NAG Library Function Document

nag_complex_svd (f02xec)

1 Purpose

nag_complex_svd (f02xec) returns all, or part, of the singular value decomposition of a general complex matrix.

2 Specification

```
#include <nag.h>
#include <nagf02.h>

void nag_complex_svd (Integer m, Integer n, Complex a[], Integer tda,
     Integer ncolb, Complex b[], Integer tdb, Nag_Boolean wantq, Complex q[],
     Integer tdq, double sv[], Nag_Boolean wantp, Complex ph[], Integer tdph,
     Integer *iter, double e[], Integer *failinfo, NagError *fail)
```

3 Description

The m by n matrix A is factorized as

$$A = QDP^H$$

where

$$D = \begin{pmatrix} S \\ 0 \end{pmatrix} \quad m > n$$
$$D = S, \quad m = n$$
$$D = \begin{pmatrix} S & 0 \end{pmatrix} \quad m < n$$

Q is an m by m unitary matrix, P is an n by n unitary matrix and S is a $\min(m,n)$ by $\min(m,n)$ diagonal matrix with real non-negative diagonal elements, $sv_1, sv_2, \ldots, sv_{\min(m,n)}$, ordered such that

$$sv_1 \geq sv_2 \geq \ldots \geq sv_{\min(m,n)} \geq 0.$$

The first $\min(m,n)$ columns of Q are the left-hand singular vectors of A, the diagonal elements of S are the singular values of A and the first $\min(m,n)$ columns of P are the right-hand singular vectors of A.

Either or both of the left-hand and right-hand singular vectors of A may be requested and the matrix C given by

$$C = Q^H B$$

where B is an m by $ncolb$ given matrix, may also be requested.

The function obtains the singular value decomposition by first reducing A to upper triangular form by means of Householder transformations, from the left when $m \geq n$ and from the right when $m < n$. The upper triangular form is then reduced to bidiagonal form by Givens plane rotations and finally the QR algorithm is used to obtain the singular value decomposition of the bidiagonal form.

Good background descriptions to the singular value decomposition are given in Dongarra *et al.* (1979), Hammarling (1985) and Wilkinson (1978). Note that this function is not based on the LINPACK routine CSVDC.

Note that if K is any unitary diagonal matrix such that

$$KK^H = I$$

then

$$A = \begin{pmatrix} Q & K \end{pmatrix} D \begin{pmatrix} P & K \end{pmatrix}^H$$

is also a singular value decomposition of A.

4 References

Dongarra J J, Moler C B, Bunch J R and Stewart G W (1979) *LINPACK Users' Guide* SIAM, Philadelphia

Hammarling S (1985) The singular value decomposition in multivariate statistics *SIGNUM Newsl.* **20(3)** 2–25

Wilkinson J H (1978) Singular Value Decomposition – Basic Aspects *Numerical Software – Needs and Availability* (ed D A H Jacobs) Academic Press

5 Arguments

1: **m** – Integer *Input*

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

Constraint: $\mathbf{m} \geq 0$.

When $\mathbf{m} = 0$ then an immediate return is effected.

2: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 0$.

When $\mathbf{n} = 0$ then an immediate return is effected.

3: **a**[**m** × **tda**] – Complex *Input/Output*

Note: the (i, j)th element of the matrix A is stored in $\mathbf{a}[(i-1) \times \mathbf{tda} + j - 1]$.

On entry: the leading m by n part of the array **a** must contain the matrix A whose singular value decomposition is required.

On exit: if $\mathbf{m} \geq \mathbf{n}$ and **wantq** = Nag_TRUE, then the leading m by n part of **a** will contain the first n columns of the unitary matrix Q.

If $\mathbf{m} < \mathbf{n}$ and **wantp** = Nag_TRUE, then the leading m by n part of **a** will contain the first m rows of the unitary matrix P^H.

If $\mathbf{m} \geq \mathbf{n}$ and **wantq** = Nag_FALSE and **wantp** = Nag_TRUE, then the leading n by n part of **a** will contain the first n rows of the unitary matrix P^H.

Otherwise the contents of the leading m by n part of **a** are indeterminate.

4: **tda** – Integer *Input*

On entry: the stride separating matrix column elements in the array **a**.

Constraint: $\mathbf{tda} \geq \mathbf{n}$.

5: **ncolb** – Integer *Input*

On entry: $ncolb$, the number of columns of the matrix B. When $\mathbf{ncolb} = 0$ the array **b** is not referenced and may be **NULL**.

Constraint: $\mathbf{ncolb} \geq 0$.

6: **b**[**m** × **tdb**] – Complex *Input/Output*

Note: the (i, j)th element of the matrix B is stored in **b**$[(i-1) \times$ **tdb** $+ j - 1]$.

On entry: if **ncolb** > 0, the leading m by $ncolb$ part of the array **b** must contain the matrix to be transformed.

If **ncolb** $= 0$ the array **b** is not referenced and may be **NULL**.

On exit: **b** is overwritten by the m by $ncolb$ matrix $Q^H B$.

7: **tdb** – Integer *Input*

On entry: the stride separating matrix column elements in the array **b**.

Constraint: if **ncolb** > 0 then **tdb** $\geq$ **ncolb**.

8: **wantq** – Nag_Boolean *Input*

On entry: **wantq** must be Nag_TRUE if the left-hand singular vectors are required. If **wantq** = Nag_FALSE then the array **q** is not referenced and may be **NULL**.

9: **q**[**m** × **tdq**] – Complex *Output*

Note: the (i, j)th element of the matrix Q is stored in **q**$[(i-1) \times$ **tdq** $+ j - 1]$.

On exit: if **m** < **n** and **wantq** = Nag_TRUE, the leading m by m part of the array **q** will contain the unitary matrix Q. Otherwise the array **q** is not referenced and may be **NULL**.

10: **tdq** – Integer *Input*

On entry: the stride separating matrix column elements in the array **q**.

Constraint: if **m** < **n** and **wantq** = Nag_TRUE, **tdq** $\geq$ **m**

11: **sv**[**min**(**m**, **n**)] – double *Output*

On exit: the min(**m**, **n**) diagonal elements of the matrix S.

12: **wantp** – Nag_Boolean *Input*

On entry: **wantp** must be Nag_TRUE if the right-hand singular vectors are required. If **wantp** = Nag_FALSE then the array **ph** is not referenced and may be **NULL**.

13: **ph**[**n** × **tdph**] – Complex *Output*

Note: the (i, j)th element of the matrix is stored in **ph**$[(i-1) \times$ **tdph** $+ j - 1]$.

On exit: if **m** $\geq$ **n** and **wantq** and **wantp** are Nag_TRUE, the leading n by n part of the array **ph** will contain the unitary matrix P^H. Otherwise the array **ph** is not referenced and may be **NULL**.

14: **tdph** – Integer *Input*

On entry: the stride separating matrix column elements in the array **ph**.

Constraint: if **m** $\geq$ **n** and **wantq** = Nag_TRUE and **wantp** = Nag_TRUE, **tdph** $\geq$ **n**

15: **iter** – Integer * *Output*

On exit: the total number of iterations taken by the QR algorithm.

16: **e**[**min**(**m**, **n**)] – double *Output*

On exit: if the error NE_QR_NOT_CONV occurs the array **e** contains the super-diagonal elements of matrix E in the factorization of A according to $A = QEP^H$. See Section 6 for further details.

17: **failinfo** – Integer * *Output*

On exit: if the error NE_QR_NOT_CONV occurs **failinfo** contains the number of singular values which may not have been found correctly. See Section 6 for details.

18: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_2_INT_ARG_LT

On entry, **tda** = $\langle value \rangle$ while **n** = $\langle value \rangle$. These arguments must satisfy **tda** $\geq$ **n**.

On entry, **tdb** = $\langle value \rangle$ while **ncolb** = $\langle value \rangle$. These arguments must satisfy **tdb** $\geq$ **ncolb**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_INT_ARG_LT

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

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

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

NE_QR_NOT_CONV

The QR algorithm has failed to converge in $\langle value \rangle$ iterations. Singular values $1, 2, \ldots,$ **failinfo** may not have been found correctly and the remaining singular values may not be the smallest. The matrix A will nevertheless have been factorized as $A = QEP^{\mathrm{T}}$, where the leading $\min(m, n)$ by $\min(m, n)$ part of E is a bidiagonal matrix with **sv**$[0]$, **sv**$[1], \ldots,$ **sv**$[\min($**m**, **n** $- 1)]$ as the diagonal elements and **e**$[0]$, **e**$[1], \ldots,$ **e**$[\min($**m′n** $- 2)]$ as the super-diagonal elements. This failure is not likely to occur.

NE_TDP_LT_N

On entry, **tdph** = $\langle value \rangle$ while **n** = $\langle value \rangle$. When **wantq** and **wantp** are Nag_TRUE and **m** $\geq$ **n** then relationship **tdph** $\geq$ **n** must be satisfied.

NE_TDQ_LT_M

On entry, **tdq** = $\langle value \rangle$ while **m** = $\langle value \rangle$. When **wantq** is Nag_TRUE and **m** $<$ **n** then relationship **tdq** $\geq$ **m** must be satisfied.

7 Accuracy

The computed factors Q, D and P satisfy the relation

$$QDP^H = A + E$$

where $\|E\| \leq c\epsilon\|A\|$, ϵ being the ***machine precision***, c is a modest function of m and n and . denotes the spectral (two) norm. Note that $\|A\| = sv_1$.

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

For this function two examples are presented. There is a single example program for nag_complex_svd (f02xec), with a main program and the code to solve the two example problems is given in the functions ex1 and ex2.

Example 1 (ex1)

To find the singular value decomposition of the 5 by 3 matrix

$$A = \begin{pmatrix} 0.5i & -0.5 + 1.5i & -1.0 + 1.0i \\ 0.4 + 0.3i & 0.9 + 1.3i & 0.2 + 1.4i \\ 0.4 & -0.4 + 0.4i & 1.8 \\ 0.3 - 0.4i & 0.1 + 0.7i & 0.0 \\ -0.3i & 0.3 + 0.3i & 2.4i \end{pmatrix}$$

together with the vector $Q^H b$ for the vector

$$b = \begin{pmatrix} -0.55 + 1.05i \\ 0.49 + 0.93i \\ 0.56 - 0.16i \\ 0.39 + 0.23i \\ 1.13 + 0.83i \end{pmatrix}.$$

Example 2 (ex2)

To find the singular value decomposition of the 3 by 5 matrix

$$A = \begin{pmatrix} 0.5i & 0.4 - 0.3i & 0.4 & 0.3 + 0.4i & 0.3i \\ -0.5 - 1.5i & 0.9 - 1.3i & -0.4 - 0.4i & 0.1 - 0.7i & 0.3 - 0.3i \\ -1.0 - 1.0i & 0.2 - 1.4i & 1.8 & 0.0 & -2.4i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_complex_svd (f02xec) Example Program.
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 1, 1990.
 * Mark 8 revised, 2004.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagf02.h>

#define COMPLEX(A)      A.re, A.im
#define COMPLEX_CONJ(A) A.re, -A.im

static int ex1(void), ex2(void);

int main(void)
{
  Integer  exit_status_ex1 = 0;
  Integer  exit_status_ex2 = 0;

  printf("nag_complex_svd (f02xec) Example Program Results\n");
  scanf(" %*[^\n]");  /* Skip heading in data file */

  exit_status_ex1 = ex1();
  exit_status_ex2 = ex2();
```

```
  return (exit_status_ex1 == 0 && exit_status_ex2 == 0) ? 0 : 1;
}


#define A(I, J)  a[(I) *tda + J]
#define B(I, J)  b[(I) *tdb + J]
#define PH(I, J) ph[(I) *tdph + J]

static int ex1(void)
{
  Nag_Boolean wantp, wantq;
  Complex     *a = 0, *b = 0, *dummy = 0, *ph = 0;
  Integer     exit_status = 0, failinfo, i, iter, j, m, n, ncolb, tda, tdb,
              tdph;
  NagError    fail;
  double      *e = 0, *sv = 0;

  INIT_FAIL(fail);

  printf("Example 1\n\n");
  scanf(" %*[^\n]");   /* Skip heading in data file */
  if (scanf("%ld%ld", &m, &n) != EOF)
    {
      if (m >= 0 && n >= 0)
        {
          ncolb = 1;
          if (!(e = NAG_ALLOC(MIN(m, n)-1, double)) ||
              !(sv = NAG_ALLOC(MIN(m, n), double)) ||
              !(a = NAG_ALLOC(m*n, Complex)) ||
              !(b = NAG_ALLOC(m*ncolb, Complex)) ||
              !(ph = NAG_ALLOC(n*n, Complex)) ||
              !(dummy = NAG_ALLOC(1, Complex)))
            {
              printf("Allocation failure\n");
              exit_status = -1;
              goto END;
            }
          tda = n;
          tdb = ncolb;
          tdph = n;
        }
      else
        {
          printf("Invalid m or n.\n");
          exit_status = 1;
          return exit_status;
        }
      for (i = 0; i < m; ++i)
        for (j = 0; j < n; ++j)
          scanf("%lf%lf", COMPLEX(&A(i, j)));
      for (i = 0; i < m; ++i)
        for (j = 0; j < ncolb; ++j)
          scanf("%lf%lf", COMPLEX(&B(i, j)));

      /* Find the SVD of A. */

      wantq = Nag_TRUE;
      wantp = Nag_TRUE;
      /* nag_complex_svd (f02xec).
       * SVD of complex matrix
       */
      nag_complex_svd(m, n, a, tda, ncolb, b, tdb, wantq,
                      dummy, (Integer) 1, sv, wantp, ph, tdph, &iter,
                      e, &failinfo, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_complex_svd (f02xec).\n%s\n",
                 fail.message);
          exit_status = 1;
          goto END;
        }
```

```
      printf("Singular value decomposition of A\n\nSingular values\n");
      for (i = 0; i < n; ++i)
        printf("%9.4f%s", sv[i], (i%5 == 4 || i == n-1)?"\n":" ");
      printf("\nLeft-hand singular vectors, by column\n");
      for (i = 0; i < m; ++i)
        for (j = 0; j < n; ++j)
          printf("%7.4f %7.4f%s", COMPLEX(A(i, j)),
                 (j%3 == 2 || j == n-1)?"\n":"   ");
      printf("\nRight-hand singular vectors, by column\n");
      for (i = 0; i < n; ++i)
        for (j = 0; j < n; ++j)
          printf("%7.4f %7.4f%s", COMPLEX_CONJ(PH(j, i)),
                 (j%3 == 2 || j == n-1)?"\n":"   ");
      printf("\nVector conjg(Q')*B\n");
      for (i = 0; i < m; ++i)
        for (j = 0; j < ncolb; ++j)
          printf("%7.4f %7.4f%s", COMPLEX(B(i, j)),
                 (i%3 == 2 || i == m-1)?"\n":"   ");
    }
 END:
  NAG_FREE(e);
  NAG_FREE(sv);
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(ph);
  NAG_FREE(dummy);

  return exit_status;
}

#define A(I, J) a[(I) *tda + J]
#define Q(I, J) q[(I) *tdq + J]

static int ex2(void)
{
  Nag_Boolean wantp, wantq;
  Complex     *a = 0, *dummy = 0, *q = 0;
  Integer     exit_status = 0, failinfo, i, iter, j, m, n, ncolb, tda, tdq;
  NagError    fail;
  double      *e = 0, *sv = 0;

  INIT_FAIL(fail);

  printf("\nExample 2\n\n");
  scanf(" %*[^\n]");   /* Skip heading in data file */
  if (scanf("%ld%ld", &m, &n) != EOF)
    {
      if (m >= 0 && n >= 0)
        {
          if (!(e = NAG_ALLOC(MIN(m, n)-1, double)) ||
              !(sv = NAG_ALLOC(MIN(m, n), double)) ||
              !(a = NAG_ALLOC(m*n, Complex)) ||
              !(q = NAG_ALLOC(m*m, Complex)) ||
              !(dummy = NAG_ALLOC(1, Complex)))
            {
              printf("Allocation failure\n");
              exit_status = -1;
              goto END;
            }
          tda = n;
          tdq = m;
        }
      else
        {
          printf("Invalid m or n.\n");
          exit_status = 1;
          return exit_status;
        }
      for (i = 0; i < m; ++i)
        for (j = 0; j < n; ++j)
```

```
          if (scanf("%lf%lf", COMPLEX(&A(i, j))) != 2)
            {
              printf("Data input error: program terminated.\n");
              exit_status = 1;
              goto END;
            }

      /* Find the SVD of A. */

      wantq = Nag_TRUE;
      wantp = Nag_TRUE;
      ncolb = 0;

      /* nag_complex_svd (f02xec), see above. */
      nag_complex_svd(m, n, a, tda, ncolb, dummy, (Integer) 1, wantq,
                      q, tdq, sv, wantp, dummy, (Integer) 1, &iter,
                      e, &failinfo, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_complex_svd (f02xec).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      printf("Singular value decomposition of A\n\nSingular values\n");
      for (i = 0; i < m; ++i)
        printf("%9.4f%s", sv[i], (i%5 == 4 || i == m-1)?"\n":" ");
      printf("\nLeft-hand singular vectors, by column\n");
      for (i = 0; i < m; ++i)
        for (j = 0; j < m; ++j)
          printf("%7.4f %7.4f%s", COMPLEX(Q(i, j)),
                  (j%3 == 2 || j == n-1)?"\n":"   ");
      printf("\nRight-hand singular vectors, by column\n");
      for (i = 0; i < n; ++i)
        for (j = 0; j < m; ++j)
          printf("%7.4f %7.4f%s", COMPLEX_CONJ(A(j, i)),
                  (j%3 == 2 || j == n-1)?"\n":"   ");
    }
 END:
  NAG_FREE(e);
  NAG_FREE(sv);
  NAG_FREE(a);
  NAG_FREE(q);
  NAG_FREE(dummy);

  return exit_status;
}
```

10.2 Program Data

```
nag_complex_svd (f02xec) Example Program Data

Example 1
  5     3

 0.00  0.50   -0.50  1.50   -1.00  1.00
 0.40  0.30    0.90  1.30    0.20  1.40
 0.40  0.00   -0.40  0.40    1.80  0.00
 0.30 -0.40    0.10  0.70    0.00  0.00
 0.00 -0.30    0.30  0.30    0.00  2.40

-0.55  1.05    0.49  0.93    0.56 -0.16
 0.39  0.23    1.13  0.83

Example 2
  3     5

 0.00 -0.50    0.40 -0.30    0.40  0.00    0.30  0.40    0.00  0.30
-0.50 -1.50    0.90 -1.30   -0.40 -0.40    0.10 -0.70    0.30 -0.30
```

```
-1.00 -1.00    0.20 -1.40    1.80  0.00    0.00  0.00    0.00 -2.40
```

10.3 Program Results

```
nag_complex_svd (f02xec) Example Program Results
Example 1

Singular value decomposition of A

Singular values
   3.9263    2.0000    0.7641

Left-hand singular vectors, by column
-0.0757 -0.5079   -0.2831 -0.2831   -0.2251  0.1594
-0.4517 -0.2441   -0.3963  0.0566   -0.0075  0.2757
-0.2366  0.2669   -0.1359 -0.6341    0.2983 -0.2082
-0.0561 -0.0513   -0.3284 -0.0340    0.1670 -0.5978
-0.4820 -0.3277    0.3737  0.1019   -0.0976 -0.5664

Right-hand singular vectors, by column
-0.1275 -0.0000   -0.2265 -0.0000    0.9656 -0.0000
-0.3899  0.2046   -0.3397  0.7926   -0.1311  0.2129
-0.5289  0.7142   -0.0000 -0.4529   -0.0698 -0.0119

Vector conjg(Q')*B
-1.9656 -0.7935    0.1132 -0.3397    0.0915  0.6086
-0.0600 -0.0200    0.0400  0.1200

Example 2

Singular value decomposition of A

Singular values
   3.9263    2.0000    0.7641

Left-hand singular vectors, by column
-0.1275 -0.0000    0.2265 -0.0000   -0.9656  0.0000
-0.3899  0.2046    0.3397 -0.7926    0.1311 -0.2129
-0.5289  0.7142    0.0000  0.4529    0.0698  0.0119

Right-hand singular vectors, by column
-0.0757 -0.5079    0.2831  0.2831    0.2251 -0.1594
-0.4517 -0.2441    0.3963 -0.0566    0.0075 -0.2757
-0.2366  0.2669    0.1359  0.6341   -0.2983  0.2082
-0.0561 -0.0513    0.3284  0.0340   -0.1670  0.5978
-0.4820 -0.3277   -0.3737 -0.1019    0.0976  0.5664
```