1 Purpose

nag_zsytrs (f07nsc) solves a complex symmetric system of linear equations with multiple right-hand sides,

\[ AX = B, \]

where \( A \) has been factorized by nag_zsytrf (f07nrc).

2 Specification

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

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

3 Description

nag_zsytrs (f07nsc) is used to solve a complex symmetric system of linear equations \( AX = B \), this function must be preceded by a call to nag_zsytrf (f07nrc) which computes the Bunch–Kaufman factorization of \( A \).

If \( \text{uplo} = \text{Nag_Upper} \), \( A = PUDU^T P^T \), where \( P \) is a permutation matrix, \( U \) is an upper triangular matrix and \( D \) is a symmetric block diagonal matrix with 1 by 1 and 2 by 2 blocks; the solution \( X \) is computed by solving \( PUDY = B \) and then \( UT^T PX = Y \).

If \( \text{uplo} = \text{Nag_Lower} \), \( A = PLDL^T P^T \), where \( L \) is a lower triangular matrix; the solution \( X \) is computed by solving \( PLDY = B \) and then \( LT^T PX = Y \).

4 References


5 Arguments

1: \( \text{order} \) – Nag_OrderType \hspace{1cm} \text{Input} \\
   \hspace{1cm} On entry: the \text{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 \text{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument. \\
   \hspace{1cm} Constraint: \text{order} = Nag_RowMajor or Nag_ColMajor.

2: \( \text{uplo} \) – Nag_UploType \hspace{1cm} \text{Input} \\
   \hspace{1cm} On entry: specifies how \( A \) has been factorized. \\
   \hspace{1cm} \text{uplo} = \text{Nag_Upper} \\
   \hspace{1cm} \hspace{1cm} A = PUDU^T P^T, \text{ where } U \text{ is upper triangular.}
\textbf{f07nsc} \hspace{1cm} NAG Library Manual

\textbf{uplo} = \texttt{Nag}\_\texttt{Lower}
\hspace{1cm} \textit{A} = \textit{PLDL}^T \textit{P}^T, \text{ where } \textit{L} \text{ is lower triangular.}

\textit{Constraint: uplo} = \texttt{Nag}\_\texttt{Upper} or \texttt{Nag}\_\texttt{Lower}.

3: \hspace{0.5cm} \textbf{n} \hspace{0.2cm} – \texttt{Integer} \hspace{1cm} \textit{Input}
\hspace{1cm} \textit{On entry: } \textit{n}, the order of the matrix \textit{A}.
\hspace{1cm} \textit{Constraint: } \textbf{n} \geq 0.

4: \hspace{0.5cm} \textbf{nrhs} \hspace{0.2cm} – \texttt{Integer} \hspace{1cm} \textit{Input}
\hspace{1cm} \textit{On entry: } \textit{r}, the number of right-hand sides.
\hspace{1cm} \textit{Constraint: } \textbf{nrhs} \geq 0.

5: \hspace{0.5cm} \textbf{a[\text{\textit{dim}}]} \hspace{0.2cm} – \texttt{const Complex} \hspace{1cm} \textit{Input}
\hspace{1cm} \textit{Note:} the dimension, \textit{\textit{dim}}, of the array \textbf{a} must be at least \textit{max(1, pda \times n)}.
\hspace{1cm} \textit{On entry:} details of the factorization of \textit{A}, as returned by \texttt{nag\_zsytrf (f07nrc)}.

6: \hspace{0.5cm} \textbf{pda} \hspace{0.2cm} – \texttt{Integer} \hspace{1cm} \textit{Input}
\hspace{1cm} \textit{On entry:} the stride separating row or column elements (depending on the value of \textit{\textbf{order}}) of the matrix in the array \textbf{a}.
\hspace{1cm} \textit{Constraint: } \textbf{pda} \geq \textit{max(1, n)}.

7: \hspace{0.5cm} \textbf{ipiv[\text{\textit{dim}}]} \hspace{0.2cm} – \texttt{const Integer} \hspace{1cm} \textit{Input}
\hspace{1cm} \textit{Note:} the dimension, \textit{\textit{dim}}, of the array \textbf{ipiv} must be at least \textit{max(1, n)}.
\hspace{1cm} \textit{On entry:} details of the interchanges and the block structure of \textit{D}, as returned by \texttt{nag\_zsytrf (f07nrc)}.

8: \hspace{0.5cm} \textbf{b[\text{\textit{dim}}]} \hspace{0.2cm} – \texttt{Complex} \hspace{1cm} \textit{Input/Output}
\hspace{1cm} \textit{Note:} the dimension, \textit{\textit{dim}}, of the array \textbf{b} must be at least
\hspace{2cm} \textit{max(1, pdb \times nrhs)} when \textit{\textbf{order}} = \texttt{Nag}\_\texttt{ColMajor};
\hspace{2cm} \textit{max(1, n \times pdb)} when \textit{\textbf{order}} = \texttt{Nag}\_\texttt{RowMajor}.
\hspace{1cm} The \((i, j)\)th element of the matrix \textit{B} is stored in
\hspace{2.5cm} \textbf{b}[(j - 1) \times \text{pdb} + i - 1] when \textit{\textbf{order}} = \texttt{Nag}\_\texttt{ColMajor};
\hspace{2.5cm} \textbf{b}[(i - 1) \times \text{pdb} + j - 1] when \textit{\textbf{order}} = \texttt{Nag}\_\texttt{RowMajor}.
\hspace{1cm} \textit{On entry:} the \textit{n} by \textit{r} right-hand side matrix \textit{B}.
\hspace{1cm} \textit{On exit:} the \textit{n} by \textit{r} solution matrix \textit{X}.

9: \hspace{0.5cm} \textbf{pdb} \hspace{0.2cm} – \texttt{Integer} \hspace{1cm} \textit{Input}
\hspace{1cm} \textit{On entry:} the stride separating row or column elements (depending on the value of \textit{\textbf{order}}) in the array \textbf{b}.
\hspace{1cm} \textit{Constraints:}
\hspace{2cm} if \textit{\textbf{order}} = \texttt{Nag}\_\texttt{ColMajor}, \textbf{pdb} \geq \textit{max(1, n)};
\hspace{2cm} if \textit{\textbf{order}} = \texttt{Nag}\_\texttt{RowMajor}, \textbf{pdb} \geq \textit{max(1, nrhs)}.

10: \hspace{0.5cm} \textbf{fail} \hspace{0.2cm} – \texttt{NagError *} \hspace{1cm} \textit{Input/Output}
\hspace{1cm} The NAG error argument (see Section 3.6 in the Essential Introduction).
6 Error Indicators and Warnings

**NE_ALLOC_FAIL**
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

**NE_BAD_PARAM**
On entry, argument ⟨value⟩ had an illegal value.

**NE_INT**
On entry, n = ⟨value⟩.
Constraint: n ≥ 0.
On entry, nrhs = ⟨value⟩.
Constraint: nrhs ≥ 0.
On entry, pda = ⟨value⟩.
Constraint: pda > 0.
On entry, pdb = ⟨value⟩.
Constraint: pdb > 0.

**NE_INT_2**
On entry, pda = ⟨value⟩ and n = ⟨value⟩.
Constraint: pda ≥ max(1, n).
On entry, pdb = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdb ≥ max(1, n).
On entry, pdb = ⟨value⟩ and nrhs = ⟨value⟩.
Constraint: pdb ≥ max(1, 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.
An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

**NE_NO_LICENCE**
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

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

if uplo = Nag_Upper, |E| ≤ c(n)εP[U]|D||U^T|P^T;  
if uplo = Nag_Lower, |E| ≤ c(n)εP[L]|D||L^T|P^T,

c(n) is a modest linear function of n, and ε is the machine precision.
If the true solution x is the solution of (A + E)x = b, 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 \( \text{cond}(A) \).

Forward and backward error bounds can be computed by calling nag_zsyrfs (f07nvc), and an estimate for \( \kappa_{\infty}(A) = \kappa_1(A) \) can be obtained by calling nag_zsycon (f07nuc).

### 8 Parallelism and Performance

nag_zsytrs (f07nsc) is not threaded by NAG in any implementation.

nag_zsytrs (f07nsc) 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 X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also 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_zsyrfs (f07nvc) to refine the solution and return an error estimate.

The real analogue of this function is nag_dsytrs (f07mec).

### 10 Example

This example solves the system of equations \( AX = B \), where

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

and

\[
B = \begin{pmatrix}
-55.64 + 41.22i & -19.09 - 35.97i \\
-48.18 + 66.00i & -12.08 - 27.02i \\
-0.49 - 1.47i & 6.95 + 20.49i \\
-6.43 + 19.24i & -4.59 - 35.53i
\end{pmatrix}
\]

Here \( A \) is symmetric and must first be factorized by nag_zsytrf (f07nrc).

### 10.1 Program Text

/* nag_zsytrs (f07nsc) Example Program. *
 * Copyright 2014 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;
    Nag_UploType uplo;

    // Rest of the function code...
NagError fail;
Nag_OrderType order;
/* Arrays */
Integer *ipiv = 0;
char nag_enum_arg[40];
Complex *a = 0, *b = 0;

#ifdef NAG_LOAD_FP
    /* The following line is needed to force the Microsoft linker to load floating point support */
    float force_loading_of_ms_float_support = 0;
#endif /* NAG_LOAD_FP */

#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_zsytrs (f07nsc) Example Program Results\n\n");

    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[^n] ");
#else
    scanf("%*[^
] ");
#endif
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[^n] ", &n, &nrhs);
#else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^n] ", &n, &nrhs);
#endif
#ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdb = n;
#else
    pda = n;
    pdb = nrhs;
#endif

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

    /* Read A and B from data file */
#ifdef _WIN32
    scanf_s(" %39s%*[^n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf(" %39s%*[^n] ", nag_enum_arg);
#endif
    /* nag_enum_name_to_value (x04nac).
    * Converts NAG enum member name to value */
    uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
    if (uplo == Nag_Upper)
    {
        for (i = 1; i <= n; ++i)
for (j = i; j <= n; ++j)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
    scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#else _WIN32
    scanf("%*

#endif
#endif
#endif

for (i = 1; i <= n; ++i)
    { for (j = 1; j <= i; ++j)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
    scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#else _WIN32
    scanf("%*
this
#endif
#endif
#endif

for (i = 1; i <= nrhs; ++i)
    { for (j = 1; j <= nrhs; ++j)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#else
    scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
#else _WIN32
    scanf("%*
#endif
#endif
#endif

/* Factorize A */
/* nag_zsytrf (f07nrc).  
 * Bunch-Kaufman factorization of complex symmetric matrix      
 */
#define _WIN32
    scanf_s("%*
#endif
#endif
#endif

/* Compute solution */
/* nag_zsytrs (f07nsc).    
 * Solution of complex symmetric system of linear equations,  
 * multiple right-hand sides, matrix already factorized by     
 * nag_zsytrf (f07nrc) */
#define _WIN32
    scanf("%*
#endif
#endif
#endif

f07nsc.6  NAG Library Manual
Mark 25
/* Print solution */
/* nag_gen_complex_mat_print_comp (x04dbc). */
* Print complex general matrix (comprehensive)
*/
fflush(stdout);
nag_gen_complex_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_complex_mat_print_comp (x04dbc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}
END:
NAG_FREE(ipiv);
NAG_FREE(a);
NAG_FREE(b);
return exit_status;

10.2 Program Data

nag_zsytrs (f07nsc) Example Program Data
4 2 :Values of n and nrhs
  Nag_Lower :Value of uplo
(-0.39,-0.71)
 ( 5.14,-0.71) ( 8.86, 1.81)
(-7.86,-2.96) (-3.52, 0.58) (-2.83,-0.03)
 ( 3.80, 0.92) ( 5.32,-1.59) (-1.54,-2.86) (-0.56, 0.12) :End of matrix A
(-55.64, 41.22) (-19.09,-35.97)
(-48.18, 66.00) (-12.08,-27.02)
 (-0.49, -1.47) ( 6.95, 20.49)
(-6.43, 19.24) (-4.59,-35.53) :End of matrix B

10.3 Program Results

nag_zsytrs (f07nsc) Example Program Results

Solution(s)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>( 1.0000, -1.0000)</td>
</tr>
<tr>
<td>2</td>
<td>(-2.0000,  5.0000)</td>
</tr>
<tr>
<td>3</td>
<td>( 3.0000, -2.0000)</td>
</tr>
<tr>
<td>4</td>
<td>(-4.0000,  3.0000)</td>
</tr>
</tbody>
</table>