NAG C Library Function Document

nag_dbdsqr (f08mec)

1 Purpose

nag_dbdsqr (f08mec) computes the singular value decomposition of a real upper or lower bidiagonal matrix, or of a real general matrix which has been reduced to bidiagonal form.

2 Specification

```
void nag_dbdsqr (Nag_OrderType order, Nag_UploType uplo, Integer n, Integer ncvt,
     Integer nru, Integer ncc, double d[], double e[], double vt[], Integer pdvt,
     double u[], Integer pdu, double c[], Integer pdc, NagError *fail)
```

3 Description

nag_dbdsqr (f08mec) computes the singular values, and optionally, the left or right singular vectors of a real upper or lower bidiagonal matrix B. In other words, it can compute the singular value decomposition (SVD) of B as

$$B = U\Sigma V^T.$$

Here Σ is a diagonal matrix with real diagonal elements σ_i (the singular values of B), such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0;$$

U is an orthogonal matrix whose columns are the left singular vectors u_i; V is an orthogonal matrix whose rows are the right singular vectors v_i. Thus

$$Bu_i = \sigma_i v_i \quad \text{and} \quad B^T v_i = \sigma_i u_i, \quad i = 1, 2, \ldots, n.$$

To compute U and/or V^T, the arrays U and/or **vt** must be initialised to the unit matrix before nag_dbdsqr (f08mec) is called.

The function may also be used to compute the SVD of a real general matrix A which has been reduced to bidiagonal form by an orthogonal transformation: $A = QBP^T$. If A is m by n with $m \geq n$, then Q is m by n and P^T is n by n; if A is n by p with $n < p$, then Q is n by n and P^T is n by p. In this case, the matrices Q and/or P^T must be formed explicitly by nag_dorgbr (f08kfc) and passed to nag_dbdsqr (f08mec) in the arrays **u** and/or **vt** respectively.

nag_dbdsqr (f08mec) also has the capability of forming $U^T C$, where C is an arbitrary real matrix; this is needed when using the SVD to solve linear least-squares problems.

nag_dbdsqr (f08mec) uses two different algorithms. If any singular vectors are required (i.e., if **ncvt** > 0 or **nru** > 0 or **ncc** > 0), the bidiagonal QR algorithm is used, switching between zero-shift and implicitly shifted forms to preserve the accuracy of small singular values, and switching between QR and QL variants in order to handle graded matrices effectively (see Demmel and Kahan (1990)). If only singular values are required (that is, if **ncvt** $=$ **nru** $=$ **ncc** $= 0$), they are computed by the differential qd algorithm (see Fernando and Parlett (1994)), which is faster and can achieve even greater accuracy.

The singular vectors are normalized so that $\|u_i\| = \|v_i\| = 1$, but are determined only to within a factor ± 1.

4 References

Demmel J W and Kahan W (1990) Accurate singular values of bidiagonal matrices *SIAM J. Sci. Statist. Comput.* **11** 873–912

Fernando K V and Parlett B N (1994) Accurate singular values and differential qd algorithms *Numer. Math.* **67** 191–229

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

5 Parameters

1: **order** – Nag_OrderType *Input*

On entry: the **order** parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: **order** = **Nag_RowMajor** or **Nag_ColMajor**.

2: **uplo** – Nag_UploType *Input*

On entry: indicates whether B is an upper or lower bidiagonal matrix as follows:

if **uplo** = **Nag_Upper**, B is an upper bidiagonal matrix;

if **uplo** = **Nag_Lower**, B is a lower bidiagonal matrix.

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

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

4: **ncvt** – Integer *Input*

On entry: $ncvt$, the number of columns of the matrix V^T of right singular vectors. Set **ncvt** $= 0$ if no right singular vectors are required.

Constraint: **ncvt** ≥ 0.

5: **nru** – Integer *Input*

On entry: nru, the number of rows of the matrix U of left singular vectors. Set **nru** $= 0$ if no left singular vectors are required.

Constraint: **nru** ≥ 0.

6: **ncc** – Integer *Input*

On entry: ncc, the number of columns of the matrix C. Set **ncc** $= 0$ if no matrix C is supplied.

Constraint: **ncc** ≥ 0.

7: **d**$[dim]$ – double *Input/Output*

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

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

On exit: the singular values in decreasing order of magnitude, unless **fail** > 0 (in which case see Section 6).

8: **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 off-diagonal elements of the bidiagonal matrix B.

On exit: the array is overwritten, but if **fail** > 0 see Section 6.

9: **vt**[dim] – double *Input/Output*

Note: the dimension, dim, of the array **vt** must be at least $\max(1, \mathbf{pdvt} \times \mathbf{ncvt})$ when **order** = **Nag_ColMajor** and at least $\max(1, \mathbf{pdvt} \times \mathbf{n})$ when **order** = **Nag_RowMajor**.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix is stored in **vt**$[(j-1) \times \mathbf{pdvt} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix is stored in **vt**$[(i-1) \times \mathbf{pdvt} + j - 1]$.

On entry: if **ncvt** > 0, **vt** must contain an n by $ncvt$ matrix. If the right singular vectors of B are required, $ncvt = n$ and **vt** must contain the unit matrix; if the right singular vectors of A are required, **vt** must contain the orthogonal matrix P^T returned by nag_dorgbr (f08kfc) with **vect** = **Nag_FormP**.

On exit: the n by $ncvt$ matrix V^T or $V^T P^T$ of right singular vectors, stored by rows.

vt is not referenced if **ncvt** $= 0$.

10: **pdvt** – Integer *Input*

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

Constraints:

if **order** = **Nag_ColMajor**,
  if **ncvt** > 0, **pdvt** $\geq \max(1, \mathbf{n})$;
  otherwise **pdvt** ≥ 1;

if **order** = **Nag_RowMajor**, **pdvt** $\geq \max(1, \mathbf{ncvt})$.

11: **u**[dim] – double *Input/Output*

Note: the dimension, dim, of the array **u** must be at least $\max(1, \mathbf{pdu} \times \mathbf{n})$ when **order** = **Nag_ColMajor** and at least $\max(1, \mathbf{pdu} \times \mathbf{nru})$ when **order** = **Nag_RowMajor**.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix U is stored in **u**$[(j-1) \times \mathbf{pdu} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix U is stored in **u**$[(i-1) \times \mathbf{pdu} + j - 1]$.

On entry: if **nru** > 0, **u** must contain an nru by n matrix. If the left singular vectors of B are required, $nru = n$ and **u** must contain the unit matrix; if the left singular vectors of A are required, **u** must contain the orthogonal matrix Q returned by nag_dorgbr (f08kfc) with **vect** = **Nag_FormQ**.

On exit: the nru by n matrix U or QU of left singular vectors, stored as columns of the matrix.

u is not referenced if **nru** $= 0$.

12: **pdu** – Integer *Input*

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

Constraints:

if **order** = **Nag_ColMajor**, **pdu** $\geq \max(1, \mathbf{nru})$;
if **order** = **Nag_RowMajor**, **pdu** $\geq \max(1, \mathbf{n})$.

13: **c**[dim] – double *Input/Output*

Note: the dimension, dim, of the array **c** must be at least $\max(1, \mathbf{pdc} \times \mathbf{ncc})$ when **order** = **Nag_ColMajor** and at least $\max(1, \mathbf{pdc} \times \mathbf{n})$ when **order** = **Nag_RowMajor**.

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix C is stored in **c**$[(j-1) \times \mathbf{pdc} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix C is stored in **c**$[(i-1) \times \mathbf{pdc} + j - 1]$.

On entry: the n by ncc matrix C if **ncc** > 0.

On exit: **c** is overwritten by the matrix $U^T C$.

c is not referenced if **ncc** $= 0$.

14: **pdc** – Integer *Input*

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

Constraints:

if **order** = **Nag_ColMajor**,
if **ncc** > 0, **pdc** $\geq \max(1, \mathbf{n})$;
otherwise **pdc** ≥ 1;

if **order** = **Nag_RowMajor**, **pdc** $\geq \max(1, \mathbf{ncc})$.

15: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

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

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

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

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

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

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

NE_INT_2

On entry, **pdvt** = ⟨*value*⟩, **ncvt** = ⟨*value*⟩.
Constraint: **pdvt** $\geq \max(1, \mathbf{ncvt})$.

On entry, **pdu** = ⟨*value*⟩, **nru** = ⟨*value*⟩.
Constraint: **pdu** $\geq \max(1, \mathbf{nru})$.

On entry, **pdu** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: **pdu** $\geq \max(1, \mathbf{n})$.

On entry, **pdc** = ⟨*value*⟩, **ncc** = ⟨*value*⟩.
Constraint: **pdc** $\geq \max(1, \mathbf{ncc})$.

NE_INT_3

On entry, **n** = ⟨*value*⟩, **ncvt** = ⟨*value*⟩, **pdvt** = ⟨*value*⟩.
Constraint: if **ncvt** > 0, **pdvt** $\geq \max(1, \mathbf{n})$;
otherwise **pdvt** ≥ 1.

On entry, **n** = ⟨*value*⟩, **ncc** = ⟨*value*⟩, **pdc** = ⟨*value*⟩.
Constraint: if **ncc** > 0, **pdc** $\geq \max(1, \mathbf{n})$;
otherwise **pdc** ≥ 1.

NE_CONVERGENCE

⟨*value*⟩ off-diagonals did not converge. The arrays **d** and **e** contain the diagonal and off-diagonal elements, respectively, of a bidiagonal matrix orthogonally equivalent to B.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

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

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 consult NAG for assistance.

7 Accuracy

Each singular value and singular vector is computed to high relative accuracy. However, the reduction to bidiagonal form (prior to calling the function) may exclude the possibility of obtaining high relative accuracy in the small singular values of the original matrix if its singular values vary widely in magnitude.

If σ_i is an exact singular value of B and $\tilde{\sigma}_i$ is the corresponding computed value, then

$$|\tilde{\sigma}_i - \sigma_i| \leq p(m,n)\epsilon\sigma_i$$

where $p(m,n)$ is a modestly increasing function of m and n, and ϵ is the ***machine precision***. If only singular values are computed, they are computed more accurately (i.e., the function $p(m,n)$ is smaller), than when some singular vectors are also computed.

If u_i is the corresponding exact left singular vector of B, and $\tilde{u}_i$ is the corresponding computed left singular vector, then the angle $\theta(\tilde{u}_i, u_i)$ between them is bounded as follows:

$$\theta(\tilde{u}_i, u_i) \leq \frac{p(m,n)\epsilon}{relgap_i}$$

where $relgap_i$ is the relative gap between σ_i and the other singular values, defined by

$$relgap_i = \min_{i \neq j} \frac{|\sigma_i - \sigma_j|}{(\sigma_i + \sigma_j)}.$$

A similar error bound holds for the right singular vectors.

8 Further Comments

The total number of floating-point operations is roughly proportional to n^2 if only the singular values are computed. About $6n^2 \times nru$ additional operations are required to compute the left singular vectors and about $6n^2 \times ncvt$ to compute the right singular vectors. The operations to compute the singular values must all be performed in scalar mode; the additional operations to compute the singular vectors can be vectorized and on some machines may be performed much faster.

The complex analogue of this function is nag_zbdsqr (f08msc).

9 Example

To compute the singular value decomposition of the upper bidiagonal matrix B, where

$$B = \begin{pmatrix} 3.62 & 1.26 & 0.00 & 0.00 \\ 0.00 & -2.41 & -1.53 & 0.00 \\ 0.00 & 0.00 & 1.92 & 1.19 \\ 0.00 & 0.00 & 0.00 & -1.43 \end{pmatrix}.$$

See also the example for nag_dorgbr (f08kfc), which illustrates the use of the function to compute the singular value decomposition of a general matrix.

9.1 Program Text

```
/* nag_dbdsqr (f08mec) 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, n, pdvt, pdu, d_len, e_len;
  Integer  exit_status=0;
  NagError fail;
  Nag_UploType  uplo;
  Nag_OrderType order;
  /* Arrays */
  char   uplo_char[2];
  double *c=0, *d=0, *e=0, *u=0, *vt=0;

  INIT_FAIL(fail);
  Vprintf("f08mec Example Program Results\n\n");

/* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%*[^\n] ", &n);
#ifdef NAG_COLUMN_MAJOR
#define U(I,J) u[(J-1)*pdu + I - 1]
#define VT(I,J) vt[(J-1)*pdvt + I - 1]
  order = Nag_ColMajor;
  pdu = n;
  pdvt = n;
#else
#define U(I,J) u[(I-1)*pdu + J - 1]
#define VT(I,J) vt[(I-1)*pdvt + J - 1]
  order = Nag_RowMajor;
  pdu = n;
  pdvt = n;
#endif
  d_len = n;
  e_len = n-1;

  /* Allocate memory */
  if ( !(c = NAG_ALLOC(1 * 1, double)) ||
       !(d = NAG_ALLOC(d_len, double)) ||
       !(e = NAG_ALLOC(e_len, double)) ||
       !(u = NAG_ALLOC(n * n, double)) ||
       !(vt = NAG_ALLOC(n * n, double)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read B from data file */
  for (i = 1; i <= n; ++i)
    Vscanf("%lf", &d[i-1]);
  Vscanf("%*[^\n] ");
  for (i = 1; i <= n-1; ++i)
    Vscanf("%lf", &e[i-1]);
```

```
  Vscanf("%*[^\n] ");
  Vscanf(" ' %ls '%*[^\n] ", uplo_char);
  if (*(unsigned char *)uplo_char == 'L')
    uplo = Nag_Lower;
  else if (*(unsigned char *)uplo_char == 'U')
    uplo = Nag_Upper;
  else
    {
      Vprintf("Unrecognised character for Nag_UploType type\n");
      exit_status = -1;
      goto END;
    }

  /* Initialise U and VT to be the unit matrix */
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= n; ++j)
        {
          U(i,j) = 0.0;
          VT(i,j) = 0.0;
        }
          U(i,i) = 1.0;
          VT(i,i) = 1.0;
    }

  /* Calculate the SVD of B */
  f08mec(order, uplo, n, n, n, 0, d, e, vt, pdvt, u, pdu, c,
         1, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08mec.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print singular values, left & right singular vectors */
  Vprintf("\nSingular values\n");
  for (i = 1; i <= n; ++i)
    Vprintf("%8.4f%s", d[i-1], i%8==0 ?"\n":" ");
  Vprintf("\n\n");

  x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
         vt, pdvt, "Right singular vectors, by row", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04cac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  Vprintf("\n");
  x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
         u, pdu, "Left singular vectors, by column", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04cac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  if (c) NAG_FREE(c);
  if (d) NAG_FREE(d);
  if (e) NAG_FREE(e);
  if (u) NAG_FREE(u);
  if (vt) NAG_FREE(vt);

  return exit_status;
}
```

9.2 Program Data

```
f08mec Example Program Data
  4                             :Value of N
  3.62  -2.41   1.92  -1.43
  1.26  -1.53   1.19            :End of matrix B
  'U'                           :Value of UPLO
```

9.3 Program Results

```
f08mec Example Program Results

Singular values
  4.0001   3.0006   1.9960   0.9998

 Right singular vectors, by row
          1        2        3        4
 1   0.8261   0.5246   0.2024   0.0369
 2   0.4512  -0.4056  -0.7350  -0.3030
 3   0.2823  -0.5644   0.1731   0.7561
 4   0.1852  -0.4916   0.6236  -0.5789

 Left singular vectors, by column
          1        2        3        4
 1   0.9129   0.3740   0.1556   0.0512
 2  -0.3935   0.7005   0.5489   0.2307
 3   0.1081  -0.5904   0.6173   0.5086
 4  -0.0132   0.1444  -0.5417   0.8280
```