NAG Toolbox

nag_lapack_dgesvj (f08kj)

1 Purpose

nag_lapack_dgesvj (f08kj) computes the one-sided Jacobi singular value decomposition (SVD) of a real m by n matrix A, $m \geq n$, with fast scaled rotations and de Rijk's pivoting, optionally computing the left and/or right singular vectors. For $m < n$, the functions nag_lapack_dgesvd (f08kb) or nag_lapack_dgesdd (f08kd) may be used.

2 Syntax

```
[a, sva, v, work, info] = nag_lapack_dgesvj(joba, jobu, jobv, a, mv, v, work, 'm', m, 'n', n)
[a, sva, v, work, info] = f08kj(joba, jobu, jobv, a, mv, v, work, 'm', m, 'n', n)
```

3 Description

The SVD is written as

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

where Σ is an n by n diagonal matrix, U is an m by n orthonormal matrix, and V is an n by n orthogonal matrix. The diagonal elements of Σ are the singular values of A in descending order of magnitude. The columns of U and V are the left and the right singular vectors of A.

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

Drmac Z and Veselic K (2008a) New fast and accurate Jacobi SVD algorithm I *SIAM J. Matrix Anal. Appl.* **29 4**

Drmac Z and Veselic K (2008b) New fast and accurate Jacobi SVD algorithm II *SIAM J. Matrix Anal. Appl.* **29 4**

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

5 Parameters

5.1 Compulsory Input Parameters

1: **joba** – CHARACTER(1)

Specifies the structure of matrix A.

joba = 'L'
The input matrix A is lower triangular.

joba = 'U'
The input matrix A is upper triangular.

joba = 'G'
The input matrix A is a general m by n matrix, **m** $\geq$ **n**.

Constraint: **joba** = 'L', 'U' or 'G'.

2: **jobu** – CHARACTER(1)

Specifies whether to compute the left singular vectors and if so whether you want to control their numerical orthogonality threshold.

jobu = 'U'
The left singular vectors corresponding to the nonzero singular values are computed and returned in the leading columns of **a**. See more details in the description of **a**. The numerical orthogonality threshold is set to approximately $tol = ctol \times \epsilon$, where ϵ is the ***machine precision*** and $ctol = \sqrt{m}$.

jobu = 'C'
Analogous to **jobu** = 'U', except that you can control the level of numerical orthogonality of the computed left singular vectors. The orthogonality threshold is set to $tol = ctol \times \epsilon$, where $ctol$ is given on input in **work**(1). The option **jobu** = 'C' can be used if $m \times \epsilon$ is a satisfactory orthogonality of the computed left singular vectors, so $ctol = \mathbf{m}$ could save a few sweeps of Jacobi rotations. See the descriptions of **a** and **work**(1).

jobu = 'N'
The matrix U is not computed. However, see the description of **a**.

Constraint: **jobu** = 'U', 'C' or 'N'.

3: **jobv** – CHARACTER(1)

Specifies whether and how to compute the right singular vectors.

jobv = 'V'
The matrix V is computed and returned in the array **v**.

jobv = 'A'
The Jacobi rotations are applied to the leading m_v by n part of the array **v**. In other words, the right singular vector matrix V is not computed explicitly, instead it is applied to an m_v by n matrix initially stored in the first **mv** rows of **v**.

jobv = 'N'
The matrix V is not computed and the array **v** is not referenced.

Constraint: **jobv** = 'V', 'A' or 'N'.

4: **a**$(lda, :)$ – REAL (KIND=nag_wp) array

The first dimension of the array **a** must be at least $\max(1, \mathbf{m})$.

The second dimension of the array **a** must be at least $\max(1, \mathbf{n})$.

The m by n matrix A.

5: **mv** – INTEGER

If **jobv** = 'A', the product of Jacobi rotations is applied to the first m_v rows of **v**.

If **jobv** $\neq$ 'A', **mv** is ignored. See the description of **jobv**.

6: **v**$(ldv, :)$ – REAL (KIND=nag_wp) array

The first dimension, ldv, of the array **v** must satisfy

if **jobv** = 'V', $ldv \geq \max(1, \mathbf{n})$;
if **jobv** = 'A', $ldv \geq \max(1, \mathbf{mv})$;
otherwise $ldv \geq 1$.

The second dimension of the array **v** must be at least $\max(1, \mathbf{n})$ if **jobv** = 'V' or 'A', and at least 1 otherwise.

If **jobv** = 'A', **v** must contain an m_v by n matrix to be premultiplied by the matrix V of right singular vectors.

7: **work**($lwork$) – REAL (KIND=nag_wp) array

$lwork$, the dimension of the array, must satisfy the constraint $lwork \geq \max(6, \mathbf{m} + \mathbf{n})$.

If **jobu** = 'C', **work**(1) = $ctol$, where $ctol$ defines the threshold for convergence. The process stops if all columns of A are mutually orthogonal up to $ctol \times \epsilon$. It is required that $ctol \geq 1$, i.e., it is not possible to force the function to obtain orthogonality below ϵ. $ctol$ greater than $1/\epsilon$ is meaningless, where ϵ is the ***machine precision***.

Constraint: if **jobu** = 'C', **work**(1) ≥ 1.0.

5.2 Optional Input Parameters

1: **m** – INTEGER

Default: the first dimension of the array **a**.

m, the number of rows of the matrix A.

Constraint: **m** ≥ 0.

2: **n** – INTEGER

Default: the second dimension of the array **a**.

n, the number of columns of the matrix A.

Constraint: **m** $\geq$ **n** ≥ 0.

5.3 Output Parameters

1: **a**(lda, :) – REAL (KIND=nag_wp) array

The first dimension of the array **a** will be $\max(1, \mathbf{m})$.

The second dimension of the array **a** will be $\max(1, \mathbf{n})$.

The matrix U containing the left singular vectors of A.

If **jobu** = 'U' or 'C'

if **info** = 0

rank(A) orthonormal columns of U are returned in the leading rank(A) columns of the array **a**. Here rank(A) $\leq$ **n** is the number of computed singular values of A that are above the safe range parameter, as returned by nag_machine_real_safe (x02am). The singular vectors corresponding to underflowed or zero singular values are not computed. The value of rank(A) is returned by rounding **work**(2) to the nearest whole number. Also see the descriptions of **sva** and **work**. The computed columns of U are mutually numerically orthogonal up to approximately $tol = \sqrt{m} \times \epsilon$; or $tol = ctol \times \epsilon$ (**jobu** = 'C'), where ϵ is the ***machine precision*** and $ctol$ is supplied on entry in **work**(1), see the description of **jobu**.

If **info** > 0

nag_lapack_dgesvj (f08kj) did not converge in 30 iterations (sweeps). In this case, the computed columns of U may not be orthogonal up to tol. The output U (stored in **a**), Σ (given by the computed singular values in **sva**) and V is still a decomposition of the input matrix A in the sense that the residual $\|A - \alpha \times U \times \Sigma \times V^{\mathrm{T}}\|_2 / \|A\|_2$ is small, where α is the value returned in **work**(1).

If **jobu** = 'N'

if **info** = 0

Note that the left singular vectors are 'for free' in the one-sided Jacobi SVD algorithm. However, if only the singular values are needed, the level of numerical orthogonality of U is not an issue and iterations are stopped when the columns of the iterated matrix are numerically orthogonal up to approximately $m \times \epsilon$. Thus, on exit, **a** contains the columns of U scaled with the corresponding singular values.

If **info** > 0
nag_lapack_dgesvj (f08kj) did not converge in 30 iterations (sweeps).

2: **sva**(**n**) – REAL (KIND=nag_wp) array

The, possibly scaled, singular values of A.

If **info** $= 0$
The singular values of A are $\sigma_i = \alpha$**sva**(i), for $i = 1, 2, \ldots, n$, where α is the scale factor stored in **work**(1). Normally $\alpha = 1$, however, if some of the singular values of A might underflow or overflow, then $\alpha \neq 1$ and the scale factor needs to be applied to obtain the singular values.

If **info** > 0
nag_lapack_dgesvj (f08kj) did not converge in 30 iterations and $\alpha \times$ **sva** may not be accurate.

3: **v**$(ldv, :)$ – REAL (KIND=nag_wp) array

The first dimension, ldv, of the array **v** will be

if **jobv** $=$ 'V', $ldv = \max(1, \mathbf{n})$;
if **jobv** $=$ 'A', $ldv = \max(1, \mathbf{mv})$;
otherwise $ldv = 1$.

The second dimension of the array **v** will be $\max(1, \mathbf{n})$ if **jobv** $=$ 'V' or 'A' and 1 otherwise.

The right singular vectors of A.

If **jobv** $=$ 'V', **v** contains the n by n matrix of the right singular vectors.

If **jobv** $=$ 'A', **v** contains the product of the computed right singular vector matrix and the initial matrix in the array **v**.

If **jobv** $=$ 'N', **v** is not referenced.

4: **work**$(lwork)$ – REAL (KIND=nag_wp) array

$lwork = \max(6, \mathbf{m} + \mathbf{n})$.

Contains information about the completed job.

work(1)
the scaling factor, α, such that $\sigma_i = \alpha$**sva**(i), for $i = 1, 2, \ldots, n$ are the computed singular values of A. (See description of **sva**.)

work(2)
nint(**work**(2))gives the number of the computed nonzero singular values.

work(3)
nint(**work**(3)) gives the number of the computed singular values that are larger than the underflow threshold.

work(4)
nint(**work**(4)) gives the number of iterations (sweeps of Jacobi rotations) needed for numerical convergence.

work(5)
$\max_{i \neq j} |\cos(A(:, i), A(:, j))|$ in the last iteration (sweep). This is useful information in cases when nag_lapack_dgesvj (f08kj) did not converge, as it can be used to estimate whether the output is still useful and for subsequent analysis.

work(6)
The largest absolute value over all sines of the Jacobi rotation angles in the last sweep. It can be useful for subsequent analysis.

5: **info** – INTEGER

info $= 0$ unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

info < 0

If **info** $= -i$, argument i had an illegal value. An explanatory message is output, and execution of the program is terminated.

info > 0 (*warning*)

nag_lapack_dgesvj (f08kj) did not converge in the allowed number of iterations (30), but its output might still be useful.

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.

See Section 6 of Drmac and Veselic (2008a) for a detailed discussion of the accuracy of the computed SVD.

8 Further Comments

This SVD algorithm is numerically superior to the bidiagonalization based QR algorithm implemented by nag_lapack_dgesvd (f08kb) and the divide and conquer algorithm implemented by nag_lapack_dgesdd (f08kd) algorithms and is considerably faster than previous implementations of the (equally accurate) Jacobi SVD method. Moreover, this algorithm can compute the SVD faster than nag_lapack_dgesvd (f08kb) and not much slower than nag_lapack_dgesdd (f08kd). See Section 3.3 of Drmac and Veselic (2008b) for the details.

9 Example

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

$$A = \begin{pmatrix} 2.27 & -1.54 & 1.15 & -1.94 \\ 0.28 & -1.67 & 0.94 & -0.78 \\ -0.48 & -3.09 & 0.99 & -0.21 \\ 1.07 & 1.22 & 0.79 & 0.63 \\ -2.35 & 2.93 & -1.45 & 2.30 \\ 0.62 & -7.39 & 1.03 & -2.57 \end{pmatrix},$$

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

9.1 Program Text

```
function f08kj_example

fprintf('f08kj example results\n\n');

m = nag_int(6);
n = nag_int(4);
a = [2.27, -1.54,  1.15, -1.94;
     0.28, -1.67,  0.94, -0.78;
    -0.48, -3.09,  0.99, -0.21;
     1.07,  1.22,  0.79,  0.63;
```

```
    -2.35,  2.93, -1.45,  2.30;
     0.62, -7.39,  1.03, -2.57];
joba = 'g';
jobu = 'u';
jobv = 'v';
work = zeros(10,1);
v    = zeros(4, 4);
mv = nag_int(0);
% Compute the singular values and left and right singular vectors
% of A (A = U*S*V, m >= n)
[a, sva, v, work, info] = f08kj( ...
                                joba, jobu, jobv, a, mv, v, work);

% Compute the approximate error bound for the computed singular values
% using the 2-norm, s(1) = norm(A), and machine precision, eps.
eps = x02aj;
serrbd = eps*sva(1);

% Print solution
fprintf('Singular values:\n');
disp(transpose(sva));
if (abs(work(1)-1) > eps)
  fprintf('\nValues nned scaling by %13.5e.\n', work(1));
end

[ifail] = x04ca( ...
                'Gen', ' ', a, 'Left singular vectors');
fprintf('\n');
[ifail] = x04ca( ...
                'Gen', ' ', v, 'Right singular vectors');

% Estimate reciprocal condition numbers for the singular vectors
[rcondu, info] = f08fl( ...
                        'Left', m, n, sva);
[rcondv, info] = f08fl( ...
                        'Right', m, n, sva);

% Approximate error bounds for the singular values and vectors
fprintf('\nError estimate for the singular values a\n');
fprintf('%11.1e\n', serrbd);
fprintf('\nError estimates for left singular vectors\n');
fprintf('%11.1e ',serrbd./rcondu);
fprintf('\n\nError estimates for right singular vectors\n');
fprintf('%11.1e ',serrbd./rcondv);
fprintf('\n');
```

9.2 Program Results

```
     f08kj example results

Singular values:
    9.9966    3.6831    1.3569    0.5000

 Left singular vectors
          1       2       3       4
 1  -0.2774  0.6003 -0.1277  0.1323
 2  -0.2020  0.0301  0.2805  0.7034
 3  -0.2918 -0.3348  0.6453  0.1906
 4   0.0938  0.3699  0.6781 -0.5399
 5   0.4213 -0.5266  0.0413 -0.0575
 6  -0.7816 -0.3353 -0.1645 -0.3957

 Right singular vectors
          1       2       3       4
 1  -0.1921  0.8030  0.0041 -0.5642
 2   0.8794  0.3926 -0.0752  0.2587
 3  -0.2140  0.2980  0.7827  0.5027
 4   0.3795 -0.3351  0.6178 -0.6017

Error estimate for the singular values a
```

```
    1.1e-15

Error estimates for left singular vectors
    1.8e-16     4.8e-16     1.3e-15     2.2e-15

Error estimates for right singular vectors
    1.8e-16     4.8e-16     1.3e-15     1.3e-15
```