NAG C Library Function Document

nag_zsyrfs (f07nvc)

1 Purpose

nag_zsyrfs (f07nvc) returns error bounds for the solution of a complex symmetric 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_zsyrfs (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_zsyrfs (f07nvc) returns the backward errors and estimated bounds on the forward errors for the solution of a complex symmetric 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_zsyrfs (f07nvc) 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 \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 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 `uplo = Nag_Upper`, the upper triangular part of $A$ is stored and $A$ is factorized as $PUDU^TP^T$, where $U$ is upper triangular;

if `uplo = Nag_Lower`, the lower triangular part of $A$ is stored and $A$ is factorized as $PLDL^TP^T$, where $L$ is lower triangular.

**Constraint:** `uplo = Nag_Upper` or `Nag_Lower`.

3: `n` – Integer

*Input*

*On entry:* $n$, the order of the matrix $A$.

*Constraint:* $n \geq 0$.

4: `nrhs` – Integer

*Input*

*On entry:* $r$, the number of right-hand sides.

*Constraint:* $nrhs \geq 0$.

5: `a[1dim]` – const Complex

*Input*

*Note:* the dimension, `dim`, of the array `a` must be at least `max(1, pda \times n)`. 

*On entry:* the $n$ by $n$ original symmetric matrix $A$ as supplied to nag_zsytrf (f07nrc).

6: `pda` – Integer

*Input*

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

*Constraint:* $pda \geq max(1, n)$.

7: `af[1dim]` – const Complex

*Input*

*Note:* the dimension, `dim`, of the array `af` must be at least `max(1, pdaf \times n)`. 

*On entry:* details of the factorization of $A$, as returned by nag_zsytrf (f07nrc).

8: `pdaf` – Integer

*Input*

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

*Constraint:* $pdaf \geq max(1, n)$.

9: `ipiv[1dim]` – const Integer

*Input*

*Note:* the dimension, `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 nag_zsytrf (f07nrc).

10: `b[1dim]` – const Complex

*Input*

*Note:* the dimension, `dim`, of the array `b` 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 $B$ is stored in $b[(j-1) \times pdb + i - 1]$ and if `order = Nag_RowMajor`, the $(i, j)$th element of the matrix $B$ is stored in $b[(i-1) \times pdb + j - 1]$.

*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 \( \text{order} = \text{Nag\_ColMajor} \), \( \text{pdb} \geq \max(1, n) \);
if \( \text{order} = \text{Nag\_RowMajor} \), \( \text{pdb} \geq \max(1, \text{nrhs}) \).

12: \( x[\text{dim}] \) – Complex

Input/Output

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

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

On entry: the \( n \) by \( r \) solution matrix \( X \), as returned by nag_zsytrs (f07nsc).

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

13: \( \text{pdx} \) – Integer

Input

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

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: \( \text{ferr}[\text{dim}] \) – double

Output

Note: the dimension, \( \text{dim} \), of the array \( \text{ferr} \) 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 \( j = 1, 2, \ldots, r \).

15: \( \text{berr}[\text{dim}] \) – double

Output

Note: the dimension, \( \text{dim} \), of the array \( \text{berr} \) 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 \( j = 1, 2, \ldots, r \).

16: \( \text{fail} \) – NagError *

Output

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

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

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

On entry, \( \text{pdx} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} > 0 \).
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_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 \( \text{nag_dsyrf} \) (f07mhc).

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

\[
A = \begin{pmatrix}
-0.39 - 0.71i & 5.14 - 0.64i & -7.86 - 2.96i & 3.80 + 0.92i \\
5.14 - 0.64i & 8.86 + 1.81i & -3.52 + 0.58i & 5.32 - 1.59i \\
-7.86 - 2.96i & -3.52 + 0.58i & -2.83 - 0.03i & -1.54 - 2.86i \\
3.80 + 0.92i & 5.32 - 1.59i & -1.54 - 2.86i & -0.56 + 0.12i \\
\end{pmatrix}
\]

and
Here $A$ is symmetric and must first be factorized by $\text{nag_zsytrf (f07nrc)}$.

### 9.1 Program Text

/* $\text{nag_zsyrsf (f07nvc)}$ 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;

    #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("f07nvc Example Program Results\n\n");
    */ Skip heading in data file */
    Vscanf("%*[\n] ");
    Vscanf("%d%d%f[\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

    [NP3645/7] f07nvc.5
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);
    }
}
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 <= nrhs; ++i)
    {
        for (j = 1; j <= n; ++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 */
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;
    }
}

if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07nrc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Compute solution in the array X */
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;
    }
}

if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07nsc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
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;
    }
}

if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07nvc.\n%s\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%s\n", fail.message);
    exit_status = 1;
    goto END;
}

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

f07nvc Example Program Data

4 2  : Values of N and NRHS

'L'  : Value of UPLO

(-0.39,-0.71)
( 5.14,-0.64) ( 8.86, 1.81)
(-7.86,-2.96) (-3.52, 0.58) (-2.83,-0.03)
( 3.80, 0.92) ( 5.32,-1.59) (-1.54,-2.86) (-0.56, 0.12)  : End of matrix A

(-55.64, 41.22) (-19.09,-35.97)
(-48.18, 66.00) (-12.08,-27.02)
(-0.49, -1.47) ( 6.95, 20.49)
(-6.43, 19.24) ( -4.59,-35.53)  : End of matrix B

9.3 Program Results

f07nvc 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) (-2.0000,-1.0000)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(-2.0000, 5.0000) ( 1.0000,-3.0000)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( 3.0000,-2.0000) ( 3.0000, 2.0000)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(-4.0000, 3.0000) (-1.0000, 1.0000)</td>
<td></td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)

1.0e-16 6.7e-17

Estimated forward error bounds (machine-dependent)

1.2e-14 1.2e-14