NAG Library Function Document

nag_zgesvd (f08kpc)

1 Purpose

nag_zgesvd (f08kpc) computes the singular value decomposition (SVD) of a complex m by n matrix A, optionally computing the left and/or right singular vectors.

2 Specification

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

```
void nag_zgesvd (Nag_OrderType order, Nag_ComputeUType jobu,
     Nag_ComputeVTType jobvt, Integer m, Integer n, Complex a[], Integer pda,
     double s[], Complex u[], Integer pdu, Complex vt[], Integer pdvt,
     double rwork[], NagError *fail)
```

3 Description

The SVD is written as

$$A = U\Sigma V^{\mathrm{H}},$$

where Σ is an m by n matrix which is zero except for its $\min(m, n)$ diagonal elements, U is an m by m unitary matrix, and V is an n by n unitary matrix. The diagonal elements of Σ are the singular values of A; they are real and non-negative, and are returned in descending order. The first $\min(m, n)$ columns of U and V are the left and right singular vectors of A.

Note that the function returns V^{H}, not V.

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: **jobu** – Nag_ComputeUType *Input*

On entry: specifies options for computing all or part of the matrix U.

jobu = Nag_AllU
All m columns of U are returned in array **u**.

jobu = Nag_SingularVecsU
The first min(m, n) columns of U (the left singular vectors) are returned in the array **u**.

jobu = Nag_Overwrite
The first min(m, n) columns of U (the left singular vectors) are overwritten on the array **a**.

jobu = Nag_NotU
No columns of U (no left singular vectors) are computed.

Constraint: **jobu** = Nag_AllU, Nag_SingularVecsU, Nag_Overwrite or Nag_NotU.

3: **jobvt** – Nag_ComputeVTType *Input*

On entry: specifies options for computing all or part of the matrix V^{H}.

jobvt = Nag_AllVT
All n rows of V^{H} are returned in the array **vt**.

jobvt = Nag_SingularVecsVT
The first min(m, n) rows of V^{H} (the right singular vectors) are returned in the array **vt**.

jobvt = Nag_OverwriteVT
The first min(m, n) rows of V^{H} (the right singular vectors) are overwritten on the array **a**.

jobvt = Nag_NotVT
No rows of V^{H} (no right singular vectors) are computed.

Constraints:

jobvt = Nag_AllVT, Nag_SingularVecsVT, Nag_OverwriteVT or Nag_NotVT;
if **jobu** = Nag_Overwrite, **jobvt** $\neq$ Nag_OverwriteVT.

4: **m** – Integer *Input*

On entry: m, the number of rows of the matrix A.

Constraint: **m** ≥ 0.

5: **n** – Integer *Input*

On entry: n, the number of columns of the matrix A.

Constraint: **n** ≥ 0.

6: **a**[*dim*] – Complex *Input/Output*

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

max(1, **pda** × **n**) when **order** = Nag_ColMajor;
max(1, **m** × **pda**) when **order** = Nag_RowMajor.

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

a[$(j-1)$ × **pda** + $i - 1$] when **order** = Nag_ColMajor;
a[$(i-1)$ × **pda** + $j - 1$] when **order** = Nag_RowMajor.

On entry: the m by n matrix A.

On exit: if **jobu** = Nag_Overwrite, **a** is overwritten with the first min(m, n) columns of U (the left singular vectors, stored column-wise).

If **jobvt** = Nag_OverwriteVT, **a** is overwritten with the first min(m, n) rows of V^{H} (the right singular vectors, stored row-wise).

If **jobu** $\neq$ Nag_Overwrite and **jobvt** $\neq$ Nag_OverwriteVT, the contents of **a** are destroyed.

7: **pda** – Integer *Input*

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

Constraints:

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

8: **s**[*dim*] – double *Output*

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

On exit: the singular values of A, sorted so that $\mathbf{s}[i-1] \geq \mathbf{s}[i]$.

9: **u**[*dim*] – Complex *Output*

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

$\max(1, \mathbf{pdu} \times \mathbf{m})$ when **jobu** = Nag_AllU;
$\max(1, \mathbf{pdu} \times \min(\mathbf{m}, \mathbf{n}))$ when **jobu** = Nag_SingularVecsU and **order** = Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pdu})$ when **jobu** = Nag_SingularVecsU and **order** = Nag_RowMajor;
$\max(1, \mathbf{m})$ 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 **jobu** = Nag_AllU, **u** contains the m by m unitary matrix U.

If **jobu** = Nag_SingularVecsU, **u** contains the first $\min(m, n)$ columns of U (the left singular vectors, stored column-wise).

If **jobu** = Nag_NotU or Nag_Overwrite, **u** is not referenced.

10: **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,

if **jobu** = Nag_AllU, **pdu** $\geq \max(1, \mathbf{m})$;
if **jobu** = Nag_SingularVecsU, **pdu** $\geq \max(1, \mathbf{m})$;
otherwise **pdu** ≥ 1.;

if **order** = Nag_RowMajor,

if **jobu** = Nag_AllU, **pdu** $\geq \max(1, \mathbf{m})$;
if **jobu** = Nag_SingularVecsU, **pdu** $\geq \max(1, \min(\mathbf{m}, \mathbf{n}))$;
otherwise **pdu** ≥ 1..

11: **vt**[*dim*] – Complex *Output*

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

$\max(1, \mathbf{pdvt} \times \mathbf{n})$ when **jobvt** = Nag_AllVT;
$\max(1, \mathbf{pdvt} \times \mathbf{n})$ when **jobvt** = Nag_SingularVecsVT and **order** = Nag_ColMajor;
$\max(1, \min(\mathbf{m}, \mathbf{n}) \times \mathbf{pdvt})$ when **jobvt** = Nag_SingularVecsVT and **order** = Nag_RowMajor;
$\max(1, \min(\mathbf{m}, \mathbf{n}))$ otherwise.

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 exit: if **jobvt** = Nag_AllVT, **vt** contains the n by n unitary matrix V^{H}.

If **jobvt** = Nag_SingularVecsVT, **vt** contains the first $\min(m, n)$ rows of V^{H} (the right singular vectors, stored row-wise).

If **jobvt** = Nag_NotVT or Nag_OverwriteVT, **vt** is not referenced.

12: **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 **jobvt** = Nag_AllVT, **pdvt** $\geq \max(1, \mathbf{n})$;
if **jobvt** = Nag_SingularVecsVT, **pdvt** $\geq \max(1, \min(\mathbf{m}, \mathbf{n}))$;
otherwise **pdvt** ≥ 1.;

if **order** = Nag_RowMajor,

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

13: **rwork**[**min**(**m**, **n**)] – double *Output*

On exit: if **fail.code** = NE_CONVERGENCE, **RWORK**$(1 : \min(\mathbf{m}, \mathbf{n}) - 1)$ (using the notation described in Section 2.3.1.4 in How to Use the NAG Library and its Documentation) contains the unconverged superdiagonal elements of an upper bidiagonal matrix B whose diagonal is in S (not necessarily sorted). B satisfies $A = UBV^{\mathrm{T}}$, so it has the same singular values as A, and singular vectors related by U and V^{T}.

14: **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_CONSTRAINT

On entry, **jobvt** = ⟨*value*⟩ and **jobu** = ⟨*value*⟩.
Constraint: **jobvt** = Nag_AllVT, Nag_SingularVecsVT, Nag_OverwriteVT or Nag_NotVT and and
if **jobu** = Nag_Overwrite, **jobvt** $\neq$ Nag_OverwriteVT.

NE_CONVERGENCE

If nag_zgesvd (f08kpc) did not converge, **fail.errnum** specifies how many superdiagonals of an intermediate bidiagonal form did not converge to zero.

NE_ENUM_INT_2

On entry, **jobu** = ⟨*value*⟩, **pdu** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: if **jobu** = Nag_AllU, **pdu** $\geq \max(1, \mathbf{m})$;
if **jobu** = Nag_SingularVecsU, **pdu** $\geq \max(1, \mathbf{m})$;
otherwise **pdu** ≥ 1.

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

NE_ENUM_INT_3

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

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

NE_INT

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

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

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

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

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

NE_INT_2

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

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

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.

7 Accuracy

The computed singular value decomposition is nearly the exact singular value decomposition for a nearby matrix $(A + E)$, where

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

and ϵ is the ***machine precision***. In addition, the computed singular vectors are nearly orthogonal to working precision. See Section 4.9 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_zgesvd (f08kpc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgesvd (f08kpc) 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 approximately proportional to mn^2 when $m > n$ and m^2n otherwise.

The singular values are returned in descending order.

The real analogue of this function is nag_dgesvd (f08kbc).

10 Example

This example finds the singular values and left and right singular vectors of the 6 by 4 matrix

$$A = \begin{pmatrix} 0.96 - 0.81i & -0.03 + 0.96i & -0.91 + 2.06i & -0.05 + 0.41i \\ -0.98 + 1.98i & -1.20 + 0.19i & -0.66 + 0.42i & -0.81 + 0.56i \\ 0.62 - 0.46i & 1.01 + 0.02i & 0.63 - 0.17i & -1.11 + 0.60i \\ -0.37 + 0.38i & 0.19 - 0.54i & -0.98 - 0.36i & 0.22 - 0.20i \\ 0.83 + 0.51i & 0.20 + 0.01i & -0.17 - 0.46i & 1.47 + 1.59i \\ 1.08 - 0.28i & 0.20 - 0.12i & -0.07 + 1.23i & 0.26 + 0.26i \end{pmatrix},$$

together with approximate error bounds for the computed singular values and vectors.

The example program for nag_zgesdd (f08krc) illustrates finding a singular value decomposition for the case $m \leq n$.

10.1 Program Text

```
/* nag_zgesvd (f08kpc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

```
#include <nagx02.h>
#include <nagx04.h>
#include <naga02.h>

int main(void)
{

  /* Scalars */
  Complex alpha, beta;
  double eps, norm, serrbd;
  Integer exit_status = 0, i, j, m, n, pda, pdd, pdu, pdvt;

  /* Arrays */
  Complex *a = 0, *d = 0, *u = 0, *vt = 0;
  double *rcondu = 0, *rcondv = 0, *s = 0, *uerrbd = 0, *verrbd = 0;
  double *rwork = 0;

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

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J - 1) * pda + I - 1]
  order = Nag_ColMajor;
#else
#define A(I, J) a[(I - 1) * pda + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zgesvd (f08kpc) 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 "%" NAG_IFMT "%*[^\n]", &m, &n);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &m, &n);
#endif
  if (m < 0 || n < 0) {
    printf("Invalid m or n\n");
    exit_status = 1;
    goto END;
  }

  /* Allocate memory: these assume that A is overwritten by U, and
   * all of VT is required.
   */
  if (!(a = NAG_ALLOC(m * n, Complex)) ||
      !(d = NAG_ALLOC(m * n, Complex)) ||
      !(u = NAG_ALLOC(1, Complex)) ||
      !(vt = NAG_ALLOC(MIN(m, n) * n, Complex)) ||
      !(rcondu = NAG_ALLOC(MIN(m, n), double)) ||
      !(rcondv = NAG_ALLOC(MIN(m, n), double)) ||
      !(s = NAG_ALLOC(MIN(m, n), double)) ||
      !(uerrbd = NAG_ALLOC(MIN(m, n), double)) ||
      !(verrbd = NAG_ALLOC(MIN(m, n), double)) ||
      !(rwork = NAG_ALLOC(MIN(m, n), double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

#ifdef NAG_COLUMN_MAJOR
  pda = m;
```

```
#else
  pda = n;
#endif
  pdu = 1;
  pdvt = n;
  pdd = pda;

  /* Read the m by n matrix A from data file. */
  for (i = 1; i <= m; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
      scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Copy A to D: nag_zge_copy (f16tfc),
   * Complex valued general matrix copy.
   */
  nag_zge_copy(order, Nag_NoTrans, m, n, a, pda, d, pdd, &fail);

  /* nag_gen_complx_mat_print_comp (x04dbc): Print matrix A */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m,
                                n, a, pda, Nag_BracketForm, "%7.4f",
                                "Matrix A", Nag_IntegerLabels, 0,
                                Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  printf("\n");
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_zgesvd (f08kpc)
   * Compute the singular values and left and right singular vectors
   * of A (A = U*S*(V^H), m.ge.n)
   */
  nag_zgesvd(order, Nag_Overwrite, Nag_AllVT, m, n, a, pda, s, u, pdu, vt,
             pdvt, rwork, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgesvd (f08kpc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Reconstruct A from its decomposition and subtract from original A:
   * first, A <- U(A)*S, then D <- D - U*S*V^H using
   * nag_zgemm (f16zac).
   */
  for (i = 1; i <= m; i++)
    for (j = 1; j <= MIN(m, n); j++)
      A(i, j).re *= s[j - 1], A(i, j).im *= s[j - 1];

  alpha = nag_complex(-1.0, 0.0);
  beta = nag_complex(1.0, 0.0);
  nag_zgemm(order, Nag_NoTrans, Nag_NoTrans, m, n, n, alpha, a, pda, vt, pdvt,
            beta, d, pdd, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgemm (f16zac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Find norm of difference matrix D and print warning if it is too large:
```

```
   * nag_zge_norm (f16uac) using one-norm.
   */
  nag_zge_norm(order, Nag_OneNorm, m, n, d, pdd, &norm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zge_norm (f16uac).\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 A-(U*S*V^H) is much greater than 0.\n"
           " Schur factorization has failed.\n");
    exit_status = 1;
    goto END;
  }
  /* Print singular values and error estimates on values and vectors. */
  printf("\nSingular values\n");
  for (i = 0; i < n; ++i)
    printf("%10.4f%s", s[i], i % 8 == 7 ? "\n" : " ");
  printf("\n\n");

  /* Approximate error bound for the computed singular values.
   * Note that for the 2-norm, s[0] = norm(A).
   */
  serrbd = eps * s[0];

  /* Call nag_ddisna (f08flc) to estimate reciprocal condition numbers for the
   * singular vectors.
   */
  nag_ddisna(Nag_LeftSingVecs, m, n, s, rcondu, &fail);
  nag_ddisna(Nag_RightSingVecs, m, n, s, rcondv, &fail);

  /* Compute the error estimates for the singular vectors */
  for (i = 0; i < n; ++i) {
    uerrbd[i] = serrbd / rcondu[i];
    verrbd[i] = serrbd / rcondv[i];
  }
  printf("Error estimate for the singular values\n%11.1e\n", serrbd);

  printf("\nError estimates for the left singular vectors\n");
  for (i = 0; i < n; ++i)
    printf(" %10.1e%s", uerrbd[i], i % 6 == 5 ? "\n" : "");

  printf("\n\nError estimates for the right singular vectors\n");
  for (i = 0; i < n; ++i)
    printf(" %10.1e%s", verrbd[i], i % 6 == 5 ? "\n" : "");
  printf("\n");

END:
  NAG_FREE(a);
  NAG_FREE(d);
  NAG_FREE(u);
  NAG_FREE(vt);
  NAG_FREE(rcondu);
  NAG_FREE(rcondv);
  NAG_FREE(s);
  NAG_FREE(uerrbd);
  NAG_FREE(verrbd);
  NAG_FREE(rwork);

  return exit_status;
}

#undef A
```

10.2 Program Data

```
nag_zgesvd (f08kpc) Example Program Data

   6           4                                     :Values of M and N

 ( 0.96,-0.81) (-0.03, 0.96) (-0.91, 2.06) (-0.05, 0.41)
 (-0.98, 1.98) (-1.20, 0.19) (-0.66, 0.42) (-0.81, 0.56)
 ( 0.62,-0.46) ( 1.01, 0.02) ( 0.63,-0.17) (-1.11, 0.60)
 (-0.37, 0.38) ( 0.19,-0.54) (-0.98,-0.36) ( 0.22,-0.20)
 ( 0.83, 0.51) ( 0.20, 0.01) (-0.17,-0.46) ( 1.47, 1.59)
 ( 1.08,-0.28) ( 0.20,-0.12) (-0.07, 1.23) ( 0.26, 0.26) :End of matrix A
```

10.3 Program Results

```
nag_zgesvd (f08kpc) Example Program Results

 Matrix A
                   1                   2                   3                   4
 1  ( 0.9600,-0.8100)  (-0.0300, 0.9600)  (-0.9100, 2.0600)  (-0.0500, 0.4100)
 2  (-0.9800, 1.9800)  (-1.2000, 0.1900)  (-0.6600, 0.4200)  (-0.8100, 0.5600)
 3  ( 0.6200,-0.4600)  ( 1.0100, 0.0200)  ( 0.6300,-0.1700)  (-1.1100, 0.6000)
 4  (-0.3700, 0.3800)  ( 0.1900,-0.5400)  (-0.9800,-0.3600)  ( 0.2200,-0.2000)
 5  ( 0.8300, 0.5100)  ( 0.2000, 0.0100)  (-0.1700,-0.4600)  ( 1.4700, 1.5900)
 6  ( 1.0800,-0.2800)  ( 0.2000,-0.1200)  (-0.0700, 1.2300)  ( 0.2600, 0.2600)


Singular values
    3.9994     3.0003     1.9944     0.9995

Error estimate for the singular values
    4.4e-16

Error estimates for the left singular vectors
    4.4e-16    4.4e-16    4.5e-16    4.5e-16

Error estimates for the right singular vectors
    4.4e-16    4.4e-16    4.5e-16    4.5e-16
```