

# NAG Library Function Document

## nag\_zgetrs (f07asc)

### 1 Purpose

nag\_zgetrs (f07asc) solves a complex system of linear equations with multiple right-hand sides,

$$AX = B, \quad A^T X = B \quad \text{or} \quad A^H X = B,$$

where  $A$  has been factorized by nag\_zgetrf (f07arc).

### 2 Specification

```
#include <nag.h>
#include <nagf07.h>

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

### 3 Description

nag\_zgetrs (f07asc) is used to solve a complex system of linear equations  $AX = B$ ,  $A^T X = B$  or  $A^H X = B$ , the function must be preceded by a call to nag\_zgetrf (f07arc) 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, the solution is computed by solving  $U^T Y = B$  and then  $L^T P^T X = Y$ .

If **trans** = Nag\_ConjTrans, the solution is computed by solving  $U^H Y = B$  and then  $L^H P^T X = Y$ .

### 4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

### 5 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: **trans** – Nag\_TransType *Input*

*On entry:* indicates the form of the equations.

**trans** = Nag\_NoTrans  
 $AX = B$  is solved for  $X$ .

**trans** = Nag\_Trans  
 $A^T X = B$  is solved for  $X$ .

**trans** = Nag\_ConjTrans  
 $A^H X = B$  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:* **nrhs**  $\geq 0$ .
- 5: **a**[*dim*] – const Complex *Input*  
**Note:** the dimension, *dim*, of the array **a** must be at least  $\max(1, \mathbf{pda} \times \mathbf{n})$ .  
The ( $i, j$ )th element of the matrix  $A$  is stored in  
 $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$  when **order** = Nag\_RowMajor.  
*On entry:* the  $LU$  factorization of  $A$ , as returned by nag\_zgetrf (f07arc).
- 6: **pda** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **a**.  
*Constraint:* **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\_zgetrf (f07arc).
- 8: **b**[*dim*] – Complex *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;  
 $\max(1, \mathbf{n} \times \mathbf{pdb})$  when **order** = Nag\_RowMajor.  
The ( $i, j$ )th element of the matrix  $B$  is stored in  
 $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$  when **order** = Nag\_RowMajor.  
*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 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: **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  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry,  $\mathbf{n} = \langle value \rangle$ .

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

On entry,  $\mathbf{nrhs} = \langle value \rangle$ .

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

On entry,  $\mathbf{pda} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} > 0$ .

On entry,  $\mathbf{pdb} = \langle value \rangle$ .

Constraint:  $\mathbf{pdb} > 0$ .

### NE\_INT\_2

On entry,  $\mathbf{pda} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} \geq \max(1, \mathbf{n})$ .

On entry,  $\mathbf{pdb} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{pdb} \geq \max(1, \mathbf{n})$ .

On entry,  $\mathbf{pdb} = \langle value \rangle$  and  $\mathbf{nrhs} = \langle value \rangle$ .

Constraint:  $\mathbf{pdb} \geq \max(1, \mathbf{nrhs})$ .

### 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

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  $\epsilon$  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  $\text{cond}(A^H)$  (which is the same as  $\text{cond}(A^T)$ ) can be much larger (or smaller) than  $\text{cond}(A)$ .

Forward and backward error bounds can be computed by calling `nag_zgerfs` (f07avc), and an estimate for  $\kappa_{\infty}(A)$  can be obtained by calling `nag_zgecon` (f07auc) with `norm = Nag_InfNorm`.

## 8 Parallelism and Performance

nag\_zgetrs (f07asc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_zgetrs (f07asc) 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

The total number of real floating-point operations is approximately  $8n^2r$ .

This function may be followed by a call to nag\_zgerfs (f07avc) to refine the solution and return an error estimate.

The real analogue of this function is nag\_dgetrs (f07aec).

## 10 Example

This example solves the system of equations  $AX = B$ , where

$$A = \begin{pmatrix} -1.34 + 2.55i & 0.28 + 3.17i & -6.39 - 2.20i & 0.72 - 0.92i \\ -0.17 - 1.41i & 3.31 - 0.15i & -0.15 + 1.34i & 1.29 + 1.38i \\ -3.29 - 2.39i & -1.91 + 4.42i & -0.14 - 1.35i & 1.72 + 1.35i \\ 2.41 + 0.39i & -0.56 + 1.47i & -0.83 - 0.69i & -1.96 + 0.67i \end{pmatrix}$$

and

$$B = \begin{pmatrix} 26.26 + 51.78i & 31.32 - 6.70i \\ 6.43 - 8.68i & 15.86 - 1.42i \\ -5.75 + 25.31i & -2.15 + 30.19i \\ 1.16 + 2.57i & -2.56 + 7.55i \end{pmatrix}.$$

Here  $A$  is nonsymmetric and must first be factorized by nag\_zgetrf (f07arc).

### 10.1 Program Text

```

/* nag_zgetrs (f07asc) 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, n, nrhs, pda, pdb;
    Integer    exit_status = 0;
    NagError   fail;
    Nag_OrderType order;
    /* Arrays */
    Complex    *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);

    printf("nag_zgetrs (f07asc) Example Program Results\n\n");
    /* Skip heading in data file */
    scanf("%*[\n] ");
    scanf("%ld%ld%*[\n] ", &n, &nrhs);
#ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdb = n;
#else
    pda = n;
    pdb = nrhs;
#endif
    ipiv_len = n;

    /* Allocate memory */
    if (!(a = NAG_ALLOC(n * n, Complex)) ||
        !(b = NAG_ALLOC(n * nrhs, Complex)) ||
        !(ipiv = NAG_ALLOC(ipiv_len, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read A and B from data file */
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= n; ++j)
        {
            scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
        }
    }
    scanf("%*[\n] ");
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= nrhs; ++j)
            scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
    }
    scanf("%*[\n] ");

    /* Factorize A */
    /* nag_zgetrf (f07arc).
     * LU factorization of complex m by n matrix
     */
    nag_zgetrf(order, n, n, a, pda, ipiv, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zgetrf (f07arc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Compute solution */
    /* nag_zgetrs (f07asc).
     * Solution of complex system of linear equations, multiple
     * right-hand sides, matrix already factorized by nag_zgetrf
     * (f07arc)
     */
    nag_zgetrs(order, Nag_NoTrans, n, nrhs, a, pda, ipiv, b, pdb, &fail);
    if (fail.code != NE_NOERROR)
    {

```

```

    printf("Error from nag_zgetrs (f07asc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print solution */
/* nag_gen_complx_mat_print_comp (x04dbc).
 * Print complex general matrix (comprehensive)
 */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                              nrhs, b, pdb, Nag_BracketForm, "%7.4f",
                              "Solution(s)", Nag_IntegerLabels, 0,
                              Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}
END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(ipiv);
return exit_status;
}

```

## 10.2 Program Data

```

nag_zgetrs (f07asc) Example Program Data
  4  2                                     :Values of N and NRHS
(-1.34, 2.55) ( 0.28, 3.17) (-6.39,-2.20) ( 0.72,-0.92)
(-0.17,-1.41) ( 3.31,-0.15) (-0.15, 1.34) ( 1.29, 1.38)
(-3.29,-2.39) (-1.91, 4.42) (-0.14,-1.35) ( 1.72, 1.35)
( 2.41, 0.39) (-0.56, 1.47) (-0.83,-0.69) (-1.96, 0.67) :End of matrix A
(26.26, 51.78) (31.32, -6.70)
( 6.43, -8.68) (15.86, -1.42)
(-5.75, 25.31) (-2.15, 30.19)
( 1.16, 2.57) (-2.56, 7.55)                                     :End of matrix B

```

## 10.3 Program Results

nag\_zgetrs (f07asc) Example Program Results

```

Solution(s)
           1           2
1 ( 1.0000, 1.0000) (-1.0000,-2.0000)
2 ( 2.0000,-3.0000) ( 5.0000, 1.0000)
3 (-4.0000,-5.0000) (-3.0000, 4.0000)
4 ( 0.0000, 6.0000) ( 2.0000,-3.0000)

```

---