

## NAG Library Function Document

### nag\_dsbtrd (f08hec)

#### 1 Purpose

nag\_dsbtrd (f08hec) reduces a real symmetric band matrix to tridiagonal form.

#### 2 Specification

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

void nag_dsbtrd (Nag_OrderType order, Nag_VectType vect, Nag_UploType uplo,
                Integer n, Integer kd, double ab[], Integer pdab, double d[], double e[],
                double q[], Integer pdq, NagError *fail)
```

#### 3 Description

nag\_dsbtrd (f08hec) reduces a symmetric band matrix  $A$  to symmetric tridiagonal form  $T$  by an orthogonal similarity transformation:

$$T = Q^T A Q.$$

The orthogonal matrix  $Q$  is determined as a product of Givens rotation matrices, and may be formed explicitly by the function if required.

The function uses a vectorizable form of the reduction, due to Kaufman (1984).

#### 4 References

Kaufman L (1984) Banded eigenvalue solvers on vector machines *ACM Trans. Math. Software* **10** 73–86  
 Parlett B N (1998) *The Symmetric Eigenvalue Problem* SIAM, Philadelphia

#### 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: **vect** – Nag\_VectType *Input*

*On entry:* indicates whether  $Q$  is to be returned.

**vect** = Nag\_FormQ  
 $Q$  is returned.

**vect** = Nag\_UpdateQ  
 $Q$  is updated (and the array **q** must contain a matrix on entry).

**vect** = Nag\_DoNotForm  
 $Q$  is not required.

*Constraint:* **vect** = Nag\_FormQ, Nag\_UpdateQ or Nag\_DoNotForm.

- 3: **uplo** – Nag\_UploType *Input*  
*On entry:* indicates whether the upper or lower triangular part of  $A$  is stored.  
**uplo** = Nag\_Upper  
 The upper triangular part of  $A$  is stored.  
**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:*  $n \geq 0$ .
- 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:*  $kd \geq 0$ .
- 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 **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 **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 **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 **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:* **pdab**  $\geq \max(1, \mathbf{kd} + 1)$ .
- 8: **d**[**n**] – double *Output*  
*On exit:* the diagonal elements of the tridiagonal matrix  $T$ .
- 9: **e**[**n** - 1] – double *Output*  
*On exit:* the off-diagonal elements of the tridiagonal matrix  $T$ .

10: **q**[*dim*] – double Input/Output

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

$\max(1, \mathbf{pdq} \times \mathbf{n})$  when **vect** = Nag\_FormQ or Nag\_UpdateQ;  
1 when **vect** = Nag\_DoNotForm.

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

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

*On entry:* if **vect** = Nag\_UpdateQ, **q** must contain the matrix formed in a previous stage of the reduction (for example, the reduction of a banded symmetric-definite generalized eigenproblem); otherwise **q** need not be set.

*On exit:* if **vect** = Nag\_FormQ or Nag\_UpdateQ, the *n* by *n* matrix *Q*.

If **vect** = Nag\_DoNotForm, **q** is not referenced.

11: **pdq** – Integer Input

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

*Constraints:*

if **vect** = Nag\_FormQ or Nag\_UpdateQ, **pdq** ≥  $\max(1, \mathbf{n})$ ;  
if **vect** = Nag\_DoNotForm, **pdq** ≥ 1.

12: **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.

### NE\_BAD\_PARAM

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

### NE\_ENUM\_INT\_2

On entry, **vect** = *⟨value⟩*, **pdq** = *⟨value⟩* and **n** = *⟨value⟩*.

Constraint: if **vect** = Nag\_FormQ or Nag\_UpdateQ, **pdq** ≥  $\max(1, \mathbf{n})$ ;  
if **vect** = Nag\_DoNotForm, **pdq** ≥ 1.

### NE\_INT

On entry, **kd** = *⟨value⟩*.

Constraint: **kd** ≥ 0.

On entry, **n** = *⟨value⟩*.

Constraint: **n** ≥ 0.

On entry, **pdab** = *⟨value⟩*.

Constraint: **pdab** > 0.

On entry, **pdq** = *⟨value⟩*.

Constraint: **pdq** > 0.

**NE\_INT\_2**

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

**7 Accuracy**

The computed tridiagonal matrix  $T$  is exactly similar to a nearby matrix  $(A + E)$ , where

$$\|E\|_2 \leq c(n)\epsilon\|A\|_2,$$

$c(n)$  is a modestly increasing function of  $n$ , and  $\epsilon$  is the *machine precision*.

The elements of  $T$  themselves may be sensitive to small perturbations in  $A$  or to rounding errors in the computation, but this does not affect the stability of the eigenvalues and eigenvectors.

The computed matrix  $Q$  differs from an exactly orthogonal matrix by a matrix  $E$  such that

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

where  $\epsilon$  is the *machine precision*.

**8 Further Comments**

The total number of floating point operations is approximately  $6n^2k$  if **vect** = Nag\_DoNotForm with  $3n^3(k-1)/k$  additional operations if **vect** = Nag\_FormQ.

The complex analogue of this function is nag\_zhbtrd (f08hsc).

**9 Example**

This example computes all the eigenvalues and eigenvectors of the matrix  $A$ , where

$$A = \begin{pmatrix} 4.99 & 0.04 & 0.22 & 0.00 \\ 0.04 & 1.05 & -0.79 & 1.04 \\ 0.22 & -0.79 & -2.31 & -1.30 \\ 0.00 & 1.04 & -1.30 & -0.43 \end{pmatrix}.$$

Here  $A$  is symmetric and is treated as a band matrix. The program first calls nag\_dsbtrd (f08hec) to reduce  $A$  to tridiagonal form  $T$ , and to form the orthogonal matrix  $Q$ ; the results are then passed to nag\_dsteqr (f08jec) which computes the eigenvalues and eigenvectors of  $A$ .

**9.1 Program Text**

```

/* nag_dsbtrd (f08hec) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

int main(void)
{
  /* Scalars */
  Integer      i, j, k, kd, n, pdab, pdz, d_len, e_len;

```

```

Integer      exit_status = 0;
NagError     fail;
Nag_UploType uplo;
Nag_OrderType order;
/* Arrays */
char         nag_enum_arg[40];
double       *ab = 0, *d = 0, *e = 0, *z = 0;

#ifdef NAG_COLUMN_MAJOR
#define AB_UPPER(I, J) ab[(J - 1) * pdab + k + I - J - 1]
#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 + k + J - I - 1]
#define Z(I, J) z[(I - 1) * pdz + J - 1]
  order = Nag_RowMajor;
#endif

INIT_FAIL(fail);

printf("nag_dsbtrd (f08hec) Example Program Results\n\n");

/* Skip heading in data file */
scanf("%*[\n] ");
scanf("%ld%ld%*[\n] ", &n, &kd);
pdab = kd + 1;
pdz = n;
d_len = n;
e_len = n - 1;

/* Allocate memory */
if (!(ab = NAG_ALLOC(pdab * n, double)) ||
    !(d = NAG_ALLOC(d_len, double)) ||
    !(e = NAG_ALLOC(e_len, double)) ||
    !(z = NAG_ALLOC(pdz * n, double)))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* Read A from data file */
scanf("%s%*[\n] ", nag_enum_arg);
/* nag_enum_name_to_value(x04nac).
 * Converts NAG enum member name to value
 */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
k = kd + 1;
if (uplo == Nag_Upper)
{
  for (i = 1; i <= n; ++i)
  {
    for (j = i; j <= MIN(i + kd, n); ++j)
      scanf("%lf", &AB_UPPER(i, j));
  }
  scanf("%*[\n] ");
}
else
{
  for (i = 1; i <= n; ++i)
  {
    for (j = MAX(1, i - kd); j <= i; ++j)
      scanf("%lf", &AB_LOWER(i, j));
  }
  scanf("%*[\n] ");
}

/* Reduce A to tridiagonal form */
/* nag_dsbtrd (f08hec).

```

```

* Orthogonal reduction of real symmetric band matrix to
* symmetric tridiagonal form
*/
nag_dsbtrd(order, Nag_FormQ, uplo, n, kd, ab, pdab, d, e,
           z, pdz, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dsbtrd (f08hec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Calculate all the eigenvalues and eigenvectors of A */
/* nag_dsteqr (f08jec).
* All eigenvalues and eigenvectors of real symmetric
* tridiagonal matrix, reduced from real symmetric matrix
* using implicit QL or QR
*/
nag_dsteqr(order, Nag_UpdateZ, n, d, e, z, pdz, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dsteqr (f08jec).\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 eigenvalues and eigenvectors */
printf("Eigenvalues\n");
for (i = 1; i <= n; ++i)
    printf("%8.4f%s", d[i-1], i%8 == 0?"\n":" ");
printf("\n\n");
/* nag_gen_real_mat_print (x04cac).
* Print real general matrix (easy-to-use)
*/
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;
}
END:
if (ab) NAG_FREE(ab);
if (d) NAG_FREE(d);
if (e) NAG_FREE(e);
if (z) NAG_FREE(z);

return exit_status;
}

```

## 9.2 Program Data

```

nag_dsbtrd (f08hec) Example Program Data
4 2 :Values of n and kd
Nag_Lower :Value of uplo
4.99
0.04 1.05
0.22 -0.79 -2.31
1.04 -1.30 -0.43 :End of matrix A

```

### 9.3 Program Results

nag\_dsbtord (f08hec) Example Program Results

Eigenvalues

-2.9943 -0.7000 1.9974 4.9969

Eigenvectors

	1	2	3	4
1	1.0000	1.0000	1.0000	1.0000
2	-2.6092	-36.0739	71.4695	0.0020
3	-35.8180	-19.3048	-26.5971	0.0311
4	-17.1000	45.9991	44.8645	-0.0071

---