NAG Library Function Document

nag_complex_form_q (f01rec)

1 Purpose

nag_complex_form_q (f01rec) returns the first $ncolq$ columns of the m by m unitary matrix Q, where Q is 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_form_q (Nag_WhereElements wheret, Integer m, Integer n,
     Integer ncolq, Complex a[], Integer tda, const Complex theta[],
     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 $\mathrm{Re}\,\gamma_k = 1.0$, 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**.

4 References

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

5 Arguments

1: **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.

2: **m** – Integer *Input*

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

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

3: **n** – Integer *Input*

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

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

4: **ncolq** – Integer *Input*

On entry: $ncolq$, the required number of columns of Q.

When $\mathbf{ncolq} = 0$ then an immediate return is effected.

Constraint: $0 \leq \mathbf{ncolq} \leq \mathbf{m}$.

5: **a**[**m** × **tda**] – Complex *Input/Output*

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**.

On exit: the first **ncolq** columns of the array **a** are overwritten by the first **ncolq** columns of the m by m unitary matrix Q. When $\mathbf{n} = 0$ then the first **ncolq** columns of **a** are overwritten by the first **ncolq** columns of the unit matrix.

6: **tda** – Integer *Input*

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

Constraint: $\mathbf{tda} \geq \max(\mathbf{n}, \mathbf{ncolq})$.

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

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

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

otherwise $\mathbf{theta}[k-1]$ 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: **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_GT

On entry, **ncolq** = ⟨*value*⟩ while **m** = ⟨*value*⟩. These arguments must satisfy $\mathbf{ncolq} \leq \mathbf{m}$.

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 $\max(\mathbf{n}, \mathbf{ncolq}) = \mathbf{ncolq}$. These arguments must satisfy **tda** $\geq$ **n**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

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

NE_INT_ARG_LT

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

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

7 Accuracy

The computed matrix Q satisfies the relation

$$Q = P + E,$$

where P is an exactly unitary matrix and

$$\|E\| \leq c\epsilon,$$

ϵ being the ***machine precision***, c is a modest function of m and . denotes the spectral (two) norm. See also Section 9 of nag_complex_qr (f01rcc).

8 Parallelism and Performance

Not applicable.

9 Further Comments

The approximate number of real floating-point operations required is given by

$$\begin{array}{cc} \frac{8}{3}n(3m-n)(2ncolq-n) - n(ncolq-n) & ncolq > n \\ \frac{8}{3}ncolq^2(3m-ncolq) & ncolq \leq n. \end{array}$$

10 Example

To obtain the 5 by 5 unitary matrix Q following the QR factorization of the 5 by 3 matrix A given by

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

10.1 Program Text

```
/* nag_complex_form_q (f01rec) 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 Q(I, J)    q[(I) *tdq + J]

int main(void)
{
  Complex  *a = 0, *q = 0, *theta = 0;
  Integer  exit_status = 0, i, j, m, n, ncolq, tda, tdq;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_complex_form_q (f01rec) 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)) ||
          !(q = NAG_ALLOC(m*m, Complex)) ||
          !(theta = NAG_ALLOC(m, Complex)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      tda = n;
      tdq = m;
    }
  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)));
  /* 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;
    }
  /* Copy the array A into Q and form the m by m matrix Q. */
  for (j = 0; j < n; ++j)
    for (i = 0; i < m; ++i)
      Q(i, j).re = A(i, j).re, Q(i, j).im = A(i, j).im;
  ncolq = m;
  /* nag_complex_form_q (f01rec).
   * Form columns of Q after factorization by nag_complex_qr
   * (f01rcc)
   */
  nag_complex_form_q(Nag_ElementsSeparate, m, n, ncolq, q, tdq,
                     theta, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_complex_form_q (f01rec).\n%s\n",
              fail.message);
```

```
      exit_status = 1;
      goto END;
    }
  printf("\nMatrix  Q\n");
  for (i = 0; i < m; ++i)
    {
      for (j = 0; j < ncolq; ++j)
        printf(" (%5.2f,%5.2f)%s", COMPLEX(Q(i, j)),
                (j%5 == 4 || j == ncolq-1)?"\n":" ");
    }
 END:
  NAG_FREE(a);
  NAG_FREE(q);
  NAG_FREE(theta);
  return exit_status;
}
```

10.2 Program Data

```
nag_complex_form_q (f01rec) Example Program Data

   5     3

  ( 0.0,  0.5 )    (-0.5,  1.5 )    (-1.0,  1.0 )
  ( 0.4,  0.3 )    ( 0.9,  1.3 )    ( 0.2,  1.4 )
  ( 0.4,  0.0 )    (-0.4,  0.4 )    ( 1.8,  0.0 )
  ( 0.3, -0.4 )    ( 0.1,  0.7 )    ( 0.0,  0.0 )
  ( 0.0, -0.3 )    ( 0.3,  0.3 )    ( 0.0,  2.4 )
```

10.3 Program Results

```
nag_complex_form_q (f01rec) Example Program Results

Matrix  Q
 ( 0.00, 0.50)  ( 0.00,-0.50)  (-0.00, 0.00)  ( 0.50, 0.00)  ( 0.40, 0.30)
 ( 0.40, 0.30)  (-0.40,-0.30)  (-0.00, 0.00)  (-0.30, 0.40)  (-0.48, 0.14)
 ( 0.40, 0.00)  ( 0.40, 0.00)  (-0.60,-0.00)  (-0.24,-0.32)  ( 0.00, 0.40)
 ( 0.30,-0.40)  ( 0.30,-0.40)  ( 0.00,-0.00)  ( 0.50,-0.00)  (-0.40,-0.30)
 ( 0.00,-0.30)  ( 0.00,-0.30)  (-0.00,-0.80)  (-0.24, 0.18)  ( 0.30, 0.00)
```