NAG Library Function Document

nag_dpttrs (f07jec)

1 Purpose

nag_dpttrs (f07jec) computes the solution to a real system of linear equations $AX = B$, where A is an n by n symmetric positive definite tridiagonal matrix and X and B are n by r matrices, using the LDL^{T} factorization returned by nag_dpttrf (f07jdc).

2 Specification

```
#include <nag.h>
#include <nagf07.h>
void nag_dpttrs (Nag_OrderType order, Integer n, Integer nrhs,
     const double d[], const double e[], double b[], Integer pdb,
     NagError *fail)
```

3 Description

nag_dpttrs (f07jec) should be preceded by a call to nag_dpttrf (f07jdc), which computes a modified Cholesky factorization of the matrix A as

$$A = LDL^{\mathrm{T}},$$

where L is a unit lower bidiagonal matrix and D is a diagonal matrix, with positive diagonal elements. nag_dpttrs (f07jec) then utilizes the factorization to solve the required equations. Note that the factorization may also be regarded as having the form $U^{\mathrm{T}}DU$, where U is a unit upper bidiagonal matrix.

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

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: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

3: **nrhs** – Integer *Input*

On entry: r, the number of right-hand sides, i.e., the number of columns of the matrix B.

Constraint: **nrhs** ≥ 0.

4: **d**[dim] – const double *Input*

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

On entry: must contain the n diagonal elements of the diagonal matrix D from the LDL^{T} factorization of A.

5: **e**[dim] – const double *Input*

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

On entry: must contain the $(n-1)$ subdiagonal elements of the unit lower bidiagonal matrix L. (**e** can also be regarded as the superdiagonal of the unit upper bidiagonal matrix U from the $U^{\mathrm{T}}DU$ factorization of A.)

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 matrix of right-hand sides 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** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdb** $\geq \max(1, \mathbf{n})$.

On entry, **pdb** $= \langle value \rangle$ and **nrhs** $= \langle value \rangle$.
Constraint: **pdb** $\geq \max(1, \mathbf{nrhs})$.

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

The computed solution for a single right-hand side, $\hat{x}$, satisfies an equation of the form

$$(A + E)\hat{x} = b,$$

where

$$\|E\|_1 = O(\epsilon)\|A\|_1$$

and ϵ is the ***machine precision***. An approximate error bound for the computed solution is given by

$$\frac{\|\hat{x} - x\|_1}{\|x\|_1} \leq \kappa(A)\frac{\|E\|_1}{\|A\|_1},$$

where $\kappa(A) = \left\|A^{-1}\right\|_1 \|A\|_1$, the condition number of A with respect to the solution of the linear equations. See Section 4.4 of Anderson *et al.* (1999) for further details.

Following the use of this function nag_dptcon (f07jgc) can be used to estimate the condition number of A and nag_dptrfs (f07jhc) can be used to obtain approximate error bounds.

8 Parallelism and Performance

nag_dpttrs (f07jec) is not threaded by NAG in any implementation.

nag_dpttrs (f07jec) 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 required to solve the equations $AX = B$ is proportional to nr.

The complex analogue of this function is nag_zpttrs (f07jsc).

10 Example

This example solves the equations

$$AX = B,$$

where A is the symmetric positive definite tridiagonal matrix

$$A = \begin{pmatrix} 4.0 & -2.0 & 0 & 0 & 0 \\ -2.0 & 10.0 & -6.0 & 0 & 0 \\ 0 & -6.0 & 29.0 & 15.0 & 0 \\ 0 & 0 & 15.0 & 25.0 & 8.0 \\ 0 & 0 & 0 & 8.0 & 5.0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 6.0 & 10.0 \\ 9.0 & 4.0 \\ 2.0 & 9.0 \\ 14.0 & 65.0 \\ 7.0 & 23.0 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_dpttrs (f07jec) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

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

int main(void)
{
  /* Scalars */
  Integer       exit_status = 0, i, j, n, nrhs, pdb;

  /* Arrays */
  double        *b = 0, *d = 0, *e = 0;

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;

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

  INIT_FAIL(fail);

  printf("nag_dpttrs (f07jec) 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
  if (n < 0 || nrhs < 0)
    {
      printf("Invalid n or nrhs\n");
      exit_status = 1;
      goto END;
```

```
    }
  /* Allocate memory */
  if (!(b = NAG_ALLOC(n * nrhs, double)) ||
      !(d = NAG_ALLOC(n, double)) ||
      !(e = NAG_ALLOC(n-1, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
#else
  pdb = nrhs;
#endif

  /* Read the upper bidiagonal part of the tridiagonal matrix A from */
  /* data file */
#ifdef _WIN32
  for (i = 0; i < n - 1; ++i) scanf_s("%lf", &e[i]);
#else
  for (i = 0; i < n - 1; ++i) scanf("%lf", &e[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  for (i = 0; i < n; ++i) scanf_s("%lf", &d[i]);
#else
  for (i = 0; i < n; ++i) scanf("%lf", &d[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Read the right hand matrix B */
  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= nrhs; ++j) scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= nrhs; ++j) scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Factorize the tridiagonal matrix A using nag_dpttrf (f07jdc). */
  nag_dpttrf(n, d, e, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dpttrf (f07jdc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Solve the equations AX = B using nag_dpttrs (f07jec). */
  nag_dpttrs(order, n, nrhs, d, e, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dpttrs (f07jec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print the solution using nag_gen_real_mat_print (x04cac). */
```

```
  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(b);
  NAG_FREE(d);
  NAG_FREE(e);

  return exit_status;
}

#undef B
```

10.2 Program Data

```
nag_dpttrs (f07jec) Example Program Data
  5     2                       : n and nrhs
       -2.0  -6.0  15.0   8.0 : super-diagonal e
  4.0  10.0  29.0  25.0   5.0 : diagonal d
  6.0  10.0
  9.0   4.0
  2.0   9.0
 14.0  65.0
  7.0  23.0                    : matrix b
```

10.3 Program Results

```
nag_dpttrs (f07jec) Example Program Results

 Solution(s)
            1          2
 1      2.5000     2.0000
 2      2.0000    -1.0000
 3      1.0000    -3.0000
 4     -1.0000     6.0000
 5      3.0000    -5.0000
```