NAG C Library Function Document

nag_dpprfs (f07ghc)

1 Purpose

nag_dpprfs (f07ghc) returns error bounds for the solution of a real symmetric 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

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

3 Description

nag_dpprfs (f07ghc) 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 $AX = B$, 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_dpprfs (f07ghc) 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**, the upper triangular part of A is stored and A is factorized as U^TU, where U is upper triangular;

if **uplo** = **Nag_Lower**, 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: $\mathbf{n} \geq 0$.

4: **nrhs** – Integer *Input*

On entry: r, the number of right-hand sides.

Constraint: $\mathbf{nrhs} \geq 0$.

5: **ap**[dim] – const double *Input*

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

On entry: the n by n original symmetric positive-definite matrix A as supplied to nag_dpptrf (f07gdc).

6: **afp**[dim] – const double *Input*

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

On entry: the Cholesky factor of A stored in packed form, as returned by nag_dpptrf (f07gdc).

7: **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 $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix B is stored in $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$.

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

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

9: **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 $\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix X is stored in $\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$.

On entry: the n by r solution matrix X, as returned by nag_dpptrs (f07gec).

On exit: the improved solution matrix X.

10: **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, **n**);
if **order** = **Nag_RowMajor**, **pdx** $\geq$ 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 $j = 1, 2, \ldots, r$.

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 $j = 1, 2, \ldots, r$.

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** $\geq$ max(1, **n**).

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

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

On entry, **pdx** = ⟨*value*⟩, **nrhs** = ⟨*value*⟩.
Constraint: **pdx** $\geq$ 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 $4n^2$ 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_zpprfs (f07gvc).

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, stored in packed form, and must first be factorized by nag_dpptrf (f07gdc).

9.1 Program Text

```
/* nag_dpprfs (f07ghc) 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, pdb, pdx, ferr_len, berr_len;
  Integer  exit_status=0;
  NagError fail;
  Nag_UploType uplo_enum;
  Nag_OrderType order;
  /* Arrays */
  char   uplo[2];
  double *afp=0, *ap=0, *b=0, *berr=0, *ferr=0, *x=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("f07ghc 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;
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
  pdx = n;
#else
  pdb = nrhs;
  pdx = nrhs;
#endif

  ferr_len = nrhs;
  berr_len = nrhs;

  /* Allocate memory */
  if ( !(afp = NAG_ALLOC(ap_len, double)) ||
       !(ap = NAG_ALLOC(afp_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)) )
    {
      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(" ' %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_UPPER(i,j));
        }
      Vscanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            Vscanf("%lf", &A_LOWER(i,j));
        }
      Vscanf("%*[^\n] ");
    }
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
```

```
        Vscanf("%lf", &B(i,j));
    }
  Vscanf("%*[^\n] ");
  for (i = 0; i < n * (n + 1) / 2; ++i)
    afp[i] = ap[i];
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        X(i,j) = B(i,j);
    }
  /* Factorize A in the array AFP */
  f07gdc(order, uplo_enum, n, afp, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f07gdc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Compute solution in the array X */
  f07gec(order, uplo_enum, n, nrhs, afp, x, pdx, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f07gec.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Improve solution, and compute backward errors and */
  /* estimated bounds on the forward errors */
  f07ghc(order, uplo_enum, n, nrhs, ap, afp, b, pdb, x, pdx, ferr, berr,
         &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f07ghc.\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 ?"\n":" ");
  Vprintf("\n");
 END:
  if (afp) NAG_FREE(afp);
  if (ap) NAG_FREE(ap);
  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

```
f07ghc 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

```
f07ghc 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)
    8.3e-17     5.2e-17
Estimated forward error bounds (machine-dependent)
    2.4e-14     2.2e-14
```