NAG Library Function Document

nag_dtgsja (f08yec)

1 Purpose

nag_dtgsja (f08yec) computes the generalized singular value decomposition (GSVD) of two real upper trapezoidal matrices A and B, where A is an m by n matrix and B is a p by n matrix.

A and B are assumed to be in the form returned by nag_dggsvp (f08vec) or nag_dggsvp3 (f08vgc).

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_dtgsja (Nag_OrderType order, Nag_ComputeUType jobu,
     Nag_ComputeVType jobv, Nag_ComputeQType jobq, Integer m, Integer p,
     Integer n, Integer k, Integer l, double a[], Integer pda, double b[],
     Integer pdb, double tola, double tolb, double alpha[], double beta[],
     double u[], Integer pdu, double v[], Integer pdv, double q[],
     Integer pdq, Integer *ncycle, NagError *fail)
```

3 Description

nag_dtgsja (f08yec) computes the GSVD of the matrices A and B which are assumed to have the form as returned by nag_dggsvp (f08vec) or nag_dggsvp3 (f08vgc)

$$A = \begin{cases} \begin{array}{r} \\ k \\ l \\ m-k-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix} \end{array}, & \text{if } m-k-l \geq 0; \\ \begin{array}{r} \\ k \\ m-k \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \end{array}, & \text{if } m-k-l < 0; \end{cases}$$

$$B = \begin{array}{r} \\ l \\ p-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \end{array},$$

where the k by k matrix A_{12} and the l by l matrix B_{13} are nonsingular upper triangular, A_{23} is l by l upper triangular if $m-k-l \geq 0$ and is $(m-k)$ by l upper trapezoidal otherwise.

nag_dtgsja (f08yec) computes orthogonal matrices Q, U and V, diagonal matrices D_1 and D_2, and an upper triangular matrix R such that

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

Optionally Q, U and V may or may not be computed, or they may be premultiplied by matrices Q_1, U_1 and V_1 respectively.

If $(m-k-l) \geq 0$ then D_1, D_2 and R have the form

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

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

$$R = \begin{array}{c} \\ k \\ l \end{array} \begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \end{array},$$

where $C = \mathrm{diag}(\alpha_{k+1}, , , \ldots, , , \alpha_{k+l})$, $\quad S = \mathrm{diag}(\beta_{k+1}, , , \ldots, , , \beta_{k+l})$.

If $(m-k-l) < 0$ then D_1, D_2 and R have the form

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

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

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

where $C = \mathrm{diag}(\alpha_{k+1}, , , \ldots, , , \alpha_m)$, $\quad S = \mathrm{diag}(\beta_{k+1}, , , \ldots, , , \beta_m)$.

In both cases the diagonal matrix C has non-negative diagonal elements, the diagonal matrix S has positive diagonal elements, so that S is nonsingular, and $C^2 + S^2 = 1$. See Section 2.3.5.3 of Anderson *et al.* (1999) for further information.

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 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: **jobu** – Nag_ComputeUType *Input*

On entry: if **jobu** = Nag_AllU, **u** must contain an orthogonal matrix U_1 on entry, and the product $U_1 U$ is returned.

If **jobu** = Nag_InitU, **u** is initialized to the unit matrix, and the orthogonal matrix U is returned.

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

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

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

On entry: if **jobv** = Nag_ComputeV, **v** must contain an orthogonal matrix V_1 on entry, and the product V_1V is returned.

If **jobv** = Nag_InitV, **v** is initialized to the unit matrix, and the orthogonal matrix V is returned.

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

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

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

On entry: if **jobq** = Nag_ComputeQ, **q** must contain an orthogonal matrix Q_1 on entry, and the product Q_1Q is returned.

If **jobq** = Nag_InitQ, **q** is initialized to the unit matrix, and the orthogonal matrix Q is returned.

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

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

5: **m** – Integer *Input*

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

Constraint: $\mathbf{m} \geq 0$.

6: **p** – Integer *Input*

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

Constraint: $\mathbf{p} \geq 0$.

7: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 0$.

8: **k** – Integer *Input*

9: **l** – Integer *Input*

On entry: **k** and **l** specify the sizes, k and l, of the subblocks of A and B, whose GSVD is to be computed by nag_dtgsja (f08yec).

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.

Where $\mathbf{A}(i,j)$ appears in this document, it refers to the array element

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ 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: if $m-k-l \geq 0$, $\mathbf{A}(1:k+l, n-k-l+1:n)$ contains the $(k+l)$ by $(k+l)$ upper triangular matrix R.

If $m-k-l<0$, $\mathbf{A}(1:m, n-k-l+1:n)$ contains the first m rows of the $(k+l)$ by $(k+l)$ upper triangular matrix R, and the submatrix R_{33} is returned in $\mathbf{B}(m-k+1:l, n+m-k-l+1:n)$.

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

Where $\mathbf{B}(i,j)$ appears in this document, it refers to the array element

$\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$ 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: if $m-k-l<0$, $\mathbf{B}(m-k+1:l, n+m-k-l+1:n)$ contains the submatrix R_{33} of R.

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

14: **tola** – double *Input*
15: **tolb** – double *Input*

On entry: **tola** and **tolb** are the convergence criteria for the Jacobi–Kogbetliantz iteration procedure. Generally, they should be the same as used in the preprocessing step performed by nag_dggsvp (f08vec) or nag_dggsvp3 (f08vgc), say

$$\mathbf{tola} = \max(\mathbf{m}, \mathbf{n})\|A\|\epsilon,$$
$$\mathbf{tolb} = \max(\mathbf{p}, \mathbf{n})\|B\|\epsilon,$$

where ϵ is the ***machine precision***.

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

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

17: **beta**[**n**] – double *Output*

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

$\mathbf{alpha}[i] = 1$, $\mathbf{beta}[i] = 0$, for $i = 0, 1, \ldots, k-1$, and

if $m-k-l \geq 0$, $\mathbf{alpha}[i] = \alpha_i$, $\mathbf{beta}[i] = \beta_i$, for $i = k, \ldots, k+l-1$, or

if $m-k-l<0$, $\mathbf{alpha}[i] = \alpha_i$, $\mathbf{beta}[i] = \beta_i$, for $i = k, \ldots, m-1$ and $\mathbf{alpha}[i] = 0$, $\mathbf{beta}[i] = 1$, for $i = m, \ldots, k+l-1$.

Furthermore, if $k+l<n$, **alpha**$[i]$ = **beta**$[i]=0$, for $i=k+l,\ldots,n-1$.

18: **u**$[dim]$ – double *Input/Output*

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

max(1, **pdu** × **m**) when **jobu** = Nag_AllU or Nag_InitU;
1 otherwise.

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

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

On entry: if **jobu** = Nag_AllU, **u** must contain an m by m matrix U_1 (usually the orthogonal matrix returned by nag_dggsvp (f08vec) or nag_dggsvp3 (f08vgc)).

On exit: if **jobu** = Nag_AllU, **u** contains the product U_1U.

If **jobu** = Nag_InitU, **u** contains the orthogonal matrix U.

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

19: **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 or Nag_InitU, **pdu** ≥ max(1, **m**);
otherwise **pdu** ≥ 1.

20: **v**$[dim]$ – double *Input/Output*

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

max(1, **pdv** × **p**) when **jobv** = Nag_ComputeV or Nag_InitV;
1 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_ComputeV, **v** must contain an p by p matrix V_1 (usually the orthogonal matrix returned by nag_dggsvp (f08vec) or nag_dggsvp3 (f08vgc)).

On exit: if **jobv** = Nag_InitV, **v** contains the orthogonal matrix V.

If **jobv** = Nag_ComputeV, **v** contains the product V_1V.

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

21: **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 or Nag_InitV, **pdv** ≥ max(1, **p**);
otherwise **pdv** ≥ 1.

22: **q**$[dim]$ – double *Input/Output*

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

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

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

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

On entry: if **jobq** = Nag_ComputeQ, **q** must contain an n by n matrix Q_1 (usually the orthogonal matrix returned by nag_dggsvp (f08vec) or nag_dggsvp3 (f08vgc)).

On exit: if **jobq** = Nag_InitQ, **q** contains the orthogonal matrix Q.

If **jobq** = Nag_ComputeQ, **q** contains the product Q_1Q.

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

23: **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 or Nag_InitQ, **pdq** $\geq \max(1, \mathbf{n})$;
otherwise **pdq** ≥ 1.

24: **ncycle** – Integer * *Output*

On exit: the number of cycles required for convergence.

25: **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 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_CONVERGENCE

The procedure does not converge after 40 cycles.

NE_ENUM_INT_2

On entry, **jobq** = ⟨*value*⟩, **pdq** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **jobq** = Nag_ComputeQ or Nag_InitQ, **pdq** $\geq \max(1, \mathbf{n})$;
otherwise **pdq** ≥ 1.

On entry, **jobu** = ⟨*value*⟩, **pdu** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: if **jobu** = Nag_AllU or Nag_InitU, **pdu** $\geq \max(1, \mathbf{m})$;
otherwise **pdu** ≥ 1.

On entry, **jobv** = ⟨*value*⟩, **pdv** = ⟨*value*⟩ and **p** = ⟨*value*⟩.
Constraint: if **jobv** = Nag_ComputeV or Nag_InitV, **pdv** $\geq \max(1, \mathbf{p})$;
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, **pda** $= \langle value \rangle$.
Constraint: **pda** > 0.

On entry, **pdb** $= \langle value \rangle$.
Constraint: **pdb** > 0.

On entry, **pdq** $= \langle value \rangle$.
Constraint: **pdq** > 0.

On entry, **pdu** $= \langle value \rangle$.
Constraint: **pdu** > 0.

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

NE_INT_2

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

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

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

On entry, **pdb** $= \langle value \rangle$ and $\mathbf{p} = \langle value \rangle$.
Constraint: **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 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 3.6.5 in How to Use the NAG Library and its Documentation 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

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

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

8 Parallelism and Performance

nag_dtgsja (f08yec) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

The complex analogue of this function is nag_ztgsja (f08ysc).

10 Example

This example finds the generalized singular value decomposition

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

of the matrix pair (A, B), where

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

10.1 Program Text

```
/* nag_dtgsja (f08yec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagx02.h>
#include <nagf08.h>
#include <nagf16.h>

int main(void)
{
  /* Scalars */
  double norma, normb, eps, tola, tolb;
  Integer i, irank, j, k, l, m, n, ncycle, p, pda, pdb, pdq, pdu, pdv;
  Integer printq, printr, printu, printv, vsize;
  Integer exit_status = 0;

  /* Arrays */
  double *a = 0, *alpha = 0, *b = 0, *beta = 0, *q = 0, *u = 0, *v = 0;
  char nag_enum_arg[40];

  /* Nag Types */
  NagError fail;
  Nag_OrderType order;
  Nag_ComputeUType jobu;
  Nag_ComputeVType jobv;
  Nag_ComputeQType jobq;
  Nag_MatrixType genmat = Nag_GeneralMatrix, upmat = Nag_UpperMatrix;
  Nag_DiagType diag = Nag_NonUnitDiag;
```

```
  Nag_LabelType intlab = Nag_IntegerLabels;

#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_dtgsja (f08yec) 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 < 0 || n < 0 || p < 0) {
    printf("Invalid m, n or p\n");
    exit_status = 1;
    goto END;
  }
#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
   */
  jobu = (Nag_ComputeUType) 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_ComputeVType) 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
  jobq = (Nag_ComputeQType) nag_enum_name_to_value(nag_enum_arg);

  pdu = (jobu != Nag_NotU ? m : 1);
  pdv = (jobv != Nag_NotV ? p : 1);
  pdq = (jobq != Nag_NotQ ? n : 1);
  vsize = (jobv != Nag_NotV ? p * m : 1);
#ifdef NAG_COLUMN_MAJOR
  pda = m;
  pdb = p;
#else
  pda = n;
  pdb = n;
#endif

  /* Read in 0s or 1s to determine whether matrices U, V, Q or R are to be
   * printed.
   */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]",
```

```
          &printu, &printv, &printq, &printr);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]",
        &printu, &printv, &printq, &printr);
#endif

  /* Allocate memory */
  if (!(a = NAG_ALLOC(m * n, double)) ||
      !(b = NAG_ALLOC(p * n, double)) ||
      !(alpha = NAG_ALLOC(n, double)) ||
      !(beta = NAG_ALLOC(n, double)) ||
      !(q = NAG_ALLOC(pdq * pdq, double)) ||
      !(u = NAG_ALLOC(pdu * pdu, double)) || !(v = NAG_ALLOC(vsize, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    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_dge_norm(order, Nag_FrobeniusNorm, m, n, a, pda, &norma, &fail);
  nag_dge_norm(order, Nag_FrobeniusNorm, p, n, b, pdb, &normb, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Compute tola and tolb using nag_machine_precision (x02ajc) */
  eps = nag_machine_precision;
  tola = MAX(m, n) * norma * eps;
  tolb = MAX(p, n) * normb * eps;

  /* Preprocess step:
   * compute transformations to reduce (A, B) to upper triangular form
   * (A = U1*S*(Q1^T), B = V1*T*(Q1^T))
   * using nag_dggsvp (f08vec).
   */
  nag_dggsvp(order, jobu, jobv, jobq, m, p, n, a, pda, b, pdb, tola, tolb, &k,
             &l, u, pdu, v, pdv, q, pdq, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dggsvp (f08vec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
```

```
  }

  /* Compute the generalized singular value decomposition of preprocessed (A,B)
   * (A = U*D1*(0 R)*(Q^T), B = V*D2*(0 R)*(Q^T))
   * using nag_dtgsja (f08yec). */
  nag_dtgsja(order, jobu, jobv, jobq, m, p, n, k, l, a, pda, b, pdb, tola,
             tolb, alpha, beta, u, pdu, v, pdv, q, pdq, &ncycle, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtgsja (f08yec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print the generalized singular value pairs alpha, beta */
  irank = MIN(k + l, m);
  printf("Number of infinite generalized singular values (k): %5" NAG_IFMT
         "\n", k);
  printf("Number of   finite generalized singular values (l): %5" NAG_IFMT
         "\n", l);
  printf("Effective Numerical rank  of  ( A^T B^T)^T   (k+l): %5" NAG_IFMT
         "\n", irank);
  printf("\nFinite generalized singular values:\n");

  for (j = k; j < irank; ++j)
    printf("%45s%12.4e\n", "", alpha[j] / beta[j]);

  printf("\nNumber of cycles of the Kogbetliantz method: %12" NAG_IFMT "\n\n",
         ncycle);

  if (printu && jobu != Nag_NotU) {
    fflush(stdout);
    nag_gen_real_mat_print_comp(order, genmat, diag, m, m, u, pdu, "%13.4e",
                                "Orthogonal matrix U", intlab, NULL, intlab,
                                NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR)
      goto PRINTERR;
    printf("\n");
  }
  if (printv && jobv != Nag_NotV) {
    fflush(stdout);
    nag_gen_real_mat_print_comp(order, genmat, diag, p, p, v, pdv, "%13.4e",
                                "Orthogonal matrix V", intlab, NULL, intlab,
                                NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR)
      goto PRINTERR;
    printf("\n");
  }
  if (printq && jobq != Nag_NotQ) {
    fflush(stdout);
    nag_gen_real_mat_print_comp(order, genmat, diag, n, n, q, pdq, "%13.4e",
                                "Orthogonal matrix Q", intlab, NULL, intlab,
                                NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR)
      goto PRINTERR;
    printf("\n");
  }
  if (printr) {
    fflush(stdout);
    nag_gen_real_mat_print_comp(order, upmat, diag, irank, irank,
                                &A(1, n - irank + 1), pda, "%13.4e",
                                "Nonsingular upper triangular matrix R",
                                intlab, NULL, intlab, NULL, 80, 0, NULL,
                                &fail);
  }
PRINTERR:
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
           fail.message);
    exit_status = 1;
  }
```

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

  return exit_status;
}
```

10.2 Program Data

```
nag_dtgsja (f08yec) Example Program Data

  4    3    2       : m, n and p

  Nag_AllU          : jobu
  Nag_ComputeV      : jobv
  Nag_ComputeQ      : jobq

  0    0    0    0  : printing u, v, q, r?

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

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

10.3 Program Results

```
nag_dtgsja (f08yec) Example Program Results

Number of infinite generalized singular values (k):     1
Number of   finite generalized singular values (l):     2
Effective Numerical rank  of  ( A^T B^T)^T   (k+l):     3

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

Number of cycles of the Kogbetliantz method:            2
```