NAG C Library Function Document

nag_zporfs (f07fvc)

1 Purpose

nag_zporfs (f07fvc) returns error bounds for the solution of a complex Hermitian positive-definite 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_zporfs (Nag_OrderType order, Nag_UploType uplo, Integer n, Integer nrhs,
                 const Complex a[], Integer pda, const Complex af[], Integer pdaf,
                 const Complex b[], Integer pdb, Complex x[], Integer pdx,
                 double ferr[],
                 double berr[], NagError *fail)
```

3 Description

nag_zporfs (f07fvc) returns the backward errors and estimated bounds on the forward errors for the solution of a complex Hermitian positive-definite 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_zporfs (f07fvc) 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 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: \texttt{uplo} – Nag_UploType \hfill Input

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

- if \texttt{uplo} = Nag_Upper, the upper triangular part of \(A\) is stored and \(A\) is factorized as \(U^H U\), where \(U\) is upper triangular;

- if \texttt{uplo} = Nag_Lower, the lower triangular part of \(A\) is stored and \(A\) is factorized as \(LL^H\), where \(L\) is lower triangular.

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

3: \texttt{n} – Integer \hfill Input

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

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

4: \texttt{nrhs} – Integer \hfill Input

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

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

5: \texttt{a[dim]} – const Complex \hfill Input

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

\textit{On entry:} the \(n\) by \(n\) original Hermitian positive-definite matrix \(A\) as supplied to nag_zpotrf (f07frc).

6: \texttt{pda} – Integer \hfill Input

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

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

7: \texttt{af[dim]} – const Complex \hfill Input

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

\textit{On entry:} the Cholesky factor of \(A\), as returned by nag_zpotrf (f07frc).

8: \texttt{pdaf} – Integer \hfill Input

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

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

9: \texttt{b[dim]} – const Complex \hfill Input

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

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

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

10: \texttt{pdb} – Integer \hfill 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:}

- if \texttt{order} = Nag_ColMajor, \texttt{pdb} \(\geq \max(1, \texttt{n})\);
if order = Nag_RowMajor, pdb \geq \max(1, nrhs).

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

\textbf{Note:} the dimension, \textit{dim}, of the array \textbf{x} must be at least \max(1, pdb \times nrhs) when order = Nag_ColMajor and at least \max(1, pdb \times n) when order = Nag_RowMajor.

If order = Nag_ColMajor, the \((i, j)\)th element of the matrix \(X\) is stored in \(x[(j - 1) \times pdb + i - 1]\) and if order = Nag_RowMajor, the \((i, j)\)th element of the matrix \(X\) is stored in \(x[(i - 1) \times pdb + j - 1]\).

On entry: the \(n\) by \(r\) solution matrix \(X\), as returned by nag_zpotrs (f07fsc).

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

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

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

\textbf{Constraints:}

if order = Nag_ColMajor, \(pdx \geq \max(1, n)\);
if order = Nag_RowMajor, \(pdx \geq \max(1, nrhs)\).

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

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

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

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

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

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

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

The NAG error parameter (see the Essential Introduction).

6 \textbf{Error Indicators and Warnings}

\textbf{NE_INT}

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

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

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

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

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

On entry, \textbf{pdx} = \langle\text{value}\rangle.
Constraint: \(pdx > 0\).
NE_INT_2

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

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

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

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

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

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 \( ferr \) 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 nag_dporfs (f07fhc).

9 Example

To solve the system of equations \( AX = B \) using iterative refinement and to compute the forward and backward error bounds, where

\[
A = \begin{pmatrix}
3.23 + 0.00i & 1.51 - 1.92i & 1.90 + 0.84i & 0.42 + 2.50i \\
1.51 + 1.92i & 3.58 + 0.00i & -0.23 + 1.11i & -1.18 + 1.37i \\
1.90 - 0.84i & -0.23 - 1.11i & 4.09 + 0.00i & 2.33 - 0.14i \\
0.42 - 2.50i & -1.18 - 1.37i & 2.33 + 0.14i & 4.29 + 0.00i
\end{pmatrix}
\]

and
Here $A$ is Hermitian positive-definite and must first be factorized by \texttt{nag_zpotrf(\texttt{f07frc})}.

### 9.1 Program Text

```c
/* \texttt{nag_zporfs(\texttt{f07fvc}) 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, ferr_len, berr_len;
  Integer exit_status=0;
  Nag_UploType uplo_enum;
  NagError fail;
  Nag_OrderType order;

  /* Arrays */
  char uplo[2];
  Complex *a=0, *af=0, *b=0, *x=0;
  double *berr=0, *ferr=0;

  INIT_FAIL(fail);
  Vprintf("f07fvc Example Program Results\n\n");
  /* Skip heading in data file */
  Vscanf("%*\[^
\] ");
  Vscanf("%ld%ld%*\[^
\] ", &n, &nrhs);
  #ifdef NAG_COLUMN_MAJOR
  pda = n;
  pdaf = n;
  pdb = n;
  pdx = n;
  #endif
  #ifndef NAG_COLUMN_MAJOR
  pda = n;
  pdaf = n;
  pdb = nrhs;
  pdx = nrhs;
  #endif
  ferr_len = nrhs;
  ```
berr_len = nrhs;

/* Allocate memory */
if ( !(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
");
    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
");
    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);
    }
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
            Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
    }
}
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        Vscanf(" ( %lf , %lf )", &B(i,j).re, &B(i,j).im);
}
/* 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;
        }
    }
}
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 */
f07frc(order, uplo_enum, n, af, pdaf, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07frc.\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Compute solution in the array X */
f07fsc(order, uplo_enum, n, nrhs, af, pdaf, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07fsc.\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07fvc(order, uplo_enum, n, nrhs, a, pda, af, pdaf, b, pdb, x, pdx,
ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07fvc.\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, 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", berr[j-1], j%4 == 0 ?\n": "");
Vprintf("\nEstimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e", ferr[j-1], j%4 == 0 ?\n": "");
Vprintf("\n");
END:
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

f07fvc Example Program Data

4 2 :Values of N and NRHS
'L' :Value of UPLO

(3.23, 0.00)
(1.51, 1.92) (3.58, 0.00)
(1.90,−0.84) (−0.23,−1.11) (4.09, 0.00)
(0.42,−2.50) (−1.18,−1.37) (2.33, 0.14) (4.29, 0.00) :End of matrix A

(3.93, −6.14) (1.48, 6.58)
(6.17, 9.42) (4.65, −4.75)
(−7.17,−21.83) (−4.91, 2.29)
(1.99,−14.38) (7.64,−10.79) :End of matrix B

9.3 Program Results

f07fvc Example Program Results

Solution(s)

1 2
1 (1.0000,−1.0000) (−1.0000, 2.0000)
2 (−0.0000, 3.0000) (3.0000,−4.0000)
3 (−4.0000,−5.0000) (−2.0000, 3.0000)
4 (2.0000, 1.0000) (4.0000,−5.0000)

Backward errors (machine-dependent)

3.3e−17 5.6e−17

Estimated forward error bounds (machine-dependent)

5.7e−14 7.2e−14