

NAG C Library Function Document

nag_dgetrs (f07aec)

1 Purpose

nag_dgetrs (f07aec) solves a real system of linear equations with multiple right-hand sides, $AX = B$ or $A^T X = B$, where A has been factorized by nag_dgetrf (f07adc).

2 Specification

```
void nag_dgetrs (Nag_OrderType order, Nag_TransType trans, Integer n, Integer nrhs,
                 const double a[], Integer pda, const Integer ipiv[], double b[], Integer pdb,
                 NagError *fail)
```

3 Description

To solve a real system of linear equations $AX = B$ or $A^T X = B$, this function must be preceded by a call to nag_dgetrf (f07adc) 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: $n \geq 0$.

- 4: **nrhs** – Integer *Input*
On entry: r , the number of right-hand sides.
Constraint: $\mathbf{nrhs} \geq 0$.
- 5: **a**[dim] – const double *Input*
Note: the dimension, dim , of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$.
 If **order** = **Nag_ColMajor**, the (i, j) th element of the matrix A is stored in **a**[($j - 1$) $\times$ **pda** + $i - 1$] and
 if **order** = **Nag_RowMajor**, the (i, j) th element of the matrix A is stored in **a**[($i - 1$) $\times$ **pda** + $j - 1$].
On entry: the LU factorization of A , as returned by nag_dgetrf (f07adc).
- 6: **pda** – Integer *Input*
On entry: the stride separating matrix row or column elements (depending on the value of **order**) in the array **a**.
Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.
- 7: **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_dgetrf (f07adc).
- 8: **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 **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 .
On exit: the n by r solution matrix X .
- 9: **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**, $\mathbf{pdb} \geq \max(1, \mathbf{nrhs})$.
- 10: **fail** – NagError * *Output*
 The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

- On entry, **n** = $\langle value \rangle$.
 Constraint: $\mathbf{n} \geq 0$.
- On entry, **nrhs** = $\langle value \rangle$.
 Constraint: $\mathbf{nrhs} \geq 0$.
- On entry, **pda** = $\langle value \rangle$.
 Constraint: $\mathbf{pda} > 0$.
- On entry, **pdb** = $\langle value \rangle$.
 Constraint: $\mathbf{pdb} > 0$.

NE_INT_2

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

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

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

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

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

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

$c(n)$ is a modest linear function of n , and ϵ is the *machine precision*.

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

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

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

Forward and backward error bounds can be computed by calling nag_dgerfs (f07ahc), and an estimate for $\kappa_{\infty}(A)$ can be obtained by calling nag_dgecon (f07agc) with **norm** = **Nag.InfNorm**.

8 Further Comments

The total number of floating-point operations is approximately $2n^2r$.

This function may be followed by a call to nag_dgerfs (f07ahc) to refine the solution and return an error estimate.

The complex analogue of this function is nag_zgetrs (f07asc).

9 Example

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

$$A = \begin{pmatrix} 1.80 & 2.88 & 2.05 & -0.89 \\ 5.25 & -2.95 & -0.95 & -3.80 \\ 1.58 & -2.69 & -2.90 & -1.04 \\ -1.11 & -0.66 & -0.59 & 0.80 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 9.52 & 18.47 \\ 24.35 & 2.25 \\ 0.77 & -13.28 \\ -6.22 & -6.21 \end{pmatrix}.$$

Here A is nonsymmetric and must first be factorized by nag_dgetrf (f07adc).

9.1 Program Text

```

/* nag_dgetrs (f07aec) 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, pda, pdb;
    Integer exit_status=0;
    NagError fail;
    Nag_OrderType order;

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

#ifdef NAG_COLUMN_MAJOR
#define A(I,J) a[(J-1)*pda + I - 1]
#define B(I,J) b[(J-1)*pdb + I - 1]
    order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
#define B(I,J) b[(I-1)*pdb + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);
    Vprintf("f07aec Example Program Results\n\n");

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

    Vscanf("%ld%ld%*[^\\n] ", &n, &nrhs);

    /* Allocate memory */
    if ( !(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * nrhs, double)) ||
        !(ipiv = NAG_ALLOC(n, Integer)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
#ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdb = n;
#else
    pda = n;
    pdb = nrhs;
#endif

    /* Read A and B from data file */
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= n; ++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] ");

    /* Factorize A */
    f07adc(order, n, n, a, pda, ipiv, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f07adc.\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Compute solution */
    f07aec(order, Nag_NoTrans, n, nrhs, a, pda, ipiv, b, pdb, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f07aec.\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 (a) NAG_FREE(a);
if (b) NAG_FREE(b);
if (ipiv) NAG_FREE(ipiv);
return exit_status;
}

```

9.2 Program Data

f07aec Example Program Data

4	2				:Values of N and NRHS
1.80	2.88	2.05	-0.89		
5.25	-2.95	-0.95	-3.80		
1.58	-2.69	-2.90	-1.04		
-1.11	-0.66	-0.59	0.80	:End of matrix A	
9.52	18.47				
24.35	2.25				
0.77	-13.28				
-6.22	-6.21			:End of matrix B	

9.3 Program Results

f07aec Example Program Results

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