NAG 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

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

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 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 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 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: \texttt{uplo} – Nag_UploType \hspace{1cm} \textit{Input}

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

\texttt{uplo} = Nag_Upper
The upper triangular part of \( A \) is stored and \( A \) is factorized as \( PUDU^HPT \), where \( U \) is upper triangular.

\texttt{uplo} = Nag_Lower
The lower triangular part of \( A \) is stored and \( A \) is factorized as \( PLDL^HPT \), where \( L \) is lower triangular.

\textit{Constraint:} \texttt{uplo} = Nag_Upper or Nag_Lower.

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

\textit{On entry:} \( n \), the order of the matrix \( A \).
\textit{Constraint:} \( n \geq 0 \).

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

\textit{On entry:} \( r \), the number of right-hand sides.
\textit{Constraint:} \( nrhs \geq 0 \).

5: \texttt{a[dim]} – const Complex \hspace{1cm} \textit{Input}

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

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

6: \texttt{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 in the array \( a \).
\textit{Constraint:} \( \texttt{pda} \geq \max(1, n) \).

7: \texttt{af[dim]} – const Complex \hspace{1cm} \textit{Input}

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

\textit{On entry:} details of the factorization of \( A \), as returned by nag_zhetrf (f07mrc).

8: \texttt{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:} \( \texttt{pdaf} \geq \max(1, n) \).

9: \texttt{ipiv[dim]} – const Integer \hspace{1cm} \textit{Input}

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

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

10: \texttt{b[dim]} – const Complex \hspace{1cm} \textit{Input}

\textit{Note:} the dimension, \( dim \), of the array \( b \) must be at least \( \max(1, \texttt{pdb} \times nrhs) \) when \texttt{order} = Nag_ColMajor;
\( \max(1, n \times \texttt{pdb}) \) when \texttt{order} = Nag_RowMajor.
The \((i, j)\)th element of the matrix \(B\) is stored in 
\[
  b[(j - 1) \times \text{pdb} + i - 1] \quad \text{when } \text{order} = \text{Nag\_ColMajor}; \\
  b[(i - 1) \times \text{pdb} + j - 1] \quad \text{when } \text{order} = \text{Nag\_RowMajor}.
\]

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

11: \(\text{pdb} \rightarrow \text{Integer} \quad \text{Input}\)

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

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}).

12: \(\text{x}[\text{dim}] \rightarrow \text{Complex} \quad \text{Input/Output}\)

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

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

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

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

13: \(\text{pdx} \rightarrow \text{Integer} \quad \text{Input}\)

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

Constraints:

if \text{order} = \text{Nag\_ColMajor}, \text{pdx} \geq \max(1, n);

if \text{order} = \text{Nag\_RowMajor}, \text{pdx} \geq \max(1, \text{nrhs}).

14: \(\text{ferr}[\text{nrhs}] \rightarrow \text{double} \quad \text{Output}\)

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 \(j = 1, 2, \ldots, r\).

15: \(\text{berr}[\text{nrhs}] \rightarrow \text{double} \quad \text{Output}\)

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 \(j = 1, 2, \ldots, r\).

16: \(\text{fail} \rightarrow \text{NagError} * \quad \text{Input/Output}\)

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

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.
On entry, \( n = \langle\text{value}\rangle\).
Constraint: \( n \geq 0 \).

On entry, \( \text{nrhs} = \langle\text{value}\rangle\).
Constraint: \( \text{nrhs} \geq 0 \).

On entry, \( \text{pda} = \langle\text{value}\rangle\).
Constraint: \( \text{pda} > 0 \).

On entry, \( \text{pdaf} = \langle\text{value}\rangle\).
Constraint: \( \text{pdaf} > 0 \).

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

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

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

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

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

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

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

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

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.

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.

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

\texttt{nag\_zherfs} (f07mvc) 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

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 five steps of iterative refinement are performed, but usually only one or two 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 nag_dsytrs (f07mhc).

10 Example

This example solves 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 nag_zhetrf (f07mrc).

10.1 Program Text

/* nag_zherfs (f07mvc) Example Program. *
 * Copyright 2014 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;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    Integer *ipiv = 0;
    char nag_enum_arg[40];
    Complex *a = 0, *af = 0, *b = 0, *x = 0;
    double *berr = 0, *ferr = 0;
    #ifdef NAG_COLUMN_MAJOR

Mark 25

f07mvc.5
\#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);

printf("nag_zherfs (f07mvc) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[^\n] ");
#else
scanf("%*[^\n] ");
#endif
#ifdef _WIN32
scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &n, &nrhs);
#else
scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &n, &nrhs);
#endif
#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, 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))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* Read A and B from data file, and copy A to AF and B to X */
#ifdef _WIN32
scanf_s(" %39s%*[^\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf(" %39s%*[^\n] ", nag_enum_arg);
#endif
/* nag_enum_name_to_value (x04nac).
  * Converts NAG enum member name to value
  */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
if (uplo == Nag_Upper)


```c
{  
  for (i = 1; i <= n; ++i)  
  {  
    for (j = i; j <= n; ++j)
#ifdef _WIN32
      scanf_s("( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
      scanf("( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
  }  
#ifdef _WIN32
  scanf_s("%*[\n ] ");
#else
  scanf("%*[\n ] ");
#endif  
}
else  
{  
  for (i = 1; i <= n; ++i)  
  {  
    for (j = i; j <= i; ++j)
#ifdef _WIN32
      scanf_s("( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
      scanf("( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
  }
  #ifdef _WIN32
  scanf_s("%*[\n ] ");
#else
  scanf("%*[\n ] ");
#endif  
  for (i = 1; i <= n; ++i)  
  {  
    for (j = 1; j <= nrhs; ++j)  
#ifdef _WIN32
      scanf_s("( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#else
      scanf("( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif  
  }  
  #ifdef _WIN32
  scanf_s("%*[\n ] ");
#else
  scanf("%*[\n ] ");
#endif  
}  
/* Copy A to AF and B to X */  
if (uplo == 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;
    }  
  }  
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 */
/* nag_zhetrf (f07mrc).*/
/* Bunch-Kaufman factorization of complex Hermitian */
/* indefinite matrix */
*nag_zhetrf(order, uplo, n, af, pdaf, ipiv, &fail);
if (fail.code != NE_NOERROR)  
{  
  printf("Error from nag_zhetrf (f07mrc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Compute solution in the array X */
/* nag_zhetrs (f07msc).*/
/* Solution of complex Hermitian indefinite system of linear */
/* equations, multiple right-hand sides, matrix already */
/* factorized by nag_zhetrf (f07mrc) */
*nag_zhetrs(order, uplo, n, nrhs, af, pdaf, ipiv, x, pdx, 
&fail);
if (fail.code != NE_NOERROR)  
{  
  printf("Error from nag_zhetrs (f07msc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
/* nag_zherfs (f07mvc).*/
/* Refined solution with error bounds of complex Hermitian */
/* indefinite system of linear equations, multiple */
/* right-hand sides */
*nag_zherfs(order, uplo, n, nrhs, a, pda, af, pdaf, ipiv, 
  b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)  
{  
  printf("Error from nag_zherfs (f07mvc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Print solution */
/* nag_gen_complx_mat_print_comp (x04dbc).*/
/* Print complex general matrix (comprehensive) */
*fflush(stdout);
*nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, 
  nrhs, x, pdx, NagBracketForm, "%7.4f", 
  "Solution(s)", Nag_IntegerLabels, 
  0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)  
{  
  printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

printf("\nBackward errors (machine-dependent)\n");  
for (j = 1; j <= nrhs; ++j)  
  printf("%lle", berr[j-1], j%4 == 0?"\n": "");  
printf("Estimated forward error bounds 
"(machine-dependent)\n");  
for (j = 1; j <= nrhs; ++j)  
  printf("%lle", ferr[j-1], j%4 == 0?"\n": "");
printf("
");
END:
NAG_FREE(ipiv);
NAG_FREE(a);
NAG_FREE(AF);
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(berr);
NAG_FREE(ferr);
return exit_status;
}

10.2 Program Data

nag_zherfs (f07mvc) Example Program Data

4 2
Nag_Lower
(-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)

10.3 Program Results

nag_zherfs (f07mvc) Example Program Results

Solution(s)

1 2
1 ( 1.0000,-1.0000) ( 3.0000,-4.0000)
2 (-1.0000, 2.0000) (-1.0000, 5.0000)
3 ( 3.0000,-2.0000) ( 7.0000,-2.0000)
4 ( 2.0000, 1.0000) (-8.0000, 6.0000)

Backward errors (machine-dependent)
5.7e-17 5.8e-17
Estimated forward error bounds (machine-dependent)
2.5e-15 3.1e-15