NAG C 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
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

$$(A + \delta A)x = b + \delta b$$

$|\delta a_{ij}| \leq \beta|a_{ij}|$ and $|\delta b_i| \leq \beta|b_i|$.

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

   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

   On entry: indicates whether the upper or lower triangular part of $A$ is stored and how $A$ has been factorized, as follows:

   - Nag_Upper: The upper triangular part of $A$ is stored.
   - Nag_Lower: The lower triangular part of $A$ is stored.

[NP3645/7] f07gvc.1
if uplo = Nag_Upper, then the upper triangular part of A is stored and A is factorized as $U^H U$, where $U$ is upper triangular;

if uplo = Nag_Lower, then the lower triangular part of A is stored and A is factorized as $LL^H$, where $L$ is lower triangular.

Constraint: uplo = Nag_Upper or Nag_Lower.

3: \n\n\n\n$n$ – Integer
On entry: $n$, the order of the matrix $A$.
Constraint: $n \geq 0$.

4: \n\n\n\nrhs – Integer
On entry: $r$, the number of right-hand sides.
Constraint: $nrhs \geq 0$.

5: \n\n\n\n$ap[dim]$ – 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).

6: \n\n\n\nafp[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).

7: \n\n\n\nb[dim] – 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 $B$.

8: \n\n\n\npdb – 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, $nrhs$).

9: \n\n\n\nx[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 order = Nag_ColMajor, the $(i,j)$th element of the matrix $X$ is stored in $x[(j-1) \times pdx + i - 1]$ and if order = Nag_RowMajor, the $(i,j)$th element of the matrix $X$ is stored in $x[(i-1) \times pdx + j - 1]$.
On entry: the $n$ by $r$ solution matrix $X$, as returned by nag_zpptrs (f07gsc).
On exit: the improved solution matrix $X$.

10: \n\n\n\npdx – Integer
Input
On entry: the stride separating matrix row or column elements (depending on the value of order) in the array x.
Constraints:

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

11: ferr[dim] – double  
Output

Note: the dimension, dim, of the array ferr must be at least max(1, nrhs).

On exit: ferr[j - 1] contains an estimated error bound for the jth solution vector, that is, the jth column of X, for j = 1, 2, ..., r.

12: berr[dim] – double  
Output

Note: the dimension, dim, of the array berr must be at least max(1, nrhs).

On exit: berr[j - 1] contains the component-wise backward error bound β for the jth solution vector, that is, the jth column of X, for j = 1, 2, ..., r.

13: fail – NagError *
Output

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, n = ⟨value⟩.
Constraint: n ≥ 0.

On entry, nrhs = ⟨value⟩.
Constraint: nrhs ≥ 0.

On entry, pdb = ⟨value⟩.
Constraint: pdb > 0.

On entry, pdx = ⟨value⟩.
Constraint: pdx > 0.

NE_INT_2

On entry, pdb = ⟨value⟩, n = ⟨value⟩.
Constraint: pdb ≥ max(1, n).

On entry, pdb = ⟨value⟩, nrhs = ⟨value⟩.
Constraint: pdb ≥ max(1, nrhs).

On entry, pdx = ⟨value⟩, n = ⟨value⟩.
Constraint: pdx ≥ max(1, n).

On entry, pdx = ⟨value⟩, nrhs = ⟨value⟩.
Constraint: pdx ≥ max(1, nrhs).

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter ⟨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 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_dpprfs (f07ghc).

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

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

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

9.1 Program Text

/* nag_zpprfs (f07gvc) 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 */
    char uplo[2];
    Complex *afp=0, *ap=0, *b=0, *x=0;
    double *berr=0, *ferr=0;

    /* Compute solution and error bounds */
    nag_zpptrs(uplo_enum, n, nrhs, afp, &pdb, pdx, x, &pdb, &ferr_len, ferr);
    nag_zpptrs_fixed(uplo_enum, n, nrhs, ap, b, &pdb, pdx, &berr_len, berr);

    /* Print solution and error bounds */
    printf("Solution:
    ");
    nag_zpptrs_solution(uplo_enum, n, nrhs, x, pdb, pdx, &pdb, &pdb, 0);
    printf("\nBackward error bounds:");
    printf("%20.16e\n", ferr);
    printf("\nForward error bounds:");
    printf("%20.16e\n", berr);

    return 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);
Vprintf("f07gvc 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;
#endif

*p 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)) )
{
    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)
    Vscanf(" ( %lf , %lf )", &B(i,j).re, &B(i,j).im);
}
Vscanf("%*[\n]");
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 */
f07grc(order, uplo_enum, n, afp, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07grc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute solution in the array X */
f07gsc(order, uplo_enum, n, nrhs, afp, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07gsc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07gvc(order, uplo_enum, n, nrhs, ap, afp, b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07gvc.
%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, 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("\n%11.1e\n", berr[j-1], j%4==0 ?"\n":" ");
Vprintf("\nEstimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("\n%11.1e\n", ferr[j-1], j%4==0 ?"\n":" ");
Vprintf("\n");
END:
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

f07gvc 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) ( 3.58, 0.00)
(0.42,-2.50) (-1.18,-1.37) ( 2.33, 0.14) ( 4.09, 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

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)

Backward errors (machine-dependent)
3.3e-17 5.6e-17
Estimated forward error bounds (machine-dependent)
5.7e-14 7.2e-14