NAG C Library Function Document

nag_zhprfs (f07pvc)

1 Purpose

nag_zhprfs (f07pvc) returns error bounds for the solution of a complex Hermitian indefinite 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
void nag_zhprfs (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_zhprfs (f07pvc) 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, $AX = B$, 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_zhprfs (f07pvc) 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: **uplo** – Nag_UploType  
    *Input*  
    *On entry:* indicates whether the upper or lower triangular part of A is stored and how A has been  
    factorized, as follows:  
    - If **uplo** = Nag_Upper, the upper triangular part of A is stored and A is factorized as  
      \[ PUDU^T \]  
      where \( U \) is upper triangular;  
    - If **uplo** = Nag_Lower, the lower triangular part of A is stored and A is factorized as  
      \[ PLDL^T \]  
      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: **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 matrix A as supplied to nag_zhptrf (f07prc).

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:* details of the factorization of A stored in packed form, as returned by nag_zhptrf (f07prc).

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 nag_zhptrf (f07prc).

8: **b**[**dim**] – const Complex  
    *Input*  
    *Note:* the dimension, **dim**, of the array **b** must be at least \( \max(1, pdb \times nrhs) \) when \( \text{order} = \text{Nag_ColMajor} \) and at least \( \max(1, pdb \times n) \) when \( \text{order} = \text{Nag_RowMajor} \).  
    If \( \text{order} = \text{Nag_ColMajor} \), the \((i, j)\)th element of the matrix B is stored in \( b[(j - 1) \times pdb + i - 1] \) and  
    if \( \text{order} = \text{Nag_RowMajor} \), 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.

9: **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 **order** = Nag_ColMajor, \( pdb \geq \max(1, n) \);  
    - If **order** = Nag_RowMajor, \( pdb \geq \max(1, nrhs) \).

10: **x**[**dim**] – Complex  
    *Input/Output*  
    *Note:* the dimension, **dim**, of the array **x** must be at least \( \max(1, pdx \times nrhs) \) when  
    **order** = Nag_ColMajor and at least \( \max(1, pdx \times n) \) when **order** = Nag_RowMajor.
If \( \text{order} = \text{Nag-ColMajor} \), the \((i, j)\)th element of the matrix \( X \) is stored in \( x[(j - 1) \times \text{pdx} + i - 1] \) and if \( \text{order} = \text{Nag-RowMajor} \), the \((i, j)\)th element of the matrix \( X \) is stored in \( x[(i - 1) \times \text{pdx} + j - 1] \).

On entry: the \( n \) by \( r \) solution matrix \( X \), as returned by nag_zhptrs (f07psc).

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

11: \( \text{pdx} \) – 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 \( \text{order} = \text{Nag-RowMajor} \), \( \text{pdx} \geq \max(1, \text{nrhs}) \).

12: \( \text{ferr}[\text{dim}] \) – 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 \( j = 1, 2, \ldots, r \).

13: \( \text{berr}[\text{dim}] \) – 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 \( j = 1, 2, \ldots, r \).

14: \( \text{fail} \) – 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, \( \text{pdx} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} > 0 \).

NE_INT_2

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

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

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

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

On entry, parameter (value) had an illegal value.

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.

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.

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_dsprfs (f07phc).

To solve 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, stored in packed form, and must first be factorized by nag_zhptrf (f07prc).

/* nag_zhprfs (f07pvc) 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, ap_len, afp_len;
    Integer berr_len, ferr_len, pdb, pdx;
    Integer exit_status=0;
    NagError fail;
    Nag_UploType uplo_enum;
    Nag_OrderType order;
    /* Arrays */
    Integer *ipiv=0;
    char uplo[2];
    Complex *afp=0, *ap=0, *b=0, *x=0;
    double *berr=0, *ferr=0;
    #ifdef NAG_COLUMN_MAJOR
    #define A_LOWER(I,J) ap[(2*n-J)*(J-1)/2 + I - 1]
    #define A_UPPER(I,J) ap[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);
    Vprintf("f07pvc Example Program Results\n\n");
    /* Skip heading in data file */
    Vscanf("%*[\n"] ;
    Vscanf("%ld%ld%*[\n"] , &n, &nrhs);
    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 ( !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(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)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Read A and B from data file, and copy A to AFP and B to X */
    Vscanf("%s %*[\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_UPPER(i,j).re, &A_UPPER(i,j).im);
    }
    Vscanf("%*[\n] ");
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
            Vscanf(" ( %lf , %lf )", &A_LOWER(i,j).re, &A_LOWER(i,j).im);
    }
    Vscanf("%*[\n] ");
}
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 */
f07prc(order, uplo_enum, n, afp, ipiv, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07prc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute solution in the array X */
f07psc(order, uplo_enum, n, nrhs, afp, ipiv, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07psc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07pvc(order, uplo_enum, n, nrhs, ap, afp, ipiv, b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07pvc.
%s
", 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.
%s
", fail.message);
exit_status = 1;
goto END;
}
Vprintf("\nBackward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
  Vprintf("%11.1e%s", berr[j-1], j%4==0 ?"\n":" ");
Vprintf("\nEstimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
  Vprintf("%11.1e%s", ferr[j-1], j%4==0 ?"\n":" ");
Vprintf("\n");
END:
if (ipiv) NAG_FREE(ipiv);
if (afp) NAG_FREE(afp);
if (ap) NAG_FREE(ap);
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

f07pvc Example Program Data

4 2
'L'
(-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) :End of matrix A
( 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) :End of matrix B

9.3 Program Results

f07pvc 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)
9.0e-17 5.8e-17
Estimated forward error bounds (machine-dependent)
2.6e-15 3.0e-15