1 Purpose

nag_dsprfs (f07phc) returns error bounds for the solution of a real symmetric 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
#include <nag.h>
#include <nagf07.h>

void nag_dsprfs (Nag_OrderType order, Nag_UploType uplo, Integer n,
                 Integer nrhs, const double ap[], const double afp[],
                 const Integer ipiv[], const double b[], Integer pdb,
                 double x[], Integer pdx, double ferr[], double berr[], NagError *fail)
```

3 Description

nag_dsprfs (f07phc) returns the backward errors and estimated bounds on the forward errors for the solution of a real symmetric 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_dsprfs (f07phc) 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

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

with \( |\delta a_{ij}| \leq \beta |a_{ij}| \) 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 Arguments

1. **order** – Nag_OrderType  
   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.
2: **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 $PUDU^TPT$, where $U$ is upper triangular.

**uplo** = Nag_Lower

The lower triangular part of $A$ is stored and $A$ is factorized as $PLDL^TPT$, 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 double

*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 symmetric matrix $A$ as supplied to nag_dsptrf (f07pdc).

6: **afp[dim]** – const double

*Input*

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

On entry: the factorization of $A$ stored in packed form, as returned by nag_dsptrf (f07pdc).

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_dsptrf (f07pdc).

8: **b[dim]** – const double

*Input*

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

$$\max(1, \text{pdb} \times \text{nrhs})$$ when **order** = Nag_ColMajor;

$$\max(1, n \times \text{pdb})$$ when **order** = Nag_RowMajor.

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

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

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

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

9: **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, $\text{pdb} \geq \max(1, n)$;
- if **order** = Nag_RowMajor, $\text{pdb} \geq \max(1, \text{nrhs})$. 

---

**f07phc.2 Mark 25**
Note: the dimension, \( \text{dim} \), of the array \( x \) must be at least
\[
\max(1, \text{pdx} \times \text{nrhs}) \text{ when } \text{order} = \text{Nag_COL_MAJOR};
\]
\[
\max(1, \text{n} \times \text{pdx}) \text{ when } \text{order} = \text{Nag_ROW_MAJOR}.
\]

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

On entry: the \( n \) by \( r \) solution matrix \( X \), as returned by nag_dsptrs (f07pec).

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

**pdx** – Integer

Input

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

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

**ferr**[\( \text{nrhs} \)] – double

Output

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 \( j = 1, 2, \ldots, r \).

**berr**[\( \text{nrhs} \)] – double

Output

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 \( j = 1, 2, \ldots, r \).

**fail** – NagError *

Input/Output

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

6 Error Indicators and Warnings

**NE_ALLOCFAIL**

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

**NE_BAD_PARAM**

On entry, argument \(<\text{value}>\) had an illegal value.

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

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

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

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

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

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
\nag_dsprfs (f07phc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\nag_dsprfs (f07phc) 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 \( 4n^2 \) floating-point operations. Each step of iterative refinement involves an additional \( 6n^2 \) operations. At most five 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 analogues of this function are \text{nag_zhprfs} (f07pvc) for Hermitian matrices and \text{nag_zsprfs} (f07qvc) for symmetric matrices.
10 Example

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

$$A = \begin{pmatrix} 2.07 & 3.87 & 4.20 & -1.15 \\ 3.87 & -0.21 & 1.87 & 0.63 \\ 4.20 & 1.87 & 1.15 & 2.06 \\ -1.15 & 0.63 & 2.06 & -1.81 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -9.50 \\ -8.38 \\ -6.07 \\ -0.96 \end{pmatrix}.$$ 

Here $A$ is symmetric indefinite, stored in packed form, and must first be factorized by nag_dsptrf (f07pdc).

10.1 Program Text

/* nag_dsprfs (f07phc) Example Program. */
/* Copyright 2014 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, pdb, pdx, ferr_len, berr_len;
    Integer exit_status = 0;
    NagError fail;
    Nag_UploType uplo;
    Nag_OrderType order;
    /* Arrays */
    Integer *ipiv = 0;
    char nag_enum_arg[40];
    double *afp = 0, *ap = 0, *b = 0, *berr = 0, *ferr = 0, *x = 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[(J-1)*pdb + I - 1]
    #define X(I, J) x[(J-1)*pdx + I - 1]
    order = Nag_RowMajor;
    #endif
    INIT_FAIL(fail);

    printf("nag_dsprfs (f07phc) Example Program Results\n\n");
/* Skip heading in data file */
    #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

    /* Make the call to nag_dsprfs */

    Call nagf07phc( &n, &nrhs, ap, &ap_len, ipiv, &ap_len, &afp, &afp_len, pdb, &pdb, &b, &pdb, &berr, &berr_len, &ferr, &ferr_len, &x, &pdx, &x_len, &exit_status, fail);

    printf("\nError in nag_dsprfs \n\n\n");
    if(exit_status != 0)
    {
        if(nag_enum_arg[0] == 'W')
            printf("\nWarning from nag_dsprfs \n\n");
        else
            printf("\nError from nag_dsprfs \n\n");
    }
    printf("\nExit Status from nag_dsprfs = %d\n\n", exit_status);
    return exit_status;
}

Mark 25
\begin{verbatim}
ap_len = n * (n + 1)/2;
apf_len = n * (n + 1)/2;
#endif NAG_COLUMN_MAJOR
pdb = n;
pdx = n;
#else
pdb = nrhs;
pdx = nrhs;
#endif
ferr_len = nrhs;
berr_len = nrhs;

/* Allocate memory */
if (!(ipiv = NAG_ALLOC(n, Integer)) ||
    !(afp = NAG_ALLOC(ap_len, double)) ||
    !(ap = NAG_ALLOC(apf_len, 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)))
{
    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 */
#elif _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", &A_UPPER(i, j));
            #else
                scanf("%lf", &A_UPPER(i, j));
            #endif
    }
    #ifdef _WIN32
        scanf_s("%*[\n ] ");
    #else
        scanf("%*[\n ] ");
    #endif
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
            #ifdef _WIN32
                scanf_s("%lf", &A_LOWER(i, j));
            #else
                scanf("%lf", &A_LOWER(i, j));
            #endif
    }
    #ifdef _WIN32
        scanf_s("%*[\n ] ");
    #else
        scanf("%*[\n ] ");
    #endif
}
#endif _WIN32

\end{verbatim}
for (i = 1; i <= n; ++i)
    { 
        for (j = 1; j <= nrhs; ++j)
            #ifdef _WIN32
                scanf_s("%lf", &B(i, j));
            #else
                scanf("%lf", &B(i, j));
            #endif
    }
    #ifdef _WIN32
        scanf_s("%*[\n] ");
    #else
        scanf("%*[\n] ");
    #endif

for (i = 1; i <= n * (n + 1) / 2; ++i)
    afp[i - 1] = ap[i - 1];
for (i = 1; i <= n; ++i)
    { 
        for (j = 1; j <= nrhs; ++j)
            X(i, j) = B(i, j);
    }

/* Factorize A in the array AFP */
/* nag_dsptrf (f07pdc). */
/* Bunch-Kaufman factorization of real symmetric indefinite */
/* matrix, packed storage */
/* nag_dsptrf(order, uplo, n, afp, ipiv, &fail); */
if (fail.code != NE_NOERROR)
    { 
        printf("Error from nag_dsptrf (f07pdc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

/* Compute solution in the array X */
/* nag_dsptrs (f07pec). */
/* Solution of real symmetric indefinite system of linear */
/* equations, multiple right-hand sides, matrix already */
/* factorized by nag_dsptrf (f07pdc), packed storage */
/* nag_dsptrs(order, uplo, n, nrhs, afp, ipiv, x, pdx, &fail); */
if (fail.code != NE_NOERROR)
    { 
        printf("Error from nag_dsptrs (f07pec).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
/* nag_dsprfs (f07phc). */
/* Refined solution with error bounds of real symmetric */
/* indefinite system of linear equations, multiple */
/* right-hand sides, packed storage */
/* nag_dsprfs(order, uplo, n, nrhs, ap, afp, ipiv, b, pdb, */
/* x, pdx, ferr, berr, &fail); */
if (fail.code != NE_NOERROR)
    { 
        printf("Error from nag_dsprfs (f07phc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

/* Print solution */
/* nag_gen_real_mat_print (x04cac). */
/* Print real general matrix (easy-to-use) */
/* fflush(stdout); */
/* nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, */
/* pdx, "Solution(s)", 0, &fail); */

Mark 25

f07phc.7
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
printf("\nBackward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", berr[j-1], j%7 == 0?"\n":" ");
printf("\nEstimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", ferr[j-1], j%7 == 0?"\n":" ");
printf("\n");
END:
NAG_FREE(ipiv);
NAG_FREE(afp);
NAG_FREE(ap);
NAG_FREE(b);
NAG_FREE(berr);
NAG_FREE(ferr);
NAG_FREE(x);
return exit_status;
}

10.2 Program Data

nag_dsprfs (f07phc) Example Program Data
4 2 :Values of n and nrhs
Nag_Lower :Value of uplo
2.07
  3.87 -0.21
  4.20 1.87 1.15
-1.15 0.63 2.06 -1.81 :End of matrix A
-9.50 27.85
-8.38 9.90
-6.07 19.25
-0.96 3.93 :End of matrix B

10.3 Program Results

nag_dsprfs (f07phc) Example Program Results

Solution(s)
  1  2
  1 -4.0000  1.0000
  2 -1.0000  4.0000
  3  2.0000  3.0000
  4  5.0000  2.0000

Backward errors (machine-dependent)
5.7e-17 1.0e-16
Estimated forward error bounds (machine-dependent)
2.3e-14 3.4e-14