NAG Library Function Document

nag_dsbevx (f08hbc)

1 Purpose

nag_dsbevx (f08hbc) computes selected eigenvalues and, optionally, eigenvectors of a real n by n symmetric band matrix A of bandwidth $(2k_d + 1)$. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

2 Specification

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

```
void nag_dsbevx (Nag_OrderType order, Nag_JobType job, Nag_RangeType range,
     Nag_UploType uplo, Integer n, Integer kd, double ab[], Integer pdab,
     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 symmetric band matrix A is first reduced to tridiagonal form, using orthogonal similarity transformations. The required eigenvalues and eigenvectors are then computed from the tridiagonal matrix; the method used depends upon whether all, or selected, eigenvalues and eigenvectors are required.

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

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

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: **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 **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 triangular part of A is stored.

If **uplo** = Nag_Lower, the lower triangular part of A is stored.

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

5: **n** – Integer *Input*

On entry: n, the order of the matrix A.

Constraint: **n** ≥ 0.

6: **kd** – Integer *Input*

On entry: if **uplo** = Nag_Upper, the number of superdiagonals, k_d, of the matrix A.

If **uplo** = Nag_Lower, the number of subdiagonals, k_d, of the matrix A.

Constraint: **kd** ≥ 0.

7: **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 $\mathbf{ab}[k_d + i - j + (j-1) \times \mathbf{pdab}]$, for $j = 1, \dots, n$ and $i = \max(1, j - k_d), \dots, j$;
if **order** = Nag_ColMajor and **uplo** = Nag_Lower,
A_{ij} is stored in $\mathbf{ab}[i - j + (j-1) \times \mathbf{pdab}]$, for $j = 1, \dots, n$ and $i = j, \dots, \min(n, j + k_d)$;
if **order** = Nag_RowMajor and **uplo** = Nag_Upper,
A_{ij} is stored in $\mathbf{ab}[j - i + (i-1) \times \mathbf{pdab}]$, for $i = 1, \dots, n$ and $j = i, \dots, \min(n, i + k_d)$;
if **order** = Nag_RowMajor and **uplo** = Nag_Lower,
A_{ij} is stored in $\mathbf{ab}[k_d + j - i + (i-1) \times \mathbf{pdab}]$, for $i = 1, \dots, n$ and $j = \max(1, i - k_d), \dots, i$.

On exit: **ab** is overwritten by values generated during the reduction to tridiagonal form.

The first superdiagonal or subdiagonal and the diagonal of the tridiagonal matrix T are returned in **ab** using the same storage format as described above.

8: **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** $\geq$ **kd** $+ 1$.

9: **q**[dim] – double *Output*

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

max(1, **pdq** $\times$ **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 orthogonal matrix used in the reduction to tridiagonal form.

If **job** $=$ Nag_EigVals, **q** is not referenced.

10: **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, **n**);
otherwise **pdq** ≥ 1.

11: **vl** – double *Input*
12: **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**.

13: **il** – Integer *Input*
14: **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**.

15: **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 A 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).

16: **m** – Integer * *Output*

On exit: the total number of eigenvalues found. $0 \leq \mathbf{m} \leq \mathbf{n}$.

If **range** = Nag_AllValues, $\mathbf{m} = \mathbf{n}$.

If **range** = Nag_Indices, $\mathbf{m} = \mathbf{iu} - \mathbf{il} + 1$.

17: $\mathbf{w}[\mathbf{n}]$ – double *Output*

On exit: the first **m** elements contain the selected eigenvalues in ascending order.

18: $\mathbf{z}[dim]$ – double *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

$\mathbf{z}[(j-1) \times \mathbf{pdz} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{z}[(i-1) \times \mathbf{pdz} + j - 1]$ when **order** = Nag_RowMajor.

On exit: if **job** = Nag_DoBoth, then

if **fail.code** = NE_NOERROR, the first **m** columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with $\mathbf{w}[i-1]$;

if an eigenvector fails to converge (**fail.code** = NE_CONVERGENCE), then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in **jfail**.

If **job** = Nag_EigVals, **z** is not referenced.

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

20: $\mathbf{jfail}[dim]$ – 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.

21: **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 $\langle value\rangle$ 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** ≥ 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** ≥ 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, **kd** $= \langle value\rangle$.
Constraint: **kd** ≥ 0.

On entry, **n** $= \langle value\rangle$.
Constraint: **n** ≥ 0.

On entry, **pdab** $= \langle value\rangle$.
Constraint: **pdab** > 0.

On entry, **pdq** $= \langle value\rangle$.
Constraint: **pdq** > 0.

On entry, **pdz** $= \langle value\rangle$.
Constraint: **pdz** > 0.

NE_INT_2

On entry, **pdab** $= \langle value\rangle$ and **kd** $= \langle value\rangle$.
Constraint: **pdab** $\geq$ **kd** $+ 1$.

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 eigenvalues and eigenvectors are exact for a nearby matrix $(A + E)$, where

$$\|E\|_2 = O(\epsilon)\|A\|_2,$$

and ϵ is the ***machine precision***. See Section 4.7 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_dsbevx (f08hbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dsbevx (f08hbc) 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 $k_d n^2$ if **job** $=$ Nag_EigVals, and is proportional to n^3 if **job** $=$ Nag_DoBoth and **range** $=$ Nag_AllValues, otherwise the number of floating-point operations will depend upon the number of computed eigenvectors.

The complex analogue of this function is nag_zhbevx (f08hpc).

10 Example

This example finds the eigenvalues in the half-open interval $(-3, 3]$, and the corresponding eigenvectors, of the symmetric band matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 0 & 0 \\ 2 & 2 & 3 & 4 & 0 \\ 3 & 3 & 3 & 4 & 5 \\ 0 & 4 & 4 & 4 & 5 \\ 0 & 0 & 5 & 5 & 5 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_dsbevx (f08hbc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#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, i, il = 0, iu = 0, j, kd, m, n, pdab, pdq,
                pdz;
  /* Arrays */
  char          nag_enum_arg[40];
  double        *ab = 0, *q = 0, *w = 0, *z = 0;
  Integer       *index = 0;
  /* Nag Types */
  Nag_OrderType order;
  Nag_UploType  uplo;
  NagError      fail, fail_print;

#ifdef NAG_COLUMN_MAJOR
#define AB_UPPER(I, J) ab[(J - 1) * pdab + kd + I - J]
#define AB_LOWER(I, J) ab[(J - 1) * pdab + 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 + kd + J - I]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dsbevx (f08hbc) 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"%*[^\n]", &n, &kd);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &n, &kd);
#endif

  /* Read uplo */
#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);

  /* Allocate memory */
  if (!(ab = NAG_ALLOC((kd+1)*n, double)) ||
      !(q = NAG_ALLOC(n*n, double)) ||
      !(w = NAG_ALLOC(n, double)) ||
      !(z = NAG_ALLOC(n*n, double)) ||
      !(index = NAG_ALLOC(n, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  pdab = kd+1;
  pdq = n;
  pdz = n;

  /* Read the lower and upper bounds of the interval to be searched,
   * and read the upper or lower triangular part of the matrix A
   * from data file.
   */
#ifdef _WIN32
  scanf_s("%lf%lf%*[^\n]", &vl, &vu);
#else
```

```
  scanf("%lf%lf%*[^\n]", &vl, &vu);
#endif
  if (uplo == Nag_Upper) {
    for (i = 1; i <= n; ++i)
      for (j = i; j <= MIN(n, i + kd); ++j)
#ifdef _WIN32
        scanf_s("%lf", &AB_UPPER(i, j));
#else
        scanf("%lf", &AB_UPPER(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif
  }
  else if (uplo == Nag_Lower) {
    for (i = 1; i <= n; ++i)
      for (j = MAX(1, i - kd); j <= i; ++j)
#ifdef _WIN32
        scanf_s("%lf", &AB_LOWER(i, j));
#else
        scanf("%lf", &AB_LOWER(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif
  }

  /* Set the absolute error tolerance for eigenvalues.
   * With abstol set to zero, the default value is used instead.
   */
  abstol = 0.0;

  /* nag_dsbevx (f08hbc).
   * Solve the band symmetric eigenvalue problem.
   */
  nag_dsbevx(order, Nag_DoBoth, Nag_Interval, uplo, n, kd, ab, pdab, 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_dsbevx (f08hbc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print solution */
  printf("Number of eigenvalues found =%5"NAG_IFMT"\n", m);

  printf("\nEigenvalues\n");
  for (j = 0; j < m; ++j)
    printf("%8.4f%s", w[j], (j+1)%8 == 0?"\n":" ");
  printf("\n\n");

  /* nag_gen_real_mat_print (x04cac).
   * Print selected eigenvectors.
   */
  INIT_FAIL(fail_print);
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m, z,
                         pdz, "Selected eigenvectors", 0, &fail_print);
  if (fail_print.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail_print.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 < m; ++j)
        printf("%8"NAG_IFMT"%s", index[j], (j+1)%8 == 0?"\n":" ");
    }

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

  return exit_status;
}

#undef AB_UPPER
#undef AB_LOWER
```

10.2 Program Data

```
nag_dsbevx (f08hbc) Example Program Data

  5    2                  :Values of n and kd
  Nag_Upper               :Value of uplo

 -3.0  3.0                :Values of vl and vu

  1.0  2.0  3.0
       2.0  3.0  4.0
            3.0  4.0  5.0
                 4.0  5.0
                      5.0  :End of matrix A
```

10.3 Program Results

```
nag_dsbevx (f08hbc) Example Program Results

Number of eigenvalues found =    2

Eigenvalues
 -2.6633   1.7511

 Selected eigenvectors
          1       2
 1  -0.6238 -0.5635
 2   0.2575  0.3896
 3   0.5900 -0.4008
 4  -0.4308  0.5581
 5  -0.1039 -0.2421
```