NAG C Library Function Document

nag_dsprfs (f07phc)

1 Purpose

nag_dsprfs (f07phc) returns error bounds for the solution of a real symmetric indefinite system of linear equations with multiple right-hand sides, $AX = B$ using packed storage. It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

```c
void nag_dsprfs (Nag_OrderType order, Nag_UploType uplo, Integer n, Integer nrhs,
               const double ap[], const double afp[], const Integer ipiv[], const double b[],
               Integer pdb, double x[], Integer pdx, double ferr[], double berr[],
               NagError *fail)
```

3 Description

nag_dsprfs (f07phc) returns the backward errors and estimated bounds on the forward errors for the solution of a real symmetric indefinite system of linear equations with multiple right-hand sides $AX = B$, using packed storage. The function handles each right-hand side vector (stored as a column of the matrix $B$) independently, so we describe the function of nag_dsprfs (f07phc) in terms of a single right-hand side $b$ and solution $x$.

Given a computed solution $x$, the function computes the component-wise backward error $\beta$. This is the size of the smallest relative perturbation in each element of $A$ and $b$ such that $x$ is the exact solution of a perturbed system

$$(A + \delta A)x = b + \delta b$$

such that $|\delta a_{ij}| \leq \beta |a_{ij}|$ and $|\delta b_i| \leq \beta |b_i|$. Then the function estimates a bound for the component-wise forward error in the computed solution, defined by:

$$\max_i |x_i - \hat{x}_i| / \max_i |x_i|$$

where $\hat{x}$ is the true solution.

For details of the method, see the f07 Chapter Introduction.

4 References


5 Parameters

1: order – Nag_OrderType

   Input

   On entry: the order parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

   Constraint: order = Nag_RowMajor or Nag_ColMajor.
2: \textbf{uplo} – Nag_UploType \hspace{1cm} \textit{Input}

\textit{On entry}: indicates whether the upper or lower triangular part of \(A\) is stored and how \(A\) is to be factorized, as follows:

- if \(\text{uplo} = \text{Nag\_Upper}\), the upper triangular part of \(A\) is stored and \(A\) is factorized as 
  \[PUU^TPT\], where \(U\) is upper triangular;
- if \(\text{uplo} = \text{Nag\_Lower}\), the lower triangular part of \(A\) is stored and \(A\) is factorized as 
  \[PLDL^TPT\], where \(L\) is lower triangular.

\textit{Constraint}: \(\text{uplo} = \text{Nag\_Upper}\) or \(\text{Nag\_Lower}\).

3: \textbf{n} – Integer \hspace{1cm} \textit{Input}

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

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

4: \textbf{nrhs} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: \(r\), the number of right-hand sides.

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

5: \textbf{ap}[\text{dim}] – const double \hspace{1cm} \textit{Input}

\textit{Note}: the dimension, \(\text{dim}\), of the array \(\text{ap}\) must be at least \(\max(1, n \times (n + 1)/2)\).

\textit{On entry}: the \(n\) by \(n\) original symmetric matrix \(A\) as supplied to nag_dsptrf (f07pdc).

6: \textbf{afp}[\text{dim}] – const double \hspace{1cm} \textit{Input}

\textit{Note}: the dimension, \(\text{dim}\), of the array \(\text{afp}\) must be at least \(\max(1, n \times (n + 1)/2)\).

\textit{On entry}: details of the factorization of \(A\) stored in packed form, as returned by nag_dsptrf (f07pdc).

7: \textbf{ipiv}[\text{dim}] – const Integer \hspace{1cm} \textit{Input}

\textit{Note}: the dimension, \(\text{dim}\), of the array \(\text{ipiv}\) must be at least \(\max(1, n)\).

\textit{On entry}: details of the interchanges and the block structure of \(D\), as returned by nag_dsptrf (f07pdc).

8: \textbf{b}[\text{dim}] – const double \hspace{1cm} \textit{Input}

\textit{Note}: the dimension, \(\text{dim}\), of the array \(b\) must be at least \(\max(1, \text{pdb} \times \text{nrhs})\) when \(\text{order} = \text{Nag\_ColMajor}\) and at least \(\max(1, \text{pdb} \times n)\) when \(\text{order} = \text{Nag\_RowMajor}\).

If \(\text{order} = \text{Nag\_ColMajor}\), the \((i, j)\)th element of the matrix \(B\) is stored in \(b[(j - 1) \times \text{pdb} + i - 1]\) and if \(\text{order} = \text{Nag\_RowMajor}\), the \((i, j)\)th element of the matrix \(B\) is stored in \(b[(i - 1) \times \text{pdb} + j - 1]\).

\textit{On entry}: the \(n\) by \(r\) right-hand side matrix \(B\).

9: \textbf{pdb} – Integer \hspace{1cm} \textit{Input}

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

\textit{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: \textbf{x}[\text{dim}] – double \hspace{1cm} \textit{Input/Output}

\textit{Note}: the dimension, \(\text{dim}\), of the array \(x\) must be at least \(\max(1, \text{pdx} \times \text{nrhs})\) when \(\text{order} = \text{Nag\_ColMajor}\) and at least \(\max(1, \text{pdx} \times n)\) when \(\text{order} = \text{Nag\_RowMajor}\).
If order = Nag_ColMajor, the \((i,j)\)th element of the matrix \(X\) is stored in \(x[(j-1) \times \text{pdx} + i - 1]\) and if order = Nag_RowMajor, the \((i,j)\)th element of the matrix \(X\) is stored in \(x[(i-1) \times \text{pdx} + j - 1]\).

On entry: the \(n\) by \(r\) solution matrix \(X\), as returned by nag_dsptrs (f07pec).

On exit: the improved solution matrix \(X\).

11: \textbf{pdx} – Integer \hspace{1cm} \textit{Input}

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

Constraints:
- if order = Nag_ColMajor, \(\text{pdx} \geq \max(1, n)\);
- if order = Nag_RowMajor, \(\text{pdx} \geq \max(1, nrhs)\).

12: \textbf{ferr}[	extit{dim}] – double \hspace{1cm} \textit{Output}

Note: the dimension, \textit{dim}, of the array \textit{ferr} must be at least \(\max(1, \text{nrhs})\).

On exit: \(\text{ferr}[j-1]\) contains an estimated error bound for the \(j\)th solution vector, that is, the \(j\)th column of \(X\), for \(j = 1, 2, \ldots, r\).

13: \textbf{berr}[	extit{dim}] – double \hspace{1cm} \textit{Output}

Note: the dimension, \textit{dim}, of the array \textit{berr} must be at least \(\max(1, \text{nrhs})\).

On exit: \(\text{berr}[j-1]\) contains the component-wise backward error bound \(\beta\) for the \(j\)th solution vector, that is, the \(j\)th column of \(X\), for \(j = 1, 2, \ldots, r\).

14: \textbf{fail} – NagError * \hspace{1cm} \textit{Output}

The NAG error parameter (see the Essential Introduction).

6 \hspace{1cm} \textbf{Error Indicators and Warnings}

**NE_INT**

On entry, \textit{n} = \langle value\rangle.

Constraint: \(n \geq 0\).

On entry, \textit{nrhs} = \langle value\rangle.

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

On entry, \textit{pdb} = \langle value\rangle.

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

On entry, \textit{pdx} = \langle value\rangle.

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

**NE_INT_2**

On entry, \textit{pdb} = \langle value\rangle, \textit{n} = \langle value\rangle.

Constraint: \(\text{pdb} \geq \max(1, n)\).

On entry, \textit{pdb} = \langle value\rangle, \textit{nrhs} = \langle value\rangle.

Constraint: \(\text{pdb} \geq \max(1, \text{nrhs})\).

On entry, \textit{pdx} = \langle value\rangle, \textit{n} = \langle value\rangle.

Constraint: \(\text{pdx} \geq \max(1, n)\).

On entry, \textit{pdx} = \langle value\rangle, \textit{nrhs} = \langle value\rangle.

Constraint: \(\text{pdx} \geq \max(1, \text{nrhs})\).
NE_ALLOC_FAIL
Memory allocation failed.

NE_BAD_PARAM
On entry, parameter (value) had an illegal value.

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 consult NAG for assistance.

7 Accuracy
The bounds returned in ferr are not rigorous, because they are estimated, not computed exactly; but in practice they almost always overestimate the actual error.

8 Further Comments
For each right-hand side, computation of the backward error involves a minimum of $4n^2$ floating-point operations. Each step of iterative refinement involves an additional $6n^2$ operations. At most 5 steps of iterative refinement are performed, but usually only 1 or 2 steps are required.

Estimating the forward error involves solving a number of systems of linear equations of the form $Ax = b$; the number is usually 4 or 5 and never more than 11. Each solution involves approximately $2n^2$ operations.

The complex analogues of this function are nag_zhprfs (f07pvc) for Hermitian matrices and nag_zsprfs (f07qvc) for symmetric matrices.

9 Example
To solve the system of equations $AX = B$ using iterative refinement and to compute the forward and backward error bounds, where

$$A = \begin{pmatrix}
  2.07 & 3.87 & 4.20 & -1.15 \\
  3.87 & -0.21 & 1.87 & 0.63 \\
  4.20 & 1.87 & 1.15 & 2.06 \\
  -1.15 & 0.63 & 2.06 & -1.81
\end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix}
  -9.50 & 27.85 \\
  -8.38 & 9.90 \\
  -6.07 & 19.25 \\
  -0.96 & 3.93
\end{pmatrix}.$$ 

Here $A$ is symmetric indefinite, stored in packed form, and must first be factorized by nag_dsprf (f07pdc).

9.1 Program Text
/* nag_dsprfs (f07phc) Example Program. 
 * Copyright 2001 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 i, j, n, nrhs, ap_len, afp_len, pdb, pdx, ferr_len, berr_len;
  Integer exit_status=0;
  NagError fail;
Nag_UploType uplo_enum;
Nag_OrderType order;
/* Arrays */
 Integer *ipiv=0;
char uplo[2];
double *afp=0, *ap=0, *b=0, *berr=0, *ferr=0, *x=0;

#define A_LOWER(I,J) ap[(2*n-J)*(J-1)/2 + I - 1]
#define A_UPPER(I,J) ap[J*(J-1)/2 + I - 1]
#define B(I,J) b[(J-1)*pdb + I - 1]
#define X(I,J) x[(J-1)*pdx + 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]
#define X(I,J) x[(I-1)*pdx + J - 1]
order = Nag_RowMajor;
#endif

INIT_FAIL(fail);
Vprintf("f07phc Example Program Results\n\n");
/* Skip heading in data file */
Vscanf("\n");
Vscanf("%ld%ld\n", &n, &nrhs);
ap_len = n * (n + 1)/2;
afp_len = n * (n + 1)/2;
#ifdef NAG_COLUMN_MAJOR
 pdb = n;
pdx = n;
#else
 pdb = nrhs;
pdx = nrhs;
#endif
ferr_len = nrhs;
berr_len = nrhs;

/* Allocate memory */
if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
 !(afp = NAG_ALLOC(ap_len, double)) ||
 !(ap = NAG_ALLOC(afp_len, double)) ||
 !(b = NAG_ALLOC(n * nrhs, double)) ||
 !(berr = NAG_ALLOC(berr_len, double)) ||
 !(ferr = NAG_ALLOC(ferr_len, double)) ||
 !(x = NAG_ALLOC(n * nrhs, double)) )
{
 Vprintf("Allocation failure\n");
 exit_status = -1;
 goto END;
}

/* Read A and B from data file, and copy A to AFP and B to X */
Vscanf(" %ls %*[\n"] , uplo);
if (*(unsigned char *)uplo == 'L')
 uplo_enum = Nag_Lower;
else if (*(unsigned char *)uplo == 'U')
 uplo_enum = Nag_Upper;
else
{
 Vprintf("Unrecognised character for Nag_UploType type\n");
 exit_status = -1;
 goto END;
}
if (uplo_enum == Nag_Upper)
{
 for (i = 1; i <= n; ++i)
  {
   for (j = i; j <= n; ++j)
   {
Vscanf("%lf", &A_UPPER(i,j));
}
Vscanf("%*[\n ]");
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
            Vscanf("%lf", &A_LOWER(i,j));
    }
Vscanf("%*[\n ]");
}
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        Vscanf("%lf", &B(i,j));
}
Vscanf("%*[\n ]");
for (i = 1; i <= n * (n + 1) / 2; ++i)
    afp[i - 1] = ap[i - 1];
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        X(i,j) = B(i,j);
}
/* Factorize A in the array AFP */
f07pdc(order, uplo_enum, n, afp, ipiv, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07pdc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute solution in the array X */
f07pec(order, uplo_enum, n, nrhs, afp, ipiv, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07pec.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07phc(order, uplo_enum, n, nrhs, ap, afp, ipiv, b, pdb,
        x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07phc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
/* Print solution */
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, pdx,
    "Solution(s)", 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04cac.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
Vprintf("\nBackward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e %", berr[j-1], j%7==0 ?"\n":" ");
Vprintf("Estimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e %", ferr[j-1], j%7==0 ?"\n":" ");
Vprintf("\n");
END:
if (ipiv) NAG_FREE(ipiv);
if (afp) NAG_FREE(afp);
if (ap) NAG_FREE(ap);
if (b) NAG_FREE(b);
if (berr) NAG_FREE(berr);
if (ferr) NAG_FREE(ferr);
if (x) NAG_FREE(x);
return exit_status;
}

9.2 Program Data

f07phc Example Program Data
4 2 :Values of N and NRHS
'L' :Value of UPLO
2.07
3.87 -0.21
4.20 1.87 1.15
-1.15 0.63 2.06 -1.81 :End of matrix A
-9.50 27.85
-8.38 9.90
-6.07 19.25
-0.96 3.93 :End of matrix B

9.3 Program Results

f07phc Example Program Results

Solution(s)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-4.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>-1.0000</td>
<td>4.0000</td>
</tr>
<tr>
<td>3</td>
<td>2.0000</td>
<td>3.0000</td>
</tr>
<tr>
<td>4</td>
<td>5.0000</td>
<td>2.0000</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)
4.1e-17  5.5e-17

Estimated forward error bounds (machine-dependent)
2.3e-14  3.3e-14