# NAG Library Function Document

# nag_dsbev (f08hac)

## 1 Purpose

nag_dsbev (f08hac) computes all the eigenvalues and, optionally, all the eigenvectors of a real $n$ by $n$ symmetric band matrix $A$ of bandwidth $(2k_d + 1)$.

## 2 Specification

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

```
void nag_dsbev (Nag_OrderType order, Nag_JobType job, Nag_UploType uplo,
     Integer n, Integer kd, double ab[], Integer pdab, double w[],
     double z[], Integer pdz, NagError *fail)
```

## 3 Description

The symmetric band matrix $A$ is first reduced to tridiagonal form, using orthogonal similarity transformations, and then the $QR$ algorithm is applied to the tridiagonal matrix to compute the eigenvalues and (optionally) the eigenvectors.

## 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

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: **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.

4: **n** – Integer *Input*

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

*Constraint*: $\mathbf{n} \geq 0$.

5: **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*: $\mathbf{kd} \geq 0$.

6: **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.

7: **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*: $\mathbf{pdab} \geq \mathbf{kd} + 1$.

8: **w**[**n**] – double *Output*

*On exit*: the eigenvalues in ascending order.

9: **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, **z** contains the orthonormal eigenvectors of the matrix $A$, with the $i$th column of $Z$ holding the eigenvector associated with $\mathbf{w}[i-1]$.

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

10: **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 $\mathbf{pdz} \geq 1$.

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

**NE_INT**

On entry, **kd** = ⟨*value*⟩.
Constraint: $\mathbf{kd} \geq 0$.

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 0$.

On entry, **pdab** = ⟨*value*⟩.
Constraint: $\mathbf{pdab} > 0$.

On entry, **pdz** = ⟨*value*⟩.
Constraint: $\mathbf{pdz} > 0$.

**NE_INT_2**

On entry, **pdab** = ⟨*value*⟩ and **kd** = ⟨*value*⟩.
Constraint: $\mathbf{pdab} \geq \mathbf{kd} + 1$.

**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 $\epsilon$ is the ***machine precision***. See Section 4.7 of Anderson *et al.* (1999) for further details.

## 8 Parallelism and Performance

nag_dsbev (f08hac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dsbev (f08hac) 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 is proportional to $k_d n^2$ otherwise.

The complex analogue of this function is nag_zhbev (f08hnc).

## 10 Example

This example finds all the eigenvalues and eigenvectors of the symmetric band matrix

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

together with approximate error bounds for the computed eigenvalues and eigenvectors.

### 10.1 Program Text

```
/* nag_dsbev (f08hac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <math.h>
#include <stdio.h>
```

```
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  double        eerrbd, eps;
  Integer       exit_status = 0, i, j, kd, n, pdab, pdz;
  /* Arrays */
  char          nag_enum_arg[40];
  double        *ab = 0, *rcondz = 0, *w = 0, *z = 0, *zerrbd = 0;
  /* Nag Types */
  Nag_OrderType order;
  Nag_UploType  uplo;
  NagError      fail;

#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]
#define Z(I, J) z[(J - 1) * pdz + I - 1]
  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]
#define Z(I, J) z[(I - 1) * pdz + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dsbev (f08hac) 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)) ||
      !(rcondz = NAG_ALLOC(n, double)) ||
      !(w = NAG_ALLOC(n, double)) ||
      !(z = NAG_ALLOC(n*n, double)) ||
      !(zerrbd = NAG_ALLOC(n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  pdab = kd+1;
```

```
  pdz = n;

  /* Read the upper or lower triangular part of the symmetric band
   * matrix A from data file.
   */
  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
  }

  /* nag_dsbev (f08hac).
   * Solve the band symmetric eigenvalue problem.
   */
  nag_dsbev(order, Nag_DoBoth, uplo, n, kd, ab, pdab, w, z, pdz, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dsbev (f08hac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Normalize the eigenvectors */
  for(j=1; j<=n; j++)
    for(i=n; i>=1; i--)
      Z(i, j) = Z(i, j) / Z(1,j);

  /* Print solution */
  printf("Eigenvalues\n");
  for (j = 0; j < n; ++j)
    printf("%8.4f%s", w[j], (j+1)%8 == 0?"\n":" ");
  printf("\n");

  /* nag_gen_real_mat_print (x04cac).
   * Print eigenvectors.
   */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, z,
                         pdz, "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;
    }

  /* Get the machine precision, eps, using nag_machine_precision (X02AJC)
```

```
   * and compute the approximate error bound for the computed eigenvalues.
   * Note that for the 2-norm, ||A|| = max {|w[i]|, i=0..n-1}, and since
   * the eigenvalues are in ascending order ||A|| = max( |w[0]|, |w[n-1]|).
   */
  eps = nag_machine_precision;
  eerrbd = eps * MAX(fabs(w[0]), fabs(w[n-1]));

  /* nag_ddisna (f08flc).
   * Estimate reciprocal condition numbers for eigenvectors.
   */
  nag_ddisna(Nag_EigVecs, n, n, w, rcondz, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ddisna (f08flc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Compute the error estimates for the eigenvectors. */
  for (i = 0; i < n; ++i)
    zerrbd[i] = eerrbd / rcondz[i];

  /* Print the approximate error bounds for the eigenvalues and vectors. */
  printf("\nError estimate for the eigenvalues\n");
  printf("%11.1e\n", eerrbd);

  printf("\nError estimates for the eigenvectors\n");
  for (i = 0; i < n; ++i)
    printf("%11.1e%s", zerrbd[i], (i+1)%6 == 0?"\n":" ");

 END:
  NAG_FREE(ab);
  NAG_FREE(rcondz);
  NAG_FREE(w);
  NAG_FREE(z);
  NAG_FREE(zerrbd);

  return exit_status;
}

#undef AB_UPPER
#undef AB_LOWER
#undef Z
```

## 10.2 Program Data

```
nag_dsbev (f08hac) Example Program Data

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

  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_dsbev (f08hac) Example Program Results

Eigenvalues
 -3.2474  -2.6633   1.7511   4.1599  14.9997
 Eigenvectors
           1          2          3          4          5
 1      1.0000     1.0000     1.0000     1.0000     1.0000
 2     14.5267    -0.4128    -0.6915     1.1530     1.9975
 3    -11.1002    -0.9459     0.7113     0.2847     3.3349
 4    -11.2315     0.6907    -0.9905    -0.0909     3.4904
 5     13.5387     0.1665     0.4296    -1.1530     3.4128
```

```
Error estimate for the eigenvalues
    1.7e-15

Error estimates for the eigenvectors
    2.9e-15     2.9e-15     6.9e-16     6.9e-16     1.5e-16
```