NAG Library Function Document

nag_dpftri (f07wjc)

1 Purpose

nag_dpftri (f07wjc) computes the inverse of a real symmetric positive definite matrix using the Cholesky factorization computed by nag_dpftrf (f07wdc) stored in Rectangular Full Packed (RFP) format.

2 Specification

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

```
void nag_dpftri (Nag_OrderType order, Nag_RFP_Store transr,
     Nag_UploType uplo, Integer n, double ar[], NagError *fail)
```

3 Description

nag_dpftri (f07wjc) is used to compute the inverse of a real symmetric positive definite matrix A, stored in RFP format. The RFP storage format is described in Section 3.3.3 in the f07 Chapter Introduction. The function must be preceded by a call to nag_dpftrf (f07wdc), which computes the Cholesky factorization of A.

If **uplo** $=$ Nag_Upper, $A = U^{\mathrm{T}}U$ and A^{-1} is computed by first inverting U and then forming $(U^{-1})U^{-\mathrm{T}}$.

If **uplo** $=$ Nag_Lower, $A = LL^{\mathrm{T}}$ and A^{-1} is computed by first inverting L and then forming $L^{-\mathrm{T}}(L^{-1})$.

4 References

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

Gustavson F G, Waśniewski J, Dongarra J J and Langou J (2010) Rectangular full packed format for Cholesky's algorithm: factorization, solution, and inversion *ACM Trans. Math. Software* **37, 2**

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: **transr** – Nag_RFP_Store *Input*

On entry: specifies whether the RFP representation of A is normal or transposed.

transr $=$ Nag_RFP_Normal
The matrix A is stored in normal RFP format.

transr $=$ Nag_RFP_Trans
The matrix A is stored in transposed RFP format.

Constraint: **transr** $=$ Nag_RFP_Normal or Nag_RFP_Trans.

3: **uplo** – Nag_UploType *Input*

On entry: specifies how A has been factorized.

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

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

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

4: **n** – Integer *Input*

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

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

5: $\mathbf{ar}[\mathbf{n} \times (\mathbf{n}+\mathbf{1})/\mathbf{2}]$ – double *Input/Output*

On entry: the Cholesky factorization of A stored in RFP format, as returned by nag_dpftrf (f07wdc).

On exit: the factorization is overwritten by the n by n matrix A^{-1} stored in RFP format.

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 ⟨*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 3.6.6 in the Essential Introduction for further information.

NE_MAT_NOT_POS_DEF

The leading minor of order ⟨*value*⟩ is not positive definite and the factorization could not be completed. Hence A itself is not positive definite. This may indicate an error in forming the matrix A. There is no function specifically designed to invert a symmetric matrix stored in RFP format which is not positive definite; the matrix must be treated as a full symmetric matrix, by calling nag_dsytri (f07mjc).

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.

7 Accuracy

The computed inverse X satisfies

$$\|XA - I\|_2 \leq c(n)\epsilon\kappa_2(A) \quad \text{and} \quad \|AX - I\|_2 \leq c(n)\epsilon\kappa_2(A),$$

where $c(n)$ is a modest function of n, ϵ is the ***machine precision*** and $\kappa_2(A)$ is the condition number of A defined by

$$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2.$$

8 Parallelism and Performance

nag_dpftri (f07wjc) is not threaded by NAG in any implementation.

nag_dpftri (f07wjc) 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 analogue of this function is nag_zpftri (f07wwc).

10 Example

This example computes the inverse of the matrix A, where

$$A = \begin{pmatrix} 4.16 & -3.12 & 0.56 & -0.10 \\ -3.12 & 5.03 & -0.83 & 1.18 \\ 0.56 & -0.83 & 0.76 & 0.34 \\ -0.10 & 1.18 & 0.34 & 1.18 \end{pmatrix}.$$

Here A is symmetric positive definite, stored in RFP format, and must first be factorized by nag_dpftrf (f07wdc).

10.1 Program Text

```
/* nag_dpftri (f07wjc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

#include <nag.h>
#include <nag_stdlib.h>
#include <nagf01.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  Integer       exit_status = 0;
  Integer       i, j, k, lar1, lar2, lenar, n, pdar, pda, q;
  /* Arrays */
  double        *ar = 0, *a = 0;
  char          nag_enum_arg[40];
  /* NAG types */
```

```
  Nag_RFP_Store transr;
  Nag_UploType  uplo;
  Nag_OrderType order;
  NagError      fail;

#ifdef NAG_COLUMN_MAJOR
  order = Nag_ColMajor;
#define AR(I,J) ar[J*pdar + I]
#else
  order = Nag_RowMajor;
#define AR(I,J) ar[I*pdar + J]
#endif

  INIT_FAIL(fail);

  printf("nag_dpftri (f07wjc) 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);
#else
  scanf("%"NAG_IFMT"", &n);
#endif
#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%39s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n]", nag_enum_arg);
#endif
  transr = (Nag_RFP_Store) nag_enum_name_to_value(nag_enum_arg);

  lenar = (n * (n + 1))/2;
  pda = n;
  if (!(ar = NAG_ALLOC(lenar, double)) ||
      !(a = NAG_ALLOC(pda*n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Setup dimensions for RFP array ar. */
  k = n/2;
  q = n - k;
  if (transr==Nag_RFP_Normal) {
    lar1 = 2*k+1;
    lar2 = q;
  } else {
    lar1 = q;
    lar2 = 2*k+1;
  }
  if (order==Nag_RowMajor) {
    pdar = lar2;
  } else {
    pdar = lar1;
  }
  /* Read matrix into RFP array ar. */
  for (i = 0; i < lar1; i++) {
    for (j = 0; j < lar2; j++) {
#ifdef _WIN32
      scanf_s("%lf ", &AR(i,j));
#else
```

```
      scanf("%lf ", &AR(i,j));
#endif
    }
  }

  /* Factorize A using nag_dpftrf (f07wdc) which performs a Cholesky
   * factorization of a real symmetric positive definite matrix in
   * Rectangular Full Packed format
   */
  nag_dpftrf(order, transr, uplo, n, ar, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dpftrf (f07wdc)\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Compute inverse of A using nag_dpftri (f07wjc) */
  nag_dpftri(order, transr, uplo, n, ar, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dpftri (f07wjc)\n%s\n", fail.message);
    exit_status = 2;
    goto END;
  }

  /* Convert inverse to full array form using nag_dtfttr (f01vgc). */
  nag_dtfttr(order, transr, uplo, n, ar, a, pda, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtfttr (f01vgc)\n%s\n", fail.message);
    exit_status = 3;
    goto END;
  }

  /* nag_gen_real_mat_print (x04cac).
   * Print real general matrix (easy-to-use)
   */
  nag_gen_real_mat_print(order, Nag_LowerMatrix, Nag_NonUnitDiag, n, n, a,
                         pda, "Inverse", 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac)\n%s\n", fail.message);
    exit_status = 4;
  }

 END:
  NAG_FREE(ar);
  NAG_FREE(a);
  return exit_status;
}
```

10.2 Program Data

```
nag_dpftri (f07wjc) Example Program Data
  4     Nag_Lower     Nag_RFP_Normal    : n, uplo, transr

  0.76    0.34
  4.16    1.18
 -3.12    5.03
  0.56   -0.83
 -0.10    1.18                          : ar[]
```

10.3 Program Results

```
nag_dpftri (f07wjc) Example Program Results

 Inverse
           1          2          3          4
 1      0.6995
 2      0.7769     1.4239
 3      0.7508     1.8255     4.0688
 4     -0.9340    -1.8841    -2.9342     3.4978
```