

## 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).

#### 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)

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

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

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

$$U^T A Q = D_1 \begin{pmatrix} 0 & R \\ & \end{pmatrix}, \quad V^T B Q = D_2 \begin{pmatrix} 0 & R \\ & \end{pmatrix}.$$

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

$$D_1 = \begin{matrix} & k & l \\ & I & 0 \\ & 0 & C \\ m - k - l & 0 & 0 \end{matrix},$$

$$D_2 = \begin{matrix} & k & l \\ & 0 & S \\ p - l & 0 & 0 \end{matrix},$$

$$R = \begin{matrix} & k & l \\ k & R_{11} & R_{12} \\ l & 0 & R_{22} \end{matrix},$$

where  $C = \text{diag}(\alpha_{k+1}, \dots, \alpha_{k+l})$ ,  $S = \text{diag}(\beta_{k+1}, \dots, \beta_{k+l})$ .

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

$$D_1 = \begin{matrix} & k & m - k & k + l - m \\ & I & 0 & 0 \\ m - k & 0 & C & 0 \end{matrix},$$

$$D_2 = \begin{matrix} & k & m - k & k + l - m \\ m - k & 0 & S & 0 \\ k + l - m & 0 & 0 & I \\ p - l & 0 & 0 & 0 \end{matrix},$$

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

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

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 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, **u** must contain an orthogonal matrix  $U_1$  on entry, and the product  $U_1U$  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\_InitU, Nag\_AllU 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:*  $m \geq 0$ .
- 6: **p** – Integer *Input*  
*On entry:*  $p$ , the number of rows of the matrix  $B$ .  
*Constraint:*  $p \geq 0$ .
- 7: **n** – Integer *Input*  
*On entry:*  $n$ , the number of columns of the matrices  $A$  and  $B$ .  
*Constraint:*  $n \geq 0$ .
- 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, **m**);  
if **order** = Nag\_RowMajor, **pda**  $\geq$  max(1, **n**).

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

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

max(1, **pdb**  $\times$  **n**) when **order** = Nag\_ColMajor;  
max(1, **p**  $\times$  **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, **p**);  
if **order** = Nag\_RowMajor, **pdb**  $\geq$  max(1, **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\_zggsvp (f08vsc), say

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

where  $\epsilon$  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  $B$ ;

**alpha**[ $i$ ] = 1, **beta**[ $i$ ] = 0, for  $i = 0, 1, \dots, k - 1$ , and

if  $m - k - l \geq 0$ , **alpha**[ $i$ ] =  $\alpha_i$ , **beta**[ $i$ ] =  $\beta_i$ , for  $i = k, \dots, k + l - 1$ , or

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

Furthermore, if  $k + l < n$ ,  $\mathbf{alpha}[i] = \mathbf{beta}[i] = 0$ , for  $i = k + l, \dots, n - 1$ .

18:  $\mathbf{u}[dim]$  – double

*Input/Output*

**Note:** the dimension,  $dim$ , of the array  $\mathbf{u}$  must be at least

$\max(1, \mathbf{pdu} \times \mathbf{m})$  when  $\mathbf{jobu} = \text{Nag\_AllU}$  or  $\text{Nag\_InitU}$  and  $\mathbf{order} = \text{Nag\_ColMajor}$ ;  
 $\max(1, \times \mathbf{pdu})$  when  $\mathbf{jobu} = \text{Nag\_AllU}$  or  $\text{Nag\_InitU}$  and  $\mathbf{order} = \text{Nag\_RowMajor}$ ;  
 1 otherwise.

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

$\mathbf{u}[(j - 1) \times \mathbf{pdu} + i - 1]$  when  $\mathbf{order} = \text{Nag\_ColMajor}$ ;  
 $\mathbf{u}[(i - 1) \times \mathbf{pdu} + j - 1]$  when  $\mathbf{order} = \text{Nag\_RowMajor}$ .

*On entry:* if  $\mathbf{jobu} = \text{Nag\_AllU}$ ,  $\mathbf{u}$  must contain an  $m$  by  $m$  matrix  $U_1$  (usually the orthogonal matrix returned by `nag_dggsvp` (`f08vec`)).

*On exit:* if  $\mathbf{jobu} = \text{Nag\_InitU}$ ,  $\mathbf{u}$  contains the orthogonal matrix  $U$ .

If  $\mathbf{jobu} = \text{Nag\_AllU}$ ,  $\mathbf{u}$  contains the product  $U_1 U$ .

If  $\mathbf{jobu} = \text{Nag\_NotU}$ ,  $\mathbf{u}$  is not referenced.

19:  $\mathbf{pdu}$  – Integer

*Input*

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

*Constraints:*

if  $\mathbf{order} = \text{Nag\_ColMajor}$ ,  
     if  $\mathbf{jobu} \neq \text{Nag\_NotU}$ ,  $\mathbf{pdu} \geq \max(1, \mathbf{m})$ ;  
     otherwise  $\mathbf{pdu} \geq 1$ .;  
 if  $\mathbf{order} = \text{Nag\_RowMajor}$ ,  
     if  $\mathbf{jobu} = \text{Nag\_AllU}$  or  $\text{Nag\_InitU}$ ,  $\mathbf{pdu} \geq \max(1, \mathbf{m})$ ;  
     otherwise  $\mathbf{pdu} \geq 1$ .

20:  $\mathbf{v}[dim]$  – double

*Input/Output*

**Note:** the dimension,  $dim$ , of the array  $\mathbf{v}$  must be at least

$\max(1, \mathbf{pdv} \times \mathbf{p})$  when  $\mathbf{jobv} = \text{Nag\_ComputeV}$  or  $\text{Nag\_InitV}$  and  $\mathbf{order} = \text{Nag\_ColMajor}$ ;  
 $\max(1, \times \mathbf{pdv})$  when  $\mathbf{jobv} = \text{Nag\_ComputeV}$  or  $\text{Nag\_InitV}$  and  $\mathbf{order} = \text{Nag\_RowMajor}$ ;  
 1 otherwise.

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

$\mathbf{v}[(j - 1) \times \mathbf{pdv} + i - 1]$  when  $\mathbf{order} = \text{Nag\_ColMajor}$ ;  
 $\mathbf{v}[(i - 1) \times \mathbf{pdv} + j - 1]$  when  $\mathbf{order} = \text{Nag\_RowMajor}$ .

*On entry:* if  $\mathbf{jobv} = \text{Nag\_ComputeV}$ ,  $\mathbf{v}$  must contain an  $p$  by  $p$  matrix  $V_1$  (usually the orthogonal matrix returned by `nag_dggsvp` (`f08vec`)).

*On exit:* if  $\mathbf{jobv} = \text{Nag\_InitV}$ ,  $\mathbf{v}$  contains the orthogonal matrix  $V$ .

If  $\mathbf{jobv} = \text{Nag\_ComputeV}$ ,  $\mathbf{v}$  contains the product  $V_1 V$ .

If  $\mathbf{jobv} = \text{Nag\_NotV}$ ,  $\mathbf{v}$  is not referenced.

21:  $\mathbf{pdv}$  – Integer

*Input*

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

*Constraints:*

if **order** = Nag\_ColMajor,  
 if **jobv** ≠ Nag\_NotV, **pdv** ≥ max(1, **p**);  
 otherwise **pdv** ≥ 1.;  
 if **order** = Nag\_RowMajor,  
 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 and **order** = Nag\_ColMajor;  
 max(1, ×**pdq**) when **jobq** = Nag\_ComputeQ or Nag\_InitQ and **order** = Nag\_RowMajor;  
 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 entry:* if **jobq** = Nag\_ComputeQ, **q** must contain an *n* by *n* matrix  $Q_1$  (usually the orthogonal matrix returned by nag\_dggsvp (f08vec)).

*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 **order** = Nag\_ColMajor,  
 if **jobq** ≠ Nag\_NotQ, **pdq** ≥ max(1, **n**);  
 otherwise **pdq** ≥ 1.;  
 if **order** = Nag\_RowMajor,  
 if **jobq** = Nag\_ComputeQ or Nag\_InitQ, **pdq** ≥ max(1, **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 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### 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⟩*, **n** = *⟨value⟩*.  
 Constraint: if **jobq** = Nag\_ComputeQ or Nag\_InitQ, **pdq**  $\geq$  max(1, **n**);  
 otherwise **pdq**  $\geq$  1.

On entry, **jobu** = *⟨value⟩*, **pdu** = *⟨value⟩*, **m** = *⟨value⟩*.  
 Constraint: if **jobu** = Nag\_AllU or Nag\_InitU, **pdu**  $\geq$  max(1, **m**);  
 otherwise **pdu**  $\geq$  1.

On entry, **jobu** = *⟨value⟩*, **pdu** = *⟨value⟩* and **m** = *⟨value⟩*.  
 Constraint: if **jobu**  $\neq$  Nag\_NotU, **pdu**  $\geq$  max(1, **m**);  
 otherwise **pdu**  $\geq$  1.

On entry, **jobv** = *⟨value⟩*, **pdv** = *⟨value⟩*, **p** = *⟨value⟩*.  
 Constraint: if **jobv** = Nag\_ComputeV or Nag\_InitV, **pdv**  $\geq$  max(1, **p**);  
 otherwise **pdv**  $\geq$  1.

On entry, **jobv** = *⟨value⟩*, **pdv** = *⟨value⟩* and **p** = *⟨value⟩*.  
 Constraint: if **jobv**  $\neq$  Nag\_NotV, **pdv**  $\geq$  max(1, **p**);  
 otherwise **pdv**  $\geq$  1.

On entry, **pdq** = *⟨value⟩*, **jobq** = *⟨value⟩* and **n** = *⟨value⟩*.  
 Constraint: if **jobq**  $\neq$  Nag\_NotQ, **pdq**  $\geq$  max(1, **n**);  
 otherwise **pdq**  $\geq$  1.

**NE\_INT**

On entry, **m** = *⟨value⟩*.  
 Constraint: **m**  $\geq$  0.

On entry, **n** = *⟨value⟩*.  
 Constraint: **n**  $\geq$  0.

On entry, **p** = *⟨value⟩*.  
 Constraint: **p**  $\geq$  0.

On entry, **pda** = *⟨value⟩*.  
 Constraint: **pda**  $>$  0.

On entry, **pdb** = *⟨value⟩*.  
 Constraint: **pdb**  $>$  0.

On entry, **pdq** = *⟨value⟩*.  
 Constraint: **pdq**  $>$  0.

On entry, **pdu** = *⟨value⟩*.  
 Constraint: **pdu**  $>$  0.

On entry, **pdv** = *⟨value⟩*.  
 Constraint: **pdv**  $>$  0.

**NE\_INT\_2**

On entry, **pda** = *⟨value⟩* and **m** = *⟨value⟩*.  
 Constraint: **pda**  $\geq$  max(1, **m**).

On entry, **pda** = *⟨value⟩* and **n** = *⟨value⟩*.  
 Constraint: **pda**  $\geq$  max(1, **n**).

On entry, **pdb** = *⟨value⟩* and **n** = *⟨value⟩*.  
 Constraint: **pdb**  $\geq$  max(1, **n**).

On entry, **pdb** =  $\langle value \rangle$  and **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.

## 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  $\epsilon$  is the *machine precision*. See Section 4.12 of Anderson *et al.* (1999) for further details.

## 8 Parallelism and Performance

nag\_dtgsja (f08yec) is not threaded by NAG in any implementation.

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 Users' Note for 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

$$A = U\Sigma_1(0 \ R)Q^T, \quad B = V\Sigma_2(0 \ R)Q^T,$$

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

#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 */
scanf("%*[\n]");
scanf("%ld%ld%ld%*[\n]", &m, &n, &p);
if (m < 0 || n < 0 || p < 0)
{
    printf("Invalid m, n or p\n");
    exit_status = 1;
    goto END;
}
scanf(" %39s%*[\n]", nag_enum_arg);
/* nag_enum_name_to_value (x04nac).
 * Converts NAG enum member name to value
 */
jobu = (Nag_ComputeUType) nag_enum_name_to_value(nag_enum_arg);
scanf(" %39s%*[\n]", nag_enum_arg);
jobv = (Nag_ComputeVType) nag_enum_name_to_value(nag_enum_arg);
scanf(" %39s%*[\n]", nag_enum_arg);
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
#endif

/* Read in 0s or 1s to determine whether matrices U, V, Q or R are to be
 * printed.
 */
scanf("%ld%ld%ld%ld%*[\n]",
        &printu, &printv, &printq, &printr);

```

```

/* 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)
    for (j = 1; j <= n; ++j) scanf("%lf", &A(i, j));
scanf("%*[\n]");
for (i = 1; i <= p; ++i)
    for (j = 1; j <= n; ++j) scanf("%lf", &B(i, j));
scanf("%*[\n]");

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*(O R)*(Q**T), B = V*D2*(O 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 + 1, m);
printf("Number of infinite generalized singular values (k): %5ld\n", k);
printf("Number of finite generalized singular values (l): %5ld\n", l);
printf("Effective Numerical rank of (A**T B**T)**T (k+1): %5ld\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: %12ld\n\n", ncycle);

if (printu && jobv!=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",
                               "Non singular 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
```

---