NAG Library Function Document

nag_dtprfs (f07uhc)

1 Purpose

nag_dtprfs (f07uhc) returns error bounds for the solution of a real triangular system of linear equations with multiple right-hand sides, $AX = B$ or $A^{\mathrm{T}}X = B$, using packed storage.

2 Specification

```
#include <nag.h>
#include <nagf07.h>
void nag_dtprfs (Nag_OrderType order, Nag_UploType uplo,
     Nag_TransType trans, Nag_DiagType diag, Integer n, Integer nrhs,
     const double ap[], const double b[], Integer pdb, const double x[],
     Integer pdx, double ferr[], double berr[], NagError *fail)
```

3 Description

nag_dtprfs (f07uhc) returns the backward errors and estimated bounds on the forward errors for the solution of a real triangular system of linear equations with multiple right-hand sides $AX = B$ or $A^{\mathrm{T}}X = B$, 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_dtprfs (f07uhc) 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 Arguments

1: **order** – Nag_OrderType *Input*

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 A is upper or lower triangular.

uplo = Nag_Upper
A is upper triangular.

uplo = Nag_Lower
A is lower triangular.

Constraint: **uplo** = Nag_Upper or Nag_Lower.

3: **trans** – Nag_TransType *Input*

On entry: indicates the form of the equations.

trans = Nag_NoTrans
The equations are of the form $AX = B$.

trans = Nag_Trans or Nag_ConjTrans
The equations are of the form $A^{\mathrm{T}}X = B$.

Constraint: **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

4: **diag** – Nag_DiagType *Input*

On entry: indicates whether A is a nonunit or unit triangular matrix.

diag = Nag_NonUnitDiag
A is a nonunit triangular matrix.

diag = Nag_UnitDiag
A is a unit triangular matrix; the diagonal elements are not referenced and are assumed to be 1.

Constraint: **diag** = Nag_NonUnitDiag or Nag_UnitDiag.

5: **n** – Integer *Input*

On entry: n, the order of the matrix A.

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

6: **nrhs** – Integer *Input*

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

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

7: **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 triangular matrix A, packed by rows or columns.

The storage of elements A_{ij} depends on the **order** and **uplo** arguments as follows:

if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Upper',
A_{ij} is stored in **ap**$[(j-1) \times j/2 + i - 1]$, for $i \leq j$;
if **order** = 'Nag_ColMajor' and **uplo** = 'Nag_Lower',
A_{ij} is stored in **ap**$[(2n-j) \times (j-1)/2 + i - 1]$, for $i \geq j$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Upper',
A_{ij} is stored in **ap**$[(2n-i) \times (i-1)/2 + j - 1]$, for $i \leq j$;
if **order** = 'Nag_RowMajor' and **uplo** = 'Nag_Lower',
A_{ij} is stored in **ap**$[(i-1) \times i/2 + j - 1]$, for $i \geq j$.

If **diag** = 'Nag_UnitDiag', the diagonal elements of AP are assumed to be 1, and are not referenced; the same storage scheme is used whether **diag** = 'Nag_NonUnitDiag' or **diag** = 'Nag_UnitDiag'.

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

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

b$[(j-1) \times \mathbf{pdb} + i - 1]$ when **order** = Nag_ColMajor;
b$[(i-1) \times \mathbf{pdb} + j - 1]$ 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, **pdb** $\geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, **pdb** $\geq \max(1, \mathbf{nrhs})$.

10: **x**[dim] – const double *Input*

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

$\max(1, \mathbf{pdx} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdx})$ when **order** = Nag_RowMajor.

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

x$[(j-1) \times \mathbf{pdx} + i - 1]$ when **order** = Nag_ColMajor;
x$[(i-1) \times \mathbf{pdx} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the n by r solution matrix X, as returned by nag_dtptrs (f07uec).

11: **pdx** – Integer *Input*

On entry: the stride separating 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** $\geq \max(1, \mathbf{nrhs})$.

12: **ferr**[**nrhs**] – double *Output*

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

13: **berr**[**nrhs**] – double *Output*

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

14: **fail** – NagError * *Input/Output*

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

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*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdb** $\geq \max(1, \mathbf{n})$.

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

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

On entry, **pdx** = ⟨*value*⟩ and **nrhs** = ⟨*value*⟩.
Constraint: **pdx** $\geq \max(1, \mathbf{nrhs})$.

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.

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 Parallelism and Performance

nag_dtprfs (f07uhc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dtprfs (f07uhc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

A call to nag_dtprfs (f07uhc), for each right-hand side, involves solving a number of systems of linear equations of the form $Ax = b$ or $A^{\mathrm{T}}x = b$; the number is usually 4 or 5 and never more than 11. Each solution involves approximately n^2 floating-point operations.

The complex analogue of this function is nag_ztprfs (f07uvc).

10 Example

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

$$A = \begin{pmatrix} 4.30 & 0.00 & 0.00 & 0.00 \\ -3.96 & -4.87 & 0.00 & 0.00 \\ 0.40 & 0.31 & -8.02 & 0.00 \\ -0.27 & 0.07 & -5.95 & 0.12 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -12.90 & -21.50 \\ 16.75 & 14.93 \\ -17.55 & 6.33 \\ -11.04 & 8.09 \end{pmatrix},$$

using packed storage for A.

10.1 Program Text

```
/* nag_dtprfs (f07uhc) 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       ap_len, i, j, n, nrhs, berr_len, ferr_len;
  Integer       pdb, pdx;
  Integer       exit_status = 0;
  Nag_UploType  uplo;
  NagError      fail;
  Nag_OrderType order;
  /* Arrays */
  char          nag_enum_arg[40];
  double        *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);

  printf("nag_dtprfs (f07uhc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
```

```
  scanf("%ld%ld%*[^\n] ", &n, &nrhs);
  berr_len = nrhs;
  ferr_len = nrhs;
  ap_len = n*(n+1)/2;
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
  pdx = n;
#else
  pdb = nrhs;
  pdx = nrhs;
#endif

  /* Allocate memory */
  if (!(ap = NAG_ALLOC(ap_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 B to X */
  scanf(" %39s%*[^\n] ", nag_enum_arg);
  /* 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)
            scanf("%lf", &A_UPPER(i, j));
        }
      scanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            scanf("%lf", &A_LOWER(i, j));
        }
      scanf("%*[^\n] ");
    }
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        scanf("%lf", &B(i, j));
    }
  scanf("%*[^\n] ");
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        X(i, j) = B(i, j);
    }
  /* Compute solution in the array X */
  /* nag_dtptrs (f07uec).
   * Solution of real triangular system of linear equations,
   * multiple right-hand sides, packed storage
   */
  nag_dtptrs(order, uplo, Nag_NoTrans, Nag_NonUnitDiag, n,
             nrhs, ap, x, pdx, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dtptrs (f07uec).\n%s\n", fail.message);
      exit_status = 1;
```

```
      goto END;
    }

  /* Compute backward errors and estimated bounds on the */
  /* forward errors */

  /* nag_dtprfs (f07uhc).
   * Error bounds for solution of real triangular system of
   * linear equations, multiple right-hand sides, packed
   * storage
   */
  nag_dtprfs(order, uplo, Nag_NoTrans, Nag_NonUnitDiag, n,
             nrhs, ap, b, pdb, x, pdx, ferr, berr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dtprfs (f07uhc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print solution */

  printf("\n");
  /* 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);
  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 || j == nrhs)?"\n":" ");
 END:
  NAG_FREE(ap);
  NAG_FREE(b);
  NAG_FREE(berr);
  NAG_FREE(ferr);
  NAG_FREE(x);

  return exit_status;
}
```

10.2 Program Data

```
nag_dtprfs (f07uhc) Example Program Data
  4  2                        :Values of n and nrhs
  Nag_Lower                   :Value of uplo
  4.30
 -3.96  -4.87
  0.40   0.31  -8.02
 -0.27   0.07  -5.95   0.12   :End of matrix A
-12.90 -21.50
 16.75  14.93
-17.55   6.33
-11.04   8.09                 :End of matrix B
```

10.3 Program Results

```
nag_dtprfs (f07uhc) Example Program Results


 Solution(s)
             1          2
 1     -3.0000    -5.0000
 2     -1.0000     1.0000
 3      2.0000    -1.0000
 4      1.0000     6.0000

Backward errors (machine-dependent)
    6.9e-17     0.0e+00
Estimated forward error bounds (machine-dependent)
    8.3e-14     2.6e-14
```