# NAG Library Function Document

# nag_ztfsm (f16zlc)

## 1 Purpose

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

$$\begin{array}{lcl} B \leftarrow \alpha A^{-1}B, & & B \leftarrow \alpha A^{-\mathrm{H}}B, \\ B \leftarrow \alpha BA^{-1} & \text{or} & B \leftarrow \alpha BA^{-\mathrm{H}}, \end{array}$$

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. $A^{-\mathrm{H}}$ denotes $(A^{\mathrm{H}})^{-1}$ or equivalently $(A^{-1})^{\mathrm{H}}$.

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

```
#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

$$\begin{array}{lcl} AX = \alpha B, & & A^{\mathrm{H}}X = \alpha B, \\ XA = \alpha B & \text{or} & XA^{\mathrm{H}} = \alpha B, \end{array}$$

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

Gustavson F G, Waśniewski J, Dongarra J J and Langou J (2010) Rectangular full packed format for Cholesky's algorithm: factorization, solution, and inversion *ACM Trans. Math. Software* **37, 2**

## 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: **transr** – Nag_RFP_Store *Input*

*On entry*: specifies whether the normal RFP representation of $A$ or its conjugate transpose is stored.

**transr** = Nag_RFP_Normal
The matrix $A$ is stored in normal RFP format.

**transr** = Nag_RFP_ConjTrans
The conjugate transpose of the RFP representation of the matrix $A$ is stored.

*Constraint*: **transr** = Nag_RFP_Normal or Nag_RFP_ConjTrans.

3: **side** – Nag_SideType *Input*

*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.

**side** = Nag_LeftSide
$B$ is pre-multiplied from the left. The system to be solved has the form $AX = \alpha B$ or $A^{\mathrm{T}}X = \alpha B$.

**side** = Nag_RightSide
$B$ is post-multiplied from the right. The system to be solved has the form $XA = \alpha B$ or $XA^{\mathrm{T}} = \alpha B$.

*Constraint*: **side** = Nag_LeftSide or Nag_RightSide.

4: **uplo** – Nag_UploType *Input*

*On entry*: specifies whether $A$ is upper or lower triangular.

**uplo** = Nag_Upper
$A$ is upper triangular.

**uplo** = Nag_Lower
$A$ is lower triangular.

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

5: **trans** – Nag_TransType *Input*

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

**trans** = Nag_NoTrans
The operation involves $A^{-1}$, i.e., $A$ is not transpose conjugated.

**trans** = Nag_ConjTrans
The operation involves $A^{-\mathrm{H}}$, i.e., $A$ is transpose conjugated.

*Constraint*: **trans** = Nag_NoTrans or Nag_ConjTrans.

6: **diag** – Nag_DiagType *Input*

*On entry*: specifies whether $A$ has nonunit or unit diagonal elements.

**diag** = Nag_NonUnitDiag
The diagonal elements of $A$ are stored explicitly.

**diag** = Nag_UnitDiag
The diagonal elements of $A$ are assumed to be 1, the corresponding elements of **ar** are not referenced.

*Constraint*: **diag** = Nag_NonUnitDiag or Nag_UnitDiag.

7: **m** – Integer *Input*

*On entry*: $m$, the number of rows of the matrix $B$.

*Constraint*: $\mathbf{m} \geq 0$.

8: **n** – Integer *Input*

*On entry*: $n$, the number of columns of the matrix $B$.

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

9: **alpha** – Complex *Input*

*On entry*: the scalar $\alpha$.

10: **ar**[*dim*] – const Complex *Input*

**Note**: the dimension, *dim*, of the array **ar** must be at least

$\max(1, \mathbf{m} \times (\mathbf{m} + 1)/2)$ when **side** = Nag_LeftSide;
$\max(1, \mathbf{n} \times (\mathbf{n} + 1)/2)$ when **side** = Nag_RightSide.

*On entry*: the $m$ by $m$ triangular matrix $A$ if **side** = Nag_LeftSide or the $n$ by $n$ triangular matrix $A$ if **side** = Nag_RightSide, stored in RFP format (as specified by **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 *Input/Output*

**Note**: the dimension, *dim*, of the array **b** must be at least

$\max(1, \mathbf{pdb} \times \mathbf{n})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pdb})$ when **order** = 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 **order** = Nag_ColMajor, $B_{ij}$ is stored in $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$.

If **order** = Nag_RowMajor, $B_{ij}$ is stored in $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$.

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

13: **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.
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, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 0$.

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

**NE_INT_2**

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

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

**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 $\alpha$ and a 4 by 3 matrix $B$ and then performs the matrix-matrix operation $B \leftarrow \alpha A^{-1} B$.

### 10.1 Program Text

```
/* nag_ztfsm (f16zlc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 25, 2014.
 */

#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_s("%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;
  }

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

## 10.2 Program Data

```
nag_ztfsm (f16zlc) Example Program Data
 4 3                                      : m, n
 Nag_RFP_Normal Nag_LeftSide              : transr, side
 Nag_Upper      Nag_NoTrans               : uplo, trans
 (4.21,1.28)                              : alpha
 (1.1,1.1) (1.2,1.2) (1.3,1.3) (1.4,1.4)
 (2.2,2.2) (2.3,2.3) (2.4,2.4)
 (3.3,3.3) (3.4,3.4)
 (4.4,4.4)                                : matrix A
 ( 1.80,0.59) ( 2.88, 1.23) (2.05, 0.78)
 ( 5.25,0.12) ( 1.76,-2.95) (2.20,-0.95)
 ( 1.58,2.01) (-2.69, 3.18) (0.11,-2.90)
 (-1.11,1.11) (-0.66, 1.66) (1.59,-0.59) : matrix B
```

## 10.3 Program Results

```
nag_ztfsm (f16zlc) Example Program Results

 The Solution
            1           2           3
 1     -2.0339      8.6009      3.8676
        2.6429      4.3188      2.2452

 2      4.3280      1.0930      3.3517
       -4.3756     -8.8840     -0.0650

 3      2.5393     -0.9711     -2.0155
```

```
       -0.1237      2.5460     -1.5364

 4     -0.3229      0.1410      0.7955
        1.0621      1.2554     -0.8975
```