NAG C Library Function Document

nag_dgbtrs (f07bec)

1 Purpose

nag_dgbtrs (f07bec) solves a real band system of linear equations with multiple right-hand sides, $AX = B$ or $A^T X = B$, where A has been factorized by nag_dgbtrf (f07bdc).

2 Specification

```
void nag_dgbtrs (Nag_OrderType order, Nag_TransType trans, Integer n, Integer kl,
      Integer ku, Integer nrhs, const double ab[], Integer pdab,
      const Integer ipiv[], double b[], Integer pdb, NagError *fail)
```

3 Description

To solve a real band system of linear equations $AX = B$ or $A^T X = B$, this function must be preceded by a call to nag_dgbtrf (f07bdc) which computes the LU factorization of A as $A = PLU$. The solution is computed by forward and backward substitution.

If **trans** = **Nag_NoTrans**, the solution is computed by solving $PLY = B$ and then $UX = Y$.

If **trans** = **Nag_Trans** or **Nag_ConjTrans**, the solution is computed by solving $U^T Y = B$ and then $L^T P^T X = Y$.

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: **trans** – Nag_TransType *Input*

On entry: indicates the form of the equations as follows:

if **trans** = **Nag_NoTrans**, $AX = B$ is solved for X;

if **trans** = **Nag_Trans** or **Nag_ConjTrans**, $A^T X = B$ is solved for X.

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

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

4: **kl** – Integer *Input*

On entry: k_l, the number of sub-diagonals within the band of A.

Constraint: **kl** ≥ 0.

5: **ku** – Integer *Input*

On entry: k_u, the number of super-diagonals within the band of A.

Constraint: **ku** ≥ 0.

6: **nrhs** – Integer *Input*

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

Constraint: **nrhs** ≥ 0.

7: **ab**$[dim]$ – const double *Input*

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

On entry: the LU factorization of A, as returned by nag_dgbtrf (f07bdc).

8: **pdab** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix in the array **ab**.

Constraint: **pdab** $\geq 2 \times \mathbf{kl} + \mathbf{ku} + 1$.

9: **ipiv**$[dim]$ – const Integer *Input*

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

On entry: the pivot indices, as returned by nag_dgbtrf (f07bdc).

10: **b**$[dim]$ – double *Input/Output*

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.

On exit: the n by r solution matrix X.

11: **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** $\geq \max(1, \mathbf{nrhs})$.

12: **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, **kl** = ⟨*value*⟩.
Constraint: **kl** ≥ 0.

On entry, **ku** = ⟨*value*⟩.
Constraint: **ku** ≥ 0.

On entry, **nrhs** = ⟨*value*⟩.
Constraint: **nrhs** ≥ 0.

On entry, **pdab** = ⟨*value*⟩.
Constraint: **pdab** > 0.

On entry, **pdb** = ⟨*value*⟩.
Constraint: **pdb** > 0.

NE_INT_2

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

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

NE_INT_3

On entry, **pdab** = ⟨*value*⟩, **kl** = ⟨*value*⟩, **ku** = ⟨*value*⟩.
Constraint: **pdab** $\geq 2 \times \mathbf{kl} + \mathbf{ku} + 1$.

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

For each right-hand side vector b, the computed solution x is the exact solution of a perturbed system of equations $(A + E)x = b$, where

$$|E| \leq c(k)\epsilon P|L|\,|U|,$$

$c(k)$ is a modest linear function of $k = k_l + k_u + 1$, and ϵ is the ***machine precision***. This assumes $k \ll n$.

If $\hat{x}$ is the true solution, then the computed solution x satisfies a forward error bound of the form

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq c(k)\,\mathrm{cond}(A, x)\epsilon$$

where $\mathrm{cond}(A, x) = \||A^{-1}|\,|A|\,|x|\|_\infty / \|x\|_\infty \leq \mathrm{cond}(A) = \||A^{-1}|\,|A|\|_\infty \leq \kappa_\infty(A)$. Note that $\mathrm{cond}(A, x)$ can be much smaller than $\mathrm{cond}(A)$, and $\mathrm{cond}(A^T)$ can be much larger (or smaller) than $\mathrm{cond}(A)$.

Forward and backward error bounds can be computed by calling nag_dgbrfs (f07bhc), and an estimate for $\kappa_\infty(A)$ can be obtained by calling nag_dgbcon (f07bgc) with **norm** = **Nag_InfNorm**.

8 Further Comments

The total number of floating-point operations is approximately $2n(2k_l + k_u)r$, assuming $n \gg k_l$ and $n \gg k_u$.

This function may be followed by a call to nag_dgbrfs (f07bhc) to refine the solution and return an error estimate.

The complex analogue of this function is nag_zgbtrs (f07bsc).

9 Example

To solve the system of equations $AX = B$, where

$$A = \begin{pmatrix} -0.23 & 2.54 & -3.66 & 0.00 \\ -6.98 & 2.46 & -2.73 & -2.13 \\ 0.00 & 2.56 & 2.46 & 4.07 \\ 0.00 & 0.00 & -4.78 & -3.82 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 4.42 & -36.01 \\ 27.13 & -31.67 \\ -6.14 & -1.16 \\ 10.50 & -25.82 \end{pmatrix}.$$

Here A is nonsymmetric and is treated as a band matrix, which must first be factorized by nag_dgbtrf (f07bdc).

9.1 Program Text

```
/* nag_dgbtrs (f07bec) 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, ipiv_len, j, kl, ku, n, nrhs, pdab, pdb;
  Integer  exit_status=0;
  NagError fail;
  Nag_OrderType order;

  /* Arrays */
  double  *ab=0, *b=0;
  Integer *ipiv=0;

#ifdef NAG_COLUMN_MAJOR
#define AB(I,J) ab[(J-1)*pdab + kl + ku + I - J]
#define B(I,J) b[(J-1)*pdb + I - 1]
  order = Nag_ColMajor;
#else
#define AB(I,J) ab[(I-1)*pdab + kl + J - I]
#define B(I,J) b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);
  Vprintf("f07bec Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%ld%ld%ld%*[^\n] ", &n, &nrhs, &kl, &ku);
  ipiv_len = n;
#ifdef NAG_COLUMN_MAJOR
  pdab = 2*kl + ku + 1;
  pdb = n;
#else
  pdab = 2*kl + ku + 1;
  pdb = nrhs;
#endif
```

```
  /* Allocate memory */
  if ( !(ab = NAG_ALLOC((2*kl+ku+1) * n, double)) ||
       !(b = NAG_ALLOC(nrhs * n, double)) ||
       !(ipiv = NAG_ALLOC(ipiv_len, Integer)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A from data file */
  for (i = 1; i <= n; ++i)
    {
      for (j = MAX(i-kl,1); j <= MIN(i+ku,n); ++j)
        Vscanf("%lf", &AB(i,j));
    }
  Vscanf("%*[^\n] ");
  /* Read B from data file */
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        Vscanf("%lf", &B(i,j));
    }
  Vscanf("%*[^\n] ");

  /* Factorize A */
  f07bdc(order, n, n, kl, ku, ab, pdab, ipiv, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f07bdc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Compute solution */
  f07bec(order, Nag_NoTrans, n, kl, ku, nrhs, ab, pdab, ipiv,
         b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f07bec.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print solution */
  x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, b, pdb,
         "Solution(s)", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04cac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  if (ab) NAG_FREE(ab);
  if (b) NAG_FREE(b);
  if (ipiv) NAG_FREE(ipiv);
  return exit_status;
}
```

9.2 Program Data

```
f07bec Example Program Data
  4  2  1  2                  :Values of N, NRHS, KL and KU
 -0.23   2.54  -3.66
 -6.98   2.46  -2.73  -2.13
         2.56   2.46   4.07
               -4.78  -3.82   :End of matrix A
  4.42 -36.01
 27.13 -31.67
 -6.14  -1.16
```

```
 10.50 -25.82                 :End of matrix B
```

9.3 Program Results

```
f07bec Example Program Results

 Solution(s)
             1           2
 1     -2.0000      1.0000
 2      3.0000     -4.0000
 3      1.0000      7.0000
 4     -4.0000     -2.0000
```