NAG Library Function Document

nag_zhbgvd (f08uqc)

1 Purpose
nag_zhbgvd (f08uqc) computes all the eigenvalues and, optionally, the eigenvectors of a complex
geneneralized Hermitian-definite banded eigenproblem, of the form

\[ Az = \lambda Bz, \]

where \( A \) and \( B \) are Hermitian and banded, and \( B \) is also positive definite. If eigenvectors are desired, it
uses a divide-and-conquer algorithm.

2 Specification

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

void nag_zhbgvd (Nag_OrderType order, Nag_JobType job, Nag_UploType uplo,
                 Integer n, Integer ka, Integer kb, Complex ab[], Integer pdab,
                 Complex bb[], Integer pdbb, double w[], Complex z[], Integer pdz,
                 NagError *fail)
```

3 Description

The generalized Hermitian-definite band problem

\[ Az = \lambda Bz \]

is first reduced to a standard band Hermitian problem

\[ Cx = \lambda x, \]

where \( C \) is a Hermitian band matrix, using Wilkinson’s modification to Crawford’s algorithm (see
Crawford (1973) and Wilkinson (1977)). The Hermitian eigenvalue problem is then solved for the
eigenvalues and the eigenvectors, if required, which are then backtransformed to the eigenvectors of the
original problem.

The eigenvectors are normalized so that the matrix of eigenvectors, \( Z \), satisfies

\[ Z^H AZ = A \quad \text{and} \quad Z^H BZ = I, \]

where \( A \) is the diagonal matrix whose diagonal elements are the eigenvalues.

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A,
Philadelphia http://www.netlib.org/lapack/lug

Crawford C R (1973) Reduction of a band-symmetric generalized eigenvalue problem Comm. ACM 16
41–44

Press, Baltimore

5 Arguments

1: **order** – Nag_OrderType

*Input*

*On entry:* the **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 **order** = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

**Constraint:** **order** = Nag_RowMajor or Nag_ColMajor.

2: **job** – Nag_JobType

*Input*

*On entry:* indicates whether eigenvectors are computed.

**job** = Nag_EigVals

Only eigenvalues are computed.

**job** = Nag_DoBoth

Eigenvalues and eigenvectors are computed.

**Constraint:** **job** = Nag_EigVals or Nag_DoBoth.

3: **uplo** – Nag_UploType

*Input*

*On entry:* if **uplo** = Nag_Upper, the upper triangles of $A$ and $B$ are stored. If **uplo** = Nag_Lower, the lower triangles of $A$ and $B$ are stored.

**Constraint:** **uplo** = Nag_Upper or Nag_Lower.

4: **n** – Integer

*Input*

*On entry:* $n$, the order of the matrices $A$ and $B$.

**Constraint:** $n \geq 0$.

5: **ka** – Integer

*Input*

*On entry:* if **uplo** = Nag_Upper, the number of superdiagonals, $k_a$, of the matrix $A$. If **uplo** = Nag_Lower, the number of subdiagonals, $k_a$, of the matrix $A$.

**Constraint:** **ka** $\geq 0$.

6: **kb** – Integer

*Input*

*On entry:* if **uplo** = Nag_Upper, the number of superdiagonals, $k_b$, of the matrix $B$. If **uplo** = Nag_Lower, the number of subdiagonals, $k_b$, of the matrix $B$.

**Constraint:** **ka** $\geq$ **kb** $\geq 0$.

7: **ab[dim]** – Complex

*Input/Output*

**Note:** the dimension, **dim**, of the array **ab** must be at least $\max(1, \text{pdab} \times n)$.

*On entry:* the upper or lower triangle of the $n$ by $n$ Hermitian band matrix $A$.

This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements of $A_{ij}$, depends on the **order** and **uplo** arguments as follows:

if **order** = Nag_ColMajor and **uplo** = Nag_Upper,

$A_{ij}$ is stored in $\text{ab}[k_a + i - j + (j-1) \times \text{pdab}]$, for $j = 1, \ldots, n$ and $i = \max(1, j - k_a), \ldots, j$;

if **order** = Nag_ColMajor and **uplo** = Nag_Lower,

$A_{ij}$ is stored in $\text{ab}[i - j + (j-1) \times \text{pdab}]$, for $j = 1, \ldots, n$ and $i = j, \ldots, \min(n, j + k_a)$;
if order = Nag_RowMajor and uplo = Nag_Upper,
  \( A_{ij} \) is stored in \( ab[j-i+(i-1) \times pdab], \) for \( i = 1, \ldots, n \) and
  \( j = i, \ldots, \min(n, i+k_a); \)
if order = Nag_RowMajor and uplo = Nag_Lower,
  \( A_{ij} \) is stored in \( ab[k_a+j-i+(i-1) \times pdab], \) for \( i = 1, \ldots, n \) and
  \( j = \max(1, i-k_a), \ldots, i. \)

On exit: the contents of \( ab \) are overwritten.

8: \( pdab \) – Integer

\textit{Input}

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

Constraint: \( pdab \geq ka + 1. \)

9: \( bb[dim] \) – Complex

\textit{Input/Output}

Note: the dimension, \( dim, \) of the array \( bb \) must be at least \( \max(1, \texttt{pdbb} \times n). \)

On entry: the upper or lower triangle of the \( n \) by \( n \) Hermitian band matrix \( B. \)

This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements of \( B_{ij}, \) depends on the \texttt{order} and \texttt{uplo} arguments as follows:

if order = Nag_ColMajor and uplo = Nag_Upper,
  \( B_{ij} \) is stored in \( bb[k_b+i-j+(j-1) \times pdbb], \) for \( j = 1, \ldots, n \) and
  \( i = \max(1, j-k_b), \ldots, j; \)
if order = Nag_ColMajor and uplo = Nag_Lower,
  \( B_{ij} \) is stored in \( bb[i-j+(j-1) \times pdbb], \) for \( j = 1, \ldots, n \) and
  \( i = j, \ldots, \min(n, j+k_b); \)
if order = Nag_RowMajor and uplo = Nag_Upper,
  \( B_{ij} \) is stored in \( bb[j-i+(i-1) \times pdab], \) for \( i = 1, \ldots, n \) and
  \( j = \max(1, i-k_b), \ldots, i; \)
if order = Nag_RowMajor and uplo = Nag_Lower,
  \( B_{ij} \) is stored in \( bb[k_b+j-i+(i-1) \times pdab], \) for \( i = 1, \ldots, n \) and
  \( j = \max(1, i-k_b), \ldots, i. \)

On exit: the factor \( S \) from the split Cholesky factorization \( B = S^H S, \) as returned by \texttt{nag_zpbstf} (f08utc).

10: \( pdbb \) – Integer

\textit{Input}

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

Constraint: \( pdbb \geq kb + 1. \)

11: \( w[n] \) – double

\textit{Output}

On exit: the eigenvalues in ascending order.

12: \( z[dim] \) – Complex

\textit{Output}

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

\[
\max(1, \texttt{pdz} \times n) \text{ when } \texttt{job} = \text{Nag_DoBoth};
1 \text{ otherwise.}
\]

The \((i,j)\)th element of the matrix \( Z \) is stored in

\[
z[(j-1) \times \texttt{pdz} + i - 1] \text{ when } \texttt{order} = \text{Nag_ColMajor};
z[(i-1) \times \texttt{pdz} + j - 1] \text{ when } \texttt{order} = \text{Nag_RowMajor}.\]
On exit: if job = Nag_DoBoth, z contains the matrix Z of eigenvectors, with the ith column of Z holding the eigenvector associated with w[i-1]. The eigenvectors are normalized so that $Z^H B Z = I$.

If job = Nag_EigVals, z is not referenced.

13: pdz – Integer

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

Constraints:
if job = Nag_DoBoth, $pdz \geq \max(1, n)$; otherwise $pdz \geq 1$.

14: fail – NagError*

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_CONVERGENCE
The algorithm failed to converge; (value) off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

NE_ENUM_INT_2
On entry, job = (value), pdz = (value) and n = (value).
Constraint: if job = Nag_DoBoth, $pdz \geq \max(1, n)$; otherwise $pdz \geq 1$.

NE_INT
On entry, ka = (value).
Constraint: ka $\geq 0$.

On entry, n = (value).
Constraint: n $\geq 0$.

On entry, pdab = (value).
Constraint: pdab $> 0$.

On entry, pdbb = (value).
Constraint: pdbb $> 0$.

On entry, pdz = (value).
Constraint: pdz $> 0$.

NE_INT_2
On entry, ka = (value) and kb = (value).
Constraint: ka $\geq kb \geq 0$.

On entry, pdab = (value) and ka = (value).
Constraint: pdab $\geq ka + 1$. 
On entry, \( \mathbf{pdbb} = \langle \text{value} \rangle \) and \( \mathbf{kb} = \langle \text{value} \rangle \).

Constraint: \( \mathbf{pdbb} \geq \mathbf{kb} + 1 \).

**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_MAT_NOT_POS_DEF**

If \( \text{fail.errnum} = n + \langle \text{value} \rangle \), for \( 1 \leq \langle \text{value} \rangle \leq n \), then \( \text{nag_zpbsdf} (\text{f08utc}) \) returned \( \text{fail.errnum} = \langle \text{value} \rangle \): \( B \) is not positive definite. The factorization of \( B \) could not be completed and no eigenvalues or eigenvectors were computed.

**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

If \( B \) is ill-conditioned with respect to inversion, then the error bounds for the computed eigenvalues and vectors may be large, although when the diagonal elements of \( B \) differ widely in magnitude the eigenvalues and eigenvectors may be less sensitive than the condition of \( B \) would suggest. See Section 4.10 of Anderson et al. (1999) for details of the error bounds.

8 Parallelism and Performance

\( \text{nag_zhbgvd} (\text{f08uqc}) \) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\( \text{nag_zhbgvd} (\text{f08uqc}) \) 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 floating-point operations is proportional to \( n^3 \) if \( \text{job} = \text{Nag_DoBoth} \) and, assuming that \( n \gg k_n \), is approximately proportional to \( n^2k_n \) otherwise.

The real analogue of this function is \( \text{nag_dsbkgvd} (\text{f08ucc}) \).

10 Example

This example finds all the eigenvalues of the generalized band Hermitian eigenproblem \( A z = \lambda B z \), where

\[
A = \begin{pmatrix}
-1.13 & 1.94 - 2.10i & -1.40 + 0.25i & 0 \\
1.94 + 2.10i & -1.91 & -0.82 - 0.89i & -0.67 + 0.34i \\
-1.40 - 0.25i & -0.82 + 0.89i & -1.87 & -1.10 - 0.16i \\
0 & -0.67 - 0.34i & -1.10 + 0.16i & 0.50
\end{pmatrix}
\]

and
\[ B = \begin{pmatrix} 9.89 & 1.08 - 1.73i & 0 & 0 \\ 1.08 + 1.73i & 1.69 & -0.04 + 0.29i & 0 \\ 0 & -0.04 - 0.29i & 2.65 & -0.33 + 2.24i \\ 0 & 0 & -0.33 - 2.24i & 2.17 \end{pmatrix}. \]

### 10.1 Program Text

```c
/* nag_zhbgvd (f08uqc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */

#include <stdio.h>
#include <nag.h>
#include <nagf08.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer i, j, ka, kb, n, pdab, pdbb, pdz, zsize;
    Integer exit_status = 0;
    /* Arrays */
    Complex *ab = 0, *bb = 0, *z = 0;
    double *w = 0;
    char nag_enum_arg[40];
    /* Nag Types */
    NagError fail;
    Nag_UploType uplo;
    Nag_OrderType order;
    Nag_JobType job;

    #ifdef NAG_COLUMN_MAJOR
    #define AB_UPPER(I, J) ab[(J-1)*pdab + ka + I-J]
    #define AB_LOWER(I, J) ab[(J-1)*pdab + I - J]
    #define BB_UPPER(I, J) bb[(J-1)*pdbb + kb + I-J]
    #define BB_LOWER(I, J) bb[(J-1)*pdbb + I - J]
    order = Nag_ColMajor;
    #else
    #define AB_UPPER(I, J) ab[(I-1)*pdab + J - I]
    #define AB_LOWER(I, J) ab[(I-1)*pdab + ka + J - I]
    #define BB_UPPER(I, J) bb[(I-1)*pdbb + kb + J - I]
    #define BB_LOWER(I, J) bb[(I-1)*pdbb + J - I]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);

    printf("nag_zhbgvd (f08uqc) Example Program Results\n\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"%*[\n]", &n, &ka, &kb);
    #else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\n]", &n, &ka, &kb);
    #endif
    if (n < 0 || ka < kb || kb < 0)
    {
        printf("Invalid n, ka or kb\n");
    }
}
```
```c
exit_status = 1;
goto END;
#endif
} #ifdef _WIN32
scanf_s(" %39s[\n"]", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf(" %39s[\n"]", nag_enum_arg);
#endif

/* nag_enum_name_to_value (x04nac).
* Converts NAG enum member name to value
*/
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
scanf_s(" %39s[\n"]", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf(" %39s[\n"]", nag_enum_arg);
#endif
#endif
job = (Nag_JobType) nag_enum_name_to_value(nag_enum_arg);
if (job==Nag_EigVals) { 
zsize = 1;
pdz = 1;
} else { 
zsize = n*n;
pdz = n;
}

pdab = ka + 1;
pdbb = kb + 1;

/* Allocate memory */
if (! (ab = NAG_ALLOC((ka+1) * n, Complex)) ||
(! (bb = NAG_ALLOC((kb+1) * n, Complex)) ||
(! (z = NAG_ALLOC(zsize, Complex)) ||
(! (w = NAG_ALLOC(n, double)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}

/* Read the triangular parts of the matrices A and B from data file */
if (uplo == Nag_Upper) {
for (i = 1; i <= n; ++i) 
for (j = i; j <= MIN(i+ka, n); ++j)
#ifndef _WIN32
scanf(" ( %lf , %lf )", &AB_UPPER(i, j).re, &AB_UPPER(i, j).im);
#else
scanf(" ( %lf , %lf )", &AB_UPPER(i, j).re, &AB_UPPER(i, j).im);
#endif
#endif
#ifndef _WIN32
scanf("%*[\n"]");
#else
scanf("%*[\n"]");
#endif
#endif
#ifndef _WIN32
scanf("%*[\n"]");
#else
scanf("%*[\n"]");
#endif
#endif
#ifndef _WIN32
for (i = 1; i <= n; ++i) 
for (j = MAX(1, i-ka); j <= i; ++j)
#else
for (i = 1; i <= n; ++i) 
for (j = MAX(1, i-ka); j <= i; ++j)
#endif
#ifndef _WIN32
scanf(" ( %lf , %lf )", &AB_LOWER(i, j).re, &AB_LOWER(i, j).im);
#else
scanf(" ( %lf , %lf )", &AB_LOWER(i, j).re, &AB_LOWER(i, j).im);
#endif
#endif
```

(f08uqc.7)
#ifdef _WIN32
    scanf_s("%*[\n"]);
#else
    scanf("%*[\n"]);
#endif
for (i = 1; i <= n; ++i)
    for (j = MAX(1, i-kb); j <= i; ++j)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )", &BB_LOWER(i, j).re, &BB_LOWER(i, j).im);
#else
    scanf(" ( %lf , %lf )", &BB_LOWER(i, j).re, &BB_LOWER(i, j).im);
#endif
}
#ifdef _WIN32
    scanf_s("%*[\n"]);
#else
    scanf("%*[\n"]);
#endif
/* Solve the generalized Hermitian band eigenvalue problem A*x = lambda*B*x
 * using nag_zhbgvd (f08uqc).
 */
  if (fail.code != NE_NOERROR)
    {
    printf("Error from nag_zhbgvd (f08uqc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
    }
    /* Print eigensolution */
    printf(" Eigenvalues\n ");
    for (j = 0; j < n; ++j) printf(" %10.4f%s", w[j], j%6 == 5?"\n":" ");
    printf("\n");
    if (job==Nag_DoBoth) {
    /* nag_gen_complx_mat_print (x04dac): Print Matrix of eigenvectors Z. */
     printf("\n");
     fflush(stdout);
     nag_gen_complex_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
                              z, pdz, "Eigenvectors", 0, &fail);
     if (fail.code != NE_NOERROR)
       {
         printf("Error from nag_gen_complex_mat_print (x04dac).\n%s\n", fail.message);
         exit_status = 1;
       }
    }
END:
  NAG_FREE(ab);
  NAG_FREE(bb);
  NAG_FREE(z);
  NAG_FREE(w);
  return exit_status;
}

10.2 Program Data

nag_zhbgvd (f08uqc) Example Program Data

4  2  1
  Nag_Upper  : n, ka and kb
  Nag_EigVals  : uplo
(-1.13, 0.00) ( 1.94,-2.10) (-1.40, 0.25)
(-1.91, 0.00) (-0.82,-0.89) (-0.67, 0.34)
(-1.87, 0.00) (-1.10,-1.16)
( 0.50, 0.00) : matrix A
10.3 Program Results

nag_zhbgvd (f08uqc) Example Program Results

<table>
<thead>
<tr>
<th>Eigenvalues</th>
<th>-6.6089</th>
<th>-2.0416</th>
<th>0.1603</th>
<th>1.7712</th>
</tr>
</thead>
</table>

(matrix B)