NAG C Library Function Document

nag_zherfs (f07mvc)

1 Purpose

nag_zherfs (f07mvc) returns error bounds for the solution of a complex Hermitian 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_zherfs (Nag_OrderType order, Nag_UploType uplo, Integer n, Integer nrhs,
   const Complex a[], Integer pda, const Complex af[], Integer pdaf,
   const Integer ipiv[], const Complex b[], Integer pdb, Complex x[],
   Integer pdx, double ferr[], double berr[], NagError *fail)
```

3 Description

nag_zherfs (f07mvc) returns the backward errors and estimated bounds on the forward errors for the solution of a complex Hermitian 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_zherfs (f07mvc) 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

\textit{Input}

\textit{On entry:} the \textit{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 \textit{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:} \textit{order = Nag_RowMajor} or \textit{Nag_ColMajor}. 

2:  \textbf{uplo} – Nag_UploType

\textit{Input}

On entry: indicates whether the upper or lower triangular part of \(A\) is stored and how \(A\) has been factorized, as follows:

if \(\text{uplo} = \text{Nag\_Upper}\), then the upper triangular part of \(A\) is stored and \(A\) is factorized as \(PUDU^H P^T\), where \(U\) is upper triangular;

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

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

3:  \(n\) – Integer

\textit{Input}

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

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

4:  \(\text{nrhs}\) – Integer

\textit{Input}

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

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

5:  \(a[\text{dim}]\) – const Complex

\textit{Input}

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

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

On entry: the \(n\) by \(n\) original Hermitian matrix \(A\) as supplied to \text{nag\_zhetrf} (f07mrc).

6:  \(\text{pda}\) – Integer

\textit{Input}

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

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

7:  \(af[\text{dim}]\) – const Complex

\textit{Input}

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

On entry: details of the factorization of \(A\), as returned by \text{nag\_zhetrf} (f07mrc).

8:  \(\text{pdaf}\) – Integer

\textit{Input}

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

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

9:  \(\text{ipiv}[\text{dim}]\) – const Integer

\textit{Input}

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

On entry: details of the interchanges and the block structure of \(D\), as returned by \text{nag\_zhetrf} (f07mrc).

10:  \(b[\text{dim}]\) – const Complex

\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 \(b[(i-1) \times \text{pdb} + j - 1]\).

On entry: the \(n\) by \(r\) right-hand side matrix \(B\).
11:  **pdb** – Integer

*Input*

*On entry:* the stride separating matrix row or column elements (depending on the value of **order** in the array **b**.

*Constraints:*

- if **order** = Nag_ColMajor, pdb ≥ max(1, n);
- if **order** = Nag_RowMajor, pdb ≥ max(1, nrhs).

12:  **x**[**dim**] – Complex

*Input/Output*

*Note:* the dimension, **dim**, of the array **x** must be at least max(1, pdx × nrhs) when **order** = Nag_ColMajor and at least max(1, pdx × n) when **order** = Nag_RowMajor.

*On entry:* the n by r solution matrix X, as returned by nag_zhetrs (f07msc).

*On exit:* the improved solution matrix X.

13:  **pdx** – Integer

*Input*

*On entry:* the stride separating matrix row or column elements (depending on the value of **order** in the array **x**.

*Constraints:*

- if **order** = Nag_ColMajor, pdx ≥ max(1, n);
- if **order** = Nag_RowMajor, pdx ≥ max(1, nrhs).

14:  **ferr**[**dim**] – double

*Output*

*Note:* the dimension, **dim**, of the array **ferr** must be at least max(1, nrhs).

*On exit:* ferr[j – 1] contains an estimated error bound for the jth solution vector, that is, the jth column of X, for j = 1, 2, ..., r.

15:  **berr**[**dim**] – double

*Output*

*Note:* the dimension, **dim**, of the array **berr** must be at least max(1, nrhs).

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

16:  **fail** – NagError *

*Output*

The NAGError * parameter (see the Essential Introduction).

### 6 Error Indicators and Warnings

**NE_INT**

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

Constraint: n ≥ 0.

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

Constraint: nrhs ≥ 0.

*On entry,* pda = \langle value \rangle.

Constraint: pda > 0.

*On entry,* pdaf = \langle value \rangle.

Constraint: pdaf > 0.

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

Constraint: pdb > 0.
On entry, \( \text{pdx} = (\text{value}) \).
Constraint: \( \text{pdx} > 0 \).

**NE_INT_2**
On entry, \( \text{pda} = (\text{value}) \), \( \text{n} = (\text{value}) \).
Constraint: \( \text{pda} \geq \max(1, \text{n}) \).
On entry, \( \text{pdaf} = (\text{value}) \), \( \text{n} = (\text{value}) \).
Constraint: \( \text{pdaf} \geq \max(1, \text{n}) \).
On entry, \( \text{pdb} = (\text{value}) \), \( \text{n} = (\text{value}) \).
Constraint: \( \text{pdb} \geq \max(1, \text{n}) \).
On entry, \( \text{pdb} = (\text{value}) \), \( \text{nrhs} = (\text{value}) \).
Constraint: \( \text{pdb} \geq \max(1, \text{nrhs}) \).
On entry, \( \text{pdx} = (\text{value}) \), \( \text{n} = (\text{value}) \).
Constraint: \( \text{pdx} \geq \max(1, \text{n}) \).
On entry, \( \text{pdx} = (\text{value}) \), \( \text{nrhs} = (\text{value}) \).
Constraint: \( \text{pdx} \geq \max(1, \text{nrhs}) \).

**NE_SINGULAR**
The block diagonal matrix \( D \) is exactly singular.

**NE_ALLOC_FAIL**
Memory allocation failed.

**NE_BAD_PARAM**
On entry, parameter \( (\text{value}) \) 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 \( \text{ferr} \) 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 \( 16n^2 \) real floating-point operations. Each step of iterative refinement involves an additional \( 24n^2 \) real 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 5 and never more than 11. Each solution involves approximately \( 8n^2 \) real operations.

The real analogue of this function is \text{nag_dsytrfs} (\text{f07mhc}).

### 9 Example
To solve the system of equations \( AX = B \) using iterative refinement and to compute the forward and backward error bounds, where
\[
A = \begin{pmatrix}
-1.36 + 0.00i & 1.58 + 0.90i & 2.21 - 0.21i & 3.91 + 1.50i \\
1.58 - 0.90i & -8.87 + 0.00i & -1.84 - 0.03i & -1.78 + 1.18i \\
2.21 + 0.21i & -1.84 + 0.03i & -4.63 + 0.00i & 0.11 + 0.11i \\
3.91 - 1.50i & -1.78 - 1.18i & 0.11 - 0.11i & -1.84 + 0.00i \\
\end{pmatrix}
\]

and
\[
B = \begin{pmatrix}
7.79 + 5.48i & -35.39 + 18.01i \\
-0.77 - 16.05i & 4.23 - 70.02i \\
-9.58 + 3.88i & -24.79 - 8.40i \\
2.98 - 10.18i & 28.68 - 39.89i \\
\end{pmatrix}
\]

Here \( A \) is Hermitian indefinite and must first be factorized by \texttt{nag_zhetrf (f07mrc)}.

### 9.1 Program Text

/* \texttt{nag_zherfs (f07mvc)} 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, pda, pdaf, pdb, pdx;
    Integer ferr_len, berr_len;
    Integer exit_status=0;
    Nag_UploType uplo_enum;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    Integer *ipiv=0;
    char uplo[2];
    Complex *a=0, *af=0, *b=0, *x=0;
    double *berr=0, *ferr=0;

#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]
#define EXIT_TRY(fail)

    INIT_FAIL(fail);
    Vprintf("f07mvc Example Program Results\n\n");
    /* Skip heading in data file */
    Vscanf("%*[^\n] ");
    Vscanf("%d%ld%[^\n] ", &n, &nrhs);
    #ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdaf = n;
    #else
    order = Nag_RowMajor;
    #endif
    INIT_TRY(fail);
    Vprintf("%s%ld%[^\n] ", &n, &nrhs);
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, Complex)) ||
    !(af = NAG_ALLOC(n * n, Complex)) ||
    !(b = NAG_ALLOC(n * nrhs, Complex)) ||
    !(x = NAG_ALLOC(n * nrhs, Complex)) ||
    !(berr = NAG_ALLOC(berr_len, double)) ||
    !(ferr = NAG_ALLOC(ferr_len, 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(" ' %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 , %lf )", &A(i,j).re, &A(i,j).im);
  }
  Vscanf("%*[\n]\n");
}
else
{
  for (i = 1; i <= n; ++i)
  {
    for (j = 1; j <= i; ++j)
      Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
  }
  Vscanf("%*[\n]\n");
}
for (i = 1; i <= n; ++i)
{
  for (j = 1; j <= nrhs; ++j)
    Vscanf(" ( %lf , %lf )", &B(i,j).re, &B(i,j).im);
}
Vscanf("%*[\n]\n");

/* Copy A to AF and B to X */
if (uplo_enum == Nag_Upper)
{
  for (i = 1; i <= n; ++i)
  {
    for (j = i; j <= n; ++j)
    {
      AF(i,j).re = A(i,j).re;
      AF(i,j).im = A(i,j).im;
    }
  }
  Vscanf("%*[\n]\n");
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
        {
            AF(i,j).re = A(i,j).re;
            AF(i,j).im = A(i,j).im;
        }
    }
}

for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
    {
        X(i,j).re = B(i,j).re;
        X(i,j).im = B(i,j).im;
    }
}

/* Factorize A in the array AF */
f07mrc(order, uplo_enum, n, af, pdaf, ipiv, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07mrc.\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Compute solution in the array X */
f07msc(order, uplo_enum, n, nrhs, af, pdaf, ipiv, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07msc.\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07mvc(order, uplo_enum, n, nrhs, a, pdaf, af, pdaf, ipiv, b, pda, af, pdaf, ipiv,
       &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07mvc.\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print solution */
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, pdx,
   Nag_BracketForm, "%7.4f", "Solution(s)", Nag_IntegerLabels,
   0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04dbc.\n", fail.message);
    exit_status = 1;
    goto END;
}
Vprintf("\nBackward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e\n", berr[j-1], j%4 == 0 ?"\n": "");
Vprintf("\nEstimated forward error bounds \
(machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e\n", ferr[j-1], j%4 == 0 ?"\n": "");
Vprintf("\n");
END:
if (ipiv) NAG_FREE(ipiv);
if (a) NAG_FREE(a);
if (af) NAG_FREE(af);
if (b) NAG_FREE(b);
if (x) NAG_FREE(x);
if (berr) NAG_FREE(berr);
if (ferr) NAG_FREE(ferr);
return exit_status;
}

9.2 Program Data

f07mvc Example Program Data

4 2 :Values of N and NRHS
'L' :Value of UPLO
(-1.36, 0.00)
( 1.58, -0.90) (-8.87, 0.00)
( 2.21, 0.21) (-1.84, 0.03) (-4.63, 0.00)
( 3.91, -1.50) (-1.78, -1.18) ( 0.11, -0.11) (-1.84, 0.00)
( 7.79, 5.48) (-35.39, 18.01)
(-0.77, -16.05) ( 4.23, -70.02)
(-9.58, 3.88) (-24.79, -8.40)
( 2.98, -10.18) ( 28.68, -39.89)

9.3 Program Results

f07mvc Example Program Results

Solution(s)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1.0000, -1.0000)</td>
<td>( 3.0000, -4.0000)</td>
</tr>
<tr>
<td>2</td>
<td>(-1.0000, 2.0000)</td>
<td>(-1.0000, 5.0000)</td>
</tr>
<tr>
<td>3</td>
<td>( 3.0000, -2.0000)</td>
<td>( 7.0000, -2.0000)</td>
</tr>
<tr>
<td>4</td>
<td>( 2.0000, 1.0000)</td>
<td>(-8.0000, 6.0000)</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)
9.0e-17 5.8e-17

Estimated forward error bounds (machine-dependent)
2.6e-15 3.0e-15