

NAG Library Function Document

nag_ztr_load (f16tgc)

1 Purpose

nag_ztr_load (f16tgc) initializes a complex triangular matrix.

2 Specification

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

void nag_ztr_load (Nag_OrderType order, Nag_UploType uplo, Integer n,
                  Complex alpha, Complex diag, Complex a[], Integer pda, NagError *fail)
```

3 Description

nag_ztr_load (f16tgc) forms the complex n by n triangular matrix A given by

$$a_{ij} = \begin{cases} d & \text{if } i = j \\ \alpha & \text{if } i \neq j \end{cases}.$$

4 References

Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001) *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard* University of Tennessee, Knoxville, Tennessee <http://www.netlib.org/blas/blast-forum/blas-report.pdf>

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 2.3.1.3 in How to Use the NAG Library and its Documentation 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: specifies whether the upper or lower triangular part of A is stored.
uplo = Nag_Upper
 The upper triangular part of A is stored.
uplo = Nag_Lower
 The lower triangular part of A is stored.
Constraint: **uplo** = Nag_Upper or Nag_Lower.
- 3: **n** – Integer *Input*
On entry: n , the order of the matrix A .
Constraint: $n \geq 0$.

- 4: **alpha** – Complex *Input*
On entry: the value, α , to be assigned to the off-diagonal elements of A .
- 5: **diag** – Complex *Input*
On entry: the value, d , to be assigned to the diagonal elements of A .
- 6: **a**[*dim*] – Complex *Output*
Note: the dimension, *dim*, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$.
On exit: the n by n triangular matrix A with diagonal elements set to **diag** and strictly upper or lower elements set to **alpha**.
 If **order** = Nag_ColMajor, A_{ij} is stored in **a**[($j - 1$) \times **pda** + $i - 1$].
 If **order** = Nag_RowMajor, A_{ij} is stored in **a**[($i - 1$) \times **pda** + $j - 1$].
 If **uplo** = Nag_Upper, A is upper triangular and the elements of the array corresponding to the lower triangular part of A are not referenced.
 If **uplo** = Nag_Lower, A is lower triangular and the elements of the array corresponding to the upper triangular part of A are not referenced.
- 7: **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: **pda** \geq $\max(1, \mathbf{n})$.
- 8: **fail** – NagError * *Input/Output*
 The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

See Section 2.3.1.2 in How to Use the NAG Library and its Documentation 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.

NE_INT_2

On entry, **pda** = $\langle value \rangle$, **n** = $\langle value \rangle$.

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

NE_INTERNAL_ERROR

An unexpected error has been triggered by this function. Please contact NAG.

See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

The BLAS standard requires accurate implementations which avoid unnecessary over/underflow (see Section 2.7 of Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001)).

8 Parallelism and Performance

nag_ztr_load (f16tgc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example initializes a 4 by 4 lower triangular matrix A , setting diagonal elements to $9.0 + 0.0i$ and strictly lower elements to $0.5 - 0.3i$.

10.1 Program Text

```

/* nag_ztr_load (f16tgc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Complex alpha, diag;
    Integer exit_status, n, pda;

    /* Arrays */
    Complex *a = 0;
    char nag_enum_arg[40];

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

#ifdef NAG_COLUMN_MAJOR
    order = Nag_ColMajor;
#else
    order = Nag_RowMajor;
#endif

    exit_status = 0;
    INIT_FAIL(fail);

```

```

printf("nag_ztr_load (f16tgc) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif

/* Read the problem dimension */
#ifdef _WIN32
scanf_s("%" NAG_IFMT "%*[\n] ", &n);
#else
scanf("%" NAG_IFMT "%*[\n] ", &n);
#endif

/* Read the uplo parameter */
#ifdef _WIN32
scanf_s("%39s%*[\n] ", nag_enum_arg, (unsigned)_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);

/* Read scalar parameters */
#ifdef _WIN32
scanf_s(" ( %lf , %lf ) ( %lf , %lf )%*[\n] ",
        &alpha.re, &alpha.im, &diag.re, &diag.im);
#else
scanf(" ( %lf , %lf ) ( %lf , %lf )%*[\n] ",
        &alpha.re, &alpha.im, &diag.re, &diag.im);
#endif

pda = n;

if (n > 0) {
/* Allocate memory */
if (!(a = NAG_ALLOC(n * n, Complex)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}
}
else {
printf("Invalid n\n");
exit_status = 1;
return exit_status;
}

/* nag_ztr_load (f16tgc).
 * Initialize complex triangular matrix.
 */
nag_ztr_load(order, uplo, n, alpha, diag, a, pda, &fail);
if (fail.code != NE_NOERROR) {
printf("Error from nag_ztr_load.\n%s\n", fail.message);
exit_status = 1;
goto END;
}

if (uplo == Nag_Upper) {
matrix = Nag_UpperMatrix;
}
else {
matrix = Nag_LowerMatrix;
}

```

```

/* Print generated matrix A */
/* nag_gen_complx_mat_print_comp (x04dbc).
 * Print complex general matrix (comprehensive)
 */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, matrix, Nag_NonUnitDiag, n, n, a, pda,
                              Nag_BracketForm, "%5.2f",
                              "Generated Matrix A", 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(a);

    return exit_status;
}

```

10.2 Program Data

```

nag_ztr_load (f16tgc) Example Program Data
  4           : n the dimension of matrix A
  Nag_Lower   : uplo
  ( 0.5,-0.3) ( 9.0, 0.0) : alpha, diag

```

10.3 Program Results

```

nag_ztr_load (f16tgc) Example Program Results

```

```

Generated Matrix A
      1           2           3           4
1 ( 9.00, 0.00)
2 ( 0.50,-0.30) ( 9.00, 0.00)
3 ( 0.50,-0.30) ( 0.50,-0.30) ( 9.00, 0.00)
4 ( 0.50,-0.30) ( 0.50,-0.30) ( 0.50,-0.30) ( 9.00, 0.00)

```
