NAG 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

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

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 $\Sigma$ is a diagonal matrix with real diagonal elements $\sigma_i$ (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

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

To compute $U$ and/or $V^T$, the arrays $u$ and/or $vt$ must be initialized 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 $U^TC$, 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 (i.e., 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 $\pm 1$. 

Mark 25

f08mec.1
4 References


5 Arguments

1: \textbf{order} – Nag_OrderType \hspace{1cm} \textit{Input}

\textit{On entry:} the \textbf{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 \textbf{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint:} \textbf{order} = Nag_RowMajor or Nag_ColMajor.

2: \textbf{uplo} – Nag_UploType \hspace{1cm} \textit{Input}

\textit{On entry:} indicates whether \(B\) is an upper or lower bidiagonal matrix.

\textbf{uplo} = Nag_Upper

\(B\) is an upper bidiagonal matrix.

\textbf{uplo} = Nag_Lower

\(B\) is a lower bidiagonal matrix.

\textit{Constraint:} \textbf{uplo} = Nag_Upper or Nag_Lower.

3: \textbf{n} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \(n\), the order of the matrix \(B\).

\textit{Constraint:} \(n \geq 0\).

4: \textbf{ncvt} – Integer \hspace{1cm} \textit{Input}

\textit{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.

\textit{Constraint:} \(ncvt \geq 0\).

5: \textbf{nru} – Integer \hspace{1cm} \textit{Input}

\textit{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.

\textit{Constraint:} \(nru \geq 0\).

6: \textbf{ncc} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \(ncc\), the number of columns of the matrix \(C\). Set \(ncc = 0\) if no matrix \(C\) is supplied.

\textit{Constraint:} \(ncc \geq 0\).

7: \textbf{d[dim]} – double \hspace{1cm} \textit{Input/Output}

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

\textit{On entry:} the diagonal elements of the bidiagonal matrix \(B\).
On exit: the singular values in decreasing order of magnitude, unless fail.code = NE_CONVERGENCE (in which case see Section 6).

8: e[dim] – double  

Note: the dimension, dim, of the array e must be at least max(1, n - 1).

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

On exit: e is overwritten, but if fail.code = NE_CONVERGENCE see Section 6.

9: vt[dim] – double  

Note: the dimension, dim, of the array vt must be at least max(1, pdvt × ncvt) when order = Nag_ColMajor and at least max(1, pdvt × n) when order = Nag_RowMajor.

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

\[
\begin{align*}
vt[(i - 1) \times pdvt + j - 1] & \quad \text{when order = Nag_ColMajor;} \\
vt[(j - 1) \times pdvt + i - 1] & \quad \text{when order = Nag_RowMajor.}
\end{align*}
\]

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.

If ncvt = 0, vt is not referenced.

10: pdvt – Integer  

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

Constraints:

\[
\begin{align*}
\text{if order = Nag_ColMajor,} \\
& \quad \text{if ncvt > 0, pdvt \geq max(1, n);} \\
& \quad \text{otherwise pdvt \geq 1.;} \\
\text{if order = Nag_RowMajor,} \\
& \quad \text{if ncvt > 0, pdvt \geq ncvt;} \\
& \quad \text{otherwise pdvt \geq 1.}
\end{align*}
\]

11: u[dim] – double  

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

\[
\begin{align*}
\max(1, pdu \times n) & \quad \text{when order = Nag_ColMajor;} \\
\max(1, nru \times pdu) & \quad \text{when order = Nag_RowMajor.}
\end{align*}
\]

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

\[
\begin{align*}
u[(i - 1) \times pdu + j - 1] & \quad \text{when order = Nag_ColMajor;} \\
u[(j - 1) \times pdu + i - 1] & \quad \text{when order = Nag_RowMajor.}
\end{align*}
\]

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.

If nru = 0, u is not referenced.
12:  **pdu** – Integer  

*Input*

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

*Constraints:*

if *order* = Nag_ColMajor, *pdu* ≥ max(1, *nr u*);

if *order* = Nag_RowMajor, *pdu* ≥ max(1, *n*).

13:  **c[**dim**]** – double  

*Input/Output*

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

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

\[ c[j \times pdc + i - 1] \]  

when *order* = Nag_ColMajor;

\[ c[i \times pdc + j - 1] \]  

when *order* = Nag_RowMajor.

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

*On exit:* *c* is overwritten by the matrix *UᵀC*. If *ncc* = 0, *c* is not referenced.

14:  **pdc** – Integer  

*Input*

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

*Constraints:*

if *order* = Nag_ColMajor,

if *ncc* > 0, *pdc* ≥ max(1, *n*);

otherwise *pdc* ≥ 1.;

if *order* = Nag_RowMajor, *pdc* ≥ max(1, *ncc*).

15:  **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**

*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_INT**

On entry, *n* = *value*.

Constraint: *n* ≥ 0.

On entry, *nc n* = *value*.

Constraint: *nc n* ≥ 0.

On entry, *nc v* = *value*.

Constraint: *nc v* > 0.
On entry, \( \text{ncvt} = \langle \text{value} \rangle \).
Constraint: \( \text{ncvt} \geq 0 \).

On entry, \( \text{nr} = \langle \text{value} \rangle \).
Constraint: \( \text{nr} \geq 0 \).

On entry, \( \text{pdc} = \langle \text{value} \rangle \).
Constraint: \( \text{pdc} > 0 \).

On entry, \( \text{pdu} = \langle \text{value} \rangle \).
Constraint: \( \text{pdu} > 0 \).

On entry, \( \text{pdvt} = \langle \text{value} \rangle \).
Constraint: \( \text{pdvt} > 0 \).

**NE_INT_2**

On entry, \( \text{pdc} = \langle \text{value} \rangle \) and \( \text{ncc} = \langle \text{value} \rangle \).
Constraint: \( \text{pdc} \geq \max(1, \text{ncc}) \).

On entry, \( \text{pdu} = \langle \text{value} \rangle \) and \( \text{n} = \langle \text{value} \rangle \).
Constraint: \( \text{pdu} \geq \max(1, \text{n}) \).

On entry, \( \text{pdu} = \langle \text{value} \rangle \) and \( \text{nr} = \langle \text{value} \rangle \).
Constraint: \( \text{pdu} \geq \max(1, \text{nr}) \).

On entry, \( \text{pdvt} = \langle \text{value} \rangle \) and \( \text{ncvt} = \langle \text{value} \rangle \).
Constraint: if \( \text{ncvt} > 0 \), \( \text{pdvt} \geq \text{ncvt} \); otherwise \( \text{pdvt} \geq 1 \).

**NE_INT_3**

On entry, \( \text{ncc} = \langle \text{value} \rangle \), \( \text{pdc} = \langle \text{value} \rangle \) and \( \text{n} = \langle \text{value} \rangle \).
Constraint: if \( \text{ncc} > 0 \), \( \text{pdc} \geq \max(1, \text{n}) \); otherwise \( \text{pdc} \geq 1 \).

On entry, \( \text{pdvt} = \langle \text{value} \rangle \), \( \text{ncvt} = \langle \text{value} \rangle \) and \( \text{n} = \langle \text{value} \rangle \).
Constraint: if \( \text{ncvt} > 0 \), \( \text{pdvt} \geq \max(1, \text{n}) \); otherwise \( \text{pdvt} \geq 1 \).

**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

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 \( \sigma_i \) 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 \( \epsilon \) 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:

\[
\theta(\tilde{u}_i, u_i) \leq \frac{p(m, n)\epsilon}{\text{relgap}_i}
\]

where \( \text{relgap}_i \) is the relative gap between \( \sigma_i \) and the other singular values, defined by

\[
\text{relgap}_i = \min_{i \neq j} \left| \frac{\sigma_i - \sigma_j}{\sigma_i + \sigma_j} \right|
\]

A similar error bound holds for the right singular vectors.

8 Parallelism and Performance

\( \text{nag_dbdsqr (f08mec)} \) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\( \text{nag_dbdsqr (f08mec)} \) 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 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 \( \text{nag_zbdsqr (f08msc)} \).

10 Example

This example computes 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 \( \text{nag_dorgbr (f08kfc)} \), which illustrates the use of the function to compute the singular value decomposition of a general matrix.

10.1 Program Text

```c
/* nag_dbdsqr (f08mec) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 7, 2001. */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
```
```c
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer i, n, pdvt, pdu;
    Integer exit_status = 0, ncc = 0, ldc = 1;
    double zero = 0.0, one = 1.0;
    /* Arrays */
    char nag_enum_arg[40];
    double c[1];
    double *d = 0, *e = 0, *u = 0, *vt = 0;
    /* Nag Types */
    NagError fail;
    Nag_UploType uplo;
    Nag_OrderType order;

    INIT_FAIL(fail);

    printf("nag_dbdsqr (f08mec) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
        scanf_s("%*[\n]");
    #else
        scanf("%*[\n]");
    #endif
    #ifdef _WIN32
        scanf("%"NAG_IFMT"%*[\n]", &n);
    #else
        scanf("%"NAG_IFMT"%*[\n]", &n);
    #endif
    if (n < 0)
    {
        printf("Invalid n\n");
        exit_status = 1;
        goto END;
    }
    #ifdef NAG_COLUMN_MAJOR
        #define U(I, J) u[(J-1)*pdu + I - 1]
        #define VT(I, J) vt[(J-1)*pdvt + I - 1]
    #else
        #define U(I, J) u[(I-1)*pdu + J - 1]
        #define VT(I, J) vt[(I-1)*pdvt + J - 1]
    #endif
    pdu = n;
    pdvt = n;

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

    /* Read B from data file */
    #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

    return exit_status;
}
```


```c
#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("%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);
/* Initialise U and VT to be the unit matrix to obtain SVD of input
 * bidiagonal matrix nag_dge_load (f16qhc).
 * General matrix initialisation.
 */
nag_dge_load(order, n, n, zero, one, u, pdu, &fail);
nag_dge_load(order, n, n, zero, one, vt, pdvt, &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_dge_load (f16qhc).
" fail.message);
  exit_status = 1;
  goto END;
}

/* nag_dbdsqr (f08mec).
 * SVD of real bidiagonal matrix reduced from real general
 * matrix.
 */
nag_dbdsqr(order, uplo, n, n, n, ncc, d, e, vt, pdvt, u, pdu, c, ldc, &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_dbdsqr (f08mec).
" fail.message);
  exit_status = 1;
  goto END;
}

/* Print singular values, left & right singular vectors */
printf("%nSingular values\n ");
for (i = 0; i < n; ++i) printf(" %7.4f", d[i], i%8 == 7?"\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,
vt, pdvt, "Right singular vectors, by row", 0,
&fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_gen_real_mat_print (x04cac).
" fail.message);
  exit_status = 1;
  goto END;
}
printf("\n");
/* nag_gen_real_mat_print (x04cac), see above */
```

---

**f08mec**

NAG Library Manual

---

Mark 25
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, 
u, pdu, "Left singular vectors, by column", 0, 
&fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).\n\"%s\",
fail.message);
    exit_status = 1;
    goto END;
}
END:
NAG_FREE(d);
NAG_FREE(e);
NAG_FREE(u);
NAG_FREE(vt);
return exit_status;
}

10.2 Program Data

nag_dbdsqr (f08mec) Example Program Data

4
3.62 -2.41 1.92 -1.43 : n
1.26 -1.53 1.19 : main diagonal
Nag_Upper : uplo

10.3 Program Results

nag_dbdsqr (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