NAG Library Function Document

nag_dgesvd (f08kbc)

1 Purpose

nag_dgesvd (f08kbc) computes the singular value decomposition (SVD) of a real \( m \times n \) matrix \( A \), optionally computing the left and/or right singular vectors.

2 Specification

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

void nag_dgesvd (Nag_OrderType order, Nag_ComputeUType jobu,
                Nag_ComputeVTType jobvt, Integer m, Integer n,
                double a[], Integer pda, double s[], Integer pdu,
                double vt[], Integer pdvt, double work[], NagError *fail)
```

3 Description

The SVD is written as

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

where \( \Sigma \) is an \( m \times n \) matrix which is zero except for its \( \min(m, n) \) diagonal elements, \( U \) is an \( m \times m \) orthogonal matrix, and \( V \) is an \( n \times n \) orthogonal matrix. The diagonal elements of \( \Sigma \) are the singular values of \( A \); they are real and non-negative, and are returned in descending order. The first \( \min(m, n) \) columns of \( U \) and \( V \) are the left and right singular vectors of \( A \).

Note that the function returns \( V^T \), not \( V \).

4 References


5 Arguments

1: order – Nag_OrderType

   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: jobu – Nag_ComputeUType

   On entry: specifies options for computing all or part of the matrix \( U \).

   jobu = Nag_AllU

   All \( m \) columns of \( U \) are returned in array u.
jobu = Nag_SingularVecsU
    The first min(m, n) columns of U (the left singular vectors) are returned in the array u.
jobu = Nag_Overwrite
    The first min(m, n) columns of U (the left singular vectors) are overwritten on the array a.
jobu = Nag_NotU
    No columns of U (no left singular vectors) are computed.
Constraint: jobu = Nag_AllU, Nag_SingularVecsU, Nag_Overwrite or Nag_NotU.

3: jobvt – Nag_ComputeVTType
   Input
   On entry: specifies options for computing all or part of the matrix VT.
jobvt = Nag_AllVT
    All n rows of VT are returned in the array vt.
jobvt = Nag_SingularVecsVT
    The first min(m, n) rows of VT (the right singular vectors) are returned in the array vt.
jobvt = Nag_OverwriteVT
    The first min(m, n) rows of VT (the right singular vectors) are overwritten on the array a.
jobvt = Nag_NotVT
    No rows of VT (no right singular vectors) are computed.
Constraints:
    jobvt = Nag_AllVT, Nag_SingularVecsVT, Nag_OverwriteVT or Nag_NotVT;
    If jobu = Nag_Overwrite, jobvt cannot be Nag_OverwriteVT.

4: m – Integer
   Input
   On entry: m, the number of rows of the matrix A.
Constraint: m ≥ 0.

5: n – Integer
   Input
   On entry: n, the number of columns of the matrix A.
Constraint: n ≥ 0.

6: a[dim] – double
   Input/Output
   Note: the dimension, dim, of the array a must be at least
   max(1, pda × n) when order = Nag_ColMajor;
   max(1, m × 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: if jobu = Nag_Overwrite, a is overwritten with the first min(m, n) columns of U (the left singular vectors, stored column-wise).
If jobvt = Nag_OverwriteVT, a is overwritten with the first min(m, n) rows of VT (the right singular vectors, stored row-wise).
If jobu ≠ Nag_Overwrite and jobvt ≠ Nag_OverwriteVT, the contents of a are destroyed.
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).

8: s[dim] – double

Note: the dimension, dim, of the array s must be at least max(1, min(m, n)).

On exit: the singular values of A, sorted so that s[i − 1] ≥ s[i].

9: u[dim] – double

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

\[ \min(1, \text{max}(m, n)) \] when jobu = Nag_AllU;
\[ \text{min}(1, \text{max}(m, n)) \] when jobu = Nag_SingularVecsU and order = Nag_ColMajor;
\[ \text{max}(1, m \times pdu) \] when jobu = Nag_SingularVecsU and order = Nag_RowMajor;
\[ \text{max}(1, m) \] otherwise.

The \( (i, j) \)th element of the matrix \( U \) is stored in

\[ u[(j - 1) \times pdv + i - 1] \] when order = Nag_ColMajor;
\[ u[(i - 1) \times pdv + j - 1] \] when order = Nag_RowMajor.

On exit: if jobu = Nag_AllU, \( U \) contains the \( m \) by \( m \) orthogonal matrix \( U \).

If jobu = Nag_SingularVecsU, \( U \) contains the first \( \min(m, n) \) columns of \( U \) (the left singular vectors, stored column-wise).

If jobu = Nag_NotU or Nag_Overwrite, \( U \) is not referenced.

10: pdu – Integer

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

Constraints:

if order = Nag_ColMajor,

if jobu = Nag_AllU, pdu ≥ max(1, m);
if jobu = Nag_SingularVecsU, pdu ≥ max(1, m);
otherwise pdu ≥ 1.;

if order = Nag_RowMajor,

if jobu = Nag_AllU, pdu ≥ max(1, m);
if jobu = Nag_SingularVecsU, pdu ≥ max(1, \( \min(m, n) \));
otherwise pdu ≥ 1.;

11: vt[dim] – double

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

\[ \min(1, \text{max}(m, n)) \] when jobvt = Nag_AllVT;
\[ \text{max}(1, \text{min}(m, n)) \] when jobvt = Nag_SingularVecsVT and order = Nag_ColMajor;
\[ \text{max}(1, \text{min}(m, n)) \times pdvt \] when jobvt = Nag_SingularVecsVT and order = Nag_RowMajor;
\[ \text{max}(1, \text{min}(m, n)) \] otherwise.

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

\[ vt[(j - 1) \times pdvt + i - 1] \] when order = Nag_ColMajor;
\[ vt[(i - 1) \times pdvt + j - 1] \] when order = Nag_RowMajor.
On exit: if \( \text{jobvt} = \text{Nag\_AllVT} \), \( \text{vt} \) contains the \( n \) by \( n \) orthogonal matrix \( V^T \).

If \( \text{jobvt} = \text{Nag\_SingularVecsVT} \), \( \text{vt} \) contains the first \( \min(m, n) \) rows of \( V^T \) (the right singular vectors, stored row-wise).

If \( \text{jobvt} = \text{Nag\_NotVT} \) or \( \text{Nag\_OverwriteVT} \), \( \text{vt} \) is not referenced.

12: \( \text{pdvt} \) – Integer

\text{Input}

On entry: the stride separating row or column elements (depending on the value of \text{order}) in the array \( \text{vt} \).

Constraints:

if \( \text{order} = \text{Nag\_ColMajor} \),

if \( \text{jobvt} = \text{Nag\_AllVT} \), \( \text{pdvt} \geq \max(1, n) \);

if \( \text{jobvt} = \text{Nag\_SingularVecsVT} \), \( \text{pdvt} \geq \max(1, \min(m, n)) \);

otherwise \( \text{pdvt} \geq 1 \).

if \( \text{order} = \text{Nag\_RowMajor} \),

if \( \text{jobvt} = \text{Nag\_AllVT} \), \( \text{pdvt} \geq \max(1, n) \);

if \( \text{jobvt} = \text{Nag\_SingularVecsVT} \), \( \text{pdvt} \geq \max(1, n) \);

otherwise \( \text{pdvt} \geq 1 \).

13: \( \text{work}[\min(m, n)] \) – double

\text{Output}

On exit: if \( \text{fail\_code} = \text{NE\_CONVERGENCE} \), \( \text{WORK}(2: \min(m, n)) \) (using the notation described in Section 3.2.1.4 in the Essential Introduction) contains the unconverged superdiagonal elements of an upper bidiagonal matrix \( B \) whose diagonal is in \( s \) (not necessarily sorted). \( B \) satisfies \( A = UBV^T \), so it has the same singular values as \( A \), and singular vectors related by \( U \) and \( V^T \).

14: \( \text{fail} \) – NagError*

\text{Input/Output}

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

6 Error Indicators and Warnings

\text{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\text{NE\_BAD\_PARAM}

On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

\text{NE\_CONVERGENCE}

If \text{nag\_dgesvd} (f08kbc) did not converge, \( \text{fail\_errno} \) specifies how many superdiagonals of an intermediate bidiagonal form did not converge to zero.

\text{NE\_ENUM\_INT\_2}

On entry, \( \langle \text{value} \rangle, \text{pdu} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).

Constraint: if \( \text{jobu} = \text{Nag\_AllU} \), \( \text{pdu} \geq \max(1, m) \);

if \( \text{jobu} = \text{Nag\_SingularVecsU} \), \( \text{pdu} \geq \max(1, m) \);

otherwise \( \text{pdu} \geq 1 \).

On entry, \( \langle \text{value} \rangle, \text{pdvt} = \langle \text{value} \rangle, \text{n} = \langle \text{value} \rangle \).

Constraint: if \( \text{jobvt} = \text{Nag\_AllVT} \), \( \text{pdvt} \geq \max(1, n) \);

if \( \text{jobvt} = \text{Nag\_SingularVecsVT} \), \( \text{pdvt} \geq \max(1, n) \);

otherwise \( \text{pdvt} \geq 1 \).
NE_ENUM_INT_3

On entry, \( \text{jobu} = \langle \text{value} \rangle, \ p\text{du} = \langle \text{value} \rangle, \ m = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

Constraint: if \( \text{jobu} = \text{Nag}_\text{AllU}, \ p\text{du} \geq \max(1, m) \);
if \( \text{jobu} = \text{Nag}_\text{SingularVecsU}, \ p\text{du} \geq \max(1, \min(m, n)) \);
otherwise \( p\text{du} \geq 1 \).

On entry, \( \text{jobvt} = \langle \text{value} \rangle, \ p\text{dvt} = \langle \text{value} \rangle, \ m = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

Constraint: if \( \text{jobvt} = \text{Nag}_\text{AllVT}, \ p\text{dvt} \geq \max(1, n) \);
if \( \text{jobvt} = \text{Nag}_\text{SingularVecsVT}, \ p\text{dvt} \geq \max(1, \min(m, n)) \);
otherwise \( p\text{dvt} \geq 1 \).

NE_INT

On entry, \( m = \langle \text{value} \rangle \).

Constraint: \( m \geq 0 \).

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 0 \).

On entry, \( p\text{da} = \langle \text{value} \rangle \).

Constraint: \( p\text{da} > 0 \).

On entry, \( p\text{du} = \langle \text{value} \rangle \).

Constraint: \( p\text{du} > 0 \).

On entry, \( p\text{dvt} = \langle \text{value} \rangle \).

Constraint: \( p\text{dvt} > 0 \).

NE_INT_2

On entry, \( p\text{da} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).

Constraint: \( p\text{da} \geq \max(1, m) \).

On entry, \( p\text{da} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

Constraint: \( p\text{da} \geq \max(1, n) \).

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 \textit{machine precision}. In addition, the computed singular vectors are nearly orthogonal to working precision. See Section 4.9 of Anderson \textit{et al.} (1999) for further details.

8 Parallelism and Performance

\texttt{nag_dgesvd} (f08kbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.
nag_dgesvd (f08kbc) 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

The total number of floating-point operations is approximately proportional to \(mn^2\) when \(m > n\) and \(m^2n\) otherwise.

The singular values are returned in descending order.

The complex analogue of this function is nag_zgesvd (f08kpc).

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.

The example program for nag_dgesdd (f08kdc) illustrates finding a singular value decomposition for the case \(m \leq n\).

10.1 Program Text

/* nag_dgesvd (f08kbc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* * Mark 23, 2011. */
*
#include <math.h>
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double alpha, beta, eps, norm, serrbd;
    Integer exit_status = 0, i, j, m, n, pda, pdd, pdu, pdvt;

    /* Arrays */
    double *a = 0, *d = 0, *rcondu = 0, *rcondv = 0;
    double *s = 0, *u = 0, *uerrbd = 0, *vrrbd = 0, *vt = 0, *work = 0;

    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
ifdef NAG_COLUMN_MAJOR
#define A(I,J) a[(J - 1) * pda + I - 1]
#define U(I,J) u[(J - 1) * pdu + I - 1]
order = Nag_ColMajor;
#else
#define A(I,J) a[(I - 1) * pda + J - 1]
#define U(I,J) u[(I - 1) * pdu + J - 1]
order = Nag_RowMajor;
#endif
INIT_FAIL(fail);

printf("nag_dgesvd (f08kbc) 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 (m<0 || n<0)
{
    printf("Invalid m or n\n");
    exit_status = 1;
    goto END;
}
/* Allocate memory */
if (!(!a = NAG_ALLOC(m * n, double)) ||
    !(!d = NAG_ALLOC(m * n, double)) ||
    !(!rcondu = NAG_ALLOC(n, double)) ||
    !(!rcondv = NAG_ALLOC(n, double)) ||
    !(!s = NAG_ALLOC(MIN(m, n), double)) ||
    !(!u = NAG_ALLOC(m * m, double)) ||
    !(!uerrbd = NAG_ALLOC(n, double)) ||
    !(!verrbd = NAG_ALLOC(n, double)) ||
    !(!vt = NAG_ALLOC(n * n, double)) ||
    !(!work = NAG_ALLOC(MIN(m, n), double)) )
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
pdu = m;
pdvt = n;
#ifdef NAG_COLUMN_MAJOR
pda = m;
pdd = m;
#else
pda = n;
pdd = n;
#endif
/* Read the m by n matrix A from data file */
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
#ifdef _WIN32
scanf_s("%*[\n");
#else
scanf("%*[\n");
#endif

Mark 25
/* Copy a into d */
for(i = 0; i < m*n; i++) d[i] = a[i];

/* nag_gen_real_mat_print (x04cac) */
* Print real general matrix A.
*
fclose(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, n, a,
pda, "Matrix A", 0, &fail);
printf("\n");
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_gen_real_mat_print (x04cac).\n\n", fail.message);
  exit_status = 1;
  goto END;
}

/* nag_dgesvd (f08kbc). */
* Compute the singular values and left and right singular vectors
* of A (A = U*S*(V**T), m.ge.n)
*
fail.code = nag_dgesvd(order, Nag_AllU, Nag_AllVT, m, n, a, pda, s, u, pdu, vt, pdvt,
  work, &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_dgesvd (f08kbc).\n\n", fail.message);
  exit_status = 1;
  goto END;
}

/* U <- U*S */
for(i = 1; i <= m; i++)
  for(j = 1; j <= n; j++) U(i, j) *= s[j-1];

/* nag_dgemm (f06yac): */
* Compute D = D - U*S*V^T from the factorization of A
* and store in d */
alpha = -1.0;
beta = 1.0;
nag_dgemm(order, Nag_NoTrans, Nag_NoTrans, m, n, n, alpha, u, pdu,
  vt, pdvt, beta, d, pdd, &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_dgemm (f06yac).\n\n", fail.message);
  exit_status = 1;
  goto END;
}

/* nag_dge_norm (f16rac) */
* Find norm of matrix D and print warning if it is too large.
*
fail.code = nag_dge_norm(order, Nag_NoTrans, m, n, d, pdd, &norm, &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_dge_norm (f16rac).\n\n", fail.message);
  exit_status = 1;
  goto END;
}

/* nag_machine_precision (x02ajc): the machine precision. */
eps = nag_machine_precision;
if (norm > pow(eps, 0.8))
{
  printf("\nNorm of A-(U*S*V^T) is much greater than 0.\n";
  "Schur factorization has failed.\n");
  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).

serrbd = eps * s[0];

/* Estimate reciprocal condition numbers for the singular vectors using
 * nag_ddisna (f08flc).
 */

nag_ddisna(Nag_LeftSingVecs, m, n, s, rcondu, &fail);
nag_ddisna(Nag_RightSingVecs, m, n, s, rcondv, &fail);

/* Compute the error estimates for the singular vectors */
for (i = 0; i < n; ++i)
{
    uerrbd[i] = serrbd / rcondu[i];
    verrbd[i] = serrbd / rcondv[i];
}

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

/* Error estimates for the left singular vectors
*/
for (i = 0; i < n; ++i) printf(" %10.1e%s", uerrbd[i], i%6 == 5?"\n":"");

/* Error estimates for the right singular vectors
*/
for (i = 0; i < n; ++i) printf(" %10.1e%s", verrbd[i], i%6 == 5?"\n":"");
printf("\n");

END:
NAG_FREE(a);
NAG_FREE(d);
NAG_FREE(rcondu);
NAG_FREE(rcondv);
NAG_FREE(s);
NAG_FREE(u);
NAG_FREE(uerrbd);
NAG_FREE(verrbd);
NAG_FREE(vt);
NAG_FREE(work);

return exit_status;

#undef A
#undef U

10.2 Program Data

nag_dgesvd (f08kbc) Example Program Data

<table>
<thead>
<tr>
<th>6</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.27</td>
<td>-1.54</td>
</tr>
<tr>
<td>0.28</td>
<td>-1.67</td>
</tr>
<tr>
<td>-0.48</td>
<td>-3.09</td>
</tr>
<tr>
<td>1.07</td>
<td>1.22</td>
</tr>
<tr>
<td>-2.35</td>
<td>2.93</td>
</tr>
<tr>
<td>0.62</td>
<td>-7.39</td>
</tr>
</tbody>
</table>

: m and n

10.3 Program Results

nag_dgesvd (f08kbc) Example Program Results

Matrix A

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2700</td>
<td>-1.5400</td>
<td>1.1500</td>
<td>-1.9400</td>
</tr>
<tr>
<td>0.2800</td>
<td>-1.6700</td>
<td>0.9400</td>
<td>-0.7800</td>
</tr>
<tr>
<td>-0.4800</td>
<td>-3.0900</td>
<td>0.9900</td>
<td>-0.2100</td>
</tr>
<tr>
<td>1.0700</td>
<td>1.2200</td>
<td>0.7900</td>
<td>0.6300</td>
</tr>
<tr>
<td>-2.3500</td>
<td>2.9300</td>
<td>-1.4500</td>
<td>2.3000</td>
</tr>
<tr>
<td>0.6200</td>
<td>-7.3900</td>
<td>1.0300</td>
<td>-2.5700</td>
</tr>
</tbody>
</table>
Error estimate for the singular values
1.1e-15

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

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