NAG C 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
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

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

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

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 the f07 Chapter Introduction.

4 References


5 Parameters

1: \( \text{order} \) – Nag_OrderType

*Input*

On entry: the \( \text{order} \) 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 \( \text{order} = \text{Nag_RowMajor} \). See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: \( \text{order} = \text{Nag_RowMajor} \) or \( \text{Nag_ColMajor} \).
**2:** \texttt{uplo} – \texttt{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 \(\text{uplo} = \text{Nag\_Upper}\), the upper triangular part of \(A\) is stored and \(A\) is factorized as \(PUDU^TP^T\), where \(U\) is upper triangular;
- if \(\text{uplo} = \text{Nag\_Lower}\), the lower triangular part of \(A\) is stored and \(A\) is factorized as \(PLDL^TP^T\), where \(L\) is lower triangular.

\textit{Constraint:} \(\text{uplo} = \text{Nag\_Upper}\) or \(\text{Nag\_Lower}\).

**3:** \(n\) – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \(n\), the order of the matrix \(A\).

\textit{Constraint:} \(n \geq 0\).

**4:** \(\text{nrhs}\) – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \(r\), the number of right-hand sides.

\textit{Constraint:} \(\text{nrhs} \geq 0\).

**5:** \(\text{ap}[\text{dim}]\) – const Complex \hspace{1cm} \textit{Input}

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

\textit{On entry:} the \(n\) by \(n\) original symmetric matrix \(A\) as supplied to \texttt{nag\_zsptrf (f07qrc)}.

**6:** \(\text{afp}[\text{dim}]\) – const Complex \hspace{1cm} \textit{Input}

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

\textit{On entry:} details of the factorization of \(A\) stored in packed form, as returned by \texttt{nag\_zsptrf (f07qrc)}.

**7:** \(\text{ipiv}[\text{dim}]\) – const Integer \hspace{1cm} \textit{Input}

\textit{Note:} the dimension, \(\text{dim}\), of the array \(\text{ipiv}\) must be at least \(\max(1, n)\).

\textit{On entry:} details of the interchanges and the block structure of \(D\), as returned by \texttt{nag\_zsptrf (f07qrc)}.

**8:** \(\text{b}[\text{dim}]\) – const Complex \hspace{1cm} \textit{Input}

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

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

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

**9:** \(\text{pdb}\) – Integer \hspace{1cm} \textit{Input}

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

\textit{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:** \(\text{x}[\text{dim}]\) – Complex \hspace{1cm} \textit{Input/Output}

\textit{Note:} the dimension, \(\text{dim}\), of the array \(\text{x}\) must be at least \(\max(1, \text{pdx} \times \text{nrhs})\) when \(\text{order} = \text{Nag\_ColMajor}\) and at least \(\max(1, \text{pdx} \times n)\) when \(\text{order} = \text{Nag\_RowMajor}\).
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_zsptrs (f07qsc).

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

11:  pdx – 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 ≥ \(\max(1, nrhs)\).

12:  ferr\([\text{dim}]\) – double  

Output  

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

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

Output  

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

On exit: 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:  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: \(\text{pdx} \geq \max(1, \text{nrhs})\).
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_dsprfs (f07phc).

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

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

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

9.1 Program Text

/* nag_zsprfs (f07qvc) Example Program. */
* * Copyright 2001 Numerical Algorithms Group. *
* * Mark 7, 2001. */

#include <stdio.h>
#include <nag.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_UPPER(I,J) ap[(J-1)*(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("f07qvc 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("%ls %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;
    }

    /* Solve the linear system AX = B */
    /* ... */

} /* end of main */

f07qvc.5
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);
    }
}
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);
    }
}
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 */
f07qrc(order, uplo_enum, n, afp, ipiv, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07qrc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
/* Compute solution in the array X */
f07qsc(order, uplo_enum, n, nrhs, afp, ipiv, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07qsc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07qvc(order, uplo_enum, n, nrhs, ap, afp, ipiv, b, pdb, x, pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07qvc.
%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, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)
{

Vprintf("Error from x04dbc.\n%s\n", 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

f07qvc Example Program Data
4 2 :Values of N and NRHS
 'L' :Value of UPLO
(-0.39,-0.71) (-7.86,-2.96)
( 5.14,-0.64) ( 8.86, 1.81)
(-55.64, 41.22) (-48.18, 66.00)
(-4.59,-35.53) (-1.47, 6.95) :End of matrix A
( 3.80, 0.92) ( 5.32,-1.59)
(-7.86,-2.96) (-3.52, 0.58)
( -0.49, -1.47) (-12.08,-27.02)
( 3.80, 0.92) ( 5.32,-1.59) :End of matrix B

9.3 Program Results

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)

Backward errors (machine-dependent)
  1.0e-16  6.7e-17
Estimated forward error bounds (machine-dependent)
  1.2e-14  1.2e-14