NAG Library Function Document

nag_zupgtr (f08gtc)

1 Purpose

nag_zupgtr (f08gtc) generates the complex unitary matrix Q, which was determined by nag_zhptrd (f08gsc) when reducing a Hermitian matrix to tridiagonal form.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_zupgtr (Nag_OrderType order, Nag_UploType uplo, Integer n,
     const Complex ap[], const Complex tau[], Complex q[], Integer pdq,
     NagError *fail)
```

3 Description

nag_zupgtr (f08gtc) is intended to be used after a call to nag_zhptrd (f08gsc), which reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation: $A = QTQ^{\mathrm{H}}$. nag_zhptrd (f08gsc) represents the unitary matrix Q as a product of $n-1$ elementary reflectors.

This function may be used to generate Q explicitly as a square matrix.

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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: this **must** be the same argument **uplo** as supplied to nag_zhptrd (f08gsc).

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

3: **n** – Integer *Input*

On entry: n, the order of the matrix Q.

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

4: **ap**[*dim*] – const Complex *Input*

Note: the dimension, *dim*, of the array **ap** must be at least $\max(1, \mathbf{n} \times (\mathbf{n}+1)/2)$.

On entry: details of the vectors which define the elementary reflectors, as returned by nag_zhptrd (f08gsc).

5: **tau**$[dim]$ – const Complex *Input*

Note: the dimension, *dim*, of the array **tau** must be at least $\max(1, \mathbf{n} - 1)$.

On entry: further details of the elementary reflectors, as returned by nag_zhptrd (f08gsc).

6: **q**$[dim]$ – Complex *Output*

Note: the dimension, *dim*, of the array **q** must be at least $\max(1, \mathbf{pdq} \times \mathbf{n})$.

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

$\mathbf{q}[(j - 1) \times \mathbf{pdq} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{q}[(i - 1) \times \mathbf{pdq} + j - 1]$ when **order** = Nag_RowMajor.

On exit: the n by n unitary matrix Q.

7: **pdq** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **q**.

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

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

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

NE_INT_2

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

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 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 computed matrix Q differs from an exactly unitary matrix by a matrix E such that

$$\|E\|_2 = O(\epsilon),$$

where ϵ is the ***machine precision***.

8 Parallelism and Performance

nag_zupgtr (f08gtc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zupgtr (f08gtc) 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 $\frac{16}{3}n^3$.

The real analogue of this function is nag_dopgtr (f08gfc).

10 Example

This example computes all the eigenvalues and eigenvectors of the matrix A, where

$$A = \begin{pmatrix} -2.28+0.00i & 1.78-2.03i & 2.26+0.10i & -0.12+2.53i \\ 1.78+2.03i & -1.12+0.00i & 0.01+0.43i & -1.07+0.86i \\ 2.26-0.10i & 0.01-0.43i & -0.37+0.00i & 2.31-0.92i \\ -0.12-2.53i & -1.07-0.86i & 2.31+0.92i & -0.73+0.00i \end{pmatrix},$$

using packed storage. Here A is Hermitian and must first be reduced to tridiagonal form by nag_zhptrd (f08gsc). The program then calls nag_zupgtr (f08gtc) to form Q, and passes this matrix to nag_zsteqr (f08jsc) which computes the eigenvalues and eigenvectors of A.

10.1 Program Text

```
/* nag_zupgtr (f08gtc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx04.h>
#include <naga02.h>

int main(void)
```

```
{
  /* Scalars */
  Integer ap_len, i, j, n, pdz, d_len, e_len, tau_len;
  Integer exit_status = 0;
  NagError fail;
  Nag_UploType uplo;
  Nag_OrderType order;
  /* Arrays */
  char nag_enum_arg[40];
  Complex *ap = 0, *tau = 0, *z = 0;
  double *d = 0, *e = 0;

#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I, J) ap[J * (J - 1) / 2 + I - 1]
#define A_LOWER(I, J) ap[(2 * n - J) * (J - 1) / 2 + I - 1]
#define Z(I, J) z[(J - 1) * pdz + I - 1]
  order = Nag_ColMajor;
#else
#define A_LOWER(I, J) ap[I * (I - 1) / 2 + J - 1]
#define A_UPPER(I, J) ap[(2 * n - I) * (I - 1) / 2 + J - 1]
#define Z(I, J) z[(I - 1) * pdz + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zupgtr (f08gtc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &n);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &n);
#endif
#ifdef NAG_COLUMN_MAJOR
  pdz = n;
#else
  pdz = n;
#endif
  ap_len = n * (n + 1) / 2;
  tau_len = n - 1;
  d_len = n;
  e_len = n - 1;
  /* Allocate memory */
  if (!(ap = NAG_ALLOC(ap_len, Complex)) ||
      !(d = NAG_ALLOC(d_len, double)) ||
      !(e = NAG_ALLOC(e_len, double)) ||
      !(tau = NAG_ALLOC(tau_len, Complex)) ||
      !(z = NAG_ALLOC(n * n, Complex)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read A from data file */
#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);
  if (uplo == Nag_Upper) {
    for (i = 1; i <= n; ++i) {
      for (j = i; j <= n; ++j) {
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(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_LOWER(i, j).re, &A_LOWER(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A_LOWER(i, j).re, &A_LOWER(i, j).im);
#endif
      }
    }
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif
  }

  /* Reduce A to tridiagonal form T = (Q^H)*A*Q */
  /* nag_zhptrd (f08gsc).
   * Unitary reduction of complex Hermitian matrix to real
   * symmetric tridiagonal form, packed storage
   */
  nag_zhptrd(order, uplo, n, ap, d, e, tau, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zhptrd (f08gsc).\n%s\n", fail.message);
    exit_status = 1;
  }

  /* Form Q explicitly, storing the result in Z */
  /* nag_zupgtr (f08gtc).
   * Generate unitary transformation matrix from reduction to
   * tridiagonal form determined by nag_zhptrd (f08gsc)
   */
  nag_zupgtr(order, uplo, n, ap, tau, z, pdz, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zupgtr (f08gtc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Calculate all the eigenvalues and eigenvectors of A */
  /* nag_zsteqr (f08jsc).
   * All eigenvalues and eigenvectors of real symmetric
   * tridiagonal matrix, reduced from complex Hermitian
   * matrix, using implicit QL or QR
   */
  nag_zsteqr(order, Nag_UpdateZ, n, d, e, z, pdz, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zsteqr (f08jsc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Normalize the eigenvectors */
  for (j = 1; j <= n; j++) {
    for (i = n; i >= 1; i--) {
      Z(i, j) = nag_complex_divide(Z(i, j), Z(1, j));
    }
  }
  /* Print eigenvalues and eigenvectors */
  printf("Eigenvalues\n");
  for (i = 1; i <= n; ++i)
    printf("%8.4f%s", d[i - 1], i % 8 == 0 ? "\n" : "            ");
  printf("\n\n");
  /* 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,
                                n, z, pdz, Nag_BracketForm, "%7.4f",
                                "Eigenvectors", 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(ap);
  NAG_FREE(d);
  NAG_FREE(e);
  NAG_FREE(tau);
  NAG_FREE(z);

  return exit_status;
}
```

10.2 Program Data

```
nag_zupgtr (f08gtc) Example Program Data
  4                                                        :Value of n
  Nag_Lower                                                :Value of uplo
 (-2.28, 0.00)
 ( 1.78, 2.03) (-1.12, 0.00)
 ( 2.26,-0.10) ( 0.01,-0.43) (-0.37, 0.00)
 (-0.12,-2.53) (-1.07,-0.86) ( 2.31, 0.92) (-0.73, 0.00)   :End of matrix A
```

10.3 Program Results

```
nag_zupgtr (f08gtc) Example Program Results

Eigenvalues
 -6.0002            -3.0030             0.5036             3.9996

 Eigenvectors
                  1                  2                  3                  4
 1  ( 1.0000, 0.0000)  ( 1.0000,-0.0000)  ( 1.0000,-0.0000)  ( 1.0000, 0.0000)
 2  (-0.2278,-0.2824)  (-2.2999,-1.6237)  ( 1.0792, 0.4997)  ( 0.4876, 0.7282)
 3  (-0.5706,-0.1941)  ( 1.1424, 0.5807)  ( 0.5013, 1.7896)  ( 0.6025,-0.6924)
 4  ( 0.2388, 0.5702)  (-1.3415,-1.5739)  (-1.0810, 0.4883)  ( 0.4257,-1.0093)
```