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_dpttrf (f07jdc) and an initial solution returned by nag_dpttrs (f07jec). Iterative refinement is used to reduce the backward error as much as possible.

2 Specification

```c
#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_dpttrf (f07jdc) and nag_dpttrs (f07jec). nag_dpttrf (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_dpttrs (f07jec) then utilizes the factorization to compute a solution, $X$, to the required equations. Letting $\hat{x}$ denote a column of $X$, 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

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

with $|e_{ij}| \leq \beta|a_{ij}|$, 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

$$A = U^T DU,$$

where $U$ is unit upper bidiagonal.

4 References


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  \( \text{Input} \)  
*On entry:* \( n \), the order of the matrix \( A \).

*Constraint:* \( n \geq 0 \).

3:  \( \text{nrhs} \) – Integer  \( \text{Input} \)  
*On entry:* \( r \), the number of right-hand sides, i.e., the number of columns of the matrix \( B \).

*Constraint:* \( \text{nrhs} \geq 0 \).

4:  \( \text{d}[\text{dim}] \) – const double  \( \text{Input} \)  
*Note:* the dimension, \( \text{dim} \), of the array \( \text{d} \) must be at least \( \max(1, n) \).

*On entry:* must contain the \( n \) diagonal elements of the matrix of \( A \).

5:  \( \text{e}[\text{dim}] \) – const double  \( \text{Input} \)  
*Note:* the dimension, \( \text{dim} \), of the array \( \text{e} \) must be at least \( \max(1, n - 1) \).

*On entry:* must contain the \( (n - 1) \) subdiagonal elements of the matrix \( A \).

6:  \( \text{df}[\text{dim}] \) – const double  \( \text{Input} \)  
*Note:* the dimension, \( \text{dim} \), of the array \( \text{df} \) must be at least \( \max(1, n) \).

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

7:  \( \text{ef}[\text{dim}] \) – const double  \( \text{Input} \)  
*Note:* the dimension, \( \text{dim} \), of the array \( \text{ef} \) must be at least \( \max(1, n) \).

*On entry:* must contain the \( (n - 1) \) subdiagonal elements of the unit bidiagonal matrix \( L \) from the \( \text{LDL}^T \) factorization of \( A \).

8:  \( \text{b}[\text{dim}] \) – const double  \( \text{Input} \)  
*Note:* the dimension, \( \text{dim} \), of the array \( \text{b} \) must be at least

\[
\max(1, \text{pdb} \times \text{nrhs}) \quad \text{when} \quad \text{order} = \text{Nag_ColMajor}; \\
\max(1, n \times \text{pdb}) \quad \text{when} \quad \text{order} = \text{Nag_RowMajor}.
\]

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

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

*On entry:* the \( n \) by \( r \) matrix of right-hand sides \( B \).

9:  \( \text{pdb} \) – Integer  \( \text{Input} \)  
*On entry:* the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( \text{b} \).

*Constraints:*

if \( \text{order} = \text{Nag_ColMajor}, \text{pdb} \geq \max(1, n); \)
if \( \text{order} = \text{Nag_RowMajor}, \text{pdb} \geq \max(1, \text{nrhs}); \)
10: \[x[\text{dim}] \] – double

\textbf{Note:} the dimension, \textit{dim}, of the array \( x \) must be at least
\[ \max(1, \text{pdx} \times \text{nrhs}) \text{ when } \text{order} = \text{Nag\_ColMajor}; \]
\[ \max(1, \text{n} \times \text{pdx}) \text{ when } \text{order} = \text{Nag\_RowMajor}. \]

The \((i,j)\)th element of the matrix \( X \) is stored in
\[ \begin{align*}
\text{x}[(j - 1) \times \text{pdx} + i - 1] & \text{ when } \text{order} = \text{Nag\_ColMajor}; \\
\text{x}[(i - 1) \times \text{pdx} + j - 1] & \text{ when } \text{order} = \text{Nag\_RowMajor}. 
\end{align*} \]

\textit{On entry:} the \( n \) by \( r \) initial solution matrix \( X \).

\textit{On exit:} the \( n \) by \( r \) refined solution matrix \( X \).

11: \( \text{pdx} \) – Integer

\textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \textit{order}) in the array \( x \).

\textbf{Constraints:}
\[ \begin{align*}
\text{if } \text{order} = \text{Nag\_ColMajor}, & \quad \text{pdx} \geq \max(1, \text{n}); \\
\text{if } \text{order} = \text{Nag\_RowMajor}, & \quad \text{pdx} \geq \max(1, \text{nrhs}). 
\end{align*} \]

12: \( \text{ferr[} \text{nrhs}] \) – double

\textit{Output}

\textit{On exit:} estimate of the forward error bound for each computed solution vector, such that
\[ \|\hat{x}_j - x_j\|_\infty / \|x_j\|_\infty \leq \text{ferr}[j - 1], \]
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: \( \text{berr[} \text{nrhs}] \) – double

\textit{Output}

\textit{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: \( \text{fail} \) – NagError*

\textit{Input/Output}

The NAGError * 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 \( \langle \text{value} \rangle \) had an illegal value.

**NE\_INT**

On entry, \( \text{n} = \langle \text{value} \rangle \).

Constraint: \( \text{n} \geq 0 \).

On entry, \( \text{nrhs} = \langle \text{value} \rangle \).

Constraint: \( \text{nrhs} \geq 0 \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \).

Constraint: \( \text{pdb} > 0 \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \).

Constraint: \( \text{pdx} > 0 \).
NE_INT_2

On entry, \( pdb = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( pdb \geq \max(1, n) \).

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

On entry, \( pdx = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( pdx \geq \max(1, n) \).

On entry, \( pdx = \langle \text{value} \rangle \) and \( nrhs = \langle \text{value} \rangle \).
Constraint: \( pdx \geq \max(1, nrhs) \).

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

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 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 \( nr \). 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

\[
A = \begin{pmatrix}
4.0 & -2.0 & 0 & 0 & 0 \\
-2.0 & 10.0 & -6.0 & 0 & 0 \\
0 & -6.0 & 29.0 & 15.0 & 0 \\
0 & 0 & 15.0 & 25.0 & 8.0 \\
0 & 0 & 0 & 8.0 & 5.0
\end{pmatrix}
\]

and \( B = \begin{pmatrix} 6.0 \\ 10.0 \\ 9.0 \\ 4.0 \\ 2.0 \\ 9.0 \\
14.0 \\ 65.0 \\ 7.0 \\ 23.0 \end{pmatrix} \).

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

## 10.1 Program Text

```c
/* nag_dptrfs (f07jhc) 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>
#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 ]
#else
#define B(I, J) b[(I-1)*pdb+J-1 ]
#endif

init_fail(fail);
printf("nag_dptrfs (f07jhc) Example Program Results\n\n");

#ifdef _WIN32
scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*\n", &n, &nrhs);
#else
scanf("%"NAG_IFMT"%"NAG_IFMT"%*\n", &n, &nrhs);
#endif

...
#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))
    || !(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 */
#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
#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
/* 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
/* 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\n", fail.message);
    exit_status = 1;
    goto END;
  }

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

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.5000</td>
<td>2.0000</td>
</tr>
<tr>
<td>2</td>
<td>2.0000</td>
<td>-1.0000</td>
</tr>
<tr>
<td>3</td>
<td>1.0000</td>
<td>-3.0000</td>
</tr>
<tr>
<td>4</td>
<td>-1.0000</td>
<td>6.0000</td>
</tr>
<tr>
<td>5</td>
<td>3.0000</td>
<td>-5.0000</td>
</tr>
</tbody>
</table>

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

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