NAG Library Function Document

nag_real_svd (f02wec)

1 Purpose

nag_real_svd (f02wec) returns all, or part, of the singular value decomposition of a general real matrix.

2 Specification

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

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

3 Description

The $m$ by $n$ matrix $A$ is factorized as

$$A = QDP^T$$

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 $m$ orthogonal matrix, $P$ is an $n$ by $n$ orthogonal matrix and $S$ is a $\min(m,n)$ by $\min(m,n)$ diagonal matrix with 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

$$C = Q^T B$$

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 SSVDC.

Note that if $K$ is any orthogonal diagonal matrix such that

$$KK^T = I,$$

so that $K$ has elements $+1$ or $-1$ on the diagonal, then
\[ A = (Q\ K)D(P\ K)^T \]

is also a singular value decomposition of \( A \).

4 References


5 Arguments

1: \( m \) – Integer \( \text{Input} \)

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

\( Constraint: m \geq 0. \)

When \( m = 0 \) then an immediate return is effected.

2: \( n \) – Integer \( \text{Input} \)

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

\( Constraint: n \geq 0. \)

When \( n = 0 \) then an immediate return is effected.

3: \( a[m \times \text{tda}] \) – double \( \text{Input/Output} \)

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

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

\( On \ exit: \) if \( m \geq n \) and \( \text{wantq} = \text{Nag_TRUE} \), then the leading \( m \) by \( n \) part of \( a \) will contain the first \( n \) columns of the orthogonal matrix \( Q \).

If \( m < n \) and \( \text{wantp} = \text{Nag_TRUE} \), then the leading \( m \) by \( n \) part of \( a \) will contain the first \( m \) rows of the orthogonal matrix \( P^T \).

If \( m \geq n \) and \( \text{wantq} = \text{Nag_FALSE} \) and \( \text{wantp} = \text{Nag_TRUE} \), then the leading \( n \) by \( n \) part of \( a \) will contain the first \( n \) rows of the orthogonal matrix \( P^T \).

Otherwise the contents of the leading \( m \) by \( n \) part of \( a \) are indeterminate.

4: \( \text{tda} \) – Integer \( \text{Input} \)

\( On \ entry: \) the stride separating matrix column elements in the array \( a \).

\( Constraint: \text{tda} \geq n. \)

5: \( \text{ncolb} \) – Integer \( \text{Input} \)

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

\( Constraint: \text{ncolb} \geq 0. \)
6: \[ b[m \times tdb] \] – double

**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 \( Q^T B \).

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: \[ \text{wantq} \] – Nag_Boolean

**Input**

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

9: \[ q[m \times tdq] \] – double

**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 \( \text{wantq} = \) Nag_TRUE, the leading \( m \) by \( m \) part of the array \( q \) will contain the orthogonal 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 \( \text{wantq} = \text{Nag_TRUE}, \) \( tdq \geq m \)

11: \[ sv[\min(m, n)] \] – double

**Output**

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

12: \[ \text{wantp} \] – Nag_Boolean

**Input**

**On entry:** \( \text{wantp} \) must be Nag_TRUE if the right-hand singular vectors are required. If \( \text{wantp} = \) Nag_FALSE, then the array \( pt \) is not referenced and may be NULL.

13: \[ pt[n \times tdpt] \] – double

**Output**

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

**On exit:** if \( m \geq n \) and \( \text{wantq} \) and \( \text{wantp} \) are Nag_TRUE, the leading \( n \) by \( n \) part of the array \( pt \) will contain the orthogonal matrix \( P^T \). Otherwise the array \( pt \) is not referenced and may be NULL.

14: \[ tdpt \] – Integer

**Input**

**On entry:** the stride separating matrix column elements in the array \( pt \).

**Constraint:** if \( m \geq n \) and \( \text{wantq} = \text{Nag_TRUE} \) and \( \text{wantp} = \text{Nag_TRUE} \), \( tdpt \geq n \)

15: \[ \text{iter} \] – 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 \( A = QEP^T \). See Section 6 for further details.
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.

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 \geq 0.

On entry, n = \langle value \rangle.

Constraint: n \geq 0.

On entry, ncolb = \langle value \rangle.

Constraint: ncolb \geq 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^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 superdiagonal elements. This failure is not likely to occur.

NE_TDP_LT_N

On entry, tdpt = \langle value \rangle while n = \langle value \rangle. When wantq and wantp are Nag_TRUE and m \geq n then relationship tdpt \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^T = A + E

where \|E\| \leq c\|A\|, c being the machine precision, c is a modest function of m and n and . denotes the spectral (two) norm. Note that \|A\| = sv1.

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_real_svd (f02wec), 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
\[
A = \begin{pmatrix}
2.0 & 2.5 & 2.5 \\
2.0 & 2.5 & 2.5 \\
1.6 & -0.4 & 2.8 \\
2.0 & -0.5 & 0.5 \\
1.2 & -0.3 & -2.9
\end{pmatrix}
\]
together with the vector \(Q^Tb\) for the vector
\[
b = \begin{pmatrix}
1.1 \\
0.9 \\
0.6 \\
0.0 \\
-0.8
\end{pmatrix}
\]

Example 2 (ex2)
To find the singular value decomposition of the 3 by 5 matrix
\[
A = \begin{pmatrix}
2.0 & 2.0 & 1.6 & 2.0 & 1.2 \\
2.5 & 2.5 & -0.4 & -0.5 & -0.3 \\
2.5 & 2.5 & -2.8 & 0.5 & -2.9
\end{pmatrix}
\]

10.1 Program Text
/* nag_real_svd (f02wec) Example Program. *
* Copyright 2014 Numerical Algorithms Group. *
* Mark 1, 1990.
* Mark 8 revised, 2004. */

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

#define EX1_MMAX 20
#define EX1_NMAX 10

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

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

    printf("nag_real_svd (f02wec) Example Program Results\n");
    #ifdef _WIN32
    scanf_s("%*[\n]"); /* Skip heading in data file */
    #else
    scanf("%*[\n]"); /* Skip heading in data file */
    

#endif

    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 PT(I, J) pt[(I) *tdpt + J]
#define Q(I, J) q[(I) *tdq + J]

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

    INIT_FAIL(fail);

    printf("Example 1\n");
    #ifdef _WIN32
        scanf_s(" %*[\n]"); /* Skip Example 1 heading */
    #else
        scanf(" %*[\n]"); /* Skip Example 1 heading */
    #endif
    #ifdef _WIN32
        scanf_s("%"NAG_IFMT"%"NAG_IFMT"", &m, &n);
    #else
        scanf("%"NAG_IFMT"%"NAG_IFMT"", &m, &n);
    #endif
    if (m >= 0 && n >= 0)
    {
        ncolb = 1;
        if (!((a = NAG_ALLOC(m*n, double)) ||
             (b = NAG_ALLOC(m*ncolb, double)) ||
             (e = NAG_ALLOC(MIN(m, n)-1, double)) ||
             (pt = NAG_ALLOC(n*n, double)) ||
             (sv = NAG_ALLOC(MIN(m, n), double)) ||
             (dummy = NAG_ALLOC(1, double))))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
        tda = n;
        tdb = ncolb;
        tdpt = n;
    }
    else
    {
        printf("Invalid m or n.\n");
        exit_status = 1;
        return exit_status;
    }
    #ifdef _WIN32
        scanf_s("%"NAG_IFMT\n");
    #else
        scanf("%"NAG_IFMT\n");
    #endif
    for (i = 0; i < m; ++i)
        for (j = 0; j < n; ++j)
            #ifdef _WIN32
                f02wec
            #else
                f02wec
            #endif
                Exit_status = ex1();

                return (exit_status == 0) ? 0 : 1;
}

#define A(I, J) a[(I) *tda + J]
#define B(I, J) b[(I) *tdb + J]
#define PT(I, J) pt[(I) *tdpt + J]
#define Q(I, J) q[(I) *tdq + J]
```c
    scanf_s("%lf", &A(i, j));
#else
    scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
    for (i = 0; i < m; ++i)
        for (j = 0; j < ncolb; ++j)
#ifdef _WIN32
    scanf_s("%lf", &B(i, j));
#else
    scanf("%lf", &B(i, j));
#endif
/* Find the SVD of A. */
wantq = Nag_TRUE;
wantp = Nag_TRUE;
/* nag_real_svd (f02wec). */
    nag_real_svd(m, n, a, tda, ncolb, b, tdb, wantq,
        dummy, (Integer) 1, sv, wantp, pt, tdpt, &iter,
        e, &failinfo, &fail);
    if (fail.code != NE_NOERROR)
        {
            printf("Error from nag_real_svd (f02wec).\n");
            exit_status = 1;
            goto END;
        }
    printf("Singular value decomposition of A\n\n");
    printf("Singular values\n");
    for (i = 0; i < n; ++i)
        printf("%8.4f", sv[i]);
    printf("\n\n");
    printf("Left-hand singular vectors, by column\n");
    for (i = 0; i < m; ++i)
        {
            for (j = 0; j < n; ++j)
                printf("%8.4f", A(i, j));
            printf("\n");
        }
    printf("\n");
    printf("Right-hand singular vectors, by column\n");
    for (i = 0; i < n; ++i)
        {
            for (j = 0; j < m; ++j)
                printf("%8.4f", PT(j, i));
            printf("\n");
        }
    printf("\n");
    printf("Vector Q'*B\n");
    for (i = 0; i < m; ++i)
        printf("%8.4f", b[i]);
    printf("\n\n");
END:
    NAG_FREE(a);
    NAG_FREE(b);
    NAG_FREE(e);
    NAG_FREE(pt);
    NAG_FREE(sv);
    NAG_FREE(dummy);
    return exit_status;
}

static int ex2(void)
{
    Nag_Boolean wantp, wantq;
    Integer exit_status = 0, failinfo, i, iter, j, m, n, ncolb, tda, tdq;
```
INIT_FAIL(fail);
printf("\nExample 2\n");
#ifdef _WIN32
scanf_s(" %*[\n]"); /* Skip Example 2 heading */
#else
scanf(" %*[\n]"); /* Skip Example 2 heading */
#endif
#ifdef _WIN32
scanf_s(" %*[\n]”);
#else
scanf(" %*[\n]”);
#endif
if (m >= 0 && n >= 0)
{
if (!(a = NAG_ALLOC(m*n, double)) ||
!(q = NAG_ALLOC(m*m, double)) ||
!(e = NAG_ALLOC(MIN(m, n)-1, double)) ||
!(sv = NAG_ALLOC(MIN(m, n), double)) ||
!(dummy = NAG_ALLOC(1, double)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}
else
{
printf("Invalid m or n.\n");
exit_status = 1;
return exit_status;
}
tda = n;
tdq = m;
#ifdef _WIN32
scanf_s(" %*[\n]”);
#else
scanf(" %*[\n]”);
#endif
for (i = 0; i < m; ++i)
for (j = 0; j < n; ++j)
#ifdef _WIN32
scanf_s("%lf", &A(i, j));
#else
scanf("%lf", &A(i, j));
#endif
/* Find the SVD of A. */
wantq = Nag_TRUE;
wantp = Nag_TRUE;
ncolb = 0;
/* nag_real_svd (f02wec), see above. */
nag_real_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_real_svd (f02wec).\n\n", fail.message);
exit_status = 1;
goto END;
}
printf("Singular value decomposition of A
\n\n\n");
printf("Singular values\n\n");
for (i = 0; i < m; ++i)
  printf(" %8.4f", sv[i]);
printf("\n\n");
printf("Left-hand singular vectors, by column\n\n");
for (i = 0; i < m; ++i)
  {
    for (j = 0; j < m; ++j)
      printf(" %8.4f", Q(i, j));
    printf("\n");
  }
printf("Right-hand singular vectors, by column\n\n");
for (i = 0; i < n; ++i)
  {
    for (j = 0; j < m; ++j)
      printf(" %8.4f", A(j, i));
    printf("\n");
  }
END:
NAG_FREE(a);
NAG_FREE(q);
NAG_FREE(e);
NAG_FREE(sv);
NAG_FREE(dummy);
return exit_status;
}

10.2 Program Data

nag_real_svd (f02wec) Example Program Data

Example 1
Values of m and n
  5   3

Matrix A
  2.0  2.5  2.5
  2.0  2.5  2.5
  1.6 -0.4  2.8
  2.0 -0.5  0.5
  1.2 -0.3 -2.9

Vector B
  1.1  0.9  0.6  0.0 -0.8

Example 2
Values of m and n
  3   5

Matrix A
  2.0  2.0  1.6  2.0  1.2
  2.5  2.5 -0.4 -0.5 -0.3
  2.5  2.5  2.8  0.5 -2.9

10.3 Program Results

nag_real_svd (f02wec) Example Program Results
Example 1
Singular value decomposition of A

Singular values
  6.5616  3.0000  2.4384

Left-hand singular vectors, by column
  0.6011 -0.1961 -0.3165
  0.6011 -0.1961 -0.3165
  0.4166  0.1569  0.6941

Mark 25
Right-hand singular vectors, by column
-0.6011  0.1961  0.3165
-0.6011  0.1961  0.3165
-0.4166  0.1569  0.6941
-0.1688  0.3922  0.5636
0.2742  0.8629  0.0139

Example 2
Singular value decomposition of A

Singular values
6.5616  3.0000  2.4384

Left-hand singular vectors, by column
-0.4694  0.7845  0.4054
-0.4324  0.1961  0.8801
-0.7699  0.5883  0.2471

Right-hand singular vectors, by column
-0.1688  0.3922  0.5636
-0.2742  0.8629  0.0139

Vector Q'*B
1.6716  0.3922  0.2276  -0.1000  -0.1000