NAG Library Function Document

nag_zsprfs (f07qvc)

1 Purpose

nag_zsprfs (f07qvc) returns error bounds for the solution of a complex symmetric system of linear equations with multiple right-hand sides, \( AX = B \), using packed storage. It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

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

void nag_zsprfs (Nag_OrderType order, Nag_UploType uplo, Integer n, 
    Integer nrhs, const Complex ap[], const Complex afp[], 
    const Integer ipiv[], const Complex b[], Integer pdb, Complex x[], 
    Integer pdx, double ferr[], double berr[], NagError *fail)
```

3 Description

nag_zsprfs (f07qvc) 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 \( AX = B \), using packed storage. The function handles each right-hand side vector (stored as a column of the matrix \( B \)) independently, so we describe the function of nag_zsprfs (f07qvc) 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

\[
(A + \delta_A)x = b + \delta b
\]

where

\[
|\delta a_{ij}| \leq \beta |a_{ij}| \quad \text{and} \quad |\delta b_i| \leq \beta |b_i|.
\]

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 \( \text{order} = \text{Nag_RowMajor} \). See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

   **Constraint:** \( \text{order} = \text{Nag_RowMajor} \) or \( \text{Nag_ColMajor} \).
2: uplo – Nag_UploType

*Input*

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

- \( \text{uplo} = \text{Nag Upper} \)
  - The upper triangular part of \( A \) is stored and \( A \) is factorized as \( PUDU^T P^T \), where \( U \) is upper triangular.

- \( \text{uplo} = \text{Nag Lower} \)
  - The lower triangular part of \( A \) is stored and \( A \) is factorized as \( PLDL^T P^T \), where \( L \) is lower triangular.

*Constraint:* \( \text{uplo} = \text{Nag Upper} \) or \( \text{Nag Lower} \).

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: ap[dim] – const Complex

*Input*

*Note:* the dimension, \( dim \), of the array \( ap \) must be at least \( \max(1, n \times (n + 1)/2) \).

*On entry:* the \( n \) by \( n \) original symmetric matrix \( A \) as supplied to \( \text{nag_zsptrf} \) (f07qrc).

6: afp[dim] – const Complex

*Input*

*Note:* the dimension, \( dim \), of the array \( afp \) must be at least \( \max(1, n \times (n + 1)/2) \).

*On entry:* the factorization of \( A \) stored in packed form, as returned by \( \text{nag_zsptrf} \) (f07qrc).

7: ipiv[dim] – 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 \( \text{nag_zsptrf} \) (f07qrc).

8: b[dim] – const Complex

*Input*

*Note:* the dimension, \( dim \), of the array \( b \) must be at least \( \max(1, \text{pdb} \times \text{nrhs}) \) when \( \text{order} = \text{Nag ColMajor} \);
\[ \max(1, n \times \text{pdb}) \] when \( \text{order} = \text{Nag RowMajor} \).

The \((i, j)\)th element of the matrix \( B \) is stored in

- \( b[(j - 1) \times \text{pdb} + i - 1] \) when \( \text{order} = \text{Nag ColMajor} \);
- \( b[(i - 1) \times \text{pdb} + j - 1] \) when \( \text{order} = \text{Nag RowMajor} \).

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

9: pdb – Integer

*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 \( \text{order} = \text{Nag RowMajor} \), \( \text{pdb} \geq \max(1, \text{nrhs}) \).


10: \( \mathbf{x}[\text{dim}] \) – Complex  

\text{Input/Output}  

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

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

\text{On entry}: the \( n \) by \( r \) solution matrix \( \mathbf{X} \), as returned by \text{nag\_zsptrs} (f07qsc).  
\text{On exit}: the improved solution matrix \( \mathbf{X} \).

11: \( \text{pdx} \) – Integer  

\text{Input}  

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

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

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

\text{Output}  

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

13: \( \text{berr}[\text{nrhs}] \) – double  

\text{Output}  

\text{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 \( \mathbf{X} \), for \( j = 1, 2, \ldots, r \).  

14: \( \text{fail} \) – NagError*  

\text{Input/Output}  

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

6 \ Error Indicators and Warnings

\textbf{NE\textunderscore ALLOC\_FAIL}  
Dynamic memory allocation failed.  
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE\textunderscore BAD\_PARAM}  
On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

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

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

On entry, \( \text{pdx} = \langle \text{value} \rangle \).  
Constraint: \( \text{pdx} > 0 \).
On entry, $\text{pdb} = \langle \text{value} \rangle$ and $n = \langle \text{value} \rangle$.
Constraint: $\text{pdb} \geq \text{max}(1, n)$.

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

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

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

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 ferr are not rigorous, because they are estimated, not computed exactly; but in practice they almost always overestimate the actual error.

nag_zsprfs (f07qvc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zsprfs (f07qvc) 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.

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_dsprfs (f07phc).
10 Example

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

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

and

\[
B = \begin{pmatrix}
-55.64 + 41.22i & -19.09 - 35.97i \\
-48.18 + 66.00i & -12.08 - 27.02i \\
-0.49 - 1.47i & 6.95 + 20.49i \\
-6.43 + 19.24i & -4.59 - 35.53i
\end{pmatrix}
\]

Here \( A \) is symmetric, stored in packed form, and must first be factorized by nag_zsptrf (f07qrc).

10.1 Program Text

/* nag_zsprfs (f07qvc) 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, ap_len, afp_len;
    Integer berr_len, ferr_len, pdb, pdx;
    Integer exit_status = 0;
    NagError fail;
    Nag_UploType uplo;
    Nag_OrderType order;
    /* Arrays */
    Integer *ipiv = 0;
    char nag_enum_arg[40];
    Complex *afp = 0, *ap = 0, *b = 0, *x = 0;
    double *berr = 0, *ferr = 0;
    INIT_FAIL(fail);
    printf("nag_zsprfs (f07qvc) Example Program Results

");
    /* Skip heading in data file */
    #ifdef _WIN32
    #else
    #endif
    #define A_UPPER(I, J) ap[J*(J-1)/2 + I - 1]
    #define A_LOWER(I, J) ap[(2*n-J)*(J-1)/2 + 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_LOWER(I, J) ap[I*(I-1)/2 + J - 1]
    #define A_UPPER(I, J) ap[(2*n-I)*(I-1)/2 + 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);

    /* Skip heading in data file */
    #ifdef _WIN32
    printf("nag_zsprfs (f07qvc) Example Program Results\n\n");
    #endif

    /* Skip heading in data file */
    #ifdef _WIN32
    printf("nag_zsprfs (f07qvc) Example Program Results\n\n");
    #endif
}
```c
scanf_s("%*[\n"]);
#else
scanf("%*[\n"]);
#endif
#else
scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[\n"] , &n, &nrhs);
#else
scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n"] , &n, &nrhs);
#endif
 ap_len = n * (n + 1)/2;
 afp_len = n * (n + 1)/2;
 berr_len = nrhs;
 ferr_len = nrhs;
#endif
 if (uplo == Nag_Upper)
 { for (i = 1; i <= n; ++i)
      for (j = i; j <= n; ++j)
 #ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &A_UPPER(i, j).re,
             &A_UPPER(i, j).im);
 #else
      scanf(" ( %lf , %lf )", &A_UPPER(i, j).re,
             &A_UPPER(i, j).im);
 #endif
 } #ifdef _WIN32
 scanf_s("%*[\n"]);
#else
 scanf("%*[\n"]);
#endif
 else
 { for (i = 1; i <= n; ++i)
      for (j = 1; j <= i; ++j)
 #ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &A_LOWER(i, j).re,
             &A_LOWER(i, j).im);
 #else
      scanf(" ( %lf , %lf )", &A_LOWER(i, j).re,
             &A_LOWER(i, j).im);
 #endif
 } #ifdef _WIN32
 scanf_s("%*[\n"]);
#else
 scanf("%*[\n"]);
#endif
 else
 { for (i = 1; i <= n; ++i)
      for (j = 1; j <= i; ++j)
 #ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &A_LOWER(i, j).re,
```
&A_LOWER(i, j).im);
#endif
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
for (i = 1; i <= n * (n + 1) / 2; ++i)
    { afp[i-1].re = ap[i-1].re;
      afp[i-1].im = ap[i-1].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 AFP */
    /* nag_zsptrf (f07qrc). */
    /* Bunch-Kaufman factorization of complex symmetric matrix, *
     * packed storage */
    nag_zsptrf(order, uplo, n, afp, ipiv, &fail);
    if (fail.code != NE_NOERROR)
    { printf("Error from nag_zsptrf (f07qrc).\n", fail.message);
        exit_status = 1;
        goto END;
    }
    
    /* Compute solution in the array X */
    /* nag_zsptrs (f07qsc). */
    /* Solution of complex symmetric system of linear equations, *
     * multiple right-hand sides, matrix already factorized by *
     * nag_zsptrf (f07qrc), packed storage */
    nag_zsptrs(order, uplo, n, nrhs, afp, ipiv, x, pdx, &fail);
    if (fail.code != NE_NOERROR)
    { printf("Error from nag_zsptrs (f07qsc).\n", fail.message);
        exit_status = 1;
        goto END;
    }
    
    /* Improve solution, and compute backward errors and */
    /* estimated bounds on the forward errors */
    /* nag_zsprfs (f07qvc). */
    /* Refined solution with error bounds of complex symmetric *
     * system of linear equations, multiple right-hand sides, *
     * packed storage */

nag_zsprfs(order, uplo, n, nrhs, ap, afp, ipiv, b, pdb,
x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zsprfs (f07qvc).\n\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print solution */
/* nag_gen_complex_mat_print_comp (x04dbc). */
/* Print complex general matrix (comprehensive) */
fflush(stdout);
NAG_FREE(ipiv);
NAG_FREE(afp);
NAG_FREE(ap);
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(berr);
NAG_FREE(ferr);
return exit_status;}

10.2 Program Data
nag_zsprfs (f07qvc) Example Program Data

4 2 :Values of n and nrhs
   Nag_Lower :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)
   (-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)

10.3 Program Results
nag_zsprfs (f07qvc) Example Program Results

Solution(s)

1 2
1 ( 1.0000, -1.0000) ( -2.0000, -1.0000)
2 ( -2.0000, 5.0000) ( 1.0000, -3.0000)
3 ( 3.0000, -2.0000) ( 3.0000, 2.0000)
4 ( -4.0000, 3.0000) ( -1.0000, 1.0000)
<table>
<thead>
<tr>
<th>Backward errors (machine-dependent)</th>
<th>8.2e-17</th>
<th>4.9e-17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated forward error bounds (machine-dependent)</td>
<td>1.2e-14</td>
<td>1.2e-14</td>
</tr>
</tbody>
</table>