

# NAG Library Function Document

## nag\_dptrfs (f07jhc)

### 1 Purpose

nag\_dptrfs (f07jhc) computes error bounds and refines 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 modified Cholesky factorization returned by nag\_dpstrf (f07jdc) and an initial solution returned by nag\_dpstrs (f07jec). Iterative refinement is used to reduce the backward error as much as possible.

### 2 Specification

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

void nag_dptrfs (Nag_OrderType order, Integer n, Integer nrhs,
                const double d[], const double e[], const double df[],
                const double ef[], const double b[], Integer pdb, double x[],
                Integer pdx, double ferr[], double berr[], NagError *fail)
```

### 3 Description

nag\_dptrfs (f07jhc) should normally be preceded by calls to nag\_dpstrf (f07jdc) and nag\_dpstrs (f07jec). nag\_dpstrf (f07jdc) computes a modified Cholesky factorization of the matrix  $A$  as

$$A = LDL^T,$$

where  $L$  is a unit lower bidiagonal matrix and  $D$  is a diagonal matrix, with positive diagonal elements. nag\_dpstrs (f07jec) then utilizes the factorization to compute a solution,  $\hat{X}$ , to the required equations. Letting  $\hat{x}$  denote a column of  $\hat{X}$ , nag\_dptrfs (f07jhc) computes a *component-wise backward error*,  $\beta$ , the smallest relative perturbation in each element of  $A$  and  $b$  such that  $\hat{x}$  is the exact solution of a perturbed system

$$(A + E)\hat{x} = b + f, \quad \text{with } |e_{ij}| \leq \beta|a_{ij}|, \quad \text{and } |f_j| \leq \beta|b_j|.$$

The function also estimates a bound for the *component-wise forward error* in the computed solution defined by  $\max |x_i - \hat{x}_i| / \max |\hat{x}_i|$ , where  $x$  is the corresponding column of the exact solution,  $X$ .

Note that the modified Cholesky factorization of  $A$  can also be expressed as

$$A = U^T D U,$$

where  $U$  is unit upper bidiagonal.

### 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:*  $n \geq 0$ .
- 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**  $\geq 0$ .
- 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 matrix 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 matrix  $A$ .
- 6: **df**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **df** 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  $A$ .
- 7: **ef**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **ef** must be at least  $\max(1, \mathbf{n})$ .  
*On entry:* must contain the  $(n - 1)$  subdiagonal elements of the unit bidiagonal matrix  $L$  from the  $LDL^T$  factorization of  $A$ .
- 8: **b**[*dim*] – const double *Input*  
**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  
 $\mathbf{b}[(j - 1) \times \mathbf{pdb} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{b}[(i - 1) \times \mathbf{pdb} + j - 1]$  when **order** = Nag\_RowMajor.  
*On entry:* the  $n$  by  $r$  matrix of right-hand sides  $B$ .
- 9: **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**  $\geq \max(1, \mathbf{nrhs})$ .

- 10: **x**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **x** must be at least  
 $\max(1, \mathbf{pdx} \times \mathbf{nrhs})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{n} \times \mathbf{pdx})$  when **order** = Nag\_RowMajor.  
The (*i*, *j*)th element of the matrix *X* is stored in  
 $\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$  when **order** = Nag\_RowMajor.  
*On entry:* the *n* by *r* initial solution matrix *X*.  
*On exit:* the *n* by *r* refined solution matrix *X*.
- 11: **pdx** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **x**.  
*Constraints:*  
if **order** = Nag\_ColMajor, **pdx**  $\geq \max(1, \mathbf{n})$ ;  
if **order** = Nag\_RowMajor, **pdx**  $\geq \max(1, \mathbf{nrhs})$ .
- 12: **ferr**[**nrhs**] – double *Output*  
*On exit:* estimate of the forward error bound for each computed solution vector, such that  $\|\hat{x}_j - x_j\|_\infty / \|\hat{x}_j\|_\infty \leq \mathbf{ferr}[j - 1]$ , where  $\hat{x}_j$  is the *j*th column of the computed solution returned in the array **x** and  $x_j$  is the corresponding column of the exact solution *X*. The estimate is almost always a slight overestimate of the true error.
- 13: **berr**[**nrhs**] – double *Output*  
*On exit:* estimate of the component-wise relative backward error of each computed solution vector  $\hat{x}_j$  (i.e., the smallest relative change in any element of *A* or *B* that makes  $\hat{x}_j$  an exact solution).
- 14: **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.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry, **n** =  $\langle value \rangle$ .

Constraint: **n**  $\geq 0$ .

On entry, **nrhs** =  $\langle value \rangle$ .

Constraint: **nrhs**  $\geq 0$ .

On entry, **pdb** =  $\langle value \rangle$ .

Constraint: **pdb**  $> 0$ .

On entry, **pdx** =  $\langle value \rangle$ .

Constraint: **pdx**  $> 0$ .

**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**  $\geq \max(1, \mathbf{nrhs})$ .

On entry, **pdx** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: **pdx**  $\geq \max(1, \mathbf{n})$ .

On entry, **pdx** =  $\langle value \rangle$  and **nrhs** =  $\langle value \rangle$ .

Constraint: **pdx**  $\geq \max(1, \mathbf{nrhs})$ .

**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.

**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

$$\|E\|_{\infty} = O(\epsilon)\|A\|_{\infty}$$

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

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

where  $\kappa(A) = \|A^{-1}\|_{\infty}\|A\|_{\infty}$ , 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.

Function nag\_dptcon (f07jgc) can be used to compute the condition number of  $A$ .

**8 Parallelism and Performance**

nag\_dptrfs (f07jhc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_dptrfs (f07jhc) 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 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  $nr$ . At most five steps of iterative refinement are performed, but usually only one or two steps are required.

The complex analogue of this function is nag\_zptrfs (f07jvc).

## 10 Example

This example solves the equations

$$AX = B,$$

where  $A$  is the symmetric positive definite tridiagonal matrix

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

Estimates for the backward errors and forward errors are also output.

### 10.1 Program Text

```

/* nag_dptrfs (f07jhc) Example Program.
*
* Copyright 2004 Numerical Algorithms Group.
*
* Mark 23, 2011.
*/

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagf16.h>

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

    /* Arrays */
    double       *b = 0, *berr = 0, *d = 0, *df = 0, *e = 0, *ef = 0, *ferr = 0;
    double       *x = 0;

    /* Nag Types */
    NagError      fail;

#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_dptrfs (f07jhc) Example Program Results\n\n");

    /* Skip heading in data file */
    scanf("%*[\n]");
    scanf("%ld%ld%*[\n]", &n, &nrhs);
    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)) ||
        !(berr = NAG_ALLOC(nrhs, double)) ||

```

```

    !(d      = NAG_ALLOC(n, double)) ||
    !(df     = NAG_ALLOC(n, double)) ||
    !(e      = NAG_ALLOC(n-1, double)) ||
    !(ef     = NAG_ALLOC(n-1, double)) ||
    !(ferr   = NAG_ALLOC(nrhs, double)) ||
    !(x      = NAG_ALLOC(n*nrhs, double))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
  pdx = n;
#else
  pdb = nrhs;
  pdx = nrhs;
#endif

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

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

  /* Copy A into DF and EF */
  for (i = 0; i < n; ++i) df[i] = d[i];
  for (i = 0; i < n-1; ++i) ef[i] = e[i];

  /* Copy B into X using nag_dge_copy (f16qfc). */
  nag_dge_copy(order, Nag_NoTrans, n, nrhs, b, pdb, x, pdx, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dge_copy (f16qfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

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

  /* Solve the equations AX = B using nag_dpstrs (f07jec). */
  nag_dpstrs(order, n, nrhs, df, ef, x, pdx, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dpstrs (f07jec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Improve the solution and compute error estimates
   * using nag_dpstrfs (f07jhc).
   */
  nag_dpstrfs(order, n, nrhs, d, e, df, ef, b, pdb, x, pdx, ferr, berr, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dpstrfs (f07jhc).\n%s\n", fail.message);
    exit_status = 1;
  }

```

```

        goto END;
    }

/* Print the solution and the forward and backward error estimates
 * using nag_gen_real_mat_print (x04cac).
 */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x,
                       pdx, "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;
}
printf("\nBackward errors (machine-dependent)\n");
for (j = 0; j < nrhs; ++j) printf("%11.1e%s", berr[j], j%7 == 6?"\n":" ");

printf("\n\nEstimated forward error bounds (machine-dependent)\n");
for (j = 0; j < nrhs; ++j) printf("%11.1e%s", ferr[j], j%7 == 6?"\n":" ");

printf("\n");

END:
NAG_FREE(b);
NAG_FREE(berr);
NAG_FREE(d);
NAG_FREE(df);
NAG_FREE(e);
NAG_FREE(ef);
NAG_FREE(ferr);
NAG_FREE(x);

return exit_status;
}
#undef B

```

## 10.2 Program Data

```

nag_dptrfs (f07jhc) Example Program Data
  5      2      : n and nrhs
  4.0  10.0  29.0  25.0   5.0 : diagonal d
 -2.0  -6.0  15.0   8.0      : super-diagonal e
  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_dptrfs (f07jhc) 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

```

```

Backward errors (machine-dependent)
  0.0e+00      7.4e-17

```

```

Estimated forward error bounds (machine-dependent)
  2.4e-14      4.7e-14

```

---