

## NAG Library Function Document

### nag\_zhbgvx (f08upc)

#### 1 Purpose

nag\_zhbgvx (f08upc) computes selected the eigenvalues and, optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem, of the form

$$Az = \lambda Bz,$$

where  $A$  and  $B$  are Hermitian and banded, and  $B$  is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either all eigenvalues, a range of values or a range of indices for the desired eigenvalues.

#### 2 Specification

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

void nag_zhbgvx (Nag_OrderType order, Nag_JobType job, Nag_RangeType range,
                Nag_UploType uplo, Integer n, Integer ka, Integer kb, Complex ab[],
                Integer pdab, Complex bb[], Integer pddb, Complex q[], Integer pdq,
                double vl, double vu, Integer il, Integer iu, double abstol, Integer *m,
                double w[], Complex z[], Integer pdz, Integer jfail[], 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 required eigenvalues and eigenvectors, and the eigenvectors are then backtransformed to the eigenvectors of the original problem.

The eigenvectors are normalized so that

$$z^H A z = \lambda \quad \text{and} \quad z^H B z = 1.$$

#### 4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia <http://www.netlib.org/lapack/lug>

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

Demmel J W and Kahan W (1990) Accurate singular values of bidiagonal matrices *SIAM J. Sci. Statist. Comput.* **11** 873–912

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

Wilkinson J H (1977) Some recent advances in numerical linear algebra *The State of the Art in Numerical Analysis* (ed D A H Jacobs) Academic Press

## 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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: **range** – Nag\_RangeType *Input*  
*On entry:* if **range** = Nag\_AllValues, all eigenvalues will be found.  
 If **range** = Nag\_Interval, all eigenvalues in the half-open interval (**vl**, **vu**] will be found.  
 If **range** = Nag\_Indices, the **ilth** to **iuth** eigenvalues will be found.  
*Constraint:* **range** = Nag\_AllValues, Nag\_Interval or Nag\_Indices.
- 4: **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.
- 5: **n** – Integer *Input*  
*On entry:* *n*, the order of the matrices *A* and *B*.  
*Constraint:* **n** ≥ 0.
- 6: **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** ≥ 0.
- 7: **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** ≥ **kb** ≥ 0.
- 8: **ab**[*dim*] – Complex *Input/Output*  
**Note:** the dimension, *dim*, of the array **ab** must be at least  $\max(1, \mathbf{pdab} \times \mathbf{n})$ .  
*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 **ab** $[k_a + i - j + (j - 1) \times \text{pdab}]$ , for  $j = 1, \dots, n$  and  
 $i = \max(1, j - k_a), \dots, j$ ;  
 if **order** = Nag\_ColMajor and **uplo** = Nag\_Lower,  
 $A_{ij}$  is stored in **ab** $[i - j + (j - 1) \times \text{pdab}]$ , for  $j = 1, \dots, n$  and  
 $i = j, \dots, \min(n, j + k_a)$ ;  
 if **order** = Nag\_RowMajor and **uplo** = Nag\_Upper,  
 $A_{ij}$  is stored in **ab** $[j - i + (i - 1) \times \text{pdab}]$ , for  $i = 1, \dots, n$  and  
 $j = i, \dots, \min(n, i + k_a)$ ;  
 if **order** = Nag\_RowMajor and **uplo** = Nag\_Lower,  
 $A_{ij}$  is stored in **ab** $[k_a + j - i + (i - 1) \times \text{pdab}]$ , for  $i = 1, \dots, n$  and  
 $j = \max(1, i - k_a), \dots, i$ .

*On exit:* the contents of **ab** are overwritten.

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**  $\geq$  **ka** + 1.

10: **bb** $[dim]$  – Complex *Input/Output*

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

*On entry:* the upper or lower triangle of the  $n$  by  $n$  Hermitian positive definite 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 **order** and **uplo** arguments as follows:

if **order** = Nag\_ColMajor and **uplo** = Nag\_Upper,  
 $B_{ij}$  is stored in **bb** $[k_b + i - j + (j - 1) \times \text{pdbb}]$ , for  $j = 1, \dots, n$  and  
 $i = \max(1, j - k_b), \dots, j$ ;  
 if **order** = Nag\_ColMajor and **uplo** = Nag\_Lower,  
 $B_{ij}$  is stored in **bb** $[i - j + (j - 1) \times \text{pdbb}]$ , for  $j = 1, \dots, n$  and  
 $i = j, \dots, \min(n, j + k_b)$ ;  
 if **order** = Nag\_RowMajor and **uplo** = Nag\_Upper,  
 $B_{ij}$  is stored in **bb** $[j - i + (i - 1) \times \text{pdbb}]$ , for  $i = 1, \dots, n$  and  
 $j = i, \dots, \min(n, i + k_b)$ ;  
 if **order** = Nag\_RowMajor and **uplo** = Nag\_Lower,  
 $B_{ij}$  is stored in **bb** $[k_b + j - i + (i - 1) \times \text{pdbb}]$ , for  $i = 1, \dots, n$  and  
 $j = \max(1, i - k_b), \dots, i$ .

*On exit:* the factor  $S$  from the split Cholesky factorization  $B = S^H S$ , as returned by nag\_zpbstf (f08utc).

11: **pdbb** – Integer *Input*

*On entry:* the stride separating row or column elements (depending on the value of **order**) of the matrix  $B$  in the array **bb**.

*Constraint:* **pdbb**  $\geq$  **kb** + 1.

- 12: **q**[*dim*] – Complex *Output*
- Note:** the dimension, *dim*, of the array **q** must be at least  
 $\max(1, \mathbf{pdq} \times \mathbf{n})$  when **job** = Nag\_DoBoth;  
 1 otherwise.
- The (*i, j*)th element of the matrix *Q* is stored in  
 $\mathbf{q}[(j-1) \times \mathbf{pdq} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{q}[(i-1) \times \mathbf{pdq} + j - 1]$  when **order** = Nag\_RowMajor.
- On exit:* if **job** = Nag\_DoBoth, the *n* by *n* matrix, *Q* used in the reduction of the standard form, i. e.,  $Cx = \lambda x$ , from symmetric banded to tridiagonal form.
- If **job** = Nag\_EigVals, **q** is not referenced.
- 13: **pdq** – Integer *Input*
- On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **q**.
- Constraints:*  
 if **job** = Nag\_DoBoth,  $\mathbf{pdq} \geq \max(1, \mathbf{n})$ ;  
 otherwise  $\mathbf{pdq} \geq 1$ .
- 14: **vl** – double *Input*
- 15: **vu** – double *Input*
- On entry:* if **range** = Nag\_Interval, the lower and upper bounds of the interval to be searched for eigenvalues.
- If **range** = Nag\_AllValues or Nag\_Indices, **vl** and **vu** are not referenced.
- Constraint:* if **range** = Nag\_Interval,  $\mathbf{vl} < \mathbf{vu}$ .
- 16: **il** – Integer *Input*
- 17: **iu** – Integer *Input*
- On entry:* if **range** = Nag\_Indices, the indices (in ascending order) of the smallest and largest eigenvalues to be returned.
- If **range** = Nag\_AllValues or Nag\_Interval, **il** and **iu** are not referenced.
- Constraints:*  
 if **range** = Nag\_Indices and  $\mathbf{n} = 0$ ,  $\mathbf{il} = 1$  and  $\mathbf{iu} = 0$ ;  
 if **range** = Nag\_Indices and  $\mathbf{n} > 0$ ,  $1 \leq \mathbf{il} \leq \mathbf{iu} \leq \mathbf{n}$ .
- 18: **abstol** – double *Input*
- On entry:* the absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval  $[a, b]$  of width less than or equal to  

$$\mathbf{abstol} + \epsilon \max(|a|, |b|),$$
 where  $\epsilon$  is the *machine precision*. If **abstol** is less than or equal to zero, then  $\epsilon \|T\|_1$  will be used in its place, where *T* is the tridiagonal matrix obtained by reducing *C* to tridiagonal form. Eigenvalues will be computed most accurately when **abstol** is set to twice the underflow threshold  $2 \times \text{nag\_real\_safe\_small\_number}()$ , not zero. If this function returns with **fail.code** = NE\_CONVERGENCE, indicating that some eigenvectors did not converge, try setting **abstol** to  $2 \times \text{nag\_real\_safe\_small\_number}()$ . See Demmel and Kahan (1990).
- 19: **m** – Integer \* *Output*
- On exit:* the total number of eigenvalues found.  $0 \leq \mathbf{m} \leq \mathbf{n}$ .

If **range** = Nag\_AllValues, **m** = **n**.

If **range** = Nag\_Indices, **m** = **iu** – **il** + 1.

20: **w[n]** – double *Output*

*On exit:* the eigenvalues in ascending order.

21: **z[dim]** – Complex *Output*

**Note:** the dimension, *dim*, of the array **z** must be at least

$\max(1, \mathbf{pdz} \times \mathbf{n})$  when **job** = Nag\_DoBoth;  
1 otherwise.

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

**z**[(*j* – 1) × **pdz** + *i* – 1] when **order** = Nag\_ColMajor;  
**z**[(*i* – 1) × **pdz** + *j* – 1] when **order** = Nag\_RowMajor.

*On exit:* if **job** = Nag\_DoBoth, **z** contains the matrix *Z* of eigenvectors, with the *i*th 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.

22: **pdz** – Integer *Input*

*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** ≥  $\max(1, \mathbf{n})$ ;  
otherwise **pdz** ≥ 1.

23: **jfail[dim]** – Integer *Output*

**Note:** the dimension, *dim*, of the array **jfail** must be at least  $\max(1, \mathbf{n})$ .

*On exit:* if **job** = Nag\_DoBoth, then

if **fail.code** = NE\_NOERROR, the first **m** elements of **jfail** are zero;

if **fail.code** = NE\_CONVERGENCE, **jfail** contains the indices of the eigenvectors that failed to converge.

If **job** = Nag\_EigVals, **jfail** is not referenced.

24: **fail** – NagError \* *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

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

**NE\_CONVERGENCE**

The algorithm failed to converge;  $\langle value \rangle$  eigenvectors did not converge. Their indices are stored in array **jfail**.

**NE\_ENUM\_INT\_2**

On entry, **job** =  $\langle value \rangle$ , **pdq** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: if **job** = Nag\_DoBoth, **pdq**  $\geq \max(1, \mathbf{n})$ ;  
otherwise **pdq**  $\geq 1$ .

On entry, **job** =  $\langle value \rangle$ , **pdz** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: if **job** = Nag\_DoBoth, **pdz**  $\geq \max(1, \mathbf{n})$ ;  
otherwise **pdz**  $\geq 1$ .

**NE\_ENUM\_INT\_3**

On entry, **range** =  $\langle value \rangle$ , **il** =  $\langle value \rangle$ , **iu** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: if **range** = Nag\_Indices and **n** = 0, **il** = 1 and **iu** = 0;  
if **range** = Nag\_Indices and **n** > 0,  $1 \leq \mathbf{il} \leq \mathbf{iu} \leq \mathbf{n}$ .

**NE\_ENUM\_REAL\_2**

On entry, **range** =  $\langle value \rangle$ , **vl** =  $\langle value \rangle$  and **vu** =  $\langle value \rangle$ .

Constraint: if **range** = Nag\_Interval, **vl** < **vu**.

**NE\_INT**

On entry, **ka** =  $\langle value \rangle$ .

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

On entry, **n** =  $\langle value \rangle$ .

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

On entry, **pdab** =  $\langle value \rangle$ .

Constraint: **pdab** > 0.

On entry, **pdbb** =  $\langle value \rangle$ .

Constraint: **pdbb** > 0.

On entry, **pdq** =  $\langle value \rangle$ .

Constraint: **pdq** > 0.

On entry, **pdz** =  $\langle value \rangle$ .

Constraint: **pdz** > 0.

**NE\_INT\_2**

On entry, **ka** =  $\langle value \rangle$  and **kb** =  $\langle value \rangle$ .

Constraint: **ka**  $\geq \mathbf{kb} \geq 0$ .

On entry, **pdab** =  $\langle value \rangle$  and **ka** =  $\langle value \rangle$ .

Constraint: **pdab**  $\geq \mathbf{ka} + 1$ .

On entry, **pdbb** =  $\langle value \rangle$  and **kb** =  $\langle value \rangle$ .

Constraint: **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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

**NE\_MAT\_NOT\_POS\_DEF**

If **fail.errnum** = **n** +  $\langle value \rangle$ , for  $1 \leq \langle value \rangle \leq \mathbf{n}$ , then `nag_zpbstf` (f08utc) returned **fail.errnum** =  $\langle value \rangle$ :  $B$  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 2.7.5 in How to Use the NAG Library and its Documentation 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**

`nag_zhbgvx` (f08upc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

`nag_zhbgvx` (f08upc) 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 **job** = Nag\_DoBoth and **range** = Nag\_AllValues, and assuming that  $n \gg k_a$ , is approximately proportional to  $n^2 k_a$  if **job** = Nag\_EigVals. Otherwise the number of floating-point operations depends upon the number of eigenvectors computed.

The real analogue of this function is `nag_dsbgvx` (f08ubc).

**10 Example**

This example finds the eigenvalues in the half-open interval  $(0.0, 2.0]$ , and corresponding eigenvectors, of the generalized band Hermitian eigenproblem  $Az = \lambda Bz$ , 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

```

/* nag_zhbgvx (f08upc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

int main(void)
{
    /* Scalars */
    double abstol, vl, vu;
    Integer exit_status = 0, il = 1, iu = 1;
    Integer i, j, ka, kb, m, n, pdab, pddb, pdq, pdz, zsize;

    /* Arrays */
    Complex *ab = 0, *bb = 0, *q = 0, *z = 0;
    double *w = 0;
    Integer *index = 0;
    char nag_enum_arg[40];

    /* Nag Types */
    NagError fail, failp;
    Nag_OrderType order;
    Nag_UploType uplo;
    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)*pddb + kb + I - J]
#define BB_LOWER(I, J) bb[(J-1)*pddb + 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)*pddb + J - I]
#define BB_LOWER(I, J) bb[(I-1)*pddb + kb + J - I]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);

    printf("nag_zhbgvx (f08upc) 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");
        exit_status = 1;
        goto END;
    }
}

```



```

#ifdef _WIN32
    scanf_s("%39s%*[\n]", nag_enum_arg, (unsigned)_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, (unsigned)_countof(nag_enum_arg));
#else
    scanf("%39s%*[\n]", nag_enum_arg);
#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;
pddb = kb + 1;
m = n;
pdq = n;
/* Allocate memory */
if (!(ab = NAG_ALLOC((ka + 1) * n, Complex)) ||
    !(bb = NAG_ALLOC((kb + 1) * n, Complex)) ||
    !(q = NAG_ALLOC(n * n, Complex)) ||
    !(z = NAG_ALLOC(zsize, Complex)) ||
    !(w = NAG_ALLOC(n, double)) || !(index = NAG_ALLOC(n, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read the lower and upper bounds of the interval to be searched. */
#ifdef _WIN32
    scanf_s("%lf%lf%*[\n]", &vl, &vu);
#else
    scanf("%lf%lf%*[\n]", &vl, &vu);
#endif

/* 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)
#ifdef _WIN32
            scanf_s(" ( %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
#ifdef _WIN32
            scanf_s("%*[\n]");
#else
            scanf("%*[\n]");
#endif
        for (i = 1; i <= n; ++i)
            for (j = i; j <= MIN(i + kb, n); ++j)
#ifdef _WIN32
                scanf_s(" ( %lf , %lf )", &BB_UPPER(i, j).re, &BB_UPPER(i, j).im);
#else
                scanf(" ( %lf , %lf )", &BB_UPPER(i, j).re, &BB_UPPER(i, j).im);
#endif
            }
        else {
            for (i = 1; i <= n; ++i)

```

```

        for (j = MAX(1, i - ka); j <= i; ++j)
#ifdef _WIN32
            scanf_s(" ( %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
#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
#endif

    /* Use the default absolute error tolerance for eigenvalues. */
    abstol = 0.;

    /* Solve the generalized symmetric eigenvalue problem  $A*x = \lambda*B*x$ 
     * using nag_zhbgvx (f08upc).
     */
    nag_zhbgvx(order, job, Nag_Interval, uplo, n, ka, kb, ab, pdab, bb, pddb, q,
               pdq, vl, vu, il, iu, abstol, &m, w, z, pdz, index, &fail);
    if (fail.code != NE_NOERROR && fail.code != NE_CONVERGENCE) {
        printf("Error from nag_zhbgvx (f08upc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Print eigensolution */
    printf("Number of eigenvalues found =%5" NAG_IFMT "\n\n", m);
    printf(" Eigenvalues\n  ");
    for (j = 0; j < m; ++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);
        INIT_FAIL(failp);
        nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m,
                                z, pdz, "Eigenvectors", 0, &failp);
        if (failp.code != NE_NOERROR) {
            printf("Error from nag_gen_complx_mat_print (x04dac).\n%s\n",
                  failp.message);
            exit_status = 1;
        }
        if (failp.code == NE_CONVERGENCE) {
            printf("eigenvectors failed to converge\n");
            printf("Indices of eigenvectors that did not converge\n  ");
            for (j = 0; j < m; ++j)
                printf("%8" NAG_IFMT "%s", index[j], j % 6 == 5 ? "\n" : "");
            printf("\n");
        }
    }
}
END:
    NAG_FREE(ab);
    NAG_FREE(bb);
    NAG_FREE(q);
    NAG_FREE(z);

```

```

NAG_FREE(w);
NAG_FREE(index);

return exit_status;
}

```

## 10.2 Program Data

nag\_zhbgvx (f08upc) Example Program Data

```

4           2           1           : n, ka and kb

Nag_Upper           : uplo
Nag_DoBoth          : job

0.0           2.0           : vl and vu

(-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,-0.16)
              ( 0.50, 0.00) : matrix A

( 9.89, 0.00) ( 1.08,-1.73)
              ( 1.69, 0.00) (-0.04, 0.29)
              ( 2.65, 0.00) (-0.33, 2.24)
              ( 2.17, 0.00) : matrix B

```

## 10.3 Program Results

nag\_zhbgvx (f08upc) Example Program Results

Number of eigenvalues found = 2

```

Eigenvalues
  0.1603      1.7712

Eigenvectors
      1      2
1  0.1908    0.0494
   0.0137   -0.0045

2  0.1413    0.2505
   0.1012    0.4427

3  -0.0437   -0.9705
   -0.0905    0.0679

4  -0.2135    0.0606
   0.2880   -1.3227

```

---