NAG Library Function Document

nag_dgesvj (f08kjc)

1 Purpose

nag_dgesvj (f08kjc) computes the one-sided Jacobi singular value decomposition (SVD) of a real \( m \times n \) 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

```c
#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 work[], Integer lwork, NagError *fail)
```

3 Description

The SVD is written as

\[ A = U \Sigma V^T, \]

where \( \Sigma \) is an \( n \times n \) diagonal matrix, \( U \) is an \( m \times n \) orthonormal matrix, and \( V \) is an \( n \times n \) 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


5 Arguments

1: order – Nag_OrderType

\hspace{1cm} 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 3.2.1.3 in the Essential Introduction 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, $m \geq n$.
    *Constraint:* $\text{joba} = \text{Nag\_LowerMatrix, Nag\_UpperMatrix or Nag\_GeneralMatrix}$.  

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 $\text{tol} = \text{ctol} \times \epsilon$, where $\epsilon$ is the [machine precision](https://www.nag.co.uk/numeric/FLS/f08fadc.html) and $\text{ctol} = \sqrt{m}$.  
    - **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 $\text{tol} = \text{ctol} \times \epsilon$, where $\text{ctol}$ is given on input in $\text{work}[0]$. The option **jobu** = Nag_LeftVecsCtol can be used if $m \times \epsilon$ is a satisfactory orthogonality of the computed left singular vectors, so $\text{ctol} = m$ could save a few sweeps of Jacobi rotations. See the descriptions of $a$ and $\text{work}[0]$.  
    - **jobu** = Nag_NotLeftVecs  
      The matrix $U$ is not computed. However, see the description of $a$.  
    *Constraint:* $\text{jobu} = \text{Nag\_LeftSpan, Nag\_LeftVecsCtol or Nag\_NotLeftVecs}$.  

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 $m v$ rows of $v$.  
    - **jobv** = Nag_NotRightVecs  
      The matrix $V$ is not computed and the array $v$ is not referenced.  
    *Constraint:* $\text{jobv} = \text{Nag\_RightVecs, Nag\_RightVecsMV or Nag\_NotRightVecs}$.  

5:  **m** – Integer  
    *Input*  
    *On entry:* $m$, the number of rows of the matrix $A$.  
    *Constraint:* $m \geq 0$.  

6:  **n** – Integer  
    *Input*  
    *On entry:* $n$, the number of columns of the matrix $A$.  
    *Constraint:* $m \geq n \geq 0$.  

---

This is a documentation excerpt for the NAG Library function f08kjc, which performs a singular value decomposition (SVD) of a real general matrix. The function allows for the computation of the left and right singular vectors, and the structure of the input matrix. The input parameters include the matrix type, whether to compute the singular vectors, the number of rows and columns of the matrix, and additional vectors for orthogonality control.
The singular values of $A$ are $\sigma_i = \alpha \text{s}va[i - 1]$, for $i = 1, 2, \ldots, n$, where $\alpha$ is the scale factor stored in $\text{work}[0]$. 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 α × 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 mv rows of v.
    If jobv ≠ 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 × n) when jobv = Nag_RightVecs;
        max(1, pdv × n) when jobv = Nag_RightVecsMV and order = Nag_ColMajor;
        max(1, mv × 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) × pdv + i - 1] when order = Nag_ColMajor;
        v[(i - 1) × pdv + j - 1] when order = Nag_RowMajor.
    On entry: if jobv = Nag_RightVecsMV, v must contain an mv 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 ≥ max(1, n);
            if jobv = Nag_RightVecsMV, pdv ≥ max(1, mv);
            otherwise pdv ≥ 1;
        if order = Nag_RowMajor,
            if jobv = Nag_RightVecs, pdv ≥ max(1, n);
            if jobv = Nag_RightVecsMV, pdv ≥ max(1, n);
            otherwise pdv ≥ 1;

13: work[lwork] – double
    Communication Array
    On entry: if jobu = Nag_LeftVecsCtol, work[0] = ctol, where ctol defines the threshold for convergence. The process stops if all columns of A are mutually orthogonal up to ctol × ε. It is required that ctol ≥ 1, i.e., it is not possible to force the function to obtain orthogonality below ε. ctol greater than 1/ε is meaningless, where ε is the machine precision.
    On exit: contains information about the completed job.
    work[0]
        the scaling factor, α, such that σ_i = α * sva[i - 1], for i = 1, 2, ..., n are the computed singular values of A. (See description of sva.)
work[1] gives the number of the computed nonzero singular values.

work[2] gives the number of the computed singular values that are larger than the underflow threshold.

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.

Constraint: if \(\text{jobu} = \text{Nag LeftVecsCtol}\), \(\text{work}[0] \geq 1.0\).

14: lwork – Integer

Input

On entry: the dimension of the array work.

Constraint: \(\text{lwork} \geq \max(6, m + n)\).

15: fail – NagError *

Input/Output

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument \(\text{value}\) 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 = \(\text{value}\), pdv = \(\text{value}\), n = \(\text{value}\).

Constraint: if jobv = Nag_RightVecs, pdv \(\geq \max(1, n)\);
if jobv = Nag_RightVecsMV, pdv \(\geq \max(1, n)\);
o otherwise pdv \(\geq 1\).

NE_ENUM_INT_3

On entry, jobv = \(\text{value}\), n = \(\text{value}\), mv = \(\text{value}\) and pdv = \(\text{value}\).

Constraint: if jobv = Nag_RightVecs, pdv \(\geq \max(1, n)\);
if jobv = Nag_RightVecsMV, pdv \(\geq \max(1, mv)\);
otherwise pdv \(\geq 1\).
NE_ENUM_REAL_1
On entry, jobu = \langle value \rangle and work[0] = \langle value \rangle.
Constraint: if jobu = Nag_LeftVecsCtol, work[0] ≥ 1.0.

NE_INT
On entry, m = \langle value \rangle.
Constraint: m ≥ 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 ≥ n ≥ 0.
On entry, pda = \langle value \rangle and m = \langle value \rangle.
Constraint: pda ≥ max(1, m).
On entry, pda = \langle value \rangle and n = \langle value \rangle.
Constraint: pda ≥ max(1, n).

NE_INT_3
On entry, lwork = \langle value \rangle, m = \langle value \rangle and n = \langle value \rangle.
Constraint: lwork ≥ max(6, m + 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 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction 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) is not threaded by NAG in any implementation.

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. */
* * Copyright 2014 Numerical Algorithms Group. *
* * Mark 23, 2011. */

#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;
    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) a[(I-1)*pdv + J-1]
    order = Nag_RowMajor;
    #endif

    //...
#define V(I, J) v[(I-1)*pdv + J-1]
order = Nag_RowMajor;
#endif
INIT_FAIL(fail);

printf("nag_dgesvj (f08kjc) Example Program Results\n\n");

/* Skip heading in data file*/
#ifdef _WIN32
scanf_s("%*[\n\n]");
#else
scanf("%*[\n]\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]\n", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf(" %39s%*[\n]\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]\n", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf(" %39s%*[\n]\n", nag_enum_arg);
#endif
jobu = (Nag_LeftVecsType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
scanf_s(" %39s%*[\n]\n", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf(" %39s%*[\n]\n", nag_enum_arg);
#endif
jobv = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
scanf_s(" %39s%*[\n]\n", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf(" %39s%*[\n]\n", nag_enum_arg);
#endif

n_vcols = n;
n_vrows = n;
mv = 0;
if (jobv==Nag_RightVecsMV) {
#ifdef _WIN32
scanf("%"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]\n");
#else
scanf("%*[\n]\n");
#endif
scanf("%*[\n]");
#endif

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

lwork = n + m;
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++)
#endif_WIN32
        for (j = 1; j <= n; j++) scanf("%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++)
#endif_WIN32
            for (j = i; j <= n; j++) scanf("%lf", &A(i, j));
#else
            for (j = i; j <= n; j++) scanf("%lf", &A(i, j));
#endif
      } else {
        for (i = 1; i <= m; i++)
#endif_WIN32
            for (j = 1; j <= i; j++) scanf("%lf", &A(i, j));
#else
            for (j = 1; j <= i; j++) scanf("%lf", &A(i, j));
#endif
      }
#endif_WIN32
scanf("%*[\n]");
#else
scanf("%*[\n]");
#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++)
#endif_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
      } else {
        for (i = 1; i <= mv; i++)
#endif_WIN32
            for (j = 1; j <= i; j++) scanf_s("%lf", &V(i, j));
#else
            for (j = 1; j <= i; j++) scanf("%lf", &V(i, j));
#endif
      }
#endif_WIN32
scanf_s("%*[\n]");
#else
scanf("%*[\n]");
#endif

/* nag_dgesvj (f08kjc)
 * Compute the singular values and left and right singular vectors
 * of A (A = U*S*V, m>=n).
 *="/}
The code snippet provided includes the following functionalities:

1. **Error Handling**: It checks for errors from the function `nag_dgesvj` and prints an error message if an error is encountered.

2. **Machine Precision**: It computes the machine precision and uses it to estimate the error bound for the computed singular values.

3. **Solution Printing**: It prints the singular values and checks if the solution needs scaling.

4. **Matrix Printing**: It prints the left and right singular vectors and computes the estimated condition numbers for the singular vectors.

5. **Error Estimation**: It prints the approximate error bounds for the singular values and vectors.

6. **Utility Functions**: It includes calls to `nag_gen_real_mat_print` for printing real general matrices and `nag_ddisna` for estimating reciprocal condition numbers for the singular vectors.

The code is structured to handle errors, print relevant information, and utilize utility functions from the NAG Library to perform the necessary computations.
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