NAG Library Function Document

nag_zhetrs (f07msc)

1 Purpose

nag_zhetrs (f07msc) solves a complex Hermitian indefinite system of linear equations with multiple right-hand sides,

\[ AX = B, \]

where \( A \) has been factorized by nag_zhetrf (f07mrc).

2 Specification

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

void nag_zhetrs (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_zhetrs (f07msc) is used to solve a complex Hermitian indefinite system of linear equations \( AX = B \), this function must be preceded by a call to nag_zhetrf (f07mrc) which computes the Bunch–Kaufman factorization of \( A \).

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

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

4 References


5 Arguments

1: \( \text{order} \) – Nag\_OrderType

\( \text{Input} \)

\( \text{On entry:} \) 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} = \text{Nag\_RowMajor}. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\( \text{Constraint:} \) \text{order} = \text{Nag\_RowMajor} or \text{Nag\_ColMajor}.  

2: \( \text{uplo} \) – Nag\_UploType

\( \text{Input} \)

\( \text{On entry:} \) specifies how \( A \) has been factorized.

\( \text{uplo} = \text{Nag\_Upper} \)

\( A = PUDU^H P^T, \) where \( U \) is upper triangular.
\texttt{uplo} = \texttt{Nag\_Lower}

\[ A = PLDL^\top PT, \] where \( L \) is lower triangular.

\textit{Constraint:} \( \texttt{uplo} = \texttt{Nag\_Upper} \) or \( \texttt{Nag\_Lower} \).

3: \( n \) – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \( n \), the order of the matrix \( A \).

\textit{Constraint:} \( n \geq 0 \).

4: \( \texttt{nrhs} \) – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \( r \), the number of right-hand sides.

\textit{Constraint:} \( \texttt{nrhs} \geq 0 \).

5: \( \texttt{a[dim]} \) – const Complex \hspace{1cm} \textit{Input}

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

\textit{On entry:} details of the factorization of \( A \), as returned by \texttt{nag\_zhetrf} (f07mrc).

6: \( \texttt{pda} \) – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) of the matrix in the array \( \texttt{a} \).

\textit{Constraint:} \( \texttt{pda} \geq \max(1, \texttt{n}) \).

7: \( \texttt{ipiv[dim]} \) – const Integer \hspace{1cm} \textit{Input}

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

\textit{On entry:} details of the interchanges and the block structure of \( D \), as returned by \texttt{nag\_zhetrf} (f07mrc).

8: \( \texttt{b[dim]} \) – Complex \hspace{1cm} \textit{Input/Output}

\textit{Note:} the dimension, \( \texttt{dim} \), of the array \( \texttt{b} \) must be at least

\[ \max(1, \texttt{pdb} \times \texttt{nrhs}) \] when \( \texttt{order} = \texttt{Nag\_ColMajor} \);

\[ \max(1, \texttt{n} \times \texttt{pdb}) \] when \( \texttt{order} = \texttt{Nag\_RowMajor} \).

The \( (i,j) \)th element of the matrix \( B \) is stored in

\[ b[(j-1) \times \texttt{pdb} + i - 1] \] when \( \texttt{order} = \texttt{Nag\_ColMajor} \);

\[ b[(i-1) \times \texttt{pdb} + j - 1] \] when \( \texttt{order} = \texttt{Nag\_RowMajor} \).

\textit{On entry:} the \( n \) by \( r \) right-hand side matrix \( B \).

\textit{On exit:} the \( n \) by \( r \) solution matrix \( X \).

9: \( \texttt{pdb} \) – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) in the array \( \texttt{b} \).

\textit{Constraints:}

if \( \texttt{order} = \texttt{Nag\_ColMajor} \), \( \texttt{pdb} \geq \max(1, \texttt{n}) \); 

if \( \texttt{order} = \texttt{Nag\_RowMajor} \), \( \texttt{pdb} \geq \max(1, \texttt{nrhs}) \).

10: \( \texttt{fail} \) – \texttt{NagError} * \hspace{1cm} \textit{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.
See Section 3.2.1.2 in the Essential Introduction for further information.

**NE_BAD_PARAM**
On entry, argument \langle value \rangle had an illegal value.

**NE_INT**
On entry, \( n = \langle value \rangle \).
Constraint: \( n \geq 0 \).
On entry, \( \text{nrhs} = \langle value \rangle \).
Constraint: \( \text{nrhs} \geq 0 \).
On entry, \( \text{pda} = \langle value \rangle \).
Constraint: \( \text{pda} > 0 \).
On entry, \( \text{pdb} = \langle value \rangle \).
Constraint: \( \text{pdb} > 0 \).

**NE_INT_2**
On entry, \( \text{pda} = \langle value \rangle \) and \( n = \langle value \rangle \).
Constraint: \( \text{pda} \geq \max (1, n) \).
On entry, \( \text{pdb} = \langle value \rangle \) and \( n = \langle value \rangle \).
Constraint: \( \text{pdb} \geq \max (1, n) \).
On entry, \( \text{pdb} = \langle value \rangle \) and \( \text{nrhs} = \langle value \rangle \).
Constraint: \( \text{pdb} \geq \max (1, \text{nrhs}) \).

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

if \( \text{uplo} = \text{Nag_Upper} \), \( |E| \leq c(n)\epsilon \|U\|D\|U^H\|P^T \);

if \( \text{uplo} = \text{Nag_Lower} \), \( |E| \leq c(n)\epsilon \|L\|D\|L^H\|P^T \),

\( c(n) \) is a modest linear function of \( n \), and \( \epsilon \) is the \textbf{machine precision}.

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

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

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 \texttt{nag_zherfs} (f07mvc), and an estimate for $\kappa_\infty(A)$ ( = $\kappa_1(A)$) can be obtained by calling \texttt{nag_zhecon} (f07muc).

8 Parallelism and Performance

\texttt{nag_zhetrs} (f07msc) is not threaded by NAG in any implementation.

\texttt{nag_zhetrs} (f07msc) 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 \texttt{nag_zherfs} (f07mvc) to refine the solution and return an error estimate.

The real analogue of this function is \texttt{nag_dsytrs} (f07mec).

10 Example

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

$$A = \begin{pmatrix} -1.36 + 0.00i & 1.58 + 0.90i & 2.21 - 0.21i & 3.91 + 1.50i \\ 1.58 - 0.90i & -8.87 + 0.00i & -1.84 - 0.03i & -1.78 + 1.18i \\ 2.21 + 0.21i & -1.84 + 0.03i & -4.63 + 0.00i & 0.11 + 0.11i \\ 3.91 - 1.50i & -1.78 - 1.18i & 0.11 - 0.11i & -1.84 + 0.00i \end{pmatrix}$$

and

$$B = \begin{pmatrix} 7.79 + 5.48i & -35.39 + 18.01i \\ -0.77 - 16.05i & 4.23 - 70.02i \\ -9.58 + 3.88i & -24.79 - 8.40i \\ 2.98 - 10.18i & 28.68 - 39.89i \end{pmatrix}.$$  

Here $A$ is Hermitian indefinite and must first be factorized by \texttt{nag_zhetrf} (f07mrc).

10.1 Program Text

/* \texttt{nag_zhetrs} (f07msc) 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;
NagError fail;
Nag_OrderType order;
/* Arrays */
Integer *ipiv = 0;
char nag_enum_arg[40];
Complex *a = 0, *b = 0;

#ifndef 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 */
#ifndef 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_zhetrs (f07msc) Example Program Results\n\n");
/* Skip heading in data file */
#ifndef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
#ifndef _WIN32
    scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[\n] ", &n, &nrhs);
#else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n] ", &n, &nrhs);
#endif
#ifndef 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 */
#ifndef _WIN32
    scanf_s( "%39s%*[\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf ( "%39s%*[\n] ", nag_enum_arg);
#endif
#ifdef nag_enum_name_to_value (x04nac).
    * Converts NAG enum member name to value */
    uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
#endif
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
    #ifdef _WIN32
    scanf_s("%*[\n ] ");
    #else
    scanf("%*[\n ] ");
    #endif
    }
else
{
    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
        #ifdef _WIN32
        scanf_s("%*[\n ] ");
        #else
        scanf("%*[\n ] ");
        #endif
    }
    for (i = 1; i <= n; ++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
        #ifdef _WIN32
        scanf_s("%*[\n ] ");
        #else
        scanf("%*[\n ] ");
        #endif
    }
    /* Factorize A */
    /* nag_zhetrf (f07mrc). 
       * Bunch-Kaufman factorization of complex Hermitian 
       * indefinite matrix 
       */
    nag_zhetrf(order, uplo, n, A, pda, ipiv, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zhetrf (f07mrc).
        fail.message\n        exit_status = 1;
        goto END;
    }
    /* Compute solution */
    /* nag_zhetrs (f07msc). 
       * Solution of complex Hermitian indefinite system of linear 
       * equations, multiple right-hand sides, matrix already 
       * factorized by nag_zhetrf (f07mrc) 
       */
    nag_zhetrs(order, uplo, n, nrhs, A, pda, ipiv, B, pdb, 
              &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zhetrs (f07msc).
        fail.message\n        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(ipiv);
NAG_FREE(a);
NAG_FREE(b);
return exit_status;


10.2 Program Data

nag_zhetrs (f07msc) Example Program Data

4 2

:Values of n and nrhs
Nag_Lower :Value of uplo
(-1.36, 0.00)
(-1.58, -0.90) (-8.87, 0.00)
( 2.21, 0.21) (-1.84, 0.03) (-4.63, 0.00)
( 3.91, -1.50) (-1.78, -1.18) ( 0.11, -0.11) (-1.84, 0.00) :End of matrix A
( 7.79, 5.48) (-35.39, 18.01)
(-0.77, -16.05) ( 4.23, -70.02)
(-9.58, 3.88) (-24.79, -8.40)
( 2.98, -10.18) ( 28.68, -39.89) :End of matrix B

10.3 Program Results

nag_zhetrs (f07msc) Example Program Results

Solution(s)

1 2
1 ( 1.0000, -1.0000) ( 3.0000, -4.0000)
2 (-1.0000, 2.0000) (-1.0000, 5.0000)
3 ( 3.0000, -2.0000) ( 7.0000, -2.0000)
4 ( 2.0000, 1.0000) (-8.0000, 6.0000)