

# NAG Library Function Document

## nag\_dsptri (f07pjc)

### 1 Purpose

nag\_dsptri (f07pjc) computes the inverse 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_dsptri (Nag_OrderType order, Nag_UploType uplo, Integer n,
                double ap[], const Integer ipiv[], NagError *fail)
```

### 3 Description

nag\_dsptri (f07pjc) is used to compute the inverse of a real symmetric indefinite matrix  $A$ , the function must be preceded by a call to nag\_dsptrf (f07pdc), which computes the Bunch–Kaufman factorization of  $A$ , using packed storage.

If **uplo** = Nag\_Upper,  $A = PUDU^T P^T$  and  $A^{-1}$  is computed by solving  $U^T P^T X P U = D^{-1}$ .

If **uplo** = Nag\_Lower,  $A = PLDL^T P^T$  and  $A^{-1}$  is computed by solving  $L^T P^T X P L = D^{-1}$ .

### 4 References

Du Croz J J and Higham N J (1992) Stability of methods for matrix inversion *IMA J. Numer. Anal.* **12** 1–19

### 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  
 $A = PUDU^T P^T$ , where  $U$  is upper triangular.  
**uplo** = Nag\_Lower  
 $A = PLDL^T P^T$ , 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:*  $n \geq 0$ .

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 factorization of *A* stored in packed form, as returned by nag\_dsptf (f07pdc).

*On exit:* the factorization is overwritten by the *n* by *n* matrix  $A^{-1}$ .

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 **ap**[(*j* – 1) × *j*/2 + *i* – 1], for  $i \leq j$ ;  
 if **order** = Nag\_ColMajor and **uplo** = Nag\_Lower,  
 $A_{ij}$  is stored in **ap**[(2*n* – *j*) × (*j* – 1)/2 + *i* – 1], for  $i \geq j$ ;  
 if **order** = Nag\_RowMajor and **uplo** = Nag\_Upper,  
 $A_{ij}$  is stored in **ap**[(2*n* – *i*) × (*i* – 1)/2 + *j* – 1], for  $i \leq j$ ;  
 if **order** = Nag\_RowMajor and **uplo** = Nag\_Lower,  
 $A_{ij}$  is stored in **ap**[(*i* – 1) × *i*/2 + *j* – 1], for  $i \geq j$ .

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\_dsptf (f07pdc).

6: **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  $\langle value \rangle$  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 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\_SINGULAR

Element  $\langle value \rangle$  of the diagonal is exactly zero. *D* is singular and the inverse of *A* cannot be computed.

## 7 Accuracy

The computed inverse  $X$  satisfies a bound of the form

$$\text{if } \mathbf{uplo} = \text{Nag\_Upper}, |DU^T P^T X P U - I| \leq c(n)\epsilon(|D||U^T|P^T|X|P|U| + |D||D^{-1}|);$$

$$\text{if } \mathbf{uplo} = \text{Nag\_Lower}, |DL^T P^T X P L - I| \leq c(n)\epsilon(|D||L^T|P^T|X|P|L| + |D||D^{-1}|),$$

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

## 8 Parallelism and Performance

nag\_dsptri (f07pjc) is not threaded by NAG in any implementation.

nag\_dsptri (f07pjc) 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 total number of floating-point operations is approximately  $\frac{2}{3}n^3$ .

The complex analogues of this function are nag\_zhptri (f07pwc) for Hermitian matrices and nag\_zsptri (f07qwc) for symmetric matrices.

## 10 Example

This example computes the inverse 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).

### 10.1 Program Text

```

/* nag_dsptri (f07pjc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 */

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

int main(void)
{
    /* Scalars */
    Integer      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_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]
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_dsptri (f07pjc) 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

/* Allocate memory */
if (!(ipiv = NAG_ALLOC(n, Integer)) ||
    !(ap = NAG_ALLOC(n * (n + 1)/2, 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
    }

    /* 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 inverse of A */
    /* nag_dsptri (f07pjc).
     * Inverse of real symmetric indefinite matrix, matrix
     * already factorized by nag_dsptrf (f07pdc), packed storage
     */
    nag_dsptri(order, uplo, n, ap, ipiv, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_dsptri (f07pjc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Print inverse */
    /* nag_pack_real_mat_print (x04ccc).
     * Print real packed triangular matrix (easy-to-use)
     */
    fflush(stdout);
    nag_pack_real_mat_print(order, uplo, Nag_NonUnitDiag, n, ap,
                            "Inverse", 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(ipiv);
    NAG_FREE(ap);
    return exit_status;
}

```

## 10.2 Program Data

```
nag_dsptri (f07pjc) 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_dsptri (f07pjc) Example Program Results

Inverse
      1          2          3          4
1      0.7485
2      0.5221   -0.1605
3     -1.0058   -0.3131   1.3501
4     -1.4386   -0.7440   2.0667   2.4547
```

---