NAG Library Function Document

nag_dpptrs (f07gec)

1 Purpose

nag_dpptrs (f07gec) solves a real symmetric positive definite system of linear equations with multiple right-hand sides,

$$AX = B,$$

where A has been factorized by nag_dpptrf (f07gdc), using packed storage.

2 Specification

```
#include <nag.h>
#include <nagf07.h>
void nag_dpptrs (Nag_OrderType order, Nag_UploType uplo, Integer n,
     Integer nrhs, const double ap[], double b[], Integer pdb,
     NagError *fail)
```

3 Description

nag_dpptrs (f07gec) is used to solve a real symmetric positive definite system of linear equations $AX = B$, the function must be preceded by a call to nag_dpptrf (f07gdc) which computes the Cholesky factorization of A, using packed storage. The solution X is computed by forward and backward substitution.

If **uplo** = Nag_Upper, $A = U^{\mathrm{T}}U$, where U is upper triangular; the solution X is computed by solving $U^{\mathrm{T}}Y = B$ and then $UX = Y$.

If **uplo** = Nag_Lower, $A = LL^{\mathrm{T}}$, where L is lower triangular; the solution X is computed by solving $LY = B$ and then $L^{\mathrm{T}}X = Y$.

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

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

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

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

4: **nrhs** – Integer *Input*

On entry: r, the number of right-hand sides.

Constraint: **nrhs** ≥ 0.

5: **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 Cholesky factor of A stored in packed form, as returned by nag_dpptrf (f07gdc).

6: **b**$[dim]$ – double *Input/Output*

Note: the dimension, *dim*, of the array **b** must be at least

$\max(1, \mathbf{pdb} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdb})$ when **order** = Nag_RowMajor.

The (i,j)th element of the matrix B is stored in

b$[(j-1) \times \mathbf{pdb} + i - 1]$ when **order** = Nag_ColMajor;
b$[(i-1) \times \mathbf{pdb} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the n by r right-hand side matrix B.

On exit: the n by r solution matrix X.

7: **pdb** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, **pdb** $\geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, **pdb** $\geq \max(1, \mathbf{nrhs})$.

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.

On entry, **nrhs** = ⟨*value*⟩.
Constraint: **nrhs** ≥ 0.

On entry, **pdb** = ⟨*value*⟩.
Constraint: **pdb** > 0.

NE_INT_2

On entry, **pdb** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdb** ≥ max(1, **n**).

On entry, **pdb** = ⟨*value*⟩ and **nrhs** = ⟨*value*⟩.
Constraint: **pdb** ≥ max(1, **nrhs**).

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.

7 Accuracy

For each right-hand side vector b, the computed solution x is the exact solution of a perturbed system of equations $(A + E)x = b$, where

if **uplo** = Nag_Upper, $|E| \leq c(n)\epsilon|U^{\mathrm{T}}||U|$;

if **uplo** = Nag_Lower, $|E| \leq c(n)\epsilon|L||L^{\mathrm{T}}|$,

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

If $\hat{x}$ is the true solution, then the computed solution x satisfies a forward error bound of the form

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq c(n)\,\mathrm{cond}(A, x)\epsilon$$

where $\mathrm{cond}(A, x) = \left\| |A^{-1}||A||x| \right\|_\infty / \|x\|_\infty \leq \mathrm{cond}(A) = \left\| |A^{-1}||A| \right\|_\infty \leq \kappa_\infty(A)$.

Note that $\mathrm{cond}(A, x)$ can be much smaller than $\mathrm{cond}(A)$.

Forward and backward error bounds can be computed by calling nag_dpprfs (f07ghc), and an estimate for $\kappa_\infty(A)$ ($= \kappa_1(A)$) can be obtained by calling nag_dppcon (f07ggc).

8 Parallelism and Performance

nag_dpptrs (f07gec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dpptrs (f07gec) 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 $2n^2r$.

This function may be followed by a call to nag_dpprfs (f07ghc) to refine the solution and return an error estimate.

The complex analogue of this function is nag_zpptrs (f07gsc).

10 Example

This example solves the system of equations $AX = B$, where

$$A = \begin{pmatrix} 4.16 & -3.12 & 0.56 & -0.10 \\ -3.12 & 5.03 & -0.83 & 1.18 \\ 0.56 & -0.83 & 0.76 & 0.34 \\ -0.10 & 1.18 & 0.34 & 1.18 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 8.70 & 8.30 \\ -13.35 & 2.13 \\ 1.89 & 1.61 \\ -4.14 & 5.00 \end{pmatrix}.$$

Here A is symmetric positive definite, stored in packed form, and must first be factorized by nag_dpptrf (f07gdc).

10.1 Program Text

```
/* nag_dpptrs (f07gec) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#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 */
  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_dpptrs (f07gec) 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)) ||
      !(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, _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_dpptrf (f07gdc).
   * Cholesky factorization of real symmetric
   * positive-definite matrix, packed storage
   */
  nag_dpptrf(order, uplo, n, ap, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dpptrf (f07gdc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Compute solution */
  /* nag_dpptrs (f07gec).
   * Solution of real symmetric positive-definite system of
   * linear equations, multiple right-hand sides, matrix
   * already factorized by nag_dpptrf (f07gdc), packed storage
   */
  nag_dpptrs(order, uplo, n, nrhs, ap, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dpptrs (f07gec).\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(b);
  return exit_status;
}
```

10.2 Program Data

```
nag_dpptrs (f07gec) Example Program Data
  4  2                        :Values of n and nrhs
  Nag_Lower                   :Value of uplo
  4.16
 -3.12   5.03
  0.56  -0.83   0.76
 -0.10   1.18   0.34   1.18   :End of matrix A
  8.70   8.30
-13.35   2.13
  1.89   1.61
 -4.14   5.00                 :End of matrix B
```

10.3 Program Results

```
nag_dpptrs (f07gec) Example Program Results

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