NAG Library Function Document

nag_dsbgvx (f08ubc)

1 Purpose

nag_dsbgvx (f08ubc) computes selected eigenvalues and, optionally, eigenvectors of a real generalized symmetric-definite banded eigenproblem, of the form

$$Az = \lambda Bz,$$

where A and B are symmetric 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_dsbgvx (Nag_OrderType order, Nag_JobType job, Nag_RangeType range,
     Nag_UploType uplo, Integer n, Integer ka, Integer kb, double ab[],
     Integer pdab, double bb[], Integer pdbb, double q[], Integer pdq,
     double vl, double vu, Integer il, Integer iu, double abstol, Integer *m,
     double w[], double z[], Integer pdz, Integer jfail[], NagError *fail)
```

3 Description

The generalized symmetric-definite band problem

$$Az = \lambda Bz$$

is first reduced to a standard band symmetric problem

$$Cx = \lambda x,$$

where C is a symmetric band matrix, using Wilkinson's modification to Crawford's algorithm (see Crawford (1973) and Wilkinson (1977)). The symmetric 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

$$z^{\mathrm{T}}Az = \lambda \quad \text{and} \quad z^{\mathrm{T}}Bz = 1.$$

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 $(\mathbf{vl}, \mathbf{vu}]$ will be found.

If **range** = Nag_Indices, the **il**th to **iu**th 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: $\mathbf{n} \geq 0$.

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: $\mathbf{ka} \geq 0$.

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

8: **ab**$[dim]$ – double *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 symmetric 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**$[k_a + i - j + (j-1) \times$ **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 **ab**$[i - j + (j-1) \times$ **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.

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**$[dim]$ – double *Input/Output*

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

On entry: the upper or lower triangle of the n by n symmetric 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**$[k_b + i - j + (j-1) \times$ **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**$[j - i + (i-1) \times$ **pdbb**$]$, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k_b)$;
if **order** = Nag_RowMajor and **uplo** = Nag_Lower,
B_{ij} is stored in **bb**$[k_b + j - i + (i-1) \times$ **pdbb**$]$, for $i = 1, \ldots, n$ and $j = \max(1, i - k_b), \ldots, i$.

On exit: the factor S from the split Cholesky factorization $B = S^{\mathrm{T}}S$, as returned by nag_dpbstf (f08ufc).

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*] – double *Output*

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

max(1, **pdq** × **n**) when **job** = Nag_DoBoth;
1 otherwise.

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

q[$(j-1) \times$ **pdq** $+ i - 1$] when **order** = Nag_ColMajor;
q[$(i-1) \times$ **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, **pdq** $\geq \max(1, \mathbf{n})$;
otherwise **pdq** ≥ 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, **vl** $<$ **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 **n** = 0, **il** = 1 and **iu** = 0;
if **range** = Nag_Indices and **n** > 0, $1 \leq$ **il** $\leq$ **iu** $\leq$ **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

$$\textbf{abstol} + \epsilon \max(|a|, |b|),$$

where ϵ 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 $2 \times$ 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 $2 \times$ nag_real_safe_small_number( ). See Demmel and Kahan (1990).

19: **m** – Integer * *Output*

On exit: the total number of eigenvalues found. $0 \leq$ **m** $\leq$ **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] – double *Output*

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

max(1, **pdz** × **n**) when **job** = Nag_DoBoth;
1 otherwise.

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

z[$(j-1) \times$ **pdz** $+ i - 1$] when **order** = Nag_ColMajor;
z[$(i-1) \times$ **pdz** $+ j - 1$] when **order** = 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 $Z^{\mathrm{T}}BZ = I$.

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, **n**);
otherwise **pdz** ≥ 1.

23: **jfail**[dim] – Integer *Output*

Note: the dimension, *dim*, of the array **jfail** must be at least max(1, **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 3.2.1.2 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; ⟨*value*⟩ eigenvectors did not converge. Their indices are stored in array **jfail**.

NE_ENUM_INT_2

On entry, **job** = ⟨*value*⟩, **pdq** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **job** = Nag_DoBoth, **pdq** $\geq \max(1, \mathbf{n})$;
otherwise **pdq** ≥ 1.

On entry, **job** = ⟨*value*⟩, **pdz** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **job** = Nag_DoBoth, **pdz** $\geq \max(1, \mathbf{n})$;
otherwise **pdz** ≥ 1.

NE_ENUM_INT_3

On entry, **range** = ⟨*value*⟩, **il** = ⟨*value*⟩, **iu** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **range** = Nag_Indices and **n** $= 0$, **il** $= 1$ and **iu** $= 0$;
if **range** = Nag_Indices and **n** > 0, $1 \leq$ **il** $\leq$ **iu** $\leq$ **n**.

NE_ENUM_REAL_2

On entry, **range** = ⟨*value*⟩, **vl** = ⟨*value*⟩ and **vu** = ⟨*value*⟩.
Constraint: if **range** = Nag_Interval, **vl** $<$ **vu**.

NE_INT

On entry, **ka** = ⟨*value*⟩.
Constraint: **ka** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

On entry, **pdab** = ⟨*value*⟩.
Constraint: **pdab** > 0.

On entry, **pdbb** = ⟨*value*⟩.
Constraint: **pdbb** > 0.

On entry, **pdq** = ⟨*value*⟩.
Constraint: **pdq** > 0.

On entry, **pdz** = ⟨*value*⟩.
Constraint: **pdz** > 0.

NE_INT_2

On entry, **ka** = ⟨*value*⟩ and **kb** = ⟨*value*⟩.
Constraint: **ka** $\geq$ **kb** ≥ 0.

On entry, **pdab** = ⟨*value*⟩ and **ka** = ⟨*value*⟩.
Constraint: **pdab** $\geq$ **ka** $+ 1$.

On entry, **pdbb** = ⟨*value*⟩ and **kb** = ⟨*value*⟩.
Constraint: **pdbb** $\geq$ **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 How to Use the NAG Library and its Documentation for further information.

NE_MAT_NOT_POS_DEF

If **fail.errnum** $= \mathbf{n} + \langle value \rangle$, for $1 \leq \langle value \rangle \leq \mathbf{n}$, then nag_dpbstf (f08ufc) 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 3.6.5 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_dsbgvx (f08ubc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dsbgvx (f08ubc) 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' Notefor 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 complex analogue of this function is nag_zhbgvx (f08upc).

10 Example

This example finds the eigenvalues in the half-open interval $(0.0, 1.0]$, and corresponding eigenvectors, of the generalized band symmetric eigenproblem $Az = \lambda Bz$, where

$$A = \begin{pmatrix} 0.24 & 0.39 & 0.42 & 0 \\ 0.39 & -0.11 & 0.79 & 0.63 \\ 0.42 & 0.79 & -0.25 & 0.48 \\ 0 & 0.63 & 0.48 & -0.03 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 2.07 & 0.95 & 0 & 0 \\ 0.95 & 1.69 & -0.29 & 0 \\ 0 & -0.29 & 0.65 & -0.33 \\ 0 & 0 & -0.33 & 1.17 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_dsbgvx (f08ubc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
```

```
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx04.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, pdbb, pdq, pdz, zsize;

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

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

  INIT_FAIL(fail);

  printf("nag_dsbgvx (f08ubc) 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;
  pdbb = kb + 1;
  pdq = n;
  /* Allocate memory */
  if (!(ab = NAG_ALLOC((ka + 1) * n, double)) ||
      !(bb = NAG_ALLOC((kb + 1) * n, double)) ||
      !(q = NAG_ALLOC(n * n, double)) ||
      !(w = NAG_ALLOC(n, double)) ||
      !(z = NAG_ALLOC(zsize, 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)
#ifdef _WIN32
      for (j = i; j <= MIN(i + ka, n); ++j)
        scanf_s("%lf", &AB_UPPER(i, j));
#else
      for (j = i; j <= MIN(i + ka, n); ++j)
        scanf("%lf", &AB_UPPER(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
      for (j = i; j <= MIN(i + kb, n); ++j)
        scanf_s("%lf", &BB_UPPER(i, j));
#else
      for (j = i; j <= MIN(i + kb, n); ++j)
        scanf("%lf", &BB_UPPER(i, j));
#endif
  }
  else {
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
      for (j = MAX(1, i - ka); j <= i; ++j)
        scanf_s("%lf", &AB_LOWER(i, j));
#else
      for (j = MAX(1, i - ka); j <= i; ++j)
        scanf("%lf", &AB_LOWER(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif
```

```
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
      for (j = MAX(1, i - kb); j <= i; ++j)
        scanf_s("%lf", &BB_LOWER(i, j));
#else
      for (j = MAX(1, i - kb); j <= i; ++j)
        scanf("%lf", &BB_LOWER(i, j));
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Use the default absolute error tolerance for eigenvalues. */
  abstol = 0.0;

  /* Solve the generalized symmetric eigenvalue problem A*x = lambda*B*x
   * using nag_dsbgvx (f08ubc).
   */
  nag_dsbgvx(order, job, Nag_Interval, uplo, n, ka, kb, ab, pdab, bb, pdbb, q,
             pdq, vl, vu, il, iu, abstol, &m, w, z, pdz, index, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from in nag_dsbgvx (f08ubc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print eigensolution */
  printf("Number of eigenvalues found = %8" NAG_IFMT "\n\n", m);
  printf(" Eigenvalues\n   ");
  for (j = 0; j < m; ++j)
    printf(" %7.4f%s", w[j], j % 6 == 5 ? "\n" : " ");
  printf("\n");

  if (job == Nag_DoBoth) {
    /* nag_gen_real_mat_print (x04cac): Print Matrix of eigenvectors Z. */
    printf("\n");
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m,
                           z, pdz, "Selected eigenvectors", 0, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
    if (fail.code == NE_CONVERGENCE) {
      printf("eigenvectors failed to converge\n");
      printf("Indices of eigenvectors that did not converge\n   ");
      for (j = 0; j < n; ++j)
        printf("%8" NAG_IFMT "%s", index[j], j % 6 == 5 ? "\n" : "");
    }
  }

END:
  NAG_FREE(ab);
  NAG_FREE(bb);
  NAG_FREE(q);
  NAG_FREE(w);
  NAG_FREE(z);
  NAG_FREE(index);

  return exit_status;
}
```

10.2 Program Data

```
nag_dsbgvx (f08ubc) Example Program Data
  4      2      1            : n, ka and kb

  Nag_Upper                  : uplo
  Nag_DoBoth                 : job

  0.0    1.0                 : vl and vu
  0.24   0.39   0.42
        -0.11   0.79   0.63
               -0.25   0.48
                      -0.03  : matrix A
  2.07   0.95
         1.69  -0.29
                0.65  -0.33
                       1.17  : matrix B
```

10.3 Program Results

```
nag_dsbgvx (f08ubc) Example Program Results

Number of eigenvalues found =        1

 Eigenvalues
     0.0992

 Selected eigenvectors
         1
 1   0.6729
 2  -0.1009
 3   0.0155
 4  -0.3806
```