NAG Library Function Document
nag_zgbbrd (f08lsc)

1 Purpose
nag_zgbbrd (f08lsc) reduces a complex \( m \) by \( n \) band matrix to real upper bidiagonal form.

2 Specification

```c
#include <nag.h>
#include <nagf08.h>
void nag_zgbbrd (Nag_OrderType order, Nag_VectType vect, Integer m,
 Integer n, Integer ncc, Integer kl, Integer ku, Complex ab[],
 Integer pdab, double d[], double e[], Complex q[], Integer pdq,
 Complex pt[], Integer pdpt, Complex c[], Integer pdc, NagError *fail)
```

3 Description
nag_zgbbrd (f08lsc) reduces a complex \( m \) by \( n \) band matrix to real upper bidiagonal form \( B \) by a unitary transformation: \( A = QBP^H \). The unitary matrices \( Q \) and \( P^H \), of order \( m \) and \( n \) respectively, are determined as a product of Givens rotation matrices, and may be formed explicitly by the function if required. A matrix \( C \) may also be updated to give \( \tilde{C} = Q^H C \).

The function uses a vectorizable form of the reduction.

4 References
None.

5 Arguments

1: \( \text{order} \) – Nag_OrderType 
   \( \text{Input} \)
   On entry: the \( \text{order} \) 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 \( \text{order} = \text{Nag_RowMajor} \). See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

   Constraint: \( \text{order} = \text{Nag_RowMajor} \) or \( \text{Nag_ColMajor} \).

2: \( \text{vect} \) – Nag_VectType 
   \( \text{Input} \)
   On entry: indicates whether the matrices \( Q \) and/or \( P^H \) are generated.

   \( \text{vect} = \text{Nag_DoNotForm} \)
   Neither \( Q \) nor \( P^H \) is generated.

   \( \text{vect} = \text{Nag_FormQ} \)
   \( Q \) is generated.

   \( \text{vect} = \text{Nag_FormP} \)
   \( P^H \) is generated.

   \( \text{vect} = \text{Nag_FormBoth} \)
   Both \( Q \) and \( P^H \) are generated.

   Constraint: \( \text{vect} = \text{Nag_DoNotForm} \), \( \text{Nag_FormQ} \), \( \text{Nag_FormP} \) or \( \text{Nag_FormBoth} \).
3: \( m \) – Integer

*Input*

*On entry:* \( m \), the number of rows of the matrix \( A \).

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

4: \( n \) – Integer

*Input*

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

*Constraint:* \( n \geq 0 \).

5: \( ncc \) – Integer

*Input*

*On entry:* \( ncc \), the number of columns of the matrix \( C \).

*Constraint:* \( ncc \geq 0 \).

6: \( kl \) – Integer

*Input*

*On entry:* the number of subdiagonals, \( k_l \), within the band of \( A \).

*Constraint:* \( kl \geq 0 \).

7: \( ku \) – Integer

*Input*

*On entry:* the number of superdiagonals, \( k_u \), within the band of \( A \).

*Constraint:* \( ku \geq 0 \).

8: \( ab[dim] \) – Complex

*Input/Output*

*Note:* the dimension, \( dim \), of the array \( ab \) must be at least

\[
\max(1, pdab \times n) \quad \text{when} \quad order = \text{Nag ColMajor};
\]

\[
\max(1, m \times pdab) \quad \text{when} \quad order = \text{Nag RowMajor}.
\]

*On entry:* the original \( m \) by \( n \) band matrix \( A \).

This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements \( A_{ij} \), for row \( i = 1, \ldots, m \) and column \( j = \max(1, i - k_l), \ldots, \min(n, i + k_u) \), depends on the \( order \) argument as follows:

- if \( order = \text{Nag ColMajor} \), \( A_{ij} \) is stored as \( ab[(j - 1) \times pdab + ku + i - j] \);
- if \( order = \text{Nag RowMajor} \), \( A_{ij} \) is stored as \( ab[(i - 1) \times pdab + kl + j - i] \).

*On exit:* \( ab \) is overwritten by values generated during the reduction.

9: \( pdab \) – Integer

*Input*

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

*Constraint:* \( pdab \geq kl + ku + 1 \).

10: \( d[min(m, n)] \) – double

*Output*

*On exit:* the diagonal elements of the bidiagonal matrix \( B \).

11: \( e[min(m, n) - 1] \) – double

*Output*

*On exit:* the superdiagonal elements of the bidiagonal matrix \( B \).
12: $q[dim]$ – Complex

**Note**: the dimension, $dim$, of the array $q$ must be at least $\max(1, pdq \times m)$ when $\text{vect} = \text{Nag\_FormQ}$ or $\text{Nag\_FormBoth}$; 1 otherwise.

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

\[
q[(j - 1) \times pdq + i - 1] \text{ when } order = \text{Nag\_ColMajor}; \\
q[(i - 1) \times pdq + j - 1] \text{ when } order = \text{Nag\_RowMajor}.
\]

**On exit**: if $\text{vect} = \text{Nag\_FormQ}$ or $\text{Nag\_FormBoth}$, contains the $m$ by $m$ unitary matrix $Q$.

If $\text{vect} = \text{Nag\_DoNotForm}$ or $\text{Nag\_FormP}$, $q$ is not referenced.

13: $pdq$ – Integer

**Input**

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

**Constraints**:

- if $\text{vect} = \text{Nag\_FormQ}$ or $\text{Nag\_FormBoth}$, $pdq \geq \max(1, m)$;
- otherwise $pdq \geq 1$.

14: $pt[dim]$ – Complex

**Output**

**Note**: the dimension, $dim$, of the array $pt$ must be at least $\max(1, pdpt \times n)$ when $\text{vect} = \text{Nag\_FormP}$ or $\text{Nag\_FormBoth}$; 1 otherwise.

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

\[
pt[(j - 1) \times pdpt + i - 1] \text{ when } order = \text{Nag\_ColMajor}; \\
pt[(i - 1) \times pdpt + j - 1] \text{ when } order = \text{Nag\_RowMajor}.
\]

**On exit**: the $n$ by $n$ unitary matrix $P^H$, if $\text{vect} = \text{Nag\_FormP}$ or $\text{Nag\_FormBoth}$. If $\text{vect} = \text{Nag\_DoNotForm}$ or $\text{Nag\_FormQ}$, $pt$ is not referenced.

15: $pdpt$ – Integer

**Input**

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

**Constraints**:

- if $\text{vect} = \text{Nag\_FormP}$ or $\text{Nag\_FormBoth}$, $pdpt \geq \max(1, n)$;
- otherwise $pdpt \geq 1$.

16: $c[dim]$ – Complex

**Input/Output**

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

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

\[
c[(j - 1) \times pdc + i - 1] \text{ when } order = \text{Nag\_ColMajor}; \\
c[(i - 1) \times pdc + j - 1] \text{ when } order = \text{Nag\_RowMajor}.
\]

**On entry**: an $m$ by $n$ matrix $C$.

**On exit**: $c$ is overwritten by $Q^HC$. If $ncc = 0$, $c$ is not referenced.
17:  **pdc** – Integer  
*Input*  
*On entry:* the stride separating row or column elements (depending on the value of *order*) in the array *c.*  

*Constraints:*  
if *order* = Nag.ColMajor,  
if *ncc* > 0, *pdc* $\geq$ max(1, *m*);  
if *ncc* = 0, *pdc* $\geq$ 1;  
if *order* = Nag.RowMajor, *pdc* $\geq$ max(1, *ncc*).

18:  **fail** – NagError *  
*Input/Output*  
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

**NE_ALLOC_FAIL**  
Dynamic memory allocation failed.  
See Section 3.2.1.2 in the Essential Introduction for further information.

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

**NE_ENUM_INT_2**  
On entry, *vect* = *value*, *pdpt* = *value* and *n* = *value*.  
Constraint: if *vect* = Nag_FormP or Nag_FormBoth, *pdpt* $\geq$ max(1, *n*);  
otherwise *pdpt* $\geq$ 1.  
On entry, *vect* = *value*, *pdq* = *value* and *m* = *value*.  
Constraint: if *vect* = Nag_FormQ or Nag_FormBoth, *pdq* $\geq$ max(1, *m*);  
otherwise *pdq* $\geq$ 1.

**NE_INT**  
On entry, *kl* = *value*.  
Constraint: *kl* $\geq$ 0.  
On entry, *ku* = *value*.  
Constraint: *ku* $\geq$ 0.  
On entry, *m* = *value*.  
Constraint: *m* $\geq$ 0.  
On entry, *n* = *value*.  
Constraint: *n* $\geq$ 0.  
On entry, *ncc* = *value*.  
Constraint: *ncc* $\geq$ 0.  
On entry, *pdab* = *value*.  
Constraint: *pdab* $>$ 0.  
On entry, *pdc* = *value*.  
Constraint: *pdc* $>$ 0.  
On entry, *pdpt* = *value*.  
Constraint: *pdpt* $>$ 0.  
On entry, *pdq* = *value*.  
Constraint: *pdq* $>$ 0.
NE_INT_2
On entry, \( pdc = \langle \text{value} \rangle \) and \( ncc = \langle \text{value} \rangle \).
Constraint: \( pdc \geq \max(1, ncc) \).

NE_INT_3
On entry, \( ncc = \langle \text{value} \rangle \), \( pdc = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).
Constraint: if \( ncc > 0 \), \( pdc \geq \max(1, m) \);
if \( ncc = 0 \), \( pdc \geq 1 \).
On entry, \( pdab = \langle \text{value} \rangle \), \( kl = \langle \text{value} \rangle \) and \( ku = \langle \text{value} \rangle \).
Constraint: \( pdab \geq kl + ku + 1 \).

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 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy
The computed bidiagonal form \( B \) satisfies \( QBP^H = A + E \), where
\[
\|E\|_2 \leq c(n)\epsilon \|A\|_2,
\]
c\( (n) \) is a modestly increasing function of \( n \), and \( \epsilon \) is the \textit{machine precision}.
The elements of \( B \) themselves may be sensitive to small perturbations in \( A \) or to rounding errors in the computation, but this does not affect the stability of the singular values and vectors.
The computed matrix \( Q \) differs from an exactly unitary matrix by a matrix \( F \) such that
\[
\|F\|_2 = O(\epsilon).
\]
A similar statement holds for the computed matrix \( P^H \).

8 Parallelism and Performance
nag_zgbbrd (f08lsc) is not threaded by NAG in any implementation.
nag_zgbbrd (f08lsc) 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 the sum of:
\[20n^2k, \text{ if } \text{vect} = \text{Nag_DoNotForm} \text{ and } ncc = 0,\]\[10n^2nC(k-1)/k, \text{ if } C \text{ is updated, and}\]\[10n^3(k-1)/k, \text{ if either } Q \text{ or } P^H \text{ is generated (double this if both)},\]
where \( k = k_l + k_u \), assuming \( n \gg k \). For this section we assume that \( m = n \).

The real analogue of this function is \texttt{nag_dgbbrd (f08lec)}.

### 10 Example

This example reduces the matrix \( A \) to upper bidiagonal form, where

\[
A = \begin{pmatrix}
  0.96 - 0.8i & -0.03 + 0.96i & 0.00 + 0.00i & 0.00 + 0.00i \\
-0.98 + 1.98i & -1.20 + 0.19i & -0.66 + 0.42i & 0.00 + 0.00i \\
0.62 - 0.46i & 1.01 + 0.02i & 0.63 - 0.17i & -1.11 + 0.60i \\
0.00 + 0.00i & 0.19 - 0.54i & -0.98 - 0.36i & 0.22 - 0.20i \\
0.00 + 0.00i & 0.00 + 0.00i & -0.17 - 0.46i & 1.47 + 1.59i \\
0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i & 0.26 + 0.26i \\
\end{pmatrix}.
\]

#### 10.1 Program Text

/* \texttt{nag_zgbbrd (f08lsc)} Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 7, 2001. */

```c
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>

int main(void)
{
    /* Scalars */
    Integer i, j, kl, ku, m, n, ncc, pdab, pdc, pdq, pdpt;
    Integer d_len, e_len;
    Integer exit_status = 0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    Complex *ab = 0, *c = 0, *pt = 0, *q = 0;
    double *d = 0, *e = 0;

    #ifdef NAG_COLUMN_MAJOR
    #define AB(I, J) ab[(J - 1) * pdab + ku + I - J]
    order = Nag_ColMajor;
    #else
    #define AB(I, J) ab[(I - 1) * pdab + kl + J - I]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    printf("nag_zgbbrd (f08lsc) Example Program Results\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n ] ");
    #else
    scanf("%*[\n ] ");
    #endif
    #ifdef _WIN32
    scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\n ] ",
            &m, &n, &kl, &ku, &ncc);
    #else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\n ] ",
            &m, &n, &kl, &ku, &ncc);
    #endif
    #ifdef NAG_COLUMN_MAJOR
    pdab = kl + ku + 1;
    #endif
}
```
# else
pdab = kl + ku + 1;
pdq = m;
pdpt = n;
pdc = MAX(1, ncc);
#endif

d_len = MIN(m, n);
e_len = MIN(m, n) - 1;

/* Allocate memory */
if (!(ab = NAG_ALLOC((kl+ku+1) * m, Complex)) ||
    !(c = NAG_ALLOC(m * MAX(1, ncc), Complex)) ||
    !(d = NAG_ALLOC(d_len, double)) ||
    !(e = NAG_ALLOC(e_len, double)) ||
    !(pt = NAG_ALLOC(n * n, Complex)) ||
    !(q = NAG_ALLOC(m * m, Complex)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read A from data file */
for (i = 1; i <= m; ++i)
{
    for (j = MAX(1, i - kl); j <= MIN(n, i + ku); ++j)
        #ifdef _WIN32
            scanf_s("( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
        #else
            scanf("( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
        #endif
}
#ifdef _WIN32
    scanf_s("%*[^
] ");
#else
    scanf("%*[^
] ");
#endif

/* Reduce A to bidiagonal form */
/* nag_zgbbrd (f08lsc). */
/* Reduction of complex rectangular band matrix to upper */
/* bidiagonal form */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zgbbrd (f08lsc).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print bidiagonal form */
printf("\nDiagonal\n");
for (i = 1; i <= MIN(m, n); ++i)
    printf("%9.4f\n", d[i-1], i%8 == 0?"\n": "");
if (m >= n)
    printf("\nSuper-diagonal\n");
else
    printf("\nSub-diagonal\n");
for (i = 1; i <= MIN(m, n) - 1; ++i)
    printf("%9.4f\n", e[i-1], i%8 == 0?"\n": "");
printf("\n");

END:
NAG_FREE(ab);
NAG_FREE(c);
NAG_FREE(d);
NAG_FREE(e);
NAG_FREE(pt);
NAG_FREE(q);

    return exit_status;
}

10.2 Program Data

nag_zgbbrd (f08lsc) Example Program Data

6 4 2 1 0 : Values of M, N, KL, KU and NCC
( 0.96, -0.81) (-0.03, 0.96)
(-0.98, 1.98) (-1.20, 0.19) (-0.66, 0.42)
( 0.62, -0.46) ( 1.01, 0.02) ( 0.63, -0.17) (-1.11, 0.60)
( 0.19, -0.54) (-0.98, -0.36) ( 0.22, -0.20)
(-0.17, -0.46) ( 1.47, 1.59) : End of matrix A

10.3 Program Results

nag_zgbbrd (f08lsc) Example Program Results

Diagonal
2.6560 1.7501 2.0607 0.8658
Super-diagonal
1.7033 1.2800 0.1467