NAG C Library Function Document

nag_dsyrfs (f07mhc)

1 Purpose

nag_dsyrfs (f07mhc) returns error bounds for the solution of a real symmetric indefinite system of linear equations with multiple right-hand sides, \( AX = B \). It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

```c
void nag_dsyrfs (Nag_OrderType order, Nag_UploType uplo, Integer n, Integer nrhs,
               const double a[], Integer pda, const double af[], Integer pdaf,
               const Integer ipiv[], const double b[], Integer pdb, double x[], Integer pdx,
               double ferr[], double berr[], NagError *fail)
```

3 Description

nag_dsyrfs (f07mhc) 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 \( AX = B \).

The function handles each right-hand side vector (stored as a column of the matrix \( B \)) independently, so we describe the function of nag_dsyrfs (f07mhc) in terms of a single right-hand side \( b \) and solution \( x \).

Given a computed solution \( x \), the function computes the \textit{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

\[
(A + \delta A)x = b + \delta b \\
|\delta a_{ij}| \leq \beta |a_{ij}| \quad \text{and} \quad |\delta b_i| \leq \beta |b_i|.
\]

Then the function estimates a bound for the \textit{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: \textbf{order} – Nag_OrderType \hspace{1cm} \textit{Input}

\textit{On entry:} the \textbf{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 \textbf{order} = Nag_RowMajor. See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

\textit{Constraint:} \textbf{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:

\[
\begin{align*}
\text{Nag_Upper} & : & A_{ij} & = a_{ij}, & & i \leq j, \\
\text{Nag_Lower} & : & A_{ij} & = a_{ij}, & & i \geq j.
\end{align*}
\]
if \texttt{uplo} = \texttt{Nag\_Upper}, the upper triangular part of \(A\) is stored and \(A\) is factorized as \(PUDU^TP^T\), where \(U\) is upper triangular;

if \texttt{uplo} = \texttt{Nag\_Lower}, the lower triangular part of \(A\) is stored and \(A\) is factorized as \(PLDL^TP^T\), where \(L\) is lower triangular.

\textit{Constraint:} \texttt{uplo} = \texttt{Nag\_Upper} or \texttt{Nag\_Lower}.

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

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

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

4: \(\text{nrhs}\) – Integer \hspace{1cm} \textit{Input}

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

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

5: \(a[\text{dim}]\) – const double \hspace{1cm} \textit{Input}

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

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

\textit{On entry:} the \(n\) by \(n\) original symmetric matrix \(A\) as supplied to \texttt{nag\_dsytrf} (f07mdc).

6: \(\text{pda}\) – Integer \hspace{1cm} \textit{Input}

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

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

7: \(af[\text{dim}]\) – const double \hspace{1cm} \textit{Input}

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

\textit{On entry:} details of the factorization of \(A\), as returned by \texttt{nag\_dsytrf} (f07mdc).

8: \(\text{pdaf}\) – Integer \hspace{1cm} \textit{Input}

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

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

9: \(\text{ipiv}[\text{dim}]\) – 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 \texttt{nag\_dsytrf} (f07mdc).

10: \(b[\text{dim}]\) – 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 \texttt{order} = \texttt{Nag\_ColMajor} and at least \(\max(1, \text{pdb} \times n)\) when \texttt{order} = \texttt{Nag\_RowMajor}.

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

\textit{Input}

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

\textit{Constraints:}

\begin{align*}
\text{if } \texttt{order} = \texttt{Nag\_ColMajor}, & \quad \texttt{pdb} \geq \max(1, \texttt{n}); \\
\text{if } \texttt{order} = \texttt{Nag\_RowMajor}, & \quad \texttt{pdb} \geq \max(1, \texttt{nrhs}).
\end{align*}

12: \texttt{x[\textit{dim}]} – double

\textit{Input/Output}

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

If \texttt{order} = \texttt{Nag\_ColMajor}, the \((i, j)\)th element of the matrix \(X\) is stored in \(\texttt{x}[(j - 1) \times \texttt{pdx} + i - 1]\) and if \texttt{order} = \texttt{Nag\_RowMajor}, the \((i, j)\)th element of the matrix \(X\) is stored in \(\texttt{x}[(i - 1) \times \texttt{pdx} + j - 1]\).

\textit{On entry:} the \(n\) by \(r\) solution matrix \(X\), as returned by nag_dsytrs (f07mec).

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

13: \texttt{pdx} – Integer

\textit{Input}

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

\textit{Constraints:}

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

14: \texttt{ferr[\textit{dim}]} – double

\textit{Output}

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

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

15: \texttt{berr[\textit{dim}]} – double

\textit{Output}

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

\textit{On exit:} \texttt{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 \(j = 1, 2, \ldots, r\).

16: \texttt{fail} – NagError *

\textit{Output}

The NAGError * parameter (see the Essential Introduction).

6 \textbf{Error Indicators and Warnings}

\textbf{NE\_INT}

\textit{On entry,} \texttt{n} = ⟨\textit{value}⟩.
\textit{Constraint:} \texttt{n} \geq 0.

\textit{On entry,} \texttt{nrhs} = ⟨\textit{value}⟩.
\textit{Constraint:} \texttt{nrhs} \geq 0.

\textit{On entry,} \texttt{pda} = ⟨\textit{value}⟩.
\textit{Constraint:} \texttt{pda} > 0.

\textit{On entry,} \texttt{pdaf} = ⟨\textit{value}⟩.
\textit{Constraint:} \texttt{pdaf} > 0.

\textit{On entry,} \texttt{pdb} = ⟨\textit{value}⟩.
\textit{Constraint:} \texttt{pdb} > 0.
On entry, \( \text{pdx} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} > 0 \).

NE_INT_2

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

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

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

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

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

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

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter \( \langle \text{value} \rangle \) 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 \( \text{ferr} \) 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 \text{nag__zherfs (f07mvc)} for Hermitian matrices and \text{nag__zsyri}fs
(f07nvc) 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
Here $A$ is symmetric indefinite and must first be factorized by nag_dsytrf (f07mdc).

### 9.1 Program Text

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

int main(void) {
    /* Scalars */
    Integer berr_len, ferr_len, i, j, n, nrhs;
    Integer pda, pdaf, pdb, pdx;
    Integer exit_status=0;
    Nag_UploType uplo_enum;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    char uplo[2];
    Integer *ipiv=0;
    double *a=0, *af=0, *b=0, *berr=0, *ferr=0, *x=0;

    #ifdef NAG_COLUMN_MAJOR
    #define A(I,J) a[(J-1)*pda+I-1]
    #define AF(I,J) af[(J-1)*pdaf +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(I,J) a[(I-1)*pda+J-1]
    #define AF(I,J) af[(I-1)*pdaf +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("f07mhc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[\n"]);
    Vscanf("%ld%ld%*[\n"] , &n, &nrhs);

    #ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdaf = n;
    pdb = n;
    pdx = n;
    #else
    pda = n;
    pdaf = n;
    pdb = nrhs;
    pdx = nrhs;
    #endif
```
ferr_len = nrhs;
berr_len = nrhs;

/* Allocate memory */
if ( !(ipiv = NAG_ALLOC(n, Integer)) ||
    !(a = NAG_ALLOC(n * n, double)) ||
    !(af = NAG_ALLOC(n * n, 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 AF and B to X */
Vscanf(" ' %1s '%*['\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(i,j));
        Vscanf("%*['\n"] ");
    }
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
            Vscanf("%lf", &A(i,j));
        Vscanf("%*['\n"] ");
    }
}

for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        Vscanf("%lf", &B(i,j));
}
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        X(i,j) = B(i,j);
}

} /* Factorize A in the array AF */
f07mdc(order, uplo_enum, n, af, pdaf, ipiv, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07mdc.\n\n", fail.message);
    exit_status = 1;
    goto END;
}

} /* Compute solution in the array X */
f07mec(order, uplo_enum, n, nrhs, af, pdaf, ipiv, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07mec.\n\n", fail.message);
    exit_status = 1;
    goto END;
}

} /* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07mhc(order, uplo_enum, n, nrhs, a, pda, af, pdaf, ipiv, b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07mhc.\n\n", 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.\n\n", fail.message);
    exit_status = 1;
    goto END;
}

Vprintf("\n\nBackward errors (machine-dependent)\n\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e%s", berr[j-1], j%7==0 ?"\n":" ");
Vprintf("\nEstimated forward error bounds" "(machine-dependent)\n"");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e", ferr[j-1], j%7==0 || j==nrhs ?"\n":" ");

END:
if (ipiv) NAG_FREE(ipiv);
if (a) NAG_FREE(a);
if (af) NAG_FREE(af);
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

f07mhc 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

[NP3645/7]
9.3 Program Results

f07mhc Example Program Results

Solution(s)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<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