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 $n \times n$ 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/blas-report.pdf

5 Arguments
1: order – Nag_OrderType
   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: uplo – Nag_UploType
   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
   On entry: $n$, the order of the matrix $A$.
   Constraint: $n \geq 0$. 
**alpha** – Complex $\quad$ Input

On entry: the value, $\alpha$, to be assigned to the off-diagonal elements of $A$.

**diag** – Complex $\quad$ Input

On entry: the value, $d$, to be assigned to the diagonal elements of $A$.

**a**[$dim$] – Complex $\quad$ Output

Note: the dimension, $dim$, of the array $a$ must be at least $\max(1, pd\times n)$.

On exit: the $n$ by $n$ triangular matrix $A$ with diagonal elements set to $\text{diag}$ and strictly upper or lower elements set to $\alpha$.

If $\text{order} = \text{Nag\_ColMajor}$, $A_{ij}$ is stored in $a[(j-1) \times pd + i - 1]$.

If $\text{order} = \text{Nag\_RowMajor}$, $A_{ij}$ is stored in $a[(i-1) \times pd + j - 1]$.

If $\text{uplo} = \text{Nag\_Upper}$, $A$ is upper triangular and the elements of the array corresponding to the lower triangular part of $A$ are not referenced.

If $\text{uplo} = \text{Nag\_Lower}$, $A$ is lower triangular and the elements of the array corresponding to the upper triangular part of $A$ are not referenced.

**pda** – Integer $\quad$ Input

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

Constraint: $pda \geq \max(1, n)$.

**fail** – NagError $\quad$ 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: $n \geq 0$.

**NE_INT_2**

On entry, $pd\ = \langle value \rangle$, $n = \langle value \rangle$.

Constraint: $pda \geq \max(1, n)$.

**NE_INTERNAL_ERROR**

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
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
Not applicable.

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. *
* Copyright 2014 Numerical Algorithms Group. *
* Mark 8, 2005. */
#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 ] ");
    
Mark 25

/* 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, _countof(nag_enum_arg));
#else
    scanf("%39s%[\n] ", nag_enum_arg);
#endif

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

<table>
<thead>
<tr>
<th>Generated Matrix A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>