

# NAG Library Function Document

## nag\_dgesvj (f08kjc)

### 1 Purpose

nag\_dgesvj (f08kjc) 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\_dgesvd (f08kbc) or nag\_dgesdd (f08kdc) may be used.

### 2 Specification

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

void nag_dgesvj (Nag_OrderType order, Nag_MatrixType joba,
                 Nag_LeftVecsType jobu, Nag_RightVecsType jobv, Integer m, Integer n,
                 double a[], Integer pda, double sva[], Integer mv, double v[],
                 Integer pdv, double ctol, double work[], NagError *fail)
```

### 3 Description

The SVD is written as

$$A = U\Sigma V^T,$$

where  $\Sigma$  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  $\Sigma$  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 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:     **joba** – Nag\_MatrixType *Input*  
*On entry:* specifies the structure of matrix *A*.  
**joba** = Nag\_LowerMatrix  
The input matrix *A* is lower triangular.  
**joba** = Nag\_UpperMatrix  
The input matrix *A* is upper triangular.  
**joba** = Nag\_GeneralMatrix  
The input matrix *A* is a general *m* by *n* matrix,  $\mathbf{m} \geq \mathbf{n}$ .  
*Constraint:* **joba** = Nag\_LowerMatrix, Nag\_UpperMatrix or Nag\_GeneralMatrix.
- 3:     **jobu** – Nag\_LeftVecsType *Input*  
*On entry:* specifies whether to compute the left singular vectors and if so whether you want to control their numerical orthogonality threshold.  
**jobu** = Nag\_LeftSpan  
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  $\epsilon$  is the *machine precision* and  $ctol = \sqrt{m}$ .  
**jobu** = Nag\_LeftVecsCtol  
Analogous to **jobu** = Nag\_LeftSpan, 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**[0]. The option **jobu** = Nag\_LeftVecsCtol 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**[0].  
**jobu** = Nag\_NotLeftVecs  
The matrix *U* is not computed. However, see the description of **a**.  
*Constraint:* **jobu** = Nag\_LeftSpan, Nag\_LeftVecsCtol or Nag\_NotLeftVecs.
- 4:     **jobv** – Nag\_RightVecsType *Input*  
*On entry:* specifies whether and how to compute the right singular vectors.  
**jobv** = Nag\_RightVecs  
The matrix *V* is computed and returned in the array **v**.  
**jobv** = Nag\_RightVecsMV  
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** = Nag\_NotRightVecs  
The matrix *V* is not computed and the array **v** is not referenced.  
*Constraint:* **jobv** = Nag\_RightVecs, Nag\_RightVecsMV or Nag\_NotRightVecs.
- 5:     **m** – Integer *Input*  
*On entry:* *m*, the number of rows of the matrix *A*.  
*Constraint:*  $\mathbf{m} \geq 0$ .
- 6:     **n** – Integer *Input*  
*On entry:* *n*, the number of columns of the matrix *A*.  
*Constraint:*  $\mathbf{m} \geq \mathbf{n} \geq 0$ .

7: **a**[*dim*] – double

Input/Output

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

$\max(1, \mathbf{pda} \times \mathbf{n})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{m} \times \mathbf{pda})$  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:* the matrix *U* containing the left singular vectors of *A*.If **jobu** = Nag\_LeftSpan or Nag\_LeftVecsCtolif **fail.errnum** = 0

rank(*A*) orthonormal columns of *U* are returned in the leading rank(*A*) columns of the array **a**. Here rank(*A*) ≤ **n** is the number of computed singular values of *A* that are above the safe range parameter, as returned by nag\_real\_safe\_small\_number (X02AMC). The singular vectors corresponding to underflowed or zero singular values are not computed. The value of rank(*A*) is returned by rounding **work**[1] 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  $\text{tol} = \sqrt{m} \times \epsilon$ ; or  $\text{tol} = \text{ctol} \times \epsilon$  (**jobu** = Nag\_LeftVecsCtol), where  $\epsilon$  is the *machine precision* and *ctol* is supplied on entry in **work**[0], see the description of **jobu**.

If **fail.errnum** > 0

nag\_dgesvj (f08kjc) 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**),  $\Sigma$  (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  $\|A - \alpha \times U \times \Sigma \times V^T\|_2 / \|A\|_2$  is small, where  $\alpha$  is the value returned in **work**[0].

If **jobu** = Nag\_NotLeftVecsif **fail.errnum** = 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 **fail.errnum** > 0

nag\_dgesvj (f08kjc) did not converge in 30 iterations (sweeps).

8: **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** ≥ max(1, **m**);  
 if **order** = Nag\_RowMajor, **pda** ≥ max(1, **n**).

9: **sva**[**n**] – double

Output

*On exit:* the, possibly scaled, singular values of *A*.If **fail.errnum** = 0

The singular values of *A* are  $\sigma_i = \alpha \mathbf{sva}[i - 1]$ , for  $i = 1, 2, \dots, n$ , where  $\alpha$  is the scale factor stored in **work**[0]. 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 **fail.errnum** > 0

nag\_dgesvj (f08kjc) did not converge in 30 iterations and  $\alpha \times \mathbf{sva}$  may not be accurate.

10: **mv** – Integer

*Input*

*On entry:* if **jobv** = Nag\_RightVecsMV, the product of Jacobi rotations is applied to the first  $m_v$  rows of **v**.

If **jobv**  $\neq$  Nag\_RightVecsMV, **mv** is ignored. See the description of **jobv**.

11: **v[dim]** – double

*Input/Output*

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

max(1, **pdv**  $\times$  **n**) when **jobv** = Nag\_RightVecs;  
 max(1, **pdv**  $\times$  **n**) when **jobv** = Nag\_RightVecsMV and **order** = Nag\_ColMajor;  
 max(1, **mv**  $\times$  **pdv**) when **jobv** = Nag\_RightVecsMV and **order** = Nag\_RowMajor;  
 max(1, **mv**) otherwise.

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

**v**[(*j* − 1)  $\times$  **pdv** + *i* − 1] when **order** = Nag\_ColMajor;  
**v**[(*i* − 1)  $\times$  **pdv** + *j* − 1] when **order** = Nag\_RowMajor.

*On entry:* if **jobv** = Nag\_RightVecsMV, **v** must contain an  $m_v$  by *n* matrix to be premultiplied by the matrix *V* of right singular vectors.

*On exit:* the right singular vectors of *A*.

If **jobv** = Nag\_RightVecs, **v** contains the *n* by *n* matrix of the right singular vectors.

If **jobv** = Nag\_RightVecsMV, **v** contains the product of the computed right singular vector matrix and the initial matrix in the array **v**.

If **jobv** = Nag\_NotRightVecs, **v** is not referenced.

12: **pdv** – Integer

*Input*

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

*Constraints:*

if **order** = Nag\_ColMajor,  
     if **jobv** = Nag\_RightVecs, **pdv**  $\geq$  max(1, **n**);  
     if **jobv** = Nag\_RightVecsMV, **pdv**  $\geq$  max(1, **mv**);  
     otherwise **pdv**  $\geq$  1.;  
 if **order** = Nag\_RowMajor,  
     if **jobv** = Nag\_RightVecs, **pdv**  $\geq$  max(1, **n**);  
     if **jobv** = Nag\_RightVecsMV, **pdv**  $\geq$  max(1, **n**);  
     otherwise **pdv**  $\geq$  1..

13: **ctol** – double

*Input*

*On entry:* if **jobu** = Nag\_LeftVecsCtol, **ctol** = *ctol*, 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  $\epsilon$ . *ctol* greater than  $1/\epsilon$  is meaningless, where  $\epsilon$  is the *machine precision*.

*Constraint:* if **jobu** = Nag\_LeftVecsCtol, **ctol**  $\geq$  1.0.

14: **work[6]** – double*Output**On exit:* contains information about the completed job.**work[0]**the scaling factor,  $\alpha$ , such that  $\sigma_i = \alpha \mathbf{sva}[i-1]$ , for  $i = 1, 2, \dots, n$  are the computed singular values of  $A$ . (See description of **sva**.)**work[1]****nint(work[1])** gives the number of the computed nonzero singular values.**work[2]****nint(work[2])** gives the number of the computed singular values that are larger than the underflow threshold.**work[3]****nint(work[3])** gives the number of iterations (sweeps of Jacobi rotations) needed for numerical convergence.**work[4]** $\max_{i \neq j} |\cos(A(:, i), A(:, j))|$  in the last iteration (sweep). This is useful information in cases when **nag\_dgesvj** (f08kjc) did not converge, as it can be used to estimate whether the output is still useful and for subsequent analysis.**work[5]**

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

15: **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  $\langle value \rangle$  had an illegal value.

### NE\_CONVERGENCE

**nag\_dgesvj** (f08kjc) did not converge in the allowed number of iterations (30), but its output might still be useful.

### NE\_ENUM\_INT\_2

On entry, **jobv** =  $\langle value \rangle$ , **pdv** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ .Constraint: if **jobv** = Nag\_RightVecs, **pdv**  $\geq \max(1, \mathbf{n})$ ;if **jobv** = Nag\_RightVecsMV, **pdv**  $\geq \max(1, \mathbf{n})$ ;otherwise **pdv**  $\geq 1$ .

### NE\_ENUM\_INT\_3

On entry, **jobv** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ , **mv** =  $\langle value \rangle$  and **pdv** =  $\langle value \rangle$ .Constraint: if **jobv** = Nag\_RightVecs, **pdv**  $\geq \max(1, \mathbf{n})$ ;if **jobv** = Nag\_RightVecsMV, **pdv**  $\geq \max(1, \mathbf{mv})$ ;otherwise **pdv**  $\geq 1$ .

**NE\_ENUM\_REAL\_1**

On entry, **jobu** =  $\langle value \rangle$  and **ctol** =  $\langle value \rangle$ .  
 Constraint: if **jobu** = Nag\_LeftVecsCtol, **ctol**  $\geq 1.0$ .

**NE\_INT**

On entry, **m** =  $\langle value \rangle$ .  
 Constraint: **m**  $\geq 0$ .  
 On entry, **pda** =  $\langle value \rangle$ .  
 Constraint: **pda**  $> 0$ .  
 On entry, **pdv** =  $\langle value \rangle$ .  
 Constraint: **pdv**  $> 0$ .

**NE\_INT\_2**

On entry, **m** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
 Constraint: **m**  $\geq$  **n**  $\geq 0$ .  
 On entry, **pda** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .  
 Constraint: **pda**  $\geq \max(1, \mathbf{m})$ .  
 On entry, **pda** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
 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  $\epsilon$  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 Parallelism and Performance**

nag\_dgesvj (f08kjc) 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

This SVD algorithm is numerically superior to the bidiagonalization based  $QR$  algorithm implemented by `nag_dgesvd` (f08kbc) and the divide and conquer algorithm implemented by `nag_dgesdd` (f08kdc) 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_dgesvd` (f08kbc) and not much slower than `nag_dgesdd` (f08kdc). See Section 3.3 of Drmac and Veselic (2008b) for the details.

## 10 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.

### 10.1 Program Text

```
/* nag_dgesvj (f08kjc) 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 <nagx02.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double eps, serrbd, ctol;
    Integer exit_status = 0;
    Integer i, j, lwork, m, mv, n, n_vrows, n_vcols, pda, pdv, ranka;

    /* Arrays */
    double *a = 0, *rcondu = 0, *rcondv = 0, *s = 0, *v = 0, *work = 0;
    char nag_enum_arg[40];

    /* Nag Types */
    Nag_OrderType order;
    Nag_MatrixType joba;
    Nag_LeftVecsType jobu;
    Nag_RightVecsType jobv;
    NagError fail;

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

```

INIT_FAIL(fail);

printf("nag_dgesvj (f08kjc) 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 (n < 0 || m < n) {
        printf("Invalid m or n\n");
        exit_status = 1;
        goto END;;
    }

/* Read Nag type arguments by name and convert to value */
#ifdef _WIN32
    scanf_s(" %39s%*[\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
    scanf(" %39s%*[\n]", nag_enum_arg);
#endif
/* nag_enum_name_to_value (x04nac).
 * Converts NAG enum member name to value
 */
joba = (Nag_MatrixType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s(" %39s%*[\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
    scanf(" %39s%*[\n]", nag_enum_arg);
#endif
jobu = (Nag_LeftVecsType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s(" %39s%*[\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
    scanf(" %39s%*[\n]", nag_enum_arg);
#endif
jobv = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s(" %39s%*[\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
    scanf(" %39s%*[\n]", nag_enum_arg);
#endif

    n_vcols = n;
    n_vrows = n;
    mv = 0;
    if (jobv == Nag_RightVecsMV) {
#ifdef _WIN32
        scanf_s("%" NAG_IFMT, &mv);
#else
        scanf("%" NAG_IFMT, &mv);
#endif
        n_vrows = mv;
    }
    else if (jobv == Nag_NotRightVecs) {
        n_vrows = 1;
        n_vcols = 1;
    }
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif

```



```

#ifdef NAG_COLUMN_MAJOR
    pda = m;
    pdv = n_vrows;
#else
    pda = n;
    pdv = n_vcols;
#endif
lwork = 6;

    if (!(a = NAG_ALLOC(m * n, double)) ||
        !(rcondu = NAG_ALLOC(m, double)) ||
        !(rcondv = NAG_ALLOC(m, double)) ||
        !(s = NAG_ALLOC(n, double)) ||
        !(v = NAG_ALLOC(n_vrows * n_vcols, double)) ||
        !(work = NAG_ALLOC(lwork, double))
        )
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read the m by n matrix A from data file */
    if (joba == Nag_GeneralMatrix) {
        for (i = 1; i <= m; i++)
#ifdef _WIN32
            for (j = 1; j <= n; j++)
                scanf_s("%lf", &A(i, j));
#else
            for (j = 1; j <= n; j++)
                scanf("%lf", &A(i, j));
#endif
    }
    else if (joba == Nag_UpperMatrix) {
        for (i = 1; i <= m; i++)
#ifdef _WIN32
            for (j = i; j <= n; j++)
                scanf_s("%lf", &A(i, j));
#else
            for (j = i; j <= n; j++)
                scanf("%lf", &A(i, j));
#endif
    }
    else {
        for (i = 1; i <= m; i++)
#ifdef _WIN32
            for (j = 1; j <= i; j++)
                scanf_s("%lf", &A(i, j));
#else
            for (j = 1; j <= i; j++)
                scanf("%lf", &A(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\\n]");
#else
    scanf("%*[^\\n]");
#endif
    /* jobv==Nag_RightVecsMV means that the first mv rows of v must be set. */
    if (jobv == Nag_RightVecsMV) {
        for (i = 1; i <= mv; i++)
#ifdef _WIN32
            for (j = 1; j <= n; j++)
                scanf_s("%lf", &V(i, j));
#else
            for (j = 1; j <= n; j++)
                scanf("%lf", &V(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\\n]");
#else
    scanf("%*[^\\n]");
#endif

```

```

        scanf("%*[^\\n]");
#endif
    }
    ctol = 10.0;
    /* nag_dgesvj (f08kjc)
     * Compute the singular values and left and right singular vectors
     * of A (A = U*S*V, m>=n).
     */
    nag_dgesvj(order, joba, jobu, jobv, m, n, a, pda, s, mv, v, pdv, ctol,
               work, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dgesvj (f08kjc).\\n%s\\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Get the machine precision, eps and compute the approximate
     * error bound for the computed singular values. Note that for
     * the 2-norm, s[0] = norm(A).
     */
    eps = nag_machine_precision;
    serrbd = eps * s[0];

    /* Print solution */
    printf("Singular values\\n  ");
    for (j = 0; j < n; j++)
        printf("%8.4f", s[j]);
    printf("\\n\\n");
    if (fabs(work[0] - 1.0) > eps)
        printf("Values need scaling by factor = %13.5e\\n\\n", work[0]);

    ranka = (Integer) work[1];
    printf("Rank of A = %5" NAG_IFMT "\\n\\n", ranka);
    if (jobu != Nag_NotLeftVecs) {
        /* nag_gen_real_mat_print (x04cac)
         * Print real general matrix (easy-to-use)
         */
        fflush(stdout);
        nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m,
                               ranka, a, pda, "Left spanning singular vectors", 0,
                               &fail);
        if (fail.code != NE_NOERROR) {
            printf("Error from nag_gen_real_mat_print (x04cac).\\n%s\\n",
                   fail.message);
            exit_status = 1;
            goto END;
        }
    }

    if (jobv == Nag_RightVecs) {
        printf("\\n");
        fflush(stdout);
        nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, v,
                               pdv, "Right singular vectors", 0, &fail);
    }
    else if (jobv == Nag_RightVecsMV) {
        printf("\\n");
        fflush(stdout);
        nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, mv, n,
                               v, pdv,
                               "Right singular vectors applied to input V", 0,
                               &fail);
    }
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_gen_real_mat_print (x04cac).\\n%s\\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* nag_ddisna (f08flc)
     * 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);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_ddisna (f08flc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print the approximate error bounds for the singular values and vectors. */
printf("\nError estimate for the singular values\n");
printf("%11.1e", serrbd);

printf("\n\nError estimates for left singular vectors\n");
for (i = 0; i < n; i++)
    printf("%11.1e", serrbd / rcondu[i]);

printf("\n\nError estimates for right singular vectors\n");
for (i = 0; i < n; i++)
    printf("%11.1e", serrbd / rcondv[i]);
printf("\n");

END:
NAG_FREE(a);
NAG_FREE(rcondu);
NAG_FREE(rcondv);
NAG_FREE(s);
NAG_FREE(v);
NAG_FREE(work);

return exit_status;
}

```

## 10.2 Program Data

nag\_dgesvj (f08kjc) Example Program Data

```

      6      4              : m and n

Nag_GeneralMatrix      : joba
Nag_LeftSpan           : jobu
Nag_RightVecs          : jobv
                      : mv if jobv==Nag_RightVecsMV

  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 : matrix a
                      : mv by n matrix v if jobv==Nag_RightVecsMV

```

## 10.3 Program Results

nag\_dgesvj (f08kjc) Example Program Results

Singular values  
 9.9966 3.6831 1.3569 0.5000

Rank of A = 4

Left spanning 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  
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

---