NAG Library Function Document

nag_zsysv (f07nnc)

1 Purpose

nag_zsysv (f07nnc) computes the solution to a complex system of linear equations

$$AX = B,$$

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

2 Specification

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

```
void nag_zsysv (Nag_OrderType order, Nag_UploType uplo, Integer n,
     Integer nrhs, Complex a[], Integer pda, Integer ipiv[], Complex b[],
     Integer pdb, NagError *fail)
```

3 Description

nag_zsysv (f07nnc) uses the diagonal pivoting method to factor A as

order	**uplo**	***A***
Nag_ColMajor	Nag_Upper	UDU^{T}
Nag_ColMajor	Nag_Lower	LDL^{T}
Nag_RowMajor	Nag_Upper	$U^{\mathrm{T}}DU$
Nag_RowMajor	Nag_Lower	$L^{\mathrm{T}}DL$

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks. The factored form of A is then used to solve the system of equations $AX = B$.

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

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

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

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: if **uplo** $=$ Nag_Upper, the upper triangle of A is stored.

If **uplo** $=$ Nag_Lower, the lower triangle of A is stored.

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

3: **n** – Integer *Input*

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

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

4: **nrhs** – Integer *Input*

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

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

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

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

On entry: the n by n symmetric matrix A.

If **order** $=$ 'Nag_ColMajor', A_{ij} is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

If **order** $=$ 'Nag_RowMajor', A_{ij} is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

If **uplo** $=$ 'Nag_Upper', the upper triangular part of A must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** $=$ 'Nag_Lower', the lower triangular part of A must be stored and the elements of the array above the diagonal are not referenced.

On exit: if **fail.code** $=$ NE_NOERROR, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = UDU^{\mathrm{T}}$, $A = LDL^{\mathrm{T}}$, $A = U^{\mathrm{T}}DU$ or $A = L^{\mathrm{T}}DL$ as computed by nag_zsytrf (f07nrc).

6: **pda** – Integer *Input*

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

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

7: **ipiv**[*dim*] – Integer *Output*

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

On exit: details of the interchanges and the block structure of D. More precisely,

if $\mathbf{ipiv}[i-1] = k > 0$, d_{ii} is a 1 by 1 pivot block and the ith row and column of A were interchanged with the kth row and column;

if **uplo** $=$ Nag_Upper and $\mathbf{ipiv}[i-2] = \mathbf{ipiv}[i-1] = -l < 0$, $\begin{pmatrix} d_{i-1,i-1} & \bar{d}_{i,i-1} \\ \bar{d}_{i,i-1} & d_{ii} \end{pmatrix}$ is a 2 by 2 pivot block and the $(i-1)$th row and column of A were interchanged with the lth row and column;

if **uplo** $=$ Nag_Lower and $\mathbf{ipiv}[i-1] = \mathbf{ipiv}[i] = -m < 0$, $\begin{pmatrix} d_{ii} & d_{i+1,i} \\ d_{i+1,i} & d_{i+1,i+1} \end{pmatrix}$ is a 2 by 2 pivot block and the $(i+1)$th row and column of A were interchanged with the mth row and column.

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

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 ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n** $=$ ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 0$.

On entry, **nrhs** $=$ ⟨*value*⟩.
Constraint: $\mathbf{nrhs} \geq 0$.

On entry, **pda** $=$ ⟨*value*⟩.
Constraint: $\mathbf{pda} > 0$.

On entry, **pdb** $=$ ⟨*value*⟩.
Constraint: $\mathbf{pdb} > 0$.

NE_INT_2

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

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

On entry, **pdb** $=$ ⟨*value*⟩ and **nrhs** $=$ ⟨*value*⟩.
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.

NE_SINGULAR

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

7 Accuracy

The computed solution for a single right-hand side, $\hat{x}$, satisfies an 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 $\kappa(A) = \left\|A^{-1}\right\|_1 \|A\|_1$, the condition number of A with respect to the solution of the linear equations. See Section 4.4 of Anderson *et al.* (1999) and Chapter 11 of Higham (2002) for further details.

nag_zsysvx (f07npc) is a comprehensive LAPACK driver that returns forward and backward error bounds and an estimate of the condition number. Alternatively, nag_complex_sym_lin_solve (f04dhc) solves $Ax = b$ and returns a forward error bound and condition estimate. nag_complex_sym_lin_solve (f04dhc) calls nag_zsysv (f07nnc) to solve the equations.

8 Parallelism and Performance

nag_zsysv (f07nnc) is not threaded by NAG in any implementation.

nag_zsysv (f07nnc) 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 floating-point operations is approximately $\frac{4}{3}n^3 + 8n^2r$, where r is the number of right-hand sides.

The real analogue of this function is nag_dsysv (f07mac). The complex Hermitian analogue of this function is nag_zhesv (f07mnc).

10 Example

This example solves the equations

$$Ax = b,$$

where A is the complex symmetric matrix

$$A = \begin{pmatrix} -0.56+0.12i & -1.54-2.86i & 5.32-1.59i & 3.80+0.92i \\ -1.54-2.86i & -2.83-0.03i & -3.52+0.58i & -7.86-2.96i \\ 5.32-1.59i & -3.52+0.58i & 8.86+1.81i & 5.14-0.64i \\ 3.80+0.92i & -7.86-2.96i & 5.14-0.64i & -0.39-0.71i \end{pmatrix}$$

and

$$b = \begin{pmatrix} -6.43+19.24i \\ -0.49-\ \ 1.47i \\ -48.18+66.00i \\ -55.64+41.22i \end{pmatrix}.$$

Details of the factorization of A are also output.

10.1 Program Text

```
/* nag_zsysv (f07nnc) Example Program.
 *
 * Copyright 2004 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

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

int main(void)
{

  /* Scalars */
  Integer        exit_status = 0, i, j, n, nrhs, pda, pdb;

  /* Arrays */
  Complex        *a = 0, *b = 0;
  Integer        *ipiv = 0;
  char           nag_enum_arg[40];

  /* Nag Types */
  NagError       fail;
  Nag_UploType   uplo;
  Nag_OrderType  order;

#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_zsysv (f07nnc) 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;
      goto END;
    }
  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);

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

  pda = n;
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
#else
  pdb = nrhs;
#endif

  /* Read the triangular part of the matrix A from data file */
  if (uplo == Nag_Upper)
    for (i = 1; i <= n; ++i)
      for (j = i; j <= n; ++j)
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
  else
    for (i = 1; i <= n; ++i)
      for (j = 1; j <= i; ++j)
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
  scanf("%*[^\n]");

  /* Read b from data file */
  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_zsysv (f07nnc). */
  nag_zsysv(order, uplo, n, nrhs, a, pda, ipiv, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zsysv (f07nnc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print solution */
  printf("      Solution\n");
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= nrhs; ++j)
        printf(" (%7.4f, %7.4f)%s", B(i, j).re, B(i, j).im, j%4 == 0?"\n":"");
      printf("\n");
    }

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

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

10.2 Program Data

```
nag_zsysv (f07nnc) Example Program Data
    4              1                                                  : n, nrhs
    Nag_Lower                                                         : uplo
 ( -0.56,  0.12)
 ( -1.54, -2.86) ( -2.83 ,-0.03)
 (  5.32, -1.59) ( -3.52,  0.58) (  8.86,  1.81)
 (  3.80,  0.92) ( -7.86, -2.96) (  5.14, -0.64) ( -0.39 ,-0.71) : matrix A
 ( -6.43, 19.24) ( -0.49, -1.47) (-48.18, 66.00) (-55.64, 41.22) : vector b
```

10.3 Program Results

```
nag_zsysv (f07nnc) Example Program Results

      Solution
 (-4.0000,  3.0000)
 ( 3.0000, -2.0000)
 (-2.0000,  5.0000)
 ( 1.0000, -1.0000)
```