NAG Library Function Document

nag_dggsvd (f08vac)

1 Purpose

nag_dggsvd (f08vac) computes the generalized singular value decomposition (GSVD) of an m by n real matrix A and a p by n real matrix B.

2 Specification

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

```
void nag_dggsvd (Nag_OrderType order, Nag_ComputeUType jobu,
     Nag_ComputeVType jobv, Nag_ComputeQType jobq, Integer m, Integer n,
     Integer p, Integer *k, Integer *l, double a[], Integer pda, double b[],
     Integer pdb, double alpha[], double beta[], double u[], Integer pdu,
     double v[], Integer pdv, double q[], Integer pdq, Integer iwork[],
     NagError *fail)
```

3 Description

The generalized singular value decomposition is given by

$$U^{\mathrm{T}}AQ = D_1\begin{pmatrix} 0 & R \end{pmatrix}, \quad V^{\mathrm{T}}BQ = D_2\begin{pmatrix} 0 & R \end{pmatrix},$$

where U, V and Q are orthogonal matrices. Let $(k+l)$ be the effective numerical rank of the matrix $\begin{pmatrix} A \\ B \end{pmatrix}$, then R is a $(k+l)$ by $(k+l)$ nonsingular upper triangular matrix, D_1 and D_2 are m by $(k+l)$ and p by $(k+l)$ 'diagonal' matrices structured as follows:

if $m-k-l \geq 0$,

$$D_1 = \begin{array}{r} \\ k \\ l \\ m-k-l \end{array}\begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} I & 0 \\ 0 & C \\ 0 & 0 \end{pmatrix} \end{array}$$

$$D_2 = \begin{array}{r} \\ l \\ p-l \end{array}\begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} 0 & S \\ 0 & 0 \end{pmatrix} \end{array}$$

$$\begin{pmatrix} 0 & R \end{pmatrix} = \begin{array}{r} \\ k \\ l \end{array}\begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & R_{11} & R_{12} \\ 0 & 0 & R_{22} \end{pmatrix} \end{array}$$

where

$$C = \mathrm{diag}(\alpha_{k+1}, \dots, \alpha_{k+l}),$$

$$S = \mathrm{diag}(\beta_{k+1}, \dots, \beta_{k+l}),$$

and

$$C^2 + S^2 = I.$$

R is stored as a submatrix of A with elements R_{ij} stored as $A_{i,n-k-l+j}$ on exit.

If $m - k - l < 0$,

$$D_1 = \begin{array}{r} \\ k \\ m-k \end{array} \begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} I & 0 & 0 \\ 0 & C & 0 \end{pmatrix} \end{array}$$

$$D_2 = \begin{array}{r} \\ m-k \\ k+l-m \\ p-l \end{array} \begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & S & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$\begin{pmatrix} 0 & R \end{pmatrix} = \begin{array}{r} \\ k \\ m-k \\ k+l-m \end{array} \begin{array}{c} \begin{array}{cccc} n-k-l & k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & R_{11} & R_{12} & R_{13} \\ 0 & 0 & R_{22} & R_{23} \\ 0 & 0 & 0 & R_{33} \end{pmatrix} \end{array}$$

where

$$C = \text{diag}(\alpha_{k+1}, \dots, \alpha_m),$$

$$S = \text{diag}(\beta_{k+1}, \dots, \beta_m),$$

and

$$C^2 + S^2 = I.$$

$\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \end{pmatrix}$ is stored as a submatrix of A with R_{ij} stored as $A_{i,n-k-l+j}$, and R_{33} is stored as a submatrix of B with $(R_{33})_{ij}$ stored as $B_{m-k+i,n+m-k-l+j}$.

The function computes C, S, R and, optionally, the orthogonal transformation matrices U, V and Q.

In particular, if B is an n by n nonsingular matrix, then the GSVD of A and B implicitly gives the SVD of AB^{-1}:

$$AB^{-1} = U\left(D_1 D_2^{-1}\right)V^{\mathrm{T}}.$$

If $\begin{pmatrix} A \\ B \end{pmatrix}$ has orthonormal columns, then the GSVD of A and B is also equal to the CS decomposition of A and B. Furthermore, the GSVD can be used to derive the solution of the eigenvalue problem:

$$A^{\mathrm{T}}Ax = \lambda B^{\mathrm{T}}Bx.$$

In some literature, the GSVD of A and B is presented in the form

$$U^{\mathrm{T}}AX = \begin{pmatrix} 0 & D_1 \end{pmatrix}, \quad V^{\mathrm{T}}BX = \begin{pmatrix} 0 & D_2 \end{pmatrix},$$

where U and V are orthogonal and X is nonsingular, and D_1 and D_2 are 'diagonal'. The former GSVD form can be converted to the latter form by taking the nonsingular matrix X as

$$X = Q\begin{pmatrix} I & 0 \\ 0 & R^{-1} \end{pmatrix}.$$

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

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 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 *Input*

On entry: if **jobu** = Nag_AllU, the orthogonal matrix U is computed.

If **jobu** = Nag_NotU, U is not computed.

Constraint: **jobu** = Nag_AllU or Nag_NotU.

3: **jobv** – Nag_ComputeVType *Input*

On entry: if **jobv** = Nag_ComputeV, the orthogonal matrix V is computed.

If **jobv** = Nag_NotV, V is not computed.

Constraint: **jobv** = Nag_ComputeV or Nag_NotV.

4: **jobq** – Nag_ComputeQType *Input*

On entry: if **jobq** = Nag_ComputeQ, the orthogonal matrix Q is computed.

If **jobq** = Nag_NotQ, Q is not computed.

Constraint: **jobq** = Nag_ComputeQ or Nag_NotQ.

5: **m** – Integer *Input*

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

Constraint: **m** ≥ 0.

6: **n** – Integer *Input*

On entry: n, the number of columns of the matrices A and B.

Constraint: **n** ≥ 0.

7: **p** – Integer *Input*

On entry: p, the number of rows of the matrix B.

Constraint: **p** ≥ 0.

8: **k** – Integer * *Output*
9: **l** – Integer * *Output*

On exit: **k** and **l** specify the dimension of the subblocks k and l as described in Section 3; $(k+l)$ is the effective numerical rank of $\begin{pmatrix} A \\ B \end{pmatrix}$.

10: **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

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the m by n matrix A.

On exit: contains the triangular matrix R, or part of R. See Section 3 for details.

11: **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, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **order** = Nag_RowMajor, $\mathbf{pda} \geq \max(1, \mathbf{n})$.

12: **b**[*dim*] – double *Input/Output*

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

$\max(1, \mathbf{pdb} \times \mathbf{n})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{p} \times \mathbf{pdb})$ when **order** = Nag_RowMajor.

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

$\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the p by n matrix B.

On exit: contains the triangular matrix R if $m - k - l < 0$. See Section 3 for details.

13: **pdb** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pdb} \geq \max(1, \mathbf{p})$;
if **order** = Nag_RowMajor, $\mathbf{pdb} \geq \max(1, \mathbf{n})$.

14: **alpha**[**n**] – double *Output*

On exit: see the description of **beta**.

15: **beta**[**n**] – double *Output*

On exit: **alpha** and **beta** contain the generalized singular value pairs of A and B, α_i and β_i;

$\mathbf{ALPHA}(1 : \mathbf{k}) = 1$,

$\mathbf{BETA}(1 : \mathbf{k}) = 0$,

and if $m - k - l \geq 0$,

$\mathbf{ALPHA}(\mathbf{k} + 1 : \mathbf{k} + \mathbf{l}) = C$,

$\mathbf{BETA}(\mathbf{k} + 1 : \mathbf{k} + \mathbf{l}) = S$,

or if $m - k - l < 0$,

$\mathbf{ALPHA}(\mathbf{k}+1:\mathbf{m}) = C$,

$\mathbf{ALPHA}(\mathbf{m}+1:\mathbf{k}+\mathbf{l}) = 0$,

$\mathbf{BETA}(\mathbf{k}+1:\mathbf{m}) = S$,

$\mathbf{BETA}(\mathbf{m}+1:\mathbf{k}+\mathbf{l}) = 1$, and

$\mathbf{ALPHA}(\mathbf{k}+\mathbf{l}+1:\mathbf{n}) = 0$,

$\mathbf{BETA}(\mathbf{k}+\mathbf{l}+1:\mathbf{n}) = 0$.

The notation $\mathbf{ALPHA}(\mathbf{k}:\mathbf{n})$ above refers to consecutive elements $\mathbf{alpha}[i-1]$, for $i = \mathbf{k}, \dots, \mathbf{n}$.

16: **u**[*dim*] – double *Output*

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

$\max(1, \mathbf{pdu} \times \mathbf{m})$ when **jobu** = Nag_AllU;
1 otherwise.

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

$\mathbf{u}[(j-1) \times \mathbf{pdu} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{u}[(i-1) \times \mathbf{pdu} + j - 1]$ when **order** = Nag_RowMajor.

On exit: if **jobu** = Nag_AllU, **u** contains the m by m orthogonal matrix U.

If **jobu** = Nag_NotU, **u** is not referenced.

17: **pdu** – Integer *Input*

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

Constraints:

if **jobu** = Nag_AllU, $\mathbf{pdu} \geq \max(1, \mathbf{m})$;
otherwise $\mathbf{pdu} \geq 1$.

18: **v**[*dim*] – double *Output*

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

$\max(1, \mathbf{pdv} \times \mathbf{p})$ when **jobv** = Nag_ComputeV;
1 otherwise.

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

$\mathbf{v}[(j-1) \times \mathbf{pdv} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{v}[(i-1) \times \mathbf{pdv} + j - 1]$ when **order** = Nag_RowMajor.

On exit: if **jobv** = Nag_ComputeV, **v** contains the p by p orthogonal matrix V.

If **jobv** = Nag_NotV, **v** is not referenced.

19: **pdv** – Integer *Input*

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

Constraints:

if **jobv** = Nag_ComputeV, $\mathbf{pdv} \geq \max(1, \mathbf{p})$;
otherwise $\mathbf{pdv} \geq 1$.

20: **q**[*dim*] – double *Output*

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

max(1, **pdq** × **n**) when **jobq** = Nag_ComputeQ;
1 otherwise.

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

q[$(j-1)$ × **pdq** + $i-1$] when **order** = Nag_ColMajor;
q[$(i-1)$ × **pdq** + $j-1$] when **order** = Nag_RowMajor.

On exit: if **jobq** = Nag_ComputeQ, **q** contains the n by n orthogonal matrix Q.

If **jobq** = Nag_NotQ, **q** is not referenced.

21: **pdq** – Integer *Input*

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

Constraints:

if **jobq** = Nag_ComputeQ, **pdq** ≥ max(1, **n**);
otherwise **pdq** ≥ 1.

22: **iwork**[**n**] – Integer *Output*

On exit: stores the sorting information. More precisely, the following loop will sort **alpha** such that **alpha**[0] ≥ **alpha**[1] ≥ ⋯ ≥ **alpha**[**n** − 1].

23: **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 ⟨*value*⟩ had an illegal value.

NE_CONVERGENCE

The Jacobi-type procedure failed to converge.

NE_ENUM_INT_2

On entry, **jobq** = ⟨*value*⟩, **pdq** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **jobq** = Nag_ComputeQ, **pdq** ≥ max(1, **n**);
otherwise **pdq** ≥ 1.

On entry, **jobu** = ⟨*value*⟩, **pdu** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: if **jobu** = Nag_AllU, **pdu** ≥ max(1, **m**);
otherwise **pdu** ≥ 1.

On entry, **jobv** = ⟨*value*⟩, **pdv** = ⟨*value*⟩ and **p** = ⟨*value*⟩.
Constraint: if **jobv** = Nag_ComputeV, **pdv** ≥ max(1, **p**);
otherwise **pdv** ≥ 1.

NE_INT

On entry, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 0$.

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

On entry, $\mathbf{p} = \langle value \rangle$.
Constraint: $\mathbf{p} \geq 0$.

On entry, $\mathbf{pda} = \langle value \rangle$.
Constraint: $\mathbf{pda} > 0$.

On entry, $\mathbf{pdb} = \langle value \rangle$.
Constraint: $\mathbf{pdb} > 0$.

On entry, $\mathbf{pdq} = \langle value \rangle$.
Constraint: $\mathbf{pdq} > 0$.

On entry, $\mathbf{pdu} = \langle value \rangle$.
Constraint: $\mathbf{pdu} > 0$.

On entry, $\mathbf{pdv} = \langle value \rangle$.
Constraint: $\mathbf{pdv} > 0$.

NE_INT_2

On entry, $\mathbf{pda} = \langle value \rangle$ and $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{pda} \geq \max(1, \mathbf{m})$.

On entry, $\mathbf{pda} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

On entry, $\mathbf{pdb} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pdb} \geq \max(1, \mathbf{n})$.

On entry, $\mathbf{pdb} = \langle value \rangle$ and $\mathbf{p} = \langle value \rangle$.
Constraint: $\mathbf{pdb} \geq \max(1, \mathbf{p})$.

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 generalized singular value decomposition is nearly the exact generalized singular value decomposition for nearby matrices $(A + E)$ and $(B + F)$, where

$$\|E\|_2 = O(\epsilon)\|A\|_2 \text{ and } \|F\|_2 = O(\epsilon)\|B\|_2,$$

and ϵ is the ***machine precision***. See Section 4.12 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_dggsvd (f08vac) is not threaded by NAG in any implementation.

nag_dggsvd (f08vac) 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 complex analogue of this function is nag_zggsvd (f08vnc).

10 Example

This example finds the generalized singular value decomposition

$$A = U\Sigma_1\begin{pmatrix} 0 & R \end{pmatrix}Q^{\mathrm{T}}, \quad B = V\Sigma_2\begin{pmatrix} 0 & R \end{pmatrix}Q^{\mathrm{T}},$$

where

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 5 & 6 \\ 7 & 8 & 8 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -2 & -3 & 3 \\ 4 & 6 & 5 \end{pmatrix},$$

together with estimates for the condition number of R and the error bound for the computed generalized singular values.

The example program assumes that $m \geq n$, and would need slight modification if this is not the case.

10.1 Program Text

```
/* nag_dggsvd (f08vac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */

#include <stdio.h>
#include <nag.h>
#include <nagf08.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx02.h>

int main(void)
{

  /* Scalars */
  double        d, eps, rcond, serrbd;
  Integer       exit_status = 0, i, irank, j, k, l, m, n, p,
                pda, pdb, pdq, pdu, pdv;
  NagError      fail;
  Nag_OrderType order;

  /* Arrays */
  double        *a = 0, *alpha = 0, *b = 0, *beta = 0, *q = 0, *u = 0, *v = 0;
  Integer       *iwork = 0;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
#define B(I, J) b[(J-1)*pdb + I - 1]
  order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
```

```
#define B(I, J) b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dggsvd (f08vac) 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"%"NAG_IFMT"%*[^\n] ", &m, &n, &p);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &m, &n, &p);
#endif
  if (m <= 10 && n <= 10 && p <= 10)
    {
      /* Allocate memory */
      if (!(a = NAG_ALLOC(m*n, double)) ||
          !(alpha = NAG_ALLOC(n, double)) ||
          !(b = NAG_ALLOC(p*n, double)) ||
          !(beta = NAG_ALLOC(n, double)) ||
          !(q = NAG_ALLOC(n*n, double)) ||
          !(u = NAG_ALLOC(m*m, double)) ||
          !(v = NAG_ALLOC(p*p, double)) ||
          !(iwork = NAG_ALLOC(n, Integer)) )
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
#ifdef NAG_COLUMN_MAJOR
      pda = m;
      pdb = p;
      pdq = n;
      pdu = m;
      pdv = p;
#else
      pda = n;
      pdb = n;
      pdq = n;
      pdu = m;
      pdv = p;
#endif
    }
  else
    {
      printf("m and/or n too small\n");
      goto END;
    }
  /* Read the m by n matrix A and p by n matrix B 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

  for (i = 1; i <= p; ++i)
#ifdef _WIN32
    for (j = 1; j <= n; ++j) scanf_s("%lf", &B(i, j));
#else
```

```
    for (j = 1; j <= n; ++j) scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* nag_dggsvd (f08vac)
   * Compute the generalized singular value decomposition of (A, B)
   * (A = U*D1*(0 R)*(Q**T), B = V*D2*(0 R)*(Q**T), m.ge.n)
   */

  nag_dggsvd(order, Nag_AllU, Nag_ComputeV, Nag_ComputeQ, m, n, p, &k, &l, a,
             pda, b, pdb, alpha, beta, u, pdu, v, pdv, q, pdq, iwork, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dggsvd (f08vac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print solution */
  irank = k + l;

  printf("Number of infinite generalized singular values (K)\n");
  printf("%5"NAG_IFMT"\n", k);
  printf("\nNumber of finite generalized singular values (L)\n");
  printf("%5"NAG_IFMT"\n", l);
  printf("Numerical rank of (A**T B**T)**T (K+L)\n");
  printf("%5"NAG_IFMT"\n", irank);

  printf("\nFinite generalized singular values\n");
  for (j = k; j < irank; ++j)
    {
      d = alpha[j] / beta[j];
      printf("%13.4e%s", d, (j+1)%8 == 0 || (j+1) == irank?"\n":" ");
    }

  printf("\n");
  fflush(stdout);
  nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m,
                              m, u, pdu, "%13.4e", "Orthogonal matrix U",
                              Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
                              80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\n");
  fflush(stdout);
  nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, p,
                              p, v, pdv, "%13.4e", "Orthogonal matrix V",
                              Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
                              80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\n");
  fflush(stdout);
  nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                              n, q, pdq, "%13.4e", "Orthogonal matrix Q",
                              Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
                              80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\n");
  fflush(stdout);
  nag_gen_real_mat_print_comp(order, Nag_UpperMatrix, Nag_NonUnitDiag, irank,
                              irank, &A(1, n - irank + 1), pda, "%13.4e",
                              "Non singular upper triangular matrix R",
                              Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
                              80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_dtrcon (f07tgc)
   * estimate the reciprocal condition number of R
   */

  nag_dtrcon(order, Nag_InfNorm, Nag_Upper, Nag_NonUnitDiag, irank,
             &A(1, n - irank + 1), pda, &rcond, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dtrcon (f07tgc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }


  printf("\nEstimate of reciprocal condition number for R\n");
  printf("%11.1e\n\n", rcond);

  /* So long as irank = n, get the machine precision, eps, and compute the
   * approximate error bound for the computed generalized singular values
   */
  if (irank == n)
    {
      eps = nag_machine_precision;
      serrbd = eps / rcond;
      printf("Error estimate for the generalized singular values");
      printf("\n%11.1e\n", serrbd);
    }
  else
    {
      printf("(A**T B**T)**T is not of full rank\n");
    }

 END:
  NAG_FREE(a);
  NAG_FREE(alpha);
  NAG_FREE(b);
  NAG_FREE(beta);
  NAG_FREE(q);
  NAG_FREE(u);
  NAG_FREE(v);
  NAG_FREE(iwork);

  return exit_status;
}
```

```
#undef B
#undef A
```

10.2 Program Data

```
nag_dggsvd (f08vac) Example Program Data

  4    3    2   :Values of M, N and P

  1.0  2.0  3.0
  3.0  2.0  1.0
  4.0  5.0  6.0
  7.0  8.0  8.0 :End of matrix A

 -2.0 -3.0  3.0
  4.0  6.0  5.0 :End of matrix B
```

10.3 Program Results

```
nag_dggsvd (f08vac) Example Program Results

Number of infinite generalized singular values (K)
    1

Number of finite generalized singular values (L)
    2
Numerical rank of (A**T B**T)**T (K+L)
    3

Finite generalized singular values
   1.3151e+00    8.0185e-02

 Orthogonal matrix U
              1             2             3             4
 1   -1.3484e-01    5.2524e-01   -2.0924e-01    8.1373e-01
 2    6.7420e-01   -5.2213e-01   -3.8886e-01    3.4874e-01
 3    2.6968e-01    5.2757e-01   -6.5782e-01   -4.6499e-01
 4    6.7420e-01    4.1615e-01    6.1014e-01    1.5127e-15

 Orthogonal matrix V
              1             2
 1    3.5539e-01   -9.3472e-01
 2    9.3472e-01    3.5539e-01

 Orthogonal matrix Q
              1             2             3
 1   -8.3205e-01   -9.4633e-02   -5.4657e-01
 2    5.5470e-01   -1.4195e-01   -8.1985e-01
 3    0.0000e+00   -9.8534e-01    1.7060e-01

 Non singular upper triangular matrix R
              1             2             3
 1   -2.0569e+00   -9.0121e+00   -9.3705e+00
 2                 -1.0882e+01   -7.2688e+00
 3                               -6.0405e+00

Estimate of reciprocal condition number for R
    4.2e-02

Error estimate for the generalized singular values
    2.6e-15
```