Module 6.3: nag_svd
Singular Value Decomposition (SVD)

`nag_svd` provides a procedure for computing the singular value decomposition (SVD) of a real or complex matrix. It also provides lower-level procedures for performing two computational sub-tasks involved in this problem.

Contents

Introduction 6.3.3

Procedures

`nag_gen_svd` 6.3.5
Singular value decomposition of a general real or complex matrix

`nag_gen_bidiag_reduc` 6.3.9
Reduction of a general real or complex matrix to real bidiagonal form

`nag_bidiag_svd` 6.3.13
Singular value decomposition of a real bidiagonal matrix

Examples

Example 1: SVD of a real matrix 6.3.17

Example 2: SVD of a complex matrix, using lower-level procedures 6.3.19

Additional Examples 6.3.23

References 6.3.24

Introduction

1 Notation and Background

The *singular value decomposition (SVD)* of an $m \times n$ matrix A is given by

$A = U\Sigma V^T$, with U and V orthogonal, if A is real;

$A = U\Sigma V^H$, with U and V unitary, if A is complex.

Here

Σ is an $m \times n$ diagonal matrix, whose $\min(m,n)$ diagonal elements are the *singular values* σ_i of A; they are real and non-negative, and arranged in descending order:

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min(m,n)} \geq 0.$$

U is a real orthogonal or complex unitary matrix of order m; its leading $\min(m,n)$ columns are the *left singular vectors* of A.

V is a real orthogonal or complex unitary matrix of order n; its leading $\min(m,n)$ columns are the *right singular vectors* of A.

The singular values and vectors satisfy $Av_i = \sigma_i u_i$ and $A^H u_i = \sigma_i v_i$, where u_i and v_i are the ith columns of U and V respectively.

The largest singular value σ_1 is the value of the 2-norm of A:

$$\sigma_1 = \|A\|_2$$

and the ratio of the largest to the smallest singular value gives the condition number of A in the 2-norm:

$$\begin{aligned} \kappa_2(A) &= \sigma_1/\sigma_{\min(m,n)} \\ &= \|A\|_2.\|A^{-1}\|_2 \text{ if } A \text{ is square and non-singular} \\ &= \|A\|_2.\|A^{\dagger}\|_2 \text{ where } A^{\dagger} \text{ denotes the pseudo-inverse of } A, \text{ and } A \text{ may be rectangular.} \end{aligned}$$

The singular value decomposition is useful for the numerical determination of the rank of a matrix, and for solving linear least-squares problems, especially when they are rank deficient or nearly so (see the module `nag_lin_lsq` (6.4)).

In exact arithmetic, a matrix A has rank r if it has precisely r non-zero singular values. In numerical work, because of uncertainties in the data and rounding errors in the computation, the effective rank is defined to be the number of singular values which are greater than or equal to a specified threshold.

2 Choice of Procedures

The procedure `nag_gen_svd` should be suitable for most purposes; it computes the singular values of A, and, if optional arguments are present, either the left or right singular vectors, or both.

The module also provides lower-level procedures which perform the two computational steps in the decomposition.

`nag_gen_bidiag_reduc` reduces the matrix A to a real bidiagonal matrix B by an orthogonal transformation: $A = QBP^T$, where Q and P are orthogonal (or if A is complex, $A = QBP^H$ with Q and P unitary).

`nag_bidiag_svd` computes the SVD of the real bidiagonal matrix B as $B = U_B \Sigma V_B^T$. The singular values of B are the singular values of A, and the SVD of A is given by $A = U\Sigma V^H$, where $U = QU_B$ and $V = PV_B$.

These lower-level procedures are intended for use by more experienced users.

For the use of the SVD to solve linear least-squares problems, see the module `nag_lin_lsq` (6.4).

Procedure: nag_gen_svd

1 Description

nag_gen_svd is a generic procedure which computes part or all of the singular value decomposition (SVD) of a real or complex $m \times n$ matrix.

By default, only the singular values are computed; optionally, the left or right singular vectors, or both, may be computed.

We write the singular value decomposition as:

$A = U\Sigma V^T$, with U and V orthogonal, if A is real;

$A = U\Sigma V^H$, with U and V unitary, if A is complex.

See the Module Introduction for further details.

The leading $\min(m, n)$ columns of U are the *left singular vectors* of A; the leading $\min(m, n)$ columns of V are the *right singular vectors* of A. Note that the procedure returns V^H ($= V^T$ if real); in other words, the right singular vectors are stored row-wise (and conjugated if they are complex).

The optional argument overwrite may be used to specify that either the left or the right singular vectors are to be overwritten on A. Alternatively, optional arguments u and/or vh may be supplied for storing the singular vectors.

Each pair of singular vectors u_i and v_i is normalized so that $\|u_i\|_2 = \|v_i\|_2 = 1$, but is defined only to within a factor ± 1 if the vector is real, or a complex factor of absolute value 1 if the vector is complex.

The procedure also has an option to pre-multiply by U^H ($= U^T$ if real) either a vector c or a matrix C; this facility may be useful if the computed SVD is also being used to solve a linear least-squares problem (see the module nag_lin_lsq (6.4)).

2 Usage

```
USE nag_svd
CALL nag_gen_svd(a, sigma  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

m — the number of rows of the matrix A

n — the number of columns of the matrix A

n_U $(\min(m, n) \leq n_U \leq m)$ — the number of columns to be computed of the matrix U

m_V $(\min(m, n) \leq m_V \leq n)$ — the number of rows to be computed of the matrix V^H

k — the number of columns of the matrix C

3.1 Mandatory Arguments

a(m, n) — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the matrix A.

Output: the contents of `a` on exit according to the setting of `overwrite`.

If `overwrite` is not present or `overwrite` = `'n'` or `'N'`, A is overwritten by intermediate results;

if `overwrite` = `'u'` or `'U'`, the leading $\min(m, n)$ columns of A are overwritten by the left singular vectors, stored column-wise;

if `overwrite` = `'v'` or `'V'`, the leading $\min(m, n)$ rows of A are overwritten by the right singular vectors, stored row-wise.

sigma$(\min(m, n))$ — real(kind=*wp*), intent(out)

Output: the singular values of A in descending order.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

overwrite — character(len=1), intent(in), optional

Input: specifies whether the left or right singular vectors are to be overwritten on A.

If `overwrite` = `'n'` or `'N'`, no left or right singular vectors are computed unless either `u` or `vh` is present;

if `overwrite` = `'u'` or `'U'`, the leading $\min(m, n)$ columns of A are overwritten by the corresponding columns of U (the left singular vectors);

if `overwrite` = `'v'` or `'V'`, the leading $\min(m, n)$ rows of A are overwritten by the corresponding rows of V^H (the right singular vectors).

Default: `overwrite` = `'n'`.

Constraints: `overwrite` = `'n'`, `'N'`, `'u'`, `'U'`, `'v'` or `'V'`.

u(m, n_U) — real(kind=*wp*) / complex(kind=*wp*), intent(out), optional

Output: the first n_U columns of the matrix U. The most likely values of n_U are: $\min(m, n)$, giving the first $\min(m, n)$ columns of U (the left singular vectors); or m, giving the whole of U.

Note: if `overwrite` = `'u'` and `u` is also present, then `u` is not used and a warning is raised.

Constraints: `u` must be of the same type as `a`.

vh(m_V, n) — real(kind=*wp*) / complex(kind=*wp*), intent(out), optional

Output: the first m_V rows of the matrix V^H (= V^T if real). The most likely values of m_V are: $\min(m, n)$, giving the first $\min(m, n)$ rows of V^H (the right singular vectors); or n, giving the whole of V^H.

Note: if `overwrite` = `'v'` and `vh` is also present, then `vh` is not used and a warning is raised.

Constraints: `vh` must be of the same type as `a`.

c_vec(m) — real(kind=*wp*) / complex(kind=*wp*), intent(inout), optional

Input: a vector c.

Output: overwritten by $U^H c$ (= $U^T c$ if U is real).

Constraints: `c_vec` must be of the same type as `a`; `c_vec` and `c_mat` must not both be present.

c_mat(m, k) — real(kind=*wp*) / complex(kind=*wp*), intent(inout), optional

Input: a matrix C.

Output: overwritten by $U^H C$ (= $U^T C$ if U is real).

Constraints: `c_mat` must be of the same type as `a`; `c_mat` and `c_vec` must not both be present.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
303	Array arguments have inconsistent shapes.
304	Invalid presence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Failure to converge. (This error is not likely to occur.) The QR algorithm failed to compute all the singular values in the permitted number of iterations.

Warnings (error%level = 1):

error%code	Description
101	Optional argument present but not used. One of the following conditions has occurred: `u` is present when `overwrite` = `'u'`: the left singular vectors are returned in `a`, and `u` is not used; `vh` is present when `overwrite` = `'v'`: the right singular vectors are returned in `a`, and `vh` is not used.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

In that example, the following call statement is used to compute the left and right singular vectors in arrays `u` and `vh`:

```
CALL nag_gen_svd( a, sigma, u=u, vh=vh )
```

To compute only the right singular vectors, overwriting them on A, the call statement should be changed to:

```
CALL nag_gen_svd( a, sigma, overwrite='v' )
```

To overwrite the left singular vectors on A and to store the right singular vectors in a separate array `vh`, the call statement should be changed to:

```
CALL nag_gen_svd( a, sigma, overwrite='u', vh=vh )
```

6 Further Comments

6.1 Algorithmic Detail

The procedure performs the following steps (see Sections 5.4.3 and 8.3.2 of Golub and Van Loan [4] for more details).

1. It calls `nag_gen_bidiag_reduc` to reduce A to real bidiagonal form B, using an orthogonal or unitary transformation: $A = QBP^H$. If left singular vectors are required, it forms the matrix Q; if right singular vectors are required, it forms the matrix P^H.
2. It then calls `nag_bidiag_svd`, which uses the bidiagonal QR algorithm to compute the singular values of B, by applying orthogonal transformations, until the off-diagonal elements are negligible: thus $\Sigma = U_B^T B V_B$. Here U_B and V_B are the matrices of singular vectors of B. If the singular vectors of A are required, the transformations are applied also to Q and P^H, giving $U = QU_B$ and $V^T = V_B^T P^H$.

The algorithms are derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

The computed Σ, U and V satisfy the relation

$$A + E = U\Sigma V^H$$

where

$$\|E\|_2 = O(\epsilon)\|A\|_2 = O(\epsilon)\sigma_1$$

and $\epsilon =$ `EPSILON(1.0_wp)`.

Procedure: nag_gen_bidiag_reduc

1 Description

nag_gen_bidiag_reduc is a generic procedure which reduces a general real or complex $m \times n$ matrix A to real bidiagonal form B by an orthogonal or unitary transformation.

The transformation is written:

$A = QBP^T$ with Q and P orthogonal, if A is real;

$A = QBP^H$ with Q and P unitary, if A is complex.

B is upper bidiagonal if $m \geq n$, and lower bidiagonal if $m < n$.

By default, the transformation matrices Q and P are represented as products of elementary reflectors. Optionally, the matrices Q and/or P^H ($= P^T$ if real) can be formed explicitly (they may be required for a subsequent call to nag_bidiag_svd, to compute singular vectors).

The procedure also has an option to pre-multiply by Q^H ($= Q^T$ if real) either a vector c or a matrix C; this facility may be needed as a step in solving a linear least-squares problem (see the module nag_lin_lsq (6.4)).

2 Usage

```
USE nag_svd
CALL nag_gen_bidiag_reduc(a, d, e  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

m — the number of rows of the matrix A

n — the number of columns of the matrix A

n_Q $(\min(m,n) \leq n_Q \leq m)$ — the number of columns of the matrix Q required

m_P $(\min(m,n) \leq m_P \leq n)$ — the number of columns of the matrix P^T (P^H) required

k — the number of columns of the matrix C

3.1 Mandatory Arguments

a(m, n) — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the matrix A.

Output: the contents of a on exit according to the setting of overwrite.

If overwrite is not present or overwrite = 'n' or 'N', A is overwritten by details of Q and P, represented as products of elementary reflectors;

if overwrite = 'q' or 'Q', the leading $\min(m,n)$ columns of A are overwritten by the corresponding columns of Q;

if overwrite = 'p' or 'P', the leading $\min(m,n)$ rows of A are overwritten by the corresponding rows of P^H.

d($\min(m,n)$) — real(kind=*wp*), intent(out)

Output: the diagonal elements of the bidiagonal matrix B.

e($\min(m,n)-1$) — real(kind=*wp*), intent(out)

Output: the off-diagonal elements of the bidiagonal matrix B.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

overwrite — character(len=1), intent(in), optional

Input: specifies whether the matrix Q or P^H is to be overwritten on A.

If `overwrite` = `'n'` or `'N'`, the matrices Q and P^H are not explicitly formed unless either `q` or `ph` is present;

if `overwrite` = `'q'` or `'Q'`, the leading $\min(m,n)$ columns of A are overwritten by the corresponding columns of Q;

if `overwrite` = `'p'` or `'P'`, the leading $\min(m,n)$ rows of A are overwritten by the corresponding rows of P^H.

Default: `overwrite` = `'n'`.

Constraints: `overwrite` = `'n'`, `'N'`, `'q'`, `'Q'`, `'p'` or `'P'`.

q(m, n_Q) — real(kind=*wp*) / complex(kind=*wp*), intent(out), optional

Output: the first n_Q columns of the matrix Q. The most likely values of n_Q are $\min(m,n)$ or m.

Note: if `overwrite` = `'q'` and `q` is also present, then `q` is not used and a warning is raised.

Constraints: `q` must be of the same type as `a`.

ph(m_P, n) — real(kind=*wp*) / complex(kind=*wp*), intent(out), optional

Output: the first m_P rows of the matrix P^H ($= P^T$ if real). The most likely values of m_P are $\min(m,n)$ or n.

Note: if `overwrite` = `'p'` and `ph` is also present, then `ph` is not used and a warning is raised.

Constraints: `ph` must be of the same type as `a`.

c_vec(m) — real(kind=*wp*) / complex(kind=*wp*), intent(inout), optional

Input: a vector c.

Output: overwritten by $Q^H c$ ($= Q^T c$ if Q is real).

Constraints: `c_vec` must be of the same type as `a`; `c_vec` and `c_mat` must not both be present.

c_mat(m, k) — real(kind=*wp*) / complex(kind=*wp*), intent(inout), optional

Input: a matrix C.

Output: overwritten by $Q^H C$ ($= Q^T C$ if Q is real).

Constraints: `c_mat` must be of the same type as `a`; `c_mat` and `c_vec` must not both be present.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
303	Array arguments have inconsistent shapes.
304	Invalid presence of an optional argument.
320	The procedure was unable to allocate enough memory.

Warnings (error%level = 1):

error%code	Description
101	Optional argument present but not used. One of the following conditions has occurred: `q` is present when `overwrite = 'q'`: the leading $\min(m,n)$ columns of Q are returned in `a`, and `q` is not used. `ph` is present when `overwrite = 'p'`: the leading $\min(m,n)$ rows of P^H are returned in `a`, and `ph` is not used.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The reduction is performed by applying elementary reflectors (Householder matrices), as described in Section 5.4.3 of Golub and Van Loan [4].

The algorithm is derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

The computed matrices B, Q and P^H satisfy $QBP^H = A + E$, where

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

$c(n)$ is a modestly increasing function of n, and $\epsilon =$ `EPSILON(1.0_wp)`.

The elements of B 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 singular values and vectors.

6.3 Timing

For real data, the total number of floating-point operations performed is roughly as follows.

	$m \gg n$	$m = n$	$m \ll n$
To reduce A to bidiagonal form:	$4mn^2$	$(8/3)n^3$	$4m^2n$
To form the first n columns of Q:	$2mn^2$	$(4/3)n^3$	$(4/3)m^3$
To form the whole of Q:	$4m^2n$	$(4/3)n^3$	$(4/3)m^3$
To form the first m rows of P^H:	$(4/3)n^3$	$(4/3)n^3$	$2m^2n$
To form the whole of P^H:	$(4/3)n^3$	$(8/3)n^3$	$4mn^2$
To form Q^HC:	$4mnk$	$2n^2k$	$2m^2k$

For complex data, 4 times as many (real) floating-point operations are performed.

Procedure: nag_bidiag_svd

1 Description

nag_bidiag_svd computes part or all of the singular value decomposition (SVD) of a real upper or lower bidiagonal matrix B of order n:

$$B = U_B \Sigma V_B^T.$$

By default, only the singular values are computed. Optionally, the left or right singular vectors, or both, may be computed; for this, the optional arguments u and/or vh must be supplied and *must be initialized* to the unit matrix.

The procedure can also be used to compute the singular vectors of a real or complex $m_A \times n_A$ matrix A which has been reduced to real bidiagonal form B by nag_gen_bidiag_reduc:

$$A = QBP^T = (QU_B)\Sigma(V_B^T P^T), \text{ if } A \text{ is real;}$$

$$A = QBP^H = (QU_B)\Sigma(V_B^T P^H), \text{ if } A \text{ is complex.}$$

In this case, $n = \min(m_A, n_A)$.

To compute the singular vectors of A, the matrices Q and P^H must be formed explicitly by nag_gen_bidiag_reduc, and supplied to this procedure in the optional arguments u and vh. They are then overwritten by $U = QU_B$ and $V^H = V_B^T P^H$; the columns of U are the left singular vectors of A, and the rows of V^H are the right singular vectors of A.

Each pair of singular vectors u_i and v_i is normalized so that $\|u_i\|_2 = \|v_i\|_2 = 1$, but is defined only to within a factor ± 1 if the vector is real, or a complex factor of absolute value 1 if the vector is complex.

The procedure also has an option to pre-multiply by U_B^T either a vector c or a matrix C; this facility is needed if the procedure is used as a step in solving a linear least-squares problem (see the module nag_lin_lsq (6.4)).

2 Usage

```
USE nag_svd
CALL nag_bidiag_svd(nag_key, uplo, d, e  [, optional arguments])
```

2.1 Interfaces

Distinct interfaces are provided for the following cases.

Real / complex data

Real data: nag_key = nag_key_real, and the optional arguments u, vh, c_vec and c_mat are of type real(kind=*wp*).

Complex data: nag_key = nag_key_cmplx, and the optional arguments u, vh, c_vec and c_mat are of type complex(kind=*wp*).

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

n — the order of the matrix B

$m_A \geq n$ — the number of rows in the matrix of left singular vectors

$n_A \geq n$ — the number of columns in the matrix of right singular vectors

k — the number of columns in the matrix C

3.1 Mandatory Arguments

nag_key — a "key" argument, intent(in)

Input: must have one of the following values, which are named constants, each of a different derived type, defined by the Library, and accessible from this module:

`nag_key_real` if the arguments `u`, `vh`, `c_vec` and `c_mat` are of type `real(kind=`*wp*`)`;

`nag_key_cmplx` if the arguments `u`, `vh`, `c_vec` and `c_mat` are of type `complex(kind=`*wp*`)`.

If no optional array arguments are present, either value may be supplied. For further explanation of "key" arguments, see the Essential Introduction.

uplo — character(len=1), intent(in)

Input: specifies whether the matrix B is upper or lower bidiagonal.

If `uplo` = `'u'` or `'U'`, B is upper bidiagonal;

if `uplo` = `'l'` or `'L'`, B is lower bidiagonal.

Constraints: `uplo` = `'u'`, `'U'`, `'l'` or `'L'`.

d(n) — real(kind=*wp*), intent(inout)

Input: the diagonal elements of B.

Output: overwritten by the singular values in *descending* order.

e($n-1$) — real(kind=*wp*), intent(inout)

Input: the off-diagonal elements of B.

Output: overwritten by intermediate results.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

u(m_A, n) — real(kind=*wp*) / complex(kind=*wp*), intent(inout), optional

Input: if the left singular vectors of the bidiagonal matrix B are desired, then $m_A = n$ and `u` must be set to the unit matrix of order n.
If the left singular vectors of the general matrix A are desired, then m_A is the number of rows of A and `u` must contain the $m_A \times n$ matrix Q which was determined by `nag_gen_bidiag_reduc` when reducing A to bidiagonal form.

Output: the matrix in `u` is postmultiplied by the matrix U_B to give the desired left singular vectors.

Constraints: `u` must be of type real(kind=*wp*) if `nag_key` = `nag_key_real`, and of type complex(kind=*wp*) if `nag_key` = `nag_key_cmplex`.

vh(n, n_A) — real(kind=*wp*) / complex(kind=*wp*), intent(inout), optional

Input: if the right singular vectors of the bidiagonal matrix B are desired, then $n_A = n$ and `vh` must be set to the unit matrix of order n.
If the right singular vectors of the general matrix A are desired, then n_A is the number of columns of A and `vh` must contain the $n \times n_A$ matrix P^H which was determined by `nag_gen_bidiag_reduc` when reducing A to bidiagonal form.

Output: the matrix in `vh` is pre-multiplied by the matrix V_B^T to give the desired right singular vectors.

Constraints: `vh` must be of type real(kind=*wp*) if `nag_key` = `nag_key_real`, and of type complex(kind=*wp*) if `nag_key` = `nag_key_cmplex`.

c_vec(m_A) — real(kind=*wp*) / complex(kind=*wp*), intent(inout), optional

Input: a vector c.

Output: c is overwritten by $U_B^T c$.

Constraints: c_vec must be of type real(kind=*wp*) if nag_key = nag_key_real, and of type complex(kind=*wp*) if nag_key = nag_key_cmplex; c_vec and c_mat must not both be present.

c_mat(m_A, k) — real(kind=*wp*) / complex(kind=*wp*), intent(inout), optional

Input: a matrix C.

Output: C is overwritten by $U_B^T C$.

Constraints: c_mat must be of type real(kind=*wp*) if nag_key = nag_key_real, and of type complex(kind=*wp*) if nag_key = nag_key_cmplex; c_mat and c_vec must not both be present.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
303	Array arguments have inconsistent shapes.
304	Invalid presence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Failure to converge. (This error is not likely to occur.) The QR algorithm has failed to compute all the singular values in the permitted number of iterations; some of the off-diagonal elements of B have not become negligible. The bidiagonal matrix returned in d and e is orthogonally equivalent to B, and gives information about those singular values which have converged.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

In that example, in which the arrays a and vh are complex, the following call statement is used:

```
CALL nag_bidiag_svd(nag_key_cmplx,uplo,d,e,u=a,v=vh)
```

If the arrays a and vh are real, the call statement must be changed to:

```
CALL nag_bidiag_svd(nag_key_real,uplo,d,e,u=a,v=vh)
```

6 Further Comments

6.1 Algorithmic Detail

The procedure uses two different algorithms. If any singular vectors are required (that is, if `u`, `vh`, `c_vec` or `c_mat` is present), the procedure uses the bidiagonal QR algorithm, 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 [2]). Otherwise, if only singular values are required, they are computed by the differential qd algorithm (see Fernando and Parlett [3]), which is faster and can achieve even greater accuracy.

The algorithms are derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

Each singular value and vector is computed to high relative accuracy. However, the reduction to bidiagonal form (prior to calling this procedure) 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

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

where $c(n)$ is a modestly increasing function of n, and $\epsilon =$ `EPSILON(1.0_wp)`.

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

$$\theta(\tilde{u}_i, u_i) \leq \frac{c(n)\epsilon}{\text{relgap}_i}$$

where relgap_i is the relative gap between σ_i and any other singular value, defined by

$$\text{relgap}_i = \min_{i \neq j} \frac{|\sigma_i - \sigma_j|}{\sigma_i + \sigma_j}.$$

A similar bound holds for the right singular vectors.

6.3 Timing

The total number of floating-point operations performed is typically about $30n^2$ if only the singular values are computed, but depends on how rapidly the algorithm converges.

For real data, an additional $6n^2m_A$ operations are performed if `u` is present, and an additional $6n^2n_A$ if `vh` is present; these operations counts must be doubled if the data are complex.

When singular values only are required, the operations are all performed in scalar mode; the additional operations required to compute the singular vectors can be vectorized and on some machines may be performed much faster.

Example 1: SVD of a real matrix

Compute the singular values and left and right singular vectors of a real matrix A. This example calls the single procedure nag_gen_svd.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_svd_ex01

  ! Example Program Text for nag_svd
  ! NAG f190, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_svd, ONLY : nag_gen_svd
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, MIN
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n, ns
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), sigma(:), u(:,:), vh(:,:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_svd_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n
  ns = MIN(m,n)

  ALLOCATE (a(m,n),sigma(ns),u(m,ns),vh(ns,n)) ! Allocate storage

  READ (nag_std_in,*) (a(i,:),i=1,m)

  ! Compute the SVD

  CALL nag_gen_svd(a,sigma,u=u,vh=vh)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Singular values'
  WRITE (nag_std_out,'(4X,5(F7.4:,1X))') sigma
  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(u,int_col_labels=.TRUE., &
   row_labels=(/(' ',i=1,m)/),title= &
   'Left singular vectors (one vector per column)')

  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(vh,int_row_labels=.TRUE., &
   title='Right singular vectors (one vector per row)')

  DEALLOCATE (a,sigma,u,vh)    ! Deallocate storage

END PROGRAM nag_svd_ex01
```

2 Program Data

```
Example Program Data for nag_svd_ex01
  6  5                              : Values of m and n
 -0.09   0.14  -0.46   0.68   1.29
 -1.56   0.20   0.29   1.09   0.51
 -1.48  -0.43   0.89  -0.71  -0.96
 -1.09   0.84   0.77   2.11  -1.27
  0.08   0.55  -1.13   0.14   1.74
 -1.59  -0.72   1.06   1.24   0.34 : End of Matrix A
```

3 Program Results

```
 Example Program Results for nag_svd_ex01

 Singular values
     3.9997  2.9962  2.0001  0.9988  0.0025

 Left singular vectors (one vector per column)
          1       2       3       4       5
     0.0468 -0.5061  0.0627 -0.1233 -0.8302
    -0.3914 -0.3776  0.2058  0.3269  0.3227
    -0.3220  0.4146  0.5273  0.5287 -0.3543
    -0.6397 -0.0734 -0.6933  0.1872 -0.1228
     0.2558 -0.6094  0.1009  0.4601  0.2050
    -0.5161 -0.2293  0.4299 -0.5930  0.1549

 Right singular vectors (one vector per row)
 1   0.6554  0.0104 -0.4376 -0.5300  0.3130
 2   0.1391 -0.1857  0.2942 -0.5255 -0.7638
 3  -0.5134 -0.5066  0.1540 -0.5115  0.4409
 4  -0.5064  0.6588 -0.3827 -0.3792 -0.1389
 5  -0.1765 -0.5241 -0.7428  0.1934 -0.3239
```

Example 2: SVD of a complex matrix, using lower-level procedures

Compute the singular values and left and right singular vectors of a complex matrix A. This example calls the lower-level procedures `nag_gen_bidiag_reduc` and `nag_bidiag_svd`. It calls `nag_gen_bidiag_reduc` to reduce A to real bidiagonal form, and to compute the unitary transformation matrices Q and P^H. It then calls `nag_bidiag_svd` to compute the singular values of B, and to overwrite the matrices Q and P^H with the matrices of left and right singular vectors of A.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_svd_ex02

  ! Example Program Text for nag_svd
  ! NAG f190, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_svd, ONLY : nag_key_cmplx, nag_gen_bidiag_reduc, nag_bidiag_svd
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, MAX, MIN
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n, ns
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: d(:), e(:)
  COMPLEX (wp), ALLOCATABLE :: a(:,:), vh(:,:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_svd_ex02'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m, n

  ns = MIN(m,n)
  IF (m>=n) THEN
    uplo = 'U'
  ELSE
    uplo = 'L'
  END IF

  ALLOCATE (a(m,n),d(ns),e(MAX(0,ns-1)),vh(ns,n)) ! Allocate storage

  READ (nag_std_in,*) (a(i,:),i=1,m)

  ! Reduction to bidiagonal form

  CALL nag_gen_bidiag_reduc(a,d,e,overwrite='Q',ph=vh)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Results of the reduction to bidiagonal form'
  WRITE (nag_std_out,*) 'Diagonal'
  WRITE (nag_std_out,'(14X,4(F7.4:,11X))') d
```

```
      IF (ns>1) THEN
        SELECT CASE (uplo)
        CASE ('U','u')
          WRITE (nag_std_out,*) 'First super-diagonal'
        CASE ('L','l')
          WRITE (nag_std_out,*) 'First sub-diagonal'
        END SELECT
        WRITE (nag_std_out,'(14X,4(F7.4:,11X))') e
      END IF

      ! Compute the SVD

      CALL nag_bidiag_svd(nag_key_cmplx,uplo,d,e,u=a,vh=vh)

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) 'Singular Value Decomposition'
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) 'Singular values'
      WRITE (nag_std_out,'(14X,4(F7.4:,11X))') d
      WRITE (nag_std_out,*)

      CALL nag_write_gen_mat(a,int_col_labels=.TRUE., &
       row_labels=(/(' ',i=1,m)/),format='(F7.4)',title= &
       'Left singular vectors (one vector per column)')

      WRITE (nag_std_out,*)

      CALL nag_write_gen_mat(vh,int_row_labels=.TRUE.,format='(F7.4)', &
       title='Right singular vectors (one vector per row)')

      DEALLOCATE (a,d,e,vh)        ! Deallocate storage

    END PROGRAM nag_svd_ex02
```

2 Program Data

```
Example Program Data for nag_svd_ex02
  5  4                                                    : Values of m, n
 ( 0.47,-0.34) (-0.40, 0.54) ( 0.60, 0.01) ( 0.80,-1.02)
 (-0.32,-0.23) (-0.05, 0.20) (-0.26,-0.44) (-0.43, 0.17)
 ( 0.35,-0.60) (-0.52,-0.34) ( 0.87,-0.11) (-0.34,-0.09)
 ( 0.89, 0.71) (-0.45,-0.45) (-0.02,-0.57) ( 1.14,-0.78)
 (-0.19, 0.06) ( 0.11,-0.85) ( 1.44, 0.80) ( 0.07, 1.14) : End of Matrix A
```

3 Program Results

```
Example Program Results for nag_svd_ex02

Results of the reduction to bidiagonal form
Diagonal
            -1.5199           -1.4211            0.9249            0.0192
First super-diagonal
            -1.9194            1.9985           -1.0610

Singular Value Decomposition

Singular values
             2.9979            1.9983            1.0044            0.0064

Left singular vectors (one vector per column)
                  1                 2                 3                 4
   (-0.4388,-0.0009) ( 0.2766,-0.2091) ( 0.7189, 0.3089) (-0.1991,-0.0538)
   ( 0.0523, 0.1531) (-0.3131,-0.0540) (-0.1777, 0.0488) (-0.7147,-0.0842)
   (-0.0928,-0.0554) ( 0.1119,-0.6069) (-0.3662, 0.0072) ( 0.2567,-0.1220)
   (-0.5431,-0.2297) ( 0.2200, 0.3164) (-0.2818,-0.3674) (-0.3668,-0.1447)
   ( 0.3633,-0.5384) ( 0.1710,-0.4692) ( 0.0290,-0.0668) (-0.4068, 0.1945)

Right singular vectors (one vector per row)
1  (-0.3352, 0.0000) ( 0.3721,-0.1084) (-0.0616, 0.4811) (-0.4465, 0.5503)
2  ( 0.5389, 0.0000) ( 0.0529,-0.2020) ( 0.0597, 0.7312) ( 0.0289,-0.3564)
3  (-0.3736, 0.0000) ( 0.4361, 0.5487) ( 0.3416, 0.2185) ( 0.3581,-0.2766)
4  (-0.6765, 0.0000) (-0.3830,-0.4102) (-0.1106, 0.2234) ( 0.0465,-0.4038)
```

Additional Examples

Not all example programs supplied with NAG *fl*90 appear in full in this module document. The following additional examples, associated with this module, are available.

`nag_svd_ex03`

SVD of a real matrix, using lower-level procedures.

`nag_svd_ex04`

SVD of a complex matrix, using `nag_gen_svd`.

References

[1] Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A, Ostrouchov S and Sorensen D (1995) *LAPACK Users' Guide* (2nd Edition) SIAM, Philadelphia

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

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

[4] Golub G H and Van Loan C F (1989) *Matrix Computations* Johns Hopkins University Press (2nd Edition)