NAG C Library Function Document

nag_zupmtr (f08guc)

1 Purpose

nag_zupmtr (f08guc) multiplies an arbitrary complex matrix $C$ by the complex unitary matrix $Q$ which was determined by nag_zhptrd (f08gsc) when reducing a complex Hermitian matrix to tridiagonal form.

2 Specification

```c
void nag_zupmtr (Nag_OrderType order, Nag_SideType side, Nag_UploType uplo,
                 Nag_TransType trans, Integer m, Integer n, Complex ap[],
                 const Complex tau[], Complex c[], Integer pdc, NagError *fail)
```

3 Description

nag_zupmtr (f08guc) 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: $A = QTQ^H$. nag_zhptrd (f08gsc) represents the unitary matrix $Q$ as a product of elementary reflectors.

This function may be used to form one of the matrix products

$$QC, Q^HC, CQ$$ or $$CQ^H,$$

overwriting the result on $C$ (which may be any complex rectangular matrix).

A common application of this function is to transform a matrix $Z$ of eigenvectors of $T$ to the matrix $QZ$ of eigenvectors of $A$.

4 References


5 Parameters

1: order – Nag_OrderType

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: side – Nag_SideType

On entry: indicates how $Q$ or $Q^H$ is to be applied to $C$ as follows:

- if side = Nag_LeftSide, $Q$ or $Q^H$ is applied to $C$ from the left;
- if side = Nag_RightSide, $Q$ or $Q^H$ is applied to $C$ from the right.

Constraint: side = Nag_LeftSide or Nag_RightSide.

3: uplo – Nag_UploType

On entry: this must be the same parameter uplo as supplied to nag_zhptrd (f08gsc).

Constraint: uplo = Nag_Upper or Nag_Lower.
trans \textendash \text{Nag\_TransType} \text{ Input}

On entry: indicates whether \( Q \) or \( Q^H \) is to be applied to \( C \) as follows:

if \( \text{trans} = \text{Nag\_NoTrans} \), \( Q \) is applied to \( C \);

if \( \text{trans} = \text{Nag\_ConjTrans} \), \( Q^H \) is applied to \( C \).

Constraint: \( \text{trans} = \text{Nag\_NoTrans} \) or \( \text{Nag\_ConjTrans} \).

\( m \) \textendash \text{Integer} \text{ Input}

On entry: \( m \), the number of rows of the matrix \( C \); \( m \) is also the order of \( Q \) if \( \text{side} = \text{Nag\_LeftSide} \).

Constraint: \( m \geq 0 \).

\( n \) \textendash \text{Integer} \text{ Input}

On entry: \( n \), the number of columns of the matrix \( C \); \( n \) is also the order of \( Q \) if \( \text{side} = \text{Nag\_RightSide} \).

Constraint: \( n \geq 0 \).

\( \text{ap}[\text{dim}] \) \textendash \text{Complex} \text{ Input/Output}

Note: the dimension, \( \text{dim} \), of the array \( \text{ap} \) must be at least \( \max(1, m \times (m + 1)/2) \) when \( \text{side} = \text{Nag\_LeftSide} \) and at least \( \max(1, n \times (n + 1)/2) \) when \( \text{side} = \text{Nag\_RightSide} \).

On entry: details of the vectors which define the elementary reflectors, as returned by \text{nag\_zhptrd} (f08gsc).

On exit: \( \text{ap} \) is used as internal workspace prior to being restored and hence is unchanged.

\( \text{tau}[\text{dim}] \) \textendash \text{const Complex} \text{ Input}

Note: the dimension, \( \text{dim} \), of the array \( \text{tau} \) must be at least \( \max(1, m - 1) \) when \( \text{side} = \text{Nag\_LeftSide} \) and at least \( \max(1, n - 1) \) when \( \text{side} = \text{Nag\_RightSide} \).

On entry: further details of the elementary reflectors, as returned by \text{nag\_zhptrd} (f08gsc).

\( \text{c}[\text{dim}] \) \textendash \text{Complex} \text{ Input/Output}

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

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

On entry: the \( m \) by \( n \) matrix \( C \).

On exit: \( \text{c} \) is overwitten by \( QC \) or \( Q^H C \) or \( CQ \) or \( CQ^H \) as specified by \( \text{side} \) and \( \text{trans} \).

\( \text{pdc} \) \textendash \text{Integer} \text{ Input}

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

Constraints:

if \( \text{order} = \text{Nag\_ColMajor} \), \( \text{pdc} \geq \max(1, m) \);

if \( \text{order} = \text{Nag\_RowMajor} \), \( \text{pdc} \geq \max(1, n) \).

\( \text{fail} \) \textendash \text{Nag\_Error *} \text{ 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, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).

On entry, \( pdc = \langle \text{value} \rangle \).
Constraint: \( pdc > 0 \).

**NE_INT_2**

On entry, \( pdc = \langle \text{value} \rangle, m = \langle \text{value} \rangle \).
Constraint: \( pdc \geq \max(1, m) \).

On entry, \( pdc = \langle \text{value} \rangle, n = \langle \text{value} \rangle \).
Constraint: \( pdc \geq \max(1, n) \).

**NE_ALLOC_FAIL**

Memory allocation failed.

**NE_BAD_PARAM**

On entry, parameter \( \langle \text{value} \rangle \) 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 result differs from the exact result by a matrix \( E \) such that
\[
\|E\|_2 = O(\epsilon)\|C\|_2,
\]
where \( \epsilon \) is the machine precision.

8 Further Comments

The total number of real floating-point operations is approximately \( 8m^2n \) if \( \text{side} = \text{Nag_LeftSide} \) and \( 8mn^2 \) if \( \text{side} = \text{Nag_RightSide} \).

The real analogue of this function is \text{nag_dopmtr} (f08gfc).

9 Example

To compute the two smallest eigenvalues, and the associated 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 \( T \) by \text{nag_zhptrd} (f08gsc). The program then calls \text{nag_dstebr} (f08jjc) to compute the requested eigenvalues and \text{nag_zstein} (f08jxc) to compute the associated eigenvectors of \( T \). Finally \text{nag_zupmtr} (f08guc) is called to transform the eigenvectors to those of \( A \).
9.1 Program Text

/* nag_zupmtr (f08guc) 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 ap_len, i, j, m, n, nsplit, pdz, d_len, e_len;
    Integer tau_len;
    Integer exit_status=0;
    double vl=0.0, vu=0.0;
    NagError fail;
    Nag_UploType uplo;
    Nag_OrderType order;
    /* Arrays */
    char uplo_char[2];
    Integer *iblock=0, *ifailv=0, *isplit=0;
    Complex *ap=0, *tau=0, *z=0;
    double *d=0, *e=0, *w=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]
    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]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    Vprintf("f08guc Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[^
\] ");
    Vscanf("%ld%*[^
\] ", &n);
    pdz = n;

    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)) ||
        !(iblock = NAG_ALLOC(n, Integer)) ||
        !(ifailv = NAG_ALLOC(n, Integer)) ||
        !(isplit = NAG_ALLOC(n, Integer)) ||
        !(w = NAG_ALLOC(n, double)) ||
        !(tau = NAG_ALLOC(tau_len, Complex)) ||
        !(z = NAG_ALLOC(n * n, Complex)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read A from data file */
    Vscanf(" ' %1s ' %*[\n] ", uplo_char);
if (*((unsigned char *)uplo_char == 'L')
    uplo = Nag_Lower;
else if (*((unsigned char *)uplo_char == 'U')
    uplo = Nag_Upper;
else
    { 
    Vprintf("Unrecognised character for Nag_UploType type\n");
    exit_status = -1;
    goto END;
    }

if (uplo == Nag_Upper)
    { 
    for (i = 1; i <= n; ++i)
        { 
        for (j = i; j <= n; ++j)
            { 
            Vscanf(" ( %lf , %lf )", &A_UPPER(i,j).re,
                &A_UPPER(i,j).im);
            } 
    Vscanf("%*[\n]");
    }
    else
    { 
    for (i = 1; i <= n; ++i)
        { 
    for (j = 1; j <= i; ++j)
        { 
            Vscanf(" ( %lf , %lf )", &A_LOWER(i,j).re,
                &A_LOWER(i,j).im);
            } 
    Vscanf("%*[\n]");
    }
    }

    /* Reduce A to tridiagonal form T = (Q**H)*A*Q */
    f08gsc(order, uplo, n, ap, d, e, tau, &fail);
    if (fail.code != NE_NOERROR)
    { 
    Vprintf("Error from f08gsc.\n", fail.message);
    exit_status = 1;
    goto END;
    }
    /* Calculate the two smallest eigenvalues of T (same as A) */
    f08jjc(Nag_Indices, Nag_ByBlock, n, vl, vu, l, 2, 0.0,
        d, e, &m, &nsplit, w, iblock, isplit, &fail);
    if (fail.code != NE_NOERROR)
    { 
    Vprintf("Error from f08jjc.\n", fail.message);
    exit_status = 1;
    goto END;
    }
    /* Print eigenvalues */
    Vprintf("Eigenvalues\n");
    for (i = 0; i < m; ++i)
        
        Vprintf("%8.4f%s", w[i], (i+1)%8==0 ?"\n":" ");
        Vprintf("\n\n");
    /* Calculate the eigenvectors of T storing the result in Z */
    f08jxc(order, n, d, e, m, w, iblock, isplit, z, pdz, ifailv, 
        &fail);
    if (fail.code != NE_NOERROR)
    { 
    Vprintf("Error from f08jxc.\n", fail.message);
    exit_status = 1;
    goto END;
    }
    /* Calculate all the eigenvectors of A = Q*(eigenvectors of T) */
    f08guc(order, Nag_LeftSide, uplo, Nag_NoTrans, n, m, ap, 
        tau, z, pdz, &fail);
    if (fail.code != NE_NOERROR)
{  
    Vprintf("Error from f08guc.\n", fail.message);  
    exit_status = 1;  
    goto END;  
}  
/* Print eigenvectors */
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m,  
    z, pdz, Nag_BracketForm, "%7.4f", "Eigenvectors",  
    Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0,  
    0, &fail);  
if (fail.code != NE_NOERROR)  
{  
    Vprintf("Error from x04dbc.\n", fail.message);  
    exit_status = 1;  
    goto END;  
}  
END:  
if (ap) NAG_FREE(ap);  
if (d) NAG_FREE(d);  
if (e) NAG_FREE(e);  
if (iblock) NAG_FREE(iblock);  
if (ifailv) NAG_FREE(ifailv);  
if (isplit) NAG_FREE(isplit);  
if (tau) NAG_FREE(tau);  
if (w) NAG_FREE(w);  
if (z) NAG_FREE(z);  
return exit_status;  
}  

9.2  Program Data  

f08guc Example Program Data  
4  
'U'  
(-2.28, 0.00) ( 1.78, -2.03) ( 2.26, 0.10) (-0.12, 2.53)  
(-1.12, 0.00) ( 0.01, 0.43) (-1.07, 0.86)  
(-0.37, 0.00) ( 2.31, -0.92)  
(-0.73, 0.00) :End of matrix A  

9.3  Program Results  

f08guc Example Program Results  

Eigenvalues  
-6.0002  -3.0030  

Eigenvalues  
1  2  
1 ( 0.7299, 0.0000) (-0.2595, 0.0000)  
2 (-0.1663,-0.2061) ( 0.5969, 0.4214)  
3 (-0.4165,-0.1417) (-0.2965,-0.1507)  
4 ( 0.1743, 0.4162) ( 0.3482, 0.4085)