

NAG C Library Function Document

nag_dporfs (f07fnc)

1 Purpose

nag_dporfs (f07fnc) returns error bounds for the solution of a real symmetric positive-definite system of linear equations with multiple right-hand sides, $AX = B$. It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

```
void nag_dporfs (Nag_OrderType order, Nag_UploType uplo, Integer n, Integer nrhs,
  const double a[], Integer pda, const double af[], Integer pdaf,
  const double b[], Integer pdb, double x[], Integer pdx, double ferr[],
  double berr[], NagError *fail)
```

3 Description

nag_dporfs (f07fnc) returns the backward errors and estimated bounds on the forward errors for the solution of a real symmetric positive-definite system of linear equations with multiple right-hand sides $AX = B$. The function handles each right-hand side vector (stored as a column of the matrix B) independently, so we describe the function of nag_dporfs (f07fnc) 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* β . 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

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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**, then the upper triangular part of A is stored and A is factorized as $U^T U$, where U is upper triangular;
 if **uplo** = **Nag_Lower**, then the lower triangular part of A is stored and A is factorized as LL^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** ≥ 0 .
- 5: **a**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$.
On entry: the n by n original symmetric positive-definite matrix A as supplied to nag_dpotr (f07fdc).
- 6: **pda** – Integer *Input*
On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix in the array **a**.
Constraint: **pda** $\geq \max(1, \mathbf{n})$.
- 7: **af**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **af** must be at least $\max(1, \mathbf{pdaf} \times \mathbf{n})$.
On entry: the Cholesky factor of A , as returned by nag_dpotr (f07fdc).
- 8: **pdaf** – Integer *Input*
On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix in the array **af**.
Constraint: **pdaf** $\geq \max(1, \mathbf{n})$.
- 9: **b**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **b** must be at least $\max(1, \mathbf{pdb} \times \mathbf{nrhs})$ when **order** = **Nag_ColMajor** and at least $\max(1, \mathbf{pdb} \times \mathbf{n})$ 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 .
- 10: **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** $\geq \max(1, \mathbf{n})$;

if **order** = **Nag_RowMajor**, **pdb** \geq $\max(1, \mathbf{nrhs})$.

11: **x**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **x** must be at least $\max(1, \mathbf{pdx} \times \mathbf{nrhs})$ when **order** = **Nag_ColMajor** and at least $\max(1, \mathbf{pdx} \times \mathbf{n})$ 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_dpotrs (f07fec).

On exit: the improved solution matrix *X*.

12: **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** \geq $\max(1, \mathbf{n})$;
if **order** = **Nag_RowMajor**, **pdx** \geq $\max(1, \mathbf{nrhs})$.

13: **ferr**[*dim*] – double *Output*

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

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, ..., *r*.

14: **berr**[*dim*] – double *Output*

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

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

15: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **n** = *value*.

Constraint: **n** \geq 0.

On entry, **nrhs** = *value*.

Constraint: **nrhs** \geq 0.

On entry, **pda** = *value*.

Constraint: **pda** > 0.

On entry, **pdaf** = *value*.

Constraint: **pdaf** > 0.

On entry, **pdb** = *value*.

Constraint: **pdb** > 0.

On entry, **pdx** = *value*.

Constraint: **pdx** > 0.

NE_INT_2

On entry, **pda** = $\langle value \rangle$, **n** = $\langle value \rangle$.
 Constraint: **pda** \geq max(1, **n**).

On entry, **pdaf** = $\langle value \rangle$, **n** = $\langle value \rangle$.
 Constraint: **pdaf** \geq max(1, **n**).

On entry, **pdb** = $\langle value \rangle$, **n** = $\langle value \rangle$.
 Constraint: **pdb** \geq max(1, **n**).

On entry, **pdb** = $\langle value \rangle$, **nrhs** = $\langle value \rangle$.
 Constraint: **pdb** \geq max(1, **nrhs**).

On entry, **pdx** = $\langle value \rangle$, **n** = $\langle value \rangle$.
 Constraint: **pdx** \geq max(1, **n**).

On entry, **pdx** = $\langle value \rangle$, **nrhs** = $\langle value \rangle$.
 Constraint: **pdx** \geq max(1, **nrhs**).

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ 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 $4n^2$ floating-point operations. Each step of iterative refinement involves an additional $6n^2$ 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 4 or 5 and never more than 11. Each solution involves approximately $2n^2$ operations.

The complex analogue of this function is nag_zporfs (f07fvc).

9 Example

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

$$A = \begin{pmatrix} 4.16 & -3.12 & 0.56 & -0.10 \\ -3.12 & 5.03 & -0.83 & 1.18 \\ 0.56 & -0.83 & 0.76 & 0.34 \\ -0.10 & 1.18 & 0.34 & 1.18 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 8.70 & 8.30 \\ -13.35 & 2.13 \\ 1.89 & 1.61 \\ -4.14 & 5.00 \end{pmatrix}.$$

Here A is symmetric positive-definite and must first be factorized by nag_dpotrf (f07fdc).

9.1 Program Text

```

/* nag_dporfs (f07fhc) 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 berr_len, ferr_len, i, j, n, nrhs, pda, pdaf, pdb, pdx;
    Integer exit_status=0;
    Nag_UploType uplo_enum;
    NagError fail;
    Nag_OrderType order;

    /* Arrays */
    char uplo[2];
    double *a=0, *af=0, *b=0, *berr=0, *ferr=0, *x=0;

#ifdef NAG_COLUMN_MAJOR
#define A(I,J) a[(J-1)*pda + I - 1]
#define AF(I,J) af[(J-1)*pdaf + 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(I,J) a[(I-1)*pda + J - 1]
#define AF(I,J) af[(I-1)*pdaf + 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("f07fhc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[^\\n] ");
    Vscanf("%ld%ld%*[^\\n] ", &n, &nrhs);
#ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdaf = n;
    pdb = n;
    pdx = n;
#else
    pda = n;
    pdaf = n;
    pdb = nrhs;
    pdx = nrhs;
#endif
    ferr_len = nrhs;
    berr_len = nrhs;

    /* Allocate memory */
    if ( !(a = NAG_ALLOC(n * n, double)) ||
        !(af = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) ||
        !(berr = NAG_ALLOC(berr_len, double)) ||
        !(ferr = NAG_ALLOC(ferr_len, double)) ||
        !(x = NAG_ALLOC(n * nrhs, double)) )
    {
        Vprintf("Allocation failure\n");
    }
}

```

```

        exit_status = -1;
        goto END;
    }

/* Read A and B from data file, and copy A to AF and B to X */
Vscanf(" ' %1s '%*[\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", &A(i,j));
            }
        Vscanf("%*[\n] ");
    }
else
    {
        for (i = 1; i <= n; ++i)
            {
                for (j = 1; j <= i; ++j)
                    Vscanf("%lf", &A(i,j));
            }
        Vscanf("%*[\n] ");
    }

for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= nrhs; ++j)
            Vscanf("%lf", &B(i,j));
    }
Vscanf("%*[\n] ");
/* Copy A to AF and B to X */
if (uplo_enum == Nag_Upper)
    {
        for (i = 1; i <= n; ++i)
            {
                for (j = i; j <= n; ++j)
                    AF(i,j) = A(i,j);
            }
    }
else
    {
        for (i = 1; i <= n; ++i)
            {
                for (j = 1; j <= i; ++j)
                    AF(i,j) = A(i,j);
            }
    }

for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= nrhs; ++j)
            X(i,j) = B(i,j);
    }
/* Factorize A in the array AF */
f07fdc(order, uplo_enum, n, af, pdaf, &fail);
if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f07fdc.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

```

```

    }
    /* Compute solution in the array X */
    f07fec(order, uplo_enum, n, nrhs, af, pdaf, x, pdx, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f07fec.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Improve solution, and compute backward errors and */
    /* estimated bounds on the forward errors */
    f07fhc(order, uplo_enum, n, nrhs, a, pda, af, pdaf, b, pdb, x, pdx,
           ferr, berr, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f07fhc.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Print solution */
    x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, pdx,
           "Solution(s)", 0, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from x04cac.\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%7==0 ? "\n": " ");
    Vprintf("\nEstimated forward error bounds (machine-dependent)\n");
    for (j = 1; j <= nrhs; ++j)
        Vprintf("%11.1e%s", ferr[j-1], j%7==0 || j==nrhs ? "\n": " ");
END:
    if (a) NAG_FREE(a);
    if (af) NAG_FREE(af);
    if (b) NAG_FREE(b);
    if (berr) NAG_FREE(berr);
    if (ferr) NAG_FREE(ferr);
    if (x) NAG_FREE(x);
    return exit_status;
}

```

9.2 Program Data

f07fhc Example Program Data

```

  4  2           :Values of N and NRHS
  'L'           :Value of UPLO
  4.16
 -3.12  5.03
  0.56 -0.83  0.76
 -0.10  1.18  0.34  1.18   :End of matrix A
  8.70  8.30
-13.35  2.13
  1.89  1.61
 -4.14  5.00           :End of matrix B

```

9.3 Program Results

f07fhc Example Program Results

```

Solution(s)
           1           2
  1      1.0000      4.0000
  2     -1.0000      3.0000
  3      2.0000      2.0000
  4     -3.0000      1.0000

```

Backward errors (machine-dependent)

9.5e-17 5.2e-17

Estimated forward error bounds (machine-dependent)

2.3e-14 2.3e-14
