NAG C Library Function Document

nag_zunglq (f08awc)

1 Purpose

nag_zunglq (f08awc) generates all or part of the complex unitary matrix $Q$ from an $LQ$ factorization computed by nag_zgelqf (f08avc).

2 Specification

void nag_zunglq (Nag_OrderType order, Integer m, Integer n, Integer k, Complex a[], Integer pda, const Complex tau[], NagError *fail)

3 Description

nag_zunglq (f08awc) is intended to be used after a call to nag_zgelqf (f08avc), which performs an $LQ$ factorization of a complex matrix $A$. The unitary matrix $Q$ is represented as a product of elementary reflectors.

This function may be used to generate $Q$ explicitly as a square matrix, or to form only its leading rows. Usually $Q$ is determined from the $LQ$ factorization of a $p$ by $n$ matrix $A$ with $p \leq n$. The whole of $Q$ may be computed by:

\[
\text{nag_zunglq (order,n,n,p,&a,pda,tau,&fail)}
\]

(note that the array $a$ must have at least $n$ rows) or its leading $p$ rows by:

\[
\text{nag_zunglq (order,p,n,p,&a,pda,tau,&fail)}
\]

The rows of $Q$ returned by the last call form an orthonormal basis for the space spanned by the rows of $A$; thus nag_zgelqf (f08avc) followed by nag_zunglq (f08awc) can be used to orthogonalise the rows of $A$.

The information returned by the $LQ$ factorization functions also yields the $LQ$ factorization of the leading $k$ rows of $A$, where $k < p$. The unitary matrix arising from this factorization can be computed by:

\[
\text{nag_zunglq (order,n,n,k,&a,pda,tau,&fail)}
\]

or its leading $k$ rows by:

\[
\text{nag_zunglq (order,k,n,k,&a,pda,tau,&fail)}
\]

4 References


5 Parameters

1: order – Nag_OrderType

Input

On entry: the order parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: order = Nag_RowMajor or Nag_ColMajor.

2: m – Integer

Input

On entry: $m$, the number of rows of the matrix $Q$.

Constraint: $m \geq 0$. 

3: \( n \) – Integer

*Input*

On entry: \( n \), the number of columns of the matrix \( Q \).

*Constraint:* \( n \geq m \).

4: \( k \) – Integer

*Input*

On entry: \( k \), the number of elementary reflectors whose product defines the matrix \( Q \).

*Constraint:* \( m \geq k \geq 0 \).

5: \( a[dim] \) – Complex

*Input/Output*

**Note:** the dimension, \( dim \), of the array \( a \) must be at least \( \max(1, pda \times n) \) when \( order = \text{Nag\_ColMajor} \) and at least \( \max(1, pda \times m) \) when \( order = \text{Nag\_RowMajor} \).

If \( order = \text{Nag\_ColMajor} \), the \((i, j)\)th element of the matrix \( A \) is stored in \( a[(j - 1) \times pda + i - 1] \) and if \( order = \text{Nag\_RowMajor} \), the \((i, j)\)th element of the matrix \( A \) is stored in \( a[(i - 1) \times pda + j - 1] \).

On entry: details of the vectors which define the elementary reflectors, as returned by nag_zgelqf (f08avc).

On exit: the \( m \) by \( n \) matrix \( Q \).

6: \( pda \) – Integer

*Input*

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

*Constraints:*

\[
\begin{align*}
&\text{if } order = \text{Nag\_ColMajor}, pda \geq \max(1, m); \\
&\text{if } order = \text{Nag\_RowMajor}, pda \geq \max(1, n).
\end{align*}
\]

7: \( \tau[dim] \) – const Complex

*Input*

**Note:** the dimension, \( dim \), of the array \( \tau \) must be at least \( \max(1, k) \).

On entry: further details of the elementary reflectors, as returned by nag_zgelqf (f08avc).

8: \( \text{fail} \) – NagError *

*Output*

The NAG error parameter (see the Essential Introduction).

### 6 Error Indicators and Warnings

**NE\_INT**

On entry, \( m = \langle \text{value} \rangle \).

Constraint: \( m \geq 0 \).

On entry, \( pda = \langle \text{value} \rangle \).

Constraint: \( pda > 0 \).

**NE\_INT\_2**

On entry, \( n = \langle \text{value} \rangle, m = \langle \text{value} \rangle \).

Constraint: \( n \geq m \).

On entry, \( m = \langle \text{value} \rangle, k = \langle \text{value} \rangle \).

Constraint: \( m \geq k \geq 0 \).

On entry, \( pda = \langle \text{value} \rangle, m = \langle \text{value} \rangle \).

Constraint: \( pda \geq \max(1, m) \).

On entry, \( pda = \langle \text{value} \rangle, n = \langle \text{value} \rangle \).

Constraint: \( pda \geq \max(1, n) \).
NE_ALLOC_FAIL
Memory allocation failed.

NE_BAD_PARAM
On entry, parameter (value) had an illegal value.

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 consult NAG for assistance.

7 Accuracy
The computed matrix $Q$ differs from an exactly unitary matrix by a matrix $E$ such that

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

where $\epsilon$ is the machine precision.

8 Further Comments
The total number of real floating-point operations is approximately $16mnk - 8(m + n)k^2 + \frac{16}{3}k^3$; when $m = k$, the number is approximately $\frac{8}{3}m^2(3n - m)$.

The real analogue of this function is nag_dorglq (f08ajc).

9 Example
To form the leading 4 rows of the unitary matrix $Q$ from the $LQ$ factorization of the matrix $A$, where

$$
A = \begin{pmatrix}
0.28 - 0.36i & 0.50 - 0.86i & -0.77 - 0.48i & 1.58 + 0.66i \\
-0.50 - 1.10i & -1.21 + 0.76i & -0.32 - 0.24i & -0.27 - 1.15i \\
0.36 - 0.51i & -0.07 + 1.33i & -0.75 + 0.47i & -0.08 + 1.01i \\
\end{pmatrix},
$$

The rows of $Q$ form an orthonormal basis for the space spanned by the rows of $A$.

9.1 Program Text
/* nag_zunglq (f08awc) Example Program. *
 * Copyright 2001 Numerical Algorithms Group. *
 * Mark 7, 2001. */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer i, j, m, n, pda, tau_len;
    Integer exit_status=0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    char *title=0;
    Complex *a=0, *tau=0;

    #ifdef NAG_COLUMN_MAJOR
    #define A(I,J) a[(J-1)*pda + I-1]
    #else
    #define A(I,J) a[(I-1)*pda + J-1]
    #endif


```c
order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
order = Nag_RowMajor;
#endif

INIT_FAIL(fail);
Vprintf("f08awc Example Program Results\n\n");

/* Skip heading in data file */
Vscanf("%*[\n ]");
Vscanf("%ld%ld%*[\n ]", &m, &n);
#ifdef NAG_COLUMN_MAJOR
pda = m;
#else
pda = n;
#endif
tau_len = m;

/* Allocate memory */
if (!(title = NAG_ALLOC(31, char)) ||
    !(a = NAG_ALLOC(m * n, Complex)) ||
    !(tau = NAG_ALLOC(tau_len, Complex)) )
{
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read A from data file */
for (i = 1; i <= m; ++i)
{
    for (j = 1; j <= n; ++j)
        Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
Vscanf("%*[\n ]");

/* Compute the LQ factorization of A */
f08avc(order, m, n, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08avc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Form the leading M rows of Q explicitly */
f08awc(order, m, n, m, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08awc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print the leading M rows of Q only */
Vsprintf(title, "The leading %2ld rows of Q\n", m);
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, n,
        a, pda, Nag_BracketForm, "%7.4f", title,
        Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04dbc.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
END:
if (title) NAG_FREE(title);
if (a) NAG_FREE(a);
if (tau) NAG_FREE(tau);
```
return exit_status;
}

9.2 Program Data

f08awc Example Program Data

3 4
( 0.28,-0.36) ( 0.50,-0.86) (-0.77,-0.48) ( 1.58, 0.66)
(-0.50,-1.10) (-1.21, 0.76) (-0.32,-0.24) (-0.27,-1.15)
( 0.36,-0.51) (-0.07, 1.33) (-0.75, 0.47) (-0.08, 1.01)

:Values of M and N

End of matrix A

9.3 Program Results

f08awc Example Program Results

The leading 3 rows of Q

1 2 3 4
1 (-0.1258, 0.1618) (-0.2247, 0.3864) ( 0.3460, 0.2157) (-0.7099,-0.2966)
2 (-0.1163,-0.6380) (-0.3240, 0.4272) (-0.1995,-0.5009) (-0.0323,-0.0162)
3 (-0.4607, 0.1090) ( 0.2171,-0.4062) ( 0.2733,-0.6106) (-0.0994,-0.3261)