.2 Program Data

```
nag_dsptrf (f07pdc) Example Program Data
  4  2                       :Values of n and nrhs
  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
 -9.50  27.85
 -8.38   9.90
 -6.07  19.25
 -0.96   3.93                 :End of matrix B
```

10.3 Program Results

```
nag_dsptrf (f07pdc) Example Program Results

 Solution(s)
             1           2
 1     -4.0000      1.0000
 2     -1.0000      4.0000
 3      2.0000      3.0000
 4      5.0000      2.0000
```