NAG Library Function Document

nag_dbdsdc (f08mdc)

1 Purpose

nag_dbdsdc (f08mdc) computes the singular values and, optionally, the left and right singular vectors of a real n by n (upper or lower) bidiagonal matrix B.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_dbdsdc (Nag_OrderType order, Nag_UploType uplo,
     Nag_ComputeSingularVecsType compq, Integer n, double d[], double e[],
     double u[], Integer pdu, double vt[], Integer pdvt, double q[],
     Integer iq[], NagError *fail)
```

3 Description

nag_dbdsdc (f08mdc) computes the singular value decomposition (SVD) of the (upper or lower) bidiagonal matrix B as

$$B = USV^{\mathrm{T}},$$

where S is a diagonal matrix with non-negative diagonal elements $s_{ii} = s_i$, such that

$$s_1 \geq s_2 \geq \cdots \geq s_n \geq 0,$$

and U and V are orthogonal matrices. The diagonal elements of S are the singular values of B and the columns of U and V are respectively the corresponding left and right singular vectors of B.

When only singular values are required the function uses the QR algorithm, but when singular vectors are required a divide and conquer method is used. The singular values can optionally be returned in compact form, although currently no function is available to apply U or V when stored in compact form.

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 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: **uplo** – Nag_UploType *Input*

On entry: indicates whether B is upper or lower bidiagonal.

uplo = Nag_Upper
B is upper bidiagonal.

uplo = Nag_Lower
B is lower bidiagonal.

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

3: **compq** – Nag_ComputeSingularVecsType *Input*

On entry: specifies whether singular vectors are to be computed.

compq = Nag_NotSingularVecs
Compute singular values only.

compq = Nag_PackedSingularVecs
Compute singular values and compute singular vectors in compact form.

compq = Nag_SingularVecs
Compute singular values and singular vectors.

Constraint: **compq** = Nag_NotSingularVecs, Nag_PackedSingularVecs or Nag_SingularVecs.

4: **n** – Integer *Input*

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

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

5: **d**[dim] – double *Input/Output*

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

On entry: the n diagonal elements of the bidiagonal matrix B.

On exit: if **fail.code** = NE_NOERROR, the singular values of B.

6: **e**[dim] – double *Input/Output*

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

On entry: the $(n - 1)$ off-diagonal elements of the bidiagonal matrix B.

On exit: the contents of **e** are destroyed.

7: **u**[dim] – double *Output*

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

$\max(1, \mathbf{pdu} \times \mathbf{n})$ when **compq** = Nag_SingularVecs;
1 otherwise.

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

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

On exit: if **compq** = Nag_SingularVecs, then if **fail.code** = NE_NOERROR, **u** contains the left singular vectors of the bidiagonal matrix B.

If **compq** $\neq$ Nag_SingularVecs, **u** is not referenced.

8: **pdu** – Integer *Input*

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

Constraints:

if **compq** = Nag_SingularVecs, **pdu** $\geq \max(1, \mathbf{n})$;
otherwise **pdu** ≥ 1.

9: **vt**[*dim*] – double *Output*

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

$\max(1, \mathbf{pdvt} \times \mathbf{n})$ when **compq** = Nag_SingularVecs;
1 otherwise.

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

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

On exit: if **compq** = Nag_SingularVecs, then if **fail.code** = NE_NOERROR, the rows of **vt** contain the right singular vectors of the bidiagonal matrix B.

If **compq** $\neq$ Nag_SingularVecs, **vt** is not referenced.

10: **pdvt** – Integer *Input*

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

Constraints:

if **compq** = Nag_SingularVecs, **pdvt** $\geq \max(1, \mathbf{n})$;
otherwise **pdvt** ≥ 1.

11: **q**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **q** must be at least $\max\left(1, \mathbf{n}^2 + 5\mathbf{n}, ldq\right)$.

On exit: if **compq** = Nag_PackedSingularVecs, then if **fail.code** = NE_NOERROR, **q** and **iq** contain the left and right singular vectors in a compact form, requiring $O(\mathbf{n}\log_2 \mathbf{n})$ space instead of $2 \times \mathbf{n}^2$. In particular, **q** contains all the real data in the first $ldq = \mathbf{n} \times (11 + 2 \times smlsiz + 8 \times \text{int}(\log_2 (\mathbf{n}/(smlsiz + 1))))$ elements of **q**, where $smlsiz$ is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25).

If **compq** $\neq$ Nag_PackedSingularVecs, **q** is not referenced.

12: **iq**[*dim*] – Integer *Output*

Note: the dimension, *dim*, of the array **iq** must be at least $\max(1, ldiq)$.

On exit: if **compq** = Nag_PackedSingularVecs, then if **fail.code** = NE_NOERROR, **q** and **iq** contain the left and right singular vectors in a compact form, requiring $O(\mathbf{n}\log_2 \mathbf{n})$ space instead of $2 \times \mathbf{n}^2$. In particular, **iq** contains all integer data in the first $ldiq = \mathbf{n} \times (3 + 3 \times \text{int}(\log_2 (\mathbf{n}/(smlsiz + 1))))$ elements of **iq**, where $smlsiz$ is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25).

If **compq** $\neq$ Nag_PackedSingularVecs, **iq** is not referenced.

13: **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_ENUM_INT_2

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

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

NE_INT

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

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

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

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_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.

NE_SINGULAR

The algorithm failed to compute a singular value. The update process of divide-and-conquer failed.

7 Accuracy

Each computed singular value of B is accurate to nearly full relative precision, no matter how tiny the singular value. The ith computed singular value, $\hat{s}_i$, satisfies the bound

$$|\hat{s}_i - s_i| \leq p(n)\epsilon s_i$$

where ϵ is the ***machine precision*** and $p(n)$ is a modest function of n.

For bounds on the computed singular values, see Section 4.9.1 of Anderson *et al.* (1999). See also nag_ddisna (f08flc).

8 Parallelism and Performance

nag_dbdsdc (f08mdc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dbdsdc (f08mdc) 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

If only singular values are required, the total number of floating-point operations is approximately proportional to n^2. When singular vectors are required the number of operations is bounded above by approximately the same number of operations as nag_dbdsqr (f08mec), but for large matrices nag_dbdsdc (f08mdc) is usually much faster.

There is no complex analogue of nag_dbdsdc (f08mdc).

10 Example

This example computes the singular value decomposition of the upper bidiagonal matrix

$$B = \begin{pmatrix} 3.62 & 1.26 & 0 & 0 \\ 0 & -2.41 & -1.53 & 0 \\ 0 & 0 & 1.92 & 1.19 \\ 0 & 0 & 0 & -1.43 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_dbdsdc (f08mdc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nagx02.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagf16.h>

int main(void)
{
  /* Scalars */
  double alpha, beta, eps, norm;
  Integer abi, i, j, k1, k2, leniq, lenq, mlvl, n, pdab, pdb, pdu, pdvt;
  Integer exit_status = 0, smlsiz = 25;

  /* Arrays */
  double *ab = 0, *b = 0, *d = 0, *e = 0, *q = 0, *u = 0, *vt = 0;
  Integer *iq = 0;
  char nag_enum_arg[40];

  /* Nag Types */
  NagError fail;
  Nag_OrderType order;
```

```
  Nag_UploType uplo;
  Nag_ComputeSingularVecsType compq;

#ifdef NAG_COLUMN_MAJOR
#define B(I, J) b[(J - 1) * pdb + I - 1]
#define U(I, J) u[(J - 1) * pdu + I - 1]
  order = Nag_ColMajor;
#else
#define B(I, J) b[(I - 1) * pdb + J - 1]
#define U(I, J) u[(I - 1) * pdu + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dbdsdc (f08mdc) 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 "%*[^\n]", &n);
#else
  scanf("%" NAG_IFMT "%*[^\n]", &n);
#endif
  if (n < 0) {
    printf("Invalid n\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
  /* Starting index for main diagonal in banded storage format = abi. */
  if ((order == Nag_ColMajor && uplo == Nag_Lower) ||
      (order == Nag_RowMajor && uplo == Nag_Upper)) {
    abi = 0;
  }
  else {
    abi = 1;
  }

  compq = (Nag_ComputeSingularVecsType) nag_enum_name_to_value(nag_enum_arg);
  /* size of u, vt, q and iq depends on value of compq input */
  if (compq == Nag_SingularVecs) {
    pdu = n;
    pdvt = n;
  }
  else {
    pdu = 1;
    pdvt = 1;
  }
  if (compq == Nag_PackedSingularVecs) {
    mlvl = (Integer) (log(n / (smlsiz + 1.0)) / log(2.0)) + 1;
    if (mlvl < 1)
      mlvl = 1;
    lenq = MAX(n * n + 5 * n, n * (3 + 2 * smlsiz + 8 * mlvl));
```

```
    leniq = n * 3 * mlvl;
  }
  else {
    lenq = 1;
    leniq = 1;
  }

  pdb = n;
  pdab = 2;
  /* Allocate memory */
  if (!(b = NAG_ALLOC(n * n, double)) ||
      !(ab = NAG_ALLOC(pdab * n, double)) ||
      !(d = NAG_ALLOC(n, double)) ||
      !(e = NAG_ALLOC(n - 1, double)) ||
      !(q = NAG_ALLOC(lenq, double)) ||
      !(u = NAG_ALLOC(pdu * pdu, double)) ||
      !(vt = NAG_ALLOC(pdvt * pdvt, double)) ||
      !(iq = NAG_ALLOC(leniq, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read the bidiagonal matrix B from data file,
   * first the diagonal elements, and then the off-diagonal elements.
   */
#ifdef _WIN32
  for (i = 0; i < n; ++i)
    scanf_s("%lf", &d[i]);
#else
  for (i = 0; i < n; ++i)
    scanf("%lf", &d[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  for (i = 0; i < n - 1; ++i)
    scanf_s("%lf", &e[i]);
#else
  for (i = 0; i < n - 1; ++i)
    scanf("%lf", &e[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Store diagonal arrays in banded format in ab for printing */
  for (i = 0; i < n; i++)
    ab[2 * i + abi] = d[i];
  for (i = 0; i < n - 1; i++)
    ab[2 * i + abi + 1] = e[i];

  /* k1 = lower bandwidth, k2 = upper bandwidth */
  k1 = (uplo == Nag_Upper ? 0 : 1);
  k2 = 1 - k1;

  /* Print Bidiagonal Matrix B stored in ab.
   * nag_band_real_mat_print (x04cec).
   * Print real packed banded matrix (easy-to-use)
   */
  fflush(stdout);
  nag_band_real_mat_print(order, n, n, k1, k2, ab, 2, "Matrix B", 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_band_real_mat_print (x04cec).\n%s\n",
           fail.message);
```

```
    exit_status = 1;
    goto END;
  }

  /* Calculate the singular values and left and right singular vectors of B
   * using nag_dbdsdc (f08mdc).
   */
  nag_dbdsdc(order, uplo, compq, n, d, e, u, pdu, vt, pdvt, q, iq, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dbdsdc (f08mdc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  printf("\nSingular values\n");
  for (i = 0; i < n; ++i)
    printf(" %8.4f%s", d[i], i % 8 == 7 ? "\n" : "");
  printf("\n\n");

  if (compq == Nag_SingularVecs) {
    /* Reconstruct bidiagonal matrix from decomposition:
     * first, U <- U*S, then Compute B = U*S*V^T.
     * nag_dgemm (f16yac).
     */
    for (i = 1; i <= n; i++)
      for (j = 1; j <= n; j++)
        U(i, j) = U(i, j) * d[j - 1];
    alpha = 1.0;
    beta = 0.0;
    nag_dgemm(order, Nag_NoTrans, Nag_NoTrans, n, n, n, alpha, u, pdu,
              vt, pdvt, beta, b, pdb, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_dgemm (f16yac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* Subtract original bidiagonal matrix:
     * this should give a matrix close to zero.
     */
    for (i = 1; i <= n; i++)
      B(i, i) -= ab[2 * i - 2 + abi];
    if (uplo == Nag_Upper)
      for (i = 1; i <= n - 1; i++)
        B(i, i + 1) -= ab[2 * i - 1 + abi];
    else
      for (i = 1; i <= n - 1; i++)
        B(i + 1, i) -= ab[2 * i - 1 + abi];

    /* nag_dge_norm (f16rac): Find norm of matrix B and print warning if
     * it is too large.
     */
    nag_dge_norm(order, Nag_OneNorm, n, n, b, pdb, &norm, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
    /* Get the machine precision, using nag_machine_precision (x02ajc) */
    eps = nag_machine_precision;
    if (norm > pow(eps, 0.8)) {
      printf("Norm of B-(U*S*V^T) is much greater than 0.\nSchur "
             "factorization has failed.\n norm = %13.4e\n", norm);
    }
  }
END:
  NAG_FREE(ab);
  NAG_FREE(b);
  NAG_FREE(d);
  NAG_FREE(e);
  NAG_FREE(q);
```

```
  NAG_FREE(u);
  NAG_FREE(vt);
  NAG_FREE(iq);

  return exit_status;
}
```

10.2 Program Data

```
nag_dbdsdc (f08mdc) Example Program Data

  4                          : n
  Nag_Upper                  : uplo
  Nag_SingularVecs           : compq
  3.62  -2.41   1.92  -1.43  : diagonal elements
  1.26  -1.53   1.19         : off-diagonal elements
```

10.3 Program Results

```
nag_dbdsdc (f08mdc) Example Program Results

 Matrix B
            1          2          3          4
 1      3.6200     1.2600
 2                -2.4100    -1.5300
 3                            1.9200     1.1900
 4                                      -1.4300

Singular values
   4.0001   3.0006   1.9960   0.9998
```