NAG Library Function Document

nag_zbdsqr (f08msc)

1 Purpose

nag_zbdsqr (f08msc) computes the singular value decomposition of a complex general matrix which has been reduced to bidiagonal form.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_zbdsqr (Nag_OrderType order, Nag_UploType uplo, Integer n,
     Integer ncvt, Integer nru, Integer ncc, double d[], double e[],
     Complex vt[], Integer pdvt, Complex u[], Integer pdu, Complex c[],
     Integer pdc, NagError *fail)
```

3 Description

nag_zbdsqr (f08msc) 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

$$B = U\Sigma V^{\mathrm{T}}.$$

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

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

To compute U and/or V^{T}, the arrays **u** and/or **vt** must be initialized to the unit matrix before nag_zbdsqr (f08msc) is called.

The function stores the real orthogonal matrices U and V^{T} in complex arrays **u** and **vt**, so that it may also be used to compute the SVD of a complex general matrix A which has been reduced to bidiagonal form by a unitary transformation: $A = QBP^{\mathrm{H}}$. If A is m by n with $m \geq n$, then Q is m by n and P^{H} is n by n; if A is n by p with $n < p$, then Q is n by n and P^{H} is n by p. In this case, the matrices Q and/or P^{H} must be formed explicitly by nag_zungbr (f08ktc) and passed to nag_zbdsqr (f08msc) in the arrays **u** and/or **vt** respectively.

nag_zbdsqr (f08msc) also has the capability of forming $U^{\mathrm{H}}C$, where C is an arbitrary complex matrix; this is needed when using the SVD to solve linear least squares problems.

nag_zbdsqr (f08msc) 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 complex factor of absolute value 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 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: **uplo** – Nag_UploType *Input*

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

uplo = Nag_Upper
B is an upper bidiagonal matrix.

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^{H} of right singular vectors. Set **ncvt** = 0 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.code** = NE_CONVERGENCE (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: **e** is overwritten, but if **fail.code** = NE_CONVERGENCE see Section 6.

9: **vt**[dim] – Complex *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.

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

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

On entry: if $\mathbf{ncvt} > 0$, **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 unitary matrix P^{H} returned by nag_zungbr (f08ktc) with **vect** = Nag_FormP.

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

If $\mathbf{ncvt} = 0$, **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 **order** = Nag_ColMajor,

if $\mathbf{ncvt} > 0$, $\mathbf{pdvt} \geq \max(1, \mathbf{n})$;
otherwise $\mathbf{pdvt} \geq 1$.;

if **order** = Nag_RowMajor,

if $\mathbf{ncvt} > 0$, $\mathbf{pdvt} \geq \mathbf{ncvt}$;
otherwise $\mathbf{pdvt} \geq 1$..

11: **u**[dim] – Complex *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;
$\max(1, \mathbf{nru} \times \mathbf{pdu})$ when **order** = Nag_RowMajor.

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 entry: if $\mathbf{nru} > 0$, **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 unitary matrix Q returned by nag_zungbr (f08ktc) 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 $\mathbf{nru} = 0$, **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** $\geq \max(1, \mathbf{nru})$;
if **order** = Nag_RowMajor, **pdu** $\geq \max(1, \mathbf{n})$.

13: **c**[*dim*] – Complex *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.

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

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

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

On exit: **c** is overwritten by the matrix $U^{\mathrm{H}}C$. 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** $\geq \max(1, \mathbf{n})$;
otherwise **pdc** ≥ 1.;

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

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

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

On entry, **ncvt** $= \langle value \rangle$.
Constraint: **ncvt** ≥ 0.

On entry, **nru** $= \langle value \rangle$.
Constraint: **nru** ≥ 0.

On entry, **pdc** $= \langle value \rangle$.
Constraint: **pdc** > 0.

On entry, **pdu** $= \langle value \rangle$.
Constraint: **pdu** > 0.

On entry, **pdvt** $= \langle value \rangle$.
Constraint: **pdvt** > 0.

NE_INT_2

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

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

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

On entry, **pdvt** $= \langle value \rangle$ and **ncvt** $= \langle value \rangle$.
Constraint: if **ncvt** > 0, **pdvt** $\geq$ **ncvt**;
otherwise **pdvt** ≥ 1.

NE_INT_3

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

On entry, **pdvt** $= \langle value \rangle$, **ncvt** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: if **ncvt** > 0, **pdvt** $\geq \max(1, \mathbf{n})$;
otherwise **pdvt** ≥ 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 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 an 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 Parallelism and Performance

nag_zbdsqr (f08msc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zbdsqr (f08msc) 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 real floating-point operations is roughly proportional to n^2 if only the singular values are computed. About $12n^2 \times nru$ additional operations are required to compute the left singular vectors and about $12n^2 \times ncvt$ 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 real analogue of this function is nag_dbdsqr (f08mec).

10 Example

See Section 10 in nag_zungbr (f08ktc), which illustrates the use of the function to compute the singular value decomposition of a general matrix.