NAG Library Function Document

nag_ztfsm (f16zlc)

1 Purpose

nag_ztfsm (f16zlc) performs one of the matrix-matrix operations

\[ B \leftarrow \alpha A^{-1} B, \quad B \leftarrow \alpha A^{-H} B, \]
\[ B \leftarrow \alpha BA^{-1} \quad \text{or} \quad B \leftarrow \alpha BA^{-H}, \]

where \( A \) is a complex triangular matrix stored in Rectangular Full Packed (RFP) format, \( B \) is an \( m \) by \( n \) complex matrix, and \( \alpha \) is a complex scalar. \( A^{-H} \) denotes \((A^H)^{-1}\) or equivalently \((A^{-1})^H\).

No test for singularity or near-singularity of \( A \) is included in this function. Such tests must be performed before calling this function.

2 Specification

```c
#include <nag.h>
#include <nagf16.h>

void nag_ztfsm (Nag_OrderType order, Nag_RFP_Store transr,
               Nag_SideType side, Nag_UploType uplo, Nag_TransType trans,
               Nag_DiagType diag, Integer m, Integer n, Complex alpha,
               const Complex ar[], Complex b[], Integer pdb, NagError *fail)
```

3 Description

nag_ztfsm (f16zlc) solves (for \( X \)) a triangular linear system of one of the forms

\[ AX = \alpha B, \quad A^H X = \alpha B, \]
\[XA = \alpha B \quad \text{or} \quad XA^H = \alpha B,\]

where \( A \) is a complex triangular matrix stored in RFP format, \( B, X \) are \( m \) by \( n \) complex matrices, and \( \alpha \) is a complex scalar. The RFP storage format is described in Section 3.3.3 in the f07 Chapter Introduction.

4 References


5 Arguments

1: \textbf{order} \textendash\text{Nag<OrderType>}
\textit{Input}

\textit{On entry:} the \textbf{order} argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. \textit{C} language defined storage is specified by \textbf{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint:} \textbf{order} = Nag_RowMajor or Nag_ColMajor.
2:  \textbf{transr} – \texttt{Nag\_RFP\_Store} \hspace{1cm} \textit{Input}

\textit{On entry:} specifies whether the normal RFP representation of \( A \) or its conjugate transpose is stored.

\textbf{transr} = \texttt{Nag\_RFP\_Normal}

The matrix \( A \) is stored in normal RFP format.

\textbf{transr} = \texttt{Nag\_RFP\_ConjTrans}

The conjugate transpose of the RFP representation of the matrix \( A \) is stored.

\textit{Constraint:} \textbf{transr} = \texttt{Nag\_RFP\_Normal} or \texttt{Nag\_RFP\_ConjTrans}.

3:  \textbf{side} – \texttt{Nag\_SideType} \hspace{1cm} \textit{Input}

\textit{On entry:} specifies whether \( B \) is operated on from the left or the right, or similarly whether \( A \) (or its transpose) appears to the left or right of the solution matrix in the linear system to be solved.

\textbf{side} = \texttt{Nag\_LeftSide}

\( B \) is pre-multiplied from the left. The system to be solved has the form \( AX = \alpha B \) or \( A^TX = \alpha B \).

\textbf{side} = \texttt{Nag\_RightSide}

\( B \) is post-multiplied from the right. The system to be solved has the form \(XA = \alpha B \) or \(XA^T = \alpha B \).

\textit{Constraint:} \textbf{side} = \texttt{Nag\_LeftSide} or \texttt{Nag\_RightSide}.

4:  \textbf{uplo} – \texttt{Nag\_UploType} \hspace{1cm} \textit{Input}

\textit{On entry:} specifies whether \( A \) is upper or lower triangular.

\textbf{uplo} = \texttt{Nag\_Upper}

\( A \) is upper triangular.

\textbf{uplo} = \texttt{Nag\_Lower}

\( A \) is lower triangular.

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

5:  \textbf{trans} – \texttt{Nag\_TransType} \hspace{1cm} \textit{Input}

\textit{On entry:} specifies whether the operation involves \( A^{-1} \) or \( A^{-H} \), i.e., whether or not \( A \) is transpose conjugated in the linear system to be solved.

\textbf{trans} = \texttt{Nag\_NoTrans}

The operation involves \( A^{-1} \), i.e., \( A \) is not transpose conjugated.

\textbf{trans} = \texttt{Nag\_ConjTrans}

The operation involves \( A^{-H} \), i.e., \( A \) is transpose conjugated.

\textit{Constraint:} \textbf{trans} = \texttt{Nag\_NoTrans} or \texttt{Nag\_ConjTrans}.

6:  \textbf{diag} – \texttt{Nag\_DiagType} \hspace{1cm} \textit{Input}

\textit{On entry:} specifies whether \( A \) has nonunit or unit diagonal elements.

\textbf{diag} = \texttt{Nag\_NonUnitDiag}

The diagonal elements of \( A \) are stored explicitly.

\textbf{diag} = \texttt{Nag\_UnitDiag}

The diagonal elements of \( A \) are assumed to be 1, the corresponding elements of \texttt{ar} are not referenced.

\textit{Constraint:} \textbf{diag} = \texttt{Nag\_NonUnitDiag} or \texttt{Nag\_UnitDiag}.
7: $m$ – Integer
   
   On entry: $m$, the number of rows of the matrix $B$.
   
   Constraint: $m \geq 0$.

8: $n$ – Integer
   
   On entry: $n$, the number of columns of the matrix $B$.
   
   Constraint: $n \geq 0$.

9: $\alpha$ – Complex
   
   On entry: the scalar $\alpha$.

10: $ar[dim]$ – const Complex
   
   Note: the dimension, $dim$, of the array $ar$ must be at least
   
   $\max(1, m \times (m + 1)/2)$ when $\text{side} = \text{Nag LeftSide}$;
   $\max(1, n \times (n + 1)/2)$ when $\text{side} = \text{Nag RightSide}$.
   
   On entry: the $m$ by $m$ triangular matrix $A$ if $\text{side} = \text{Nag LeftSide}$ or the $n$ by $n$ triangular matrix $A$ if $\text{side} = \text{Nag RightSide}$, stored in RFP format (as specified by $\text{transr}$). The storage format is described in detail in Section 3.3.3 in the f07 Chapter Introduction. If $\alpha = 0.0$, $ar$ is not referenced.

11: $b[dim]$ – Complex
   
   Note: the dimension, $dim$, of the array $b$ must be at least
   
   $\max(1, pdb \times n)$ when $\text{order} = \text{Nag ColMajor}$;
   $\max(1, m \times pdb)$ when $\text{order} = \text{Nag RowMajor}$.
   
   On entry: the $m$ by $n$ matrix $B$.
   
   If $\alpha = 0$, $b$ need not be set.
   
   On exit: the updated matrix $B$, or similarly the solution matrix $X$.
   
   If $\text{order} = \text{Nag ColMajor}$, $B_{ij}$ is stored in $b[(j - 1) \times pdb + i - 1]$.
   
   If $\text{order} = \text{Nag RowMajor}$, $B_{ij}$ is stored in $b[(i - 1) \times pdb + j - 1]$.

12: $pdb$ – Integer
   
   On entry: the stride separating row or column elements (depending on the value of $\text{order}$) in the array $b$.
   
   Constraints:
   
   if $\text{order} = \text{Nag ColMajor}$, $pdb \geq \max(1, m)$;
   if $\text{order} = \text{Nag RowMajor}$, $pdb \geq \max(1, n)$.

13: $\text{fail}$ – NagError*
   
   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, m = ⟨value⟩.
Constraint: m ≥ 0.
On entry, n = ⟨value⟩.
Constraint: n ≥ 0.

NE_INT_2
On entry, pdb = ⟨value⟩, m = ⟨value⟩.
Constraint: pdb ≥ max(1, m).
On entry, pdb = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdb ≥ max(1, n).

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
Not applicable.

8 Parallelism and Performance
nag_ztfsm (f16zlc) is not threaded by NAG in any implementation.
nag_ztfsm (f16zlc) 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
None.

10 Example
This example reads in the upper triangular part of a symmetric matrix A which it converts to RFP
format. It also reads in α and a 4 by 3 matrix B and then performs the matrix-matrix operation
B ← αA⁻¹B.
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf01.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer exit_status = 0;
    Complex alpha;
    Integer i, j, m, n, pda, pdb;
    /* Arrays */
    Complex *a = 0, *ar = 0, *b = 0;
    char nag_enum_arg[40];
    /* Nag Types */
    Nag_OrderType order;
    Nag_RFP_Store transr;
    Nag_SideType side;
    Nag_UploType uplo;
    Nag_TransType trans;
    NagError fail;

    INIT_FAIL(fail);
    printf("nag_ztfsm (f16zlc) Example Program Results\n");
    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n ]");
    #else
    scanf("%*[\n ]");
    #endif
    #ifdef _WIN32
    scanf_s("%"NAG_IFMT "%"NAG_IFMT "%*[\n ]", &m, &n);
    #else
    scanf("%"NAG_IFMT "%"NAG_IFMT "%*[\n ]", &m, &n);
    #endif
    pda = m;
    #ifdef NAG_COLUMN_MAJOR
    order = Nag_ColMajor;
    pdb = m;
    #define A(I, J) a[(J-1)*pda + I-1]
    #define B(I, J) b[(J-1)*pdb + I-1]
    #else
    order = Nag_RowMajor;
    pdb = n;
    #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(pda*m, Complex)) ||
        !(ar = NAG_ALLOC((m * (m + 1))/2, Complex)) ||
        !(b = NAG_ALLOC(m*n, Complex)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Nag_RFP_Store */
    #ifdef _WIN32

scanf("%39s ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s ", nag_enum_arg);
#endif

transr = (Nag_RFP_Store) nag_enum_name_to_value (nag_enum_arg);
/* Nag_SideType */
#ifdef _WIN32
  scanf_s("%39s %*[\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s %*[\n] ", nag_enum_arg);
#endif
side = (Nag_SideType) nag_enum_name_to_value (nag_enum_arg);
/* Nag_UploType */
#ifdef _WIN32
  scanf_s("%39s ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s ", nag_enum_arg);
#endif
uplo = (Nag_UploType) nag_enum_name_to_value (nag_enum_arg);
/* Nag_TransType */
#ifdef _WIN32
  scanf_s("%39s %*[\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s %*[\n] ", nag_enum_arg);
#endif
trans = (Nag_TransType) nag_enum_name_to_value (nag_enum_arg);
#ifdef _WIN32
  scanf_s(" ( %lf , %lf ) %*[\n] ", &alpha.re, &alpha.im);
#else
  scanf(" ( %lf , %lf ) %*[\n] ", &alpha.re, &alpha.im);
#endif
/* Read upper or lower triangle of matrix A from data file */
if (uplo == Nag_Lower) {
  for (i = 1; i <= m; 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 {
  for (i = 1; i <= m; i++) {
    for (j = i; j <= m; 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
/* Read matrix B from data file */
for (i = 1; i <= m; i++) {
  for (j = 1; j <= n; 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
  }
} /* Convert complex triangular matrix A from full to rectangular full packed
* storage format (stored in ar) using nag_ztrttf (f01vfc).
* /
  nag_ztrttf(order, transr, uplo, m, a, pda, ar, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ztrttf (f01vfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  printf("\n");
  /* Solve AX = B, where complex triangular matrix A is stored using RFP format
  * in ar, using nag_ztfsm (f16zlc).
  */
  nag_ztfsm(order, transr, side, uplo, trans, Nag_NonUnitDiag, m, n, alpha, ar,
      b, pdb, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ztfsm (f16zlc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print the result using easy-to-use complex general matrix printing routine
  * nag_gen_complx_mat_print (x04dac).
  */
  nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, n, b,
      pdb, "The Solution", 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print (x04dac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

END:
NAG_FREE(a);
NAG_FREE(ar);
NAG_FREE(b);
return exit_status;
}

10.2 Program Data

nag_ztfsm (f16zlc) Example Program Data

<table>
<thead>
<tr>
<th>4 3</th>
<th>: m, n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_RFP_Normal Nag_LeftSide</td>
<td>: transr, side</td>
</tr>
<tr>
<td>Nag_Upper Nag_NoTrans</td>
<td>: uplo, trans</td>
</tr>
<tr>
<td>(4.21,1.28)</td>
<td>: alpha</td>
</tr>
<tr>
<td>(1.1,1.1)</td>
<td>(1.2,1.2)</td>
</tr>
<tr>
<td>(2.2,2.2)</td>
<td>(2.3,2.3)</td>
</tr>
<tr>
<td>(3.3,3.3)</td>
<td>(3.4,3.4)</td>
</tr>
<tr>
<td>(4.4,4.4)</td>
<td>: matrix A</td>
</tr>
<tr>
<td>( 1.80,0.59)</td>
<td>( 2.88, 1.23)</td>
</tr>
<tr>
<td>( 5.25,0.12)</td>
<td>( 1.76,−2.95)</td>
</tr>
<tr>
<td>( 1.58,2.01)</td>
<td>(−2.69, 3.18)</td>
</tr>
<tr>
<td>(−1.11,1.11)</td>
<td>(−0.66, 1.66)</td>
</tr>
</tbody>
</table>

10.3 Program Results

nag_ztfsm (f16zlc) Example Program Results

The Solution

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>−2.0339</td>
<td>8.6009</td>
</tr>
<tr>
<td></td>
<td>2.6429</td>
<td>4.3188</td>
</tr>
<tr>
<td>2</td>
<td>4.3280</td>
<td>1.0930</td>
</tr>
<tr>
<td></td>
<td>−4.3756</td>
<td>−8.8840</td>
</tr>
<tr>
<td>3</td>
<td>2.5393</td>
<td>−0.9711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>4</td>
<td>-0.1237</td>
<td>2.5460</td>
</tr>
<tr>
<td>4</td>
<td>-0.3229</td>
<td>0.1410</td>
</tr>
<tr>
<td>4</td>
<td>1.0621</td>
<td>1.2554</td>
</tr>
</tbody>
</table>