NAG Library Function Document

nag_zcgesv (f07aqc)

1 Purpose

nag_zcgesv (f07aqc) computes the solution to a complex system of linear equations

$$AX = B,$$

where A is an n by n matrix and X and B are n by r matrices.

2 Specification

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

```
void nag_zcgesv (Nag_OrderType order, Integer n, Integer nrhs, Complex a[],
     Integer pda, Integer ipiv[], const Complex b[], Integer pdb,
     Complex x[], Integer pdx, Integer *iter, NagError *fail)
```

3 Description

nag_zcgesv (f07aqc) first attempts to factorize the matrix in single precision and use this factorization within an iterative refinement procedure to produce a solution with double precision accuracy. If the approach fails the method switches to a double precision factorization and solve.

The iterative refinement process is stopped if

$$\textbf{iter} > itermax,$$

where **iter** is the number of iterations carried out thus far and $itermax$ is the maximum number of iterations allowed, which is fixed at 30 iterations. The process is also stopped if for all right-hand sides we have

$$\|resid\| < \sqrt{\mathbf{n}}\|x\|\|A\|\epsilon,$$

where $\|resid\|$ is the ∞-norm of the residual, $\|x\|$ is the ∞-norm of the solution, $\|A\|$ is the ∞-operator-norm of the matrix A and ϵ is the ***machine precision*** returned by nag_machine_precision (X02AJC).

The iterative refinement strategy used by nag_zcgesv (f07aqc) can be more efficient than the corresponding direct full precision algorithm. Since this strategy must perform iterative refinement on each right-hand side, any efficiency gains will reduce as the number of right-hand sides increases. Conversely, as the matrix size increases the cost of these iterative refinements become less significant relative to the cost of factorization. Thus, any efficiency gains will be greatest for a very small number of right-hand sides and for large matrix sizes. The cut-off values for the number of right-hand sides and matrix size, for which the iterative refinement strategy performs better, depends on the relative performance of the reduced and full precision factorization and back-substitution. For now, nag_zcgesv (f07aqc) always attempts the iterative refinement strategy first; you are advised to compare the performance of nag_zcgesv (f07aqc) with that of its full precision counterpart nag_zgesv (f07anc) to determine whether this strategy is worthwhile for your particular problem dimensions.

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

Buttari A, Dongarra J, Langou J, Langou J, Luszczek P and Kurzak J (2007) Mixed precision iterative refinement techniques for the solution of dense linear systems *International Journal of High Performance Computing Applications*

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: **n** – Integer *Input*

On entry: n, the number of linear equations, i.e., the order of the matrix A.

Constraint: **n** ≥ 0.

3: **nrhs** – Integer *Input*

On entry: r, the number of right-hand sides, i.e., the number of columns of the matrix B.

Constraint: **nrhs** ≥ 0.

4: **a**[dim] – Complex *Input/Output*

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 n by n coefficient matrix A.

On exit: if iterative refinement has been successfully used (i.e., if **fail.code** = NE_NOERROR and **iter** ≥ 0), then A is unchanged. If double precision factorization has been used (when **fail.code** = NE_NOERROR and **iter** < 0), A contains the factors L and U from the factorization $A = PLU$; the unit diagonal elements of L are not stored.

5: **pda** – Integer *Input*

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

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

6: **ipiv**[**n**] – Integer *Output*

On exit: if no constraints are violated, the pivot indices that define the permutation matrix P; at the ith step row i of the matrix was interchanged with row $\mathbf{ipiv}[i-1]$. $\mathbf{ipiv}[i-1] = i$ indicates a row interchange was not required. **ipiv** corresponds either to the single precision factorization (if **fail.code** = NE_NOERROR and **iter** ≥ 0) or to the double precision factorization (if **fail.code** = NE_NOERROR and **iter** < 0).

7: **b**[*dim*] – const Complex *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

$\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$ 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.

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

9: **x**[*dim*] – Complex *Output*

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

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

On exit: if **fail.code** = NE_NOERROR, the n by r solution matrix X.

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

11: **iter** – Integer * *Output*

On exit: if $\mathbf{iter} > 0$, iterative refinement has been successfully used and **iter** is the number of iterations carried out.

If $\mathbf{iter} < 0$, iterative refinement has failed for one of the reasons given below and double precision factorization has been carried out instead.

$\mathbf{iter} = -1$
Taking into account machine parameters, and the values of **n** and **nrhs**, it is not worth working in single precision.

$\mathbf{iter} = -2$
Overflow of an entry occurred when moving from double to single precision.

$\mathbf{iter} = -3$
An intermediate single precision factorization failed.

$\mathbf{iter} = -31$
The maximum permitted number of iterations was exceeded.

12: **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: $\mathbf{pdb} > 0$.

On entry, $\mathbf{pdx} = \langle value\rangle$.
Constraint: $\mathbf{pdx} > 0$.

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

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

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

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.

NE_SINGULAR

$U(\langle value\rangle, \langle value\rangle)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

7 Accuracy

The computed solution for a single right-hand side, $\hat{x}$, satisfies the equation of the form

$$(A + E)\hat{x} = b,$$

where

$$\|E\|_1 = O(\epsilon)\|A\|_1$$

and ϵ is the ***machine precision***. An approximate error bound for the computed solution is given by

$$\frac{\|\hat{x} - x\|_1}{\|x\|_1} \leq \kappa(A)\frac{\|E\|_1}{\|A\|_1}$$

where $\kappa(A) = \|A^{-1}\|_1\|A\|_1$, the condition number of A with respect to the solution of the linear equations. See Section 4.4 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_zcgesv (f07aqc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zcgesv (f07aqc) 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 real analogue of this function is nag_dsgesv (f07acc).

10 Example

This example solves the equations

$$Ax = b,$$

where A is the general matrix

$$A = \begin{pmatrix} -1.34 + 2.55i & 0.28 + 3.17i & -6.39 - 2.20i & 0.72 - 0.92i \\ -0.17 - 1.41i & 3.31 - 0.15i & -0.15 + 1.34i & 1.29 + 1.38i \\ -3.29 - 2.39i & -1.91 + 4.42i & -0.14 - 1.35i & 1.72 + 1.35i \\ 2.41 + 0.39i & -0.56 + 1.47i & -0.83 - 0.69i & -1.96 + 0.67i \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 26.26 + 51.78i \\ 6.43 - 8.68i \\ -5.75 + 25.31i \\ 1.16 + 2.57i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_zcgesv (f07aqc) Example Program.
 *
 * Copyright 2009, Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  Integer       exit_status = 0;
  Integer       i, iter, j, n, nrhs, pda, pdb, pdx;

  /* Arrays */
  Integer       *ipiv = 0;
  Complex       *a = 0, *b = 0, *x = 0;
```

```
  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;

  INIT_FAIL(fail);

  printf("nag_zcgesv (f07aqc) Example Program Results\n\n");
  /* Skip heading in data file*/
  scanf("%*[^\n]");
  scanf("%ld%ld%*[^\n]", &n, &nrhs);

  if (n < 0 || nrhs < 0)
    {
      printf("Invalid n or nrhs\n");
      exit_status = 1;
      return exit_status;
    }

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

  if (!(a = NAG_ALLOC(n*n, Complex)) ||
      !(b = NAG_ALLOC(n*nrhs, Complex)) ||
      !(x = NAG_ALLOC(n*nrhs, Complex)) ||
      !(ipiv = NAG_ALLOC(n, 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]");

  /* Solve the equations Ax = b for x using
   * nag_zcgesv (f07aqc)
   * Mixed precision complex system solver
   */
  nag_zcgesv(order, n, nrhs, a, pda, ipiv, b, pdb, x, pdx, &iter, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zcgesv (f07aqc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print solution using nag_gen_complx_mat_print_comp (x04dbc). */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                                nrhs, x, pdx, 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;
    }

  /* Print pivot indices*/
  printf("\n\nPivot indices\n");
  for (i = 0; i < n; i++) printf("%11ld%s", ipiv[i], (i+1)%4?" ":"\n");
  printf("\n");

 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(x);
  NAG_FREE(ipiv);

  return exit_status;
}
#undef B
#undef A
```

10.2 Program Data

```
nag_zcgesv (f07aqc) Example Program Data

   4            1                                       : n,  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)  : matrix A

 (26.26,51.78) ( 6.43,-8.68) (-5.75,25.31) ( 1.16, 2.57)  : vector b
```

10.3 Program Results

```
nag_zcgesv (f07aqc) Example Program Results

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


Pivot indices
          3           2           3           4
```