NAG Library Function Document

nag_zpprfs (f07gvc)

1 Purpose

nag_zpprfs (f07gvc) returns error bounds for the solution of a complex Hermitian positive definite 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_zpprfs (Nag_OrderType order, Nag_UploType uplo, Integer n,
                Integer nrhs, const Complex ap[], const Complex afp[],
                const Complex b[], Integer pdb, Complex x[], Integer pdx,
                double ferr[], double berr[], NagError *fail)
```

3 Description

nag_zpprfs (f07gvc) 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 $AX = B$, 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_zpprfs (f07gvc) 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.
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.

- **uplo = Nag_Upper**
  The upper triangular part of $A$ is stored and $A$ is factorized as $U^H U$, where $U$ is upper triangular.

- **uplo = Nag_Lower**
  The lower triangular part of $A$ is stored and $A$ is factorized as $L L^H$, where $L$ is lower triangular.

*Constraint: uplo = Nag_Upper or Nag_Lower.*

**n** – Integer

*Input*

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

*Constraint: $n \geq 0$.**

**nrhs** – Integer

*Input*

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

*Constraint: nrhs $\geq 0$.**

**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 Hermitian positive definite matrix $A$ as supplied to nag_zpptrf (f07grc).

**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 Cholesky factor of $A$ stored in packed form, as returned by nag_zpptrf (f07grc).

**b[dim]** – const Complex

*Input*

Note: the dimension, *dim*, of the array *b* must be at least

\[
\max(1, \text{pdb} \times \text{nrhs}) \quad \text{when order = Nag_ColMajor};
\]

\[
\max(1, n \times \text{pdb}) \quad \text{when order = Nag_RowMajor}.
\]

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

\[
b[(j - 1) \times \text{pdb} + i - 1] \quad \text{when order = Nag_ColMajor};
\]

\[
b[(i - 1) \times \text{pdb} + j - 1] \quad \text{when order = Nag_RowMajor}.
\]

On entry: the $n$ by $r$ right-hand side matrix $B$.

**pdb** – Integer

*Input*

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

*Constraints:*

- if order = Nag_ColMajor, pdb $\geq \max(1, n)$;
- if order = Nag_RowMajor, pdb $\geq \max(1, nrhs)$. 
9:  \( \mathbf{x}[\text{dim}] \) – Complex

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

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

On entry: the \( n \) by \( r \) solution matrix \( X \), as returned by \text{nag\_zpptrs (f07gsc)}.

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

10:  \text{pdx} – Integer

Input

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

Constraints:
\[
\text{if } \text{order} = \text{Nag\_ColMajor}, \text{pdx} \geq \max(1, \text{n});
\]
\[
\text{if } \text{order} = \text{Nag\_RowMajor}, \text{pdx} \geq \max(1, \text{nrhs}).
\]

11:  \text{ferr[nrhs]} – double

Output

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

12:  \text{berr[nrhs]} – double

Output

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

13:  \text{fail} – \text{NagError}\ *

Input/Output

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

6 Error Indicators and Warnings

\text{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\text{NE\_BAD\_PARAM}

On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

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

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

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

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 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.

NE_NO_LICENCE

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.

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 Parallelism and Performance

\text{nag_zpprfs (f07gvc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\text{nag_zpprfs (f07gvc)} 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 \text{nag_dpprfs (f07ghc)}. 
10 Example

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

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

and

$$B = \begin{pmatrix}
3.93 - 6.14i & 1.48 + 6.58i \\
6.17 + 9.42i & 4.65 - 4.75i \\
-7.17 - 21.83i & -4.91 + 2.29i \\
1.99 - 14.38i & 7.64 - 10.79i \\
\end{pmatrix}.$$  

Here $A$ is Hermitian positive definite, stored in packed form, and must first be factorized by nag_zpptrf (f07grc).

10.1 Program Text

/* nag_zpprfs (f07gvc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
*/

#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 */
    char nag_enum_arg[40];
    Complex *afp = 0, *ap = 0, *b = 0, *x = 0;
    double *berr = 0, *ferr = 0;

    #ifdef NAG_COLUMN_MAJOR
    #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);

    printf("nag_zpprfs (f07gvc) Example Program Results\n\n");

    /* Skip heading in data file */
}

Mark 25
```c
#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
    ap_len = n*(n+1)/2;
    afp_len = n*(n+1)/2;
    berr_len = nrhs;
    ferr_len = nrhs;
#ifdef NAG_COLUMN_MAJOR
    pdb = n;
    pdx = n;
#else
    pdb = nrhs;
    pdx = nrhs;
#endif
/* Allocate memory */
if (!(afp = NAG_ALLOC(afp_len, Complex)) ||
    !(ap = NAG_ALLOC(ap_len, 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 AFP 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)
    {
        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
        }
    }
    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
        }
    }
#endif
#endif
```
&A_LOWER(i, j).im);
#endif
#endif
}
#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
#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 * (n + 1) / 2; ++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_zpptrf (f07grc). */
* Cholesky factorization of complex Hermitian
* positive-definite matrix, packed storage
*/
nag_zpptrf(order, uplo, n, afp, &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_zpptrf (f07grc).\n", fail.message);
  exit_status = 1;
  goto END;
}
/* Compute solution in the array X */
/* nag_zpptrs (f07gsc). */
* Solution of complex Hermitian positive-definite system of
* linear equations, multiple right-hand sides, matrix
* already factorized by nag_zpptrf (f07grc), packed storage
*/
nag_zpptrs(order, uplo, n, nrhs, afp, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_zpptrs (f07gsc).\n", fail.message);
  exit_status = 1;
  goto END;
}
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
/* nag_zpprfs (f07gvc). */
* Refined solution with error bounds of complex Hermitian
* positive-definite system of linear equations, multiple
* right-hand sides, packed storage
*/
nag_zpprfs(order, uplo, n, nrhs, ap, afp, b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zpprfs (f07gvc).\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, Nag_BracketForm, "%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).
", fail.message);
    exit_status = 1;
    goto END;
}
printf("Backward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", berr[j-1], j%4 == 0?"\n":" ");
printf("Estimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", ferr[j-1], j%4 == 0?"\n":" ");
printf("\n");
END: 
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_zpprfs (f07gvc) Example Program Data
4 2 :Values of n and nrhs
   Nag_Lower :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

10.3 Program Results
nag_zpprfs (f07gvc) 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)
<table>
<thead>
<tr>
<th>Backward errors (machine-dependent)</th>
<th>(1.1e{-16})</th>
<th>(7.9e{-17})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated forward error bounds (machine-dependent)</td>
<td>(6.1e{-14})</td>
<td>(7.4e{-14})</td>
</tr>
</tbody>
</table>