NAG Library Function Document

nag_complex_apply_q (f01rdc)

1 Purpose

nag_complex_apply_q (f01rdc) performs one of the transformations

$$B := QB \quad \text{or} \quad B := Q^H B,$$

where B is an m by $ncolb$ complex matrix and Q is an m by m unitary matrix, given as the product of Householder transformation matrices.

This function is intended for use following nag_complex_qr (f01rcc).

2 Specification

```
#include <nag.h>
#include <nagf01.h>
void nag_complex_apply_q (MatrixTranspose trans, Nag_WhereElements wheret,
     Integer m, Integer n, Complex a[], Integer tda, const Complex theta[],
     Integer ncolb, Complex b[], Integer tdb, NagError *fail)
```

3 Description

The unitary matrix Q is assumed to be given by

$$Q = (Q_n Q_{n-1} \cdots Q_1)^H,$$

Q_k being given in the form

$$Q_k = \begin{pmatrix} I & 0 \\ 0 & T_k \end{pmatrix},$$

where

$$T_k = I - \gamma_k u_k u_k^H,$$

$$u_k = \begin{pmatrix} \zeta_k \\ z_k \end{pmatrix},$$

γ_k is a scalar for which Re $\gamma_k = 1.0$, ζ_k is a real scalar and z_k is an $(m-k)$ element vector.

z_k must be supplied in the $(k-1)$th column of **a** in elements $\mathbf{a}[(k) \times \mathbf{tda} + k - 1], \ldots, \mathbf{a}[(m-1) \times \mathbf{tda} + k - 1]$ and θ_k, given by

$$\theta_k = (\zeta_k, \mathrm{Im}\,\gamma_k),$$

must be supplied either in $\mathbf{a}[(k-1) \times \mathbf{tda} + k - 1]$ or in $\mathbf{theta}[k-1]$, depending upon the argument **wheret**.

To obtain Q explicitly B may be set to I and premultiplied by Q. This is more efficient than obtaining Q^H. Alternatively, nag_complex_form_q (f01rec) may be used to obtain Q overwritten on A.

4 References

Wilkinson J H (1965) *The Algebraic Eigenvalue Problem* Oxford University Press, Oxford

5 Arguments

1: **trans** – MatrixTranspose *Input*

On entry: the operation to be performed as follows:

trans = NoTranspose, perform the operation $B := QB$.

trans = ConjugateTranspose, perform the operation $B := Q^H B$.

Constraint: **trans** must be one of NoTranspose or ConjugateTranspose.

2: **wheret** – Nag_WhereElements *Input*

On entry: the elements of θ are to be found as follows:

wheret = Nag_ElementsIn The elements of θ are in A.

wheret = Nag_ElementsSeparate The elements of θ are separate from A, in **theta**.

Constraint: **wheret** must be one of Nag_ElementsIn or Nag_ElementsSeparate.

3: **m** – Integer *Input*

On entry: m, the number of rows of A.

Constraint: **m** $\geq$ **n**.

4: **n** – Integer *Input*

On entry: n, the number of columns of A.

When **n** $= 0$ then an immediate return is effected.

Constraint: **n** ≥ 0.

5: **a**[**m** $\times$ **tda**] – Complex *Input*

On entry: the leading m by n strictly lower triangular part of the array **a** must contain details of the matrix Q. In addition, when **wheret** = Nag_ElementsIn, then the diagonal elements of **a** must contain the elements of θ as described under the argument **theta**. When **wheret** = Nag_ElementsSeparate, then the diagonal elements of the array **a** are referenced, since they are used temporarily to store the ζ_k, but they contain their original values on return.

6: **tda** – Integer *Input*

On entry: the stride separating matrix column elements in the array **a**.

Constraint: **tda** $\geq$ **n**.

7: **theta**[**n**] – const Complex *Input*

On entry: with **wheret** = Nag_ElementsSeparate, the array **theta** must contain the elements of θ. If **theta**$[k-1] = 0.0$ then T_k is assumed to be I; if **theta**$[k-1] = \alpha$, with Re $\alpha < 0.0$, then T_k is assumed to be of the form

$$T_k = \begin{pmatrix} \alpha & 0 \\ 0 & I \end{pmatrix};$$

otherwise **theta**$[k-1]$ is assumed to contain θ_k given by $\theta_k = (\zeta_k, \operatorname{Im} \gamma_k)$.

When **wheret** = Nag_ElementsIn, the array **theta** is not referenced and may be **NULL**.

8: **ncolb** – Integer *Input*

On entry: $ncolb$, the number of columns of B.

When **ncolb** $= 0$ then an immediate return is effected.

Constraint: **ncolb** ≥ 0.

9: **b**[**m** × **tdb**] – Complex *Input/Output*

Note: the (i, j)th element of the matrix B is stored in **b**$[(i-1) \times$ **tdb** $+ j - 1]$.

On entry: the leading m by $ncolb$ part of the array **b** must contain the matrix to be transformed.

On exit: **b** is overwritten by the transformed matrix.

10: **tdb** – Integer *Input*

On entry: the stride separating matrix column elements in the array **b**.

Constraint: **tdb** $\geq$ **ncolb**.

11: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_2_INT_ARG_LT

On entry, **m** $= \langle$*value*$\rangle$ while **n** $= \langle$*value*$\rangle$. These arguments must satisfy **m** $\geq$ **n**.

On entry, **tda** $= \langle$*value*$\rangle$ while **n** $= \langle$*value*$\rangle$. These arguments must satisfy **tda** $\geq$ **n**.

On entry, **tdb** $= \langle$*value*$\rangle$ while **ncolb** $= \langle$*value*$\rangle$. These arguments must satisfy **tdb** $\geq$ **ncolb**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument **trans** had an illegal value.

On entry, argument **wheret** had an illegal value.

NE_INT_ARG_LT

On entry, **n** $= \langle$*value*$\rangle$.
Constraint: **n** ≥ 0.

On entry, **ncolb** $= \langle$*value*$\rangle$.
Constraint: **ncolb** ≥ 0.

7 Accuracy

Letting C denote the computed matrix $Q^H B$, C satisfies the relation

$$QC = B + E$$

where $\|E\| \leq c\epsilon\|B\|$, ϵ being the ***machine precision***, c is a modest function of m and . denotes the spectral (two) norm. An equivalent result holds for the computed matrix QB. See also Section 9.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The approximate number of real floating-point operations is given by $8n(2m - n)ncolb$.

10 Example

To obtain the matrix $Q^H B$ for the matrix B given by

$$B = \begin{pmatrix} & -0.55 + 1.05i & 0.45 + 1.05i \\ & 0.49 + 0.93i & 1.09 + 0.13i \\ & 0.56 - 0.16i & 0.64 + 0.16i \\ & 0.39 + 0.23i & -0.39 - 0.23i \\ 1.13 + 0.83i - 1.13 + 0.77i & & \end{pmatrix}$$

following the QR factorization of the 5 by 3 matrix A given by

$$A = \begin{pmatrix} 0.5i & -0.5 + 1.5i & -1.0 + 1.0i \\ 0.4 + 0.3i & 0.9 + 1.3i & 0.2 + 1.4i \\ 0.4 & -0.4 + 0.4i & 1.8 \\ 0.3 - 0.4i & 0.1 + 0.7i & 0.0 \\ -0.3i & 0.3 + 0.3i & 2.4i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_complex_apply_q (f01rdc) Example Program.
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 1, 1990.
 * Mark 8 revised, 2004.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagf01.h>

#define COMPLEX(A) A.re, A.im
#define A(I, J)    a[(I) *tda + J]
#define B(I, J)    b[(I) *tdb + J]

int main(void)
{

  Complex  *a = 0, *b = 0, *theta = 0;
  Integer  exit_status = 0, i, j, m, n, ncolb, tda, tdb;
  NagError fail;

  INIT_FAIL(fail);


  printf("nag_complex_apply_q (f01rdc) Example Program Results\n");
  /* Skip heading in data file */
  scanf("%*[^\n]");
  scanf("%ld%ld", &m, &n);
  if (n >= 0 && m >= n)
    {
      if (!(a = NAG_ALLOC(m*n, Complex)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      tda = n;
    }
  else
    {
```

```
      printf("Invalid n or m.\n");
      exit_status = 1;
      return exit_status;
    }
  for (i = 0; i < m; ++i)
    for (j = 0; j < n; ++j)
      scanf(" ( %lf,  %lf ) ", COMPLEX(&A(i, j)));
  scanf("%ld", &ncolb);
  if (ncolb >= 0)
    {
      if (!(b = NAG_ALLOC(m*ncolb, Complex)) ||
          !(theta = NAG_ALLOC(n, Complex)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      tdb = ncolb;
    }
  else
    {
      printf("Invalid ncolb.\n");
      exit_status = 1;
      return exit_status;
    }
  for (i = 0; i < m; ++i)
    for (j = 0; j < ncolb; ++j)
      scanf(" ( %lf,  %lf ) ", COMPLEX(&B(i, j)));
  /* Find the QR factorization of A. */
  /* nag_complex_qr (f01rcc).
   * QR factorization of complex m by n matrix (m >= n)
   */
  nag_complex_qr(m, n, a, tda, theta, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_complex_qr (f01rcc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Form  conjg( Q' )*B. */

  /* nag_complex_apply_q (f01rdc).
   * Compute QB or Q^HB after factorization by nag_complex_qr
   * (f01rcc)
   */
  nag_complex_apply_q(ConjugateTranspose, Nag_ElementsSeparate, m, n,
                      a, tda, theta, ncolb, b,
                      tdb, &fail);

  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_complex_apply_q (f01rdc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  printf("\nMatrix  conjg( Q' )*B\n");
  for (i = 0; i < m; ++i)
    {
      for (j = 0; j < ncolb; ++j)
        printf("  (%7.4f, %8.4f)%s", COMPLEX(B(i, j)),
                (j%2 == 1 || j == n-1)?"\n":" ");
    }
 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(theta);
  return exit_status;
}
```

10.2 Program Data

```
nag_complex_apply_q (f01rdc) Example Program Data

   5      3

  (0.00,  0.50)   (-0.50,  1.50)   (-1.00,  1.00)
  (0.40,  0.30)   ( 0.90,  1.30)   ( 0.20,  1.40)
  (0.40,  0.00)   (-0.40,  0.40)   ( 1.80,  0.00)
  (0.30, -0.40)   ( 0.10,  0.70)   ( 0.00,  0.00)
  (0.00, -0.30)   ( 0.30,  0.30)   ( 0.00,  2.40)

   2

 (-0.55,  1.05)   ( 0.45,  1.05)
  (0.49,  0.93)   ( 1.09,  0.13)
  (0.56, -0.16)   ( 0.64,  0.16)
  (0.39,  0.23)   (-0.39, -0.23)
  (1.13,  0.83)   (-1.13,  0.77)
```

10.3 Program Results

```
nag_complex_apply_q (f01rdc) Example Program Results

Matrix  conjg( Q' )*B
  ( 1.0000,   1.0000)   ( 1.0000,  -1.0000)
  (-1.0000,   0.0000)   (-1.0000,   0.0000)
  (-1.0000,   1.0000)   (-1.0000,  -1.0000)
  (-0.0600,  -0.0200)   (-0.0400,   0.1200)
  ( 0.0400,   0.1200)   (-0.0600,   0.0200)
```