1 Purpose

nag_ztgsja (f08ysc) computes the generalized singular value decomposition (GSVD) of two complex 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_zggsvp (f08vsc).

2 Specification

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

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

3 Description

nag_ztgsja (f08ysc) computes the GSVD of the matrices \( A \) and \( B \) which are assumed to have the form as returned by nag_zggsvp (f08vsc)

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

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

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_ztgsja (f08ysc) computes unitary matrices \( Q \), \( U \) and \( V \), diagonal matrices \( D_1 \) and \( D_2 \), and an upper triangular matrix \( R \) such that

\[
U^H AQ = D_1 \begin{pmatrix} \quad & R \end{pmatrix}, \quad V^H BQ = D_2 \begin{pmatrix} \quad & R \end{pmatrix}.
\]

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

\[
D_1 = \begin{pmatrix}
  k & l \\
  m - k - l & 0
\end{pmatrix},
\]

\[
D_2 = \begin{pmatrix}
  k & l \\
  p - l & 0
\end{pmatrix},
\]

\[
R = \begin{pmatrix}
  k & l \\
  \frac{k}{C_{19}} & \frac{l}{C_{18}}
\end{pmatrix},
\]

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

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

\[
D_1 = \begin{pmatrix}
  k & m - k & k + l - m \\
  m - k & 0 & 0 \\
  0 & C & 0
\end{pmatrix},
\]

\[
D_2 = \begin{pmatrix}
  m - k & k + l - m \\
  k + l - m & 0 & 0 \\
  0 & 0 & I
\end{pmatrix},
\]

\[
R = \begin{pmatrix}
  k & m - k & k + l - m \\
  m - k & 0 & 0 \\
  k + l - m & \frac{k}{R_{11}} & \frac{R_{12}}{R_{33}}
\end{pmatrix},
\]

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

In both cases the diagonal matrix \( C \) has real non-negative diagonal elements, the diagonal matrix \( S \) has real 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


5 Arguments

1: \textbf{order} – Nag_OrderType

On entry: the \textbf{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: if jobu = Nag_AllU, u must contain a unitary 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 unitary 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

On entry: if jobv = Nag_ComputeV, v must contain a unitary 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 unitary 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

On entry: if jobq = Nag_ComputeQ, q must contain a unitary 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 unitary matrix $Q$ is returned.

If jobq = Nag_NotQ, Q is not computed.

Constraint: jobq = Nag_ComputeQ, Nag_InitQ or Nag_NotQ.

5: m – Integer

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

Constraint: $m \geq 0$.

6: p – Integer

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

Constraint: $p \geq 0$.

7: n – Integer

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

Constraint: $n \geq 0$.

8: k – Integer

9: l – Integer

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_ztgsja (f08ysc).

10: a[dim] – Complex

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

$\max(1, pda \times n)$ when order = Nag_ColMajor;

$\max(1, m \times pda)$ when order = Nag_RowMajor.
Where \( A(i, j) \) appears in this document, it refers to the array element

\[
    a[j \times pda + i - 1] \quad \text{when order = Nag_ColMajor};
\]

\[
    a[i \times pda + j - 1] \quad \text{when order = Nag_RowMajor}.
\]

On entry: the \( m \) by \( n \) matrix \( A \).

On exit: if \( m - k - l \geq 0 \), \( A(1: k + l, n - k - l + 1: n) \) contains the \( (k + l) \) by \( (k + l) \) upper triangular matrix \( R \).

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

11: \( \text{pda} \) – Integer

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

Constraints:

\[
    \begin{align*}
    \text{if order = Nag_ColMajor, pda} & \geq \max(1, m); \\
    \text{if order = Nag_RowMajor, pda} & \geq \max(1, n).
\end{align*}
\]

12: \( \text{b}[\text{dim}] \) – Complex

Input/Output

Note: the dimension, \( \text{dim} \), of the array \( b \) must be at least

\[
    \max(1, \text{pdb} \times n) \quad \text{when order = Nag_ColMajor};
\]

\[
    \max(1, p \times \text{pdb}) \quad \text{when order = Nag_RowMajor}.
\]

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

\[
    \begin{align*}
    \text{b}[j \times \text{pdb} + i - 1] \quad \text{when order = Nag_ColMajor}; \\
    \text{b}[i \times \text{pdb} + j - 1] \quad \text{when order = Nag_RowMajor}.
\end{align*}
\]

On entry: the \( p \) by \( n \) matrix \( B \).

On exit: if \( m - k - l < 0 \), \( B(m - k + 1: l, n + m - k - l + 1: n) \) contains the submatrix \( R_{33} \) of \( R \).

13: \( \text{pdb} \) – Integer

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

Constraints:

\[
    \begin{align*}
    \text{if order = Nag_ColMajor, pdb} & \geq \max(1, p); \\
    \text{if order = Nag_RowMajor, pdb} & \geq \max(1, n).
\end{align*}
\]

14: \( \text{tola} \) – double

Input

On entry: \( \text{tola} \) and \( \text{tolb} \) are the convergence criteria for the Jacobi–Kogbetliantz iteration procedure. Generally, they should be the same as used in the preprocessing step performed by \( \text{nag_zggsvp (f08vsc)} \), say

\[
    \begin{align*}
    \text{tola} & = \max(m, n)\|A\|\epsilon, \\
    \text{tolb} & = \max(p, n)\|B\|\epsilon,
\end{align*}
\]

where \( \epsilon \) is the machine precision.

16: \( \text{alpha}[n] \) – double

Output

On exit: see the description of \( \text{beta} \).
17: \textbf{beta} \[ n \] – double

Output

On exit: \textbf{alpha} and \textbf{beta} contain the generalized singular value pairs of \( A \) and \( B \);

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

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

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

Furthermore, if \( k + l < n \), \( \text{alpha}[i] = \text{beta}[i] = 0, \) for \( i = k + l, \ldots, n - 1. \)

18: \textbf{u}[\text{dim}] – Complex

Input/Output

Note: the dimension, \textit{dim}, of the array \textbf{u} must be at least

\[ \max(1, \text{pdu} \times m) \text{ when } \textbf{jobu} = \text{Nag}_\text{AllU} \text{ or Nag}_\text{InitU}; \]

1 otherwise.

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

\[ \text{u}[(j - 1) \times \text{pdu} + i - 1] \text{ when } \text{order} = \text{Nag}_\text{ColMajor}; \]

\[ \text{u}[(i - 1) \times \text{pdu} + j - 1] \text{ when } \text{order} = \text{Nag}_\text{RowMajor}. \]

On entry: if \textbf{jobu} = \text{Nag}_\text{AllU}, \textbf{u} must contain an \( m \times m \) matrix \( U_1 \) (usually the unitary matrix returned by \text{zggsvp (f08vsc)}).

On exit: if \textbf{jobu} = \text{Nag}_\text{AllU}, \textbf{u} contains the product \( U_1 U \).

If \textbf{jobu} = \text{Nag}_\text{InitU}, \textbf{u} contains the unitary matrix \( U \).

If \textbf{jobu} = \text{Nag}_\text{NotU}, \textbf{u} is not referenced.

19: \textbf{pdu} – Integer

Input

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

Constraints:

\[ \text{if } \textbf{jobu} = \text{Nag}_\text{AllU} \text{ or Nag}_\text{InitU}, \textbf{pdu} \geq \max(1, m); \]

1 otherwise.

20: \textbf{v}[\text{dim}] – Complex

Input/Output

Note: the dimension, \textit{dim}, of the array \textbf{v} must be at least

\[ \max(1, \text{pdv} \times p) \text{ when } \textbf{jobv} = \text{Nag}_\text{ComputeV} \text{ or Nag}_\text{InitV}; \]

1 otherwise.

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

\[ \text{v}[(j - 1) \times \text{pdv} + i - 1] \text{ when } \text{order} = \text{Nag}_\text{ColMajor}; \]

\[ \text{v}[(i - 1) \times \text{pdv} + j - 1] \text{ when } \text{order} = \text{Nag}_\text{RowMajor}. \]

On entry: if \textbf{jobv} = \text{Nag}_\text{ComputeV}, \textbf{v} must contain an \( p \times p \) matrix \( V_1 \) (usually the unitary matrix returned by \text{zggsvp (f08vsc)}).

On exit: if \textbf{jobv} = \text{Nag}_\text{InitV}, \textbf{v} contains the unitary matrix \( V \).

If \textbf{jobv} = \text{Nag}_\text{ComputeV}, \textbf{v} contains the product \( V_1 V \).

If \textbf{jobv} = \text{Nag}_\text{NotV}, \textbf{v} is not referenced.

21: \textbf{pdv} – Integer

Input

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

if \( \text{jobv} = \text{Nag\_ComputeV} \) or \( \text{Nag\_InitV} \), \( \text{pdv} \geq \max(1, p) \);
otherwise \( \text{pdv} \geq 1 \).

22: \( \text{q}[\text{dim}] \) \(-\) Complex

Note: the dimension, \( \text{dim} \), of the array \( \text{q} \) must be at least
\( \max(1, \text{pdq} \times n) \) when \( \text{jobq} = \text{Nag\_ComputeQ} \) or \( \text{Nag\_InitQ} \);
1 otherwise.

The \((i, j)\)th element of the matrix \( Q \) is stored in
\( q[(j - 1) \times \text{pdq} + i - 1] \) when \( \text{order} = \text{Nag\_ColMajor} \);
\( q[(i - 1) \times \text{pdq} + j - 1] \) when \( \text{order} = \text{Nag\_RowMajor} \).

On entry: if \( \text{jobq} = \text{Nag\_ComputeQ} \), \( \text{q} \) must contain an \( n \) by \( n \) matrix \( Q \)1 (usually the unitary
matrix returned by nag_zggsvp (f08vsc)).

On exit: if \( \text{jobq} = \text{Nag\_InitQ} \), \( \text{q} \) contains the unitary matrix \( Q \).
If \( \text{jobq} = \text{Nag\_ComputeQ} \), \( \text{q} \) contains the product \( Q_1 Q \).
If \( \text{jobq} = \text{Nag\_NotQ} \), \( \text{q} \) is not referenced.

23: \( \text{pdq} \) \(-\) Integer

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

Constraints:

if \( \text{jobq} = \text{Nag\_ComputeQ} \) or \( \text{Nag\_InitQ} \), \( \text{pdq} \geq \max(1, n) \);
otherwise \( \text{pdq} \geq 1 \).

24: \( \text{ncycle} \) \(-\) Integer *

On exit: the number of cycles required for convergence.

25: \( \text{fail} \) \(-\) NagError *

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

6 Error Indicators and Warnings

**NE\_ALLOC\_FAIL**

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

**NE\_BAD\_PARAM**

On entry, argument \(<\text{value}>\) had an illegal value.

**NE\_CONVERGENCE**

The procedure does not converge after 40 cycles.

**NE\_ENUM\_INT\_2**

On entry, \( \text{jobq} = <\text{value}> \), \( \text{pdq} = <\text{value}> \) and \( n = <\text{value}> \).
Constraint: if \( \text{jobq} = \text{Nag\_ComputeQ} \) or \( \text{Nag\_InitQ} \), \( \text{pdq} \geq \max(1, n) \);
otherwise \( \text{pdq} \geq 1 \).
On entry, \( \text{jobu} = \langle \text{value} \rangle \), \( \text{pdu} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).
Constraint: if \( \text{jobu} = \text{Nag\_AllU} \) or \( \text{Nag\_InitU} \), \( \text{pdu} \geq \max(1,m) \);
otherwise \( \text{pdu} \geq 1 \).

On entry, \( \text{jobv} = \langle \text{value} \rangle \), \( \text{pdv} = \langle \text{value} \rangle \) and \( p = \langle \text{value} \rangle \).
Constraint: if \( \text{jobv} = \text{Nag\_ComputeV} \) or \( \text{Nag\_InitV} \), \( \text{pdv} \geq \max(1,p) \);
otherwise \( \text{pdv} \geq 1 \).

**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, \( p = \langle \text{value} \rangle \).
Constraint: \( p \geq 0 \).

On entry, \( \text{pda} = \langle \text{value} \rangle \).
Constraint: \( \text{pda} > 0 \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} > 0 \).

On entry, \( \text{pdq} = \langle \text{value} \rangle \).
Constraint: \( \text{pdq} > 0 \).

On entry, \( \text{pdu} = \langle \text{value} \rangle \).
Constraint: \( \text{pdu} > 0 \).

On entry, \( \text{pdv} = \langle \text{value} \rangle \).
Constraint: \( \text{pdv} > 0 \).

**NE\_INT\_2**

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

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

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

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

**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

\[
\|E\|_2 = O \varepsilon \|A\|_2 \quad \text{and} \quad \|F\|_2 = O \varepsilon \|B\|_2 ,
\]

and \( \varepsilon \) is the \textit{machine precision}. See Section 4.12 of Anderson et al. (1999) for further details.

8 Parallelism and Performance

\texttt{nag_ztgsja} (f08ysc) is not threaded by NAG in any implementation.

\texttt{nag_ztgsja} (f08ysc) 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 real analogue of this function is \texttt{nag_dtgsja} (f08yec).

10 Example

This example finds the generalized singular value decomposition

\[
A = U\Sigma_1 (0 \quad R) Q^H , \quad B = V\Sigma_2 (0 \quad R) Q^H ,
\]

of the matrix pair \( (A, B) \), where

\[
A = \begin{pmatrix}
0.96 - 0.81i & -0.03 + 0.96i & -0.91 + 2.06i & -0.05 + 0.41i \\
-0.98 + 1.98i & -1.20 + 0.19i & -0.66 + 0.42i & -0.81 + 0.56i \\
0.62 - 0.46i & 1.01 + 0.02i & 0.63 - 0.17i & -1.11 + 0.60i \\
0.37 + 0.38i & 0.19 - 0.54i & -0.98 - 0.36i & 0.22 - 0.20i \\
0.83 + 0.51i & 0.20 + 0.01i & -0.17 - 0.46i & 1.47 + 1.59i \\
1.08 - 0.28i & 0.20 - 0.12i & -0.07 + 1.23i & 0.26 + 0.26i
\end{pmatrix}
\]

and

\[
B = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} .
\]

10.1 Program Text

/* \texttt{nag_ztgsja} (f08ysc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */

#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 eps, norma, normb, tola, tolb;
Integer i, irank, j, k, l, m, n, ncycle, p, pda, pdb, pdu, pdv;
Integer pdq, printq, prinr, printu, printv, vsize;
Integer exit_status = 0;

/* Arrays */
Complex *a = 0, *b = 0, *q = 0, *u = 0, *v = 0;
double *alpha = 0, *beta = 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;
Nag_ComplexFormType brac = Nag_BracketForm;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
#define B(I, J) b[(J-1)*pdb + I - 1]
#else
#define A(I, J) a[(I-1)*pda+J-1]
#define B(I, J) b[(I-1)*pdb +J - 1]
#endif

INIT_FAIL(fail);
printf("nag_ztgsja (f08ysc) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[\n]");
#else
scanf("%*[\n]");
#endif

#ifdef _WIN32
scanf_s("%39s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf("%39s%*[\n]", nag_enum_arg);
#endif

#ifdef _WIN32
scanf_s("%39s%*[\n]", nag_enum_arg, _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, _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, _countof(nag_enum_arg));
#else
scanf("%39s%*[\n]", nag Enum arg);
#endif

jobq = (Nag_ComputeQType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
scanf_s("%39s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf("%39s%*[\n]", nag_enum_arg);
#endif

if (m<0 | | n<0 | | p<0) {
  printf("Invalid m, n or p\n");
  exit_status = 1;
  goto END;
}

END:
#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, Complex)) ||
    !(b = NAG_ALLOC(p*n, Complex)) ||
    !(alpha = NAG_ALLOC(n, double)) ||
    !(beta = NAG_ALLOC(n, double)) ||
    !(q = NAG_ALLOC(pdq*pdq, Complex)) ||
    !(u = NAG_ALLOC(pdu*pdu, Complex)) ||
    !(v = NAG_ALLOC(vsize, Complex))
    {
        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)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#ifdef _WIN32
        scanf_s("%[\n];
#else
        scanf("%[\n];
#endif
    for (i = 1; i <= p; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#else
        scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
#ifdef _WIN32
        scanf_s("%[\n];
#else
        scanf("%[\n];
#endif

    /* Compute tola and tolb as */
    /* tola = max(m,n)*norm(A)*macheps */
    /* tolb = max(p,n)*norm(B)*macheps */
    nag_zge_norm(order, Nag_OneNorm, m, n, a, pda, &norma, &fail);
    nag_zge_norm(order, Nag_OneNorm, p, n, b, pdb, &normb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zge_norm (f16uac).\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\textsuperscript{H}), B = V1*T*(Q1\textsuperscript{H}))
* using nag_zggsvp (f08vsc).
*/
nag_zggsvp(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_zggsvp (f08vsc).\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**H), B = V^D2*(0 R)*(Q**H))
* using nag_ztgsja (f08ysc).
*/
nag_ztgsja(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_ztgsja (f08ysc).\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"
", k);
printf("Number of finite generalized singular values (l): %5"NAG_IFMT"
", l);
printf("Effective Numerical rank of (A^H B^HT)^H (k+l): %5"NAG_IFMT"
", irank);

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

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

if (printu && jobu!=Nag_NotU) {
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, genmat, diag, m, m, u, pdu, brac,
        "%13.4e", "Orthogonal matrix U", intlab,
        NULL, intlab, NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR) goto PRINTERR;
}
if (printv && jobv!=Nag_NotV) {
    printf("\n");
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, genmat, diag, p, p, v, pdv, brac,
        "%13.4e", "Orthogonal matrix V", intlab,
        NULL, intlab, NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR) goto PRINTERR;
}
if (printq && jobq!=Nag_NotQ) {
    printf("\n");
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, genmat, diag, n, n, q, pdq, brac,
        "%13.4e", "Orthogonal matrix Q", intlab,
        NULL, intlab, NULL, 80, 0, NULL, &fail);
    if (fail.code != NE_NOERROR) goto PRINTERR;
}
```c
if (fail.code != NE_NOERROR) goto PRINTERR;
}
if (printr) {
    printf("\n");
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, upmat, diag, irank, irank,
    &A(1, n - irank + 1), pda, brac, "%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(b);
NAG_FREE(alpha);
NAG_FREE(beta);
NAG_FREE(q);
NAG_FREE(u);
NAG_FREE(v);
return exit_status;
}

10.2 Program Data

nag_ztgsja (f08ysc) Example Program Data

<table>
<thead>
<tr>
<th>6</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>n</td>
<td>p</td>
</tr>
</tbody>
</table>

Nag_AllU : jobu
Nag_ComputeV : jobv
Nag_ComputeQ : jobq

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

( 0.96, -0.81) (-0.03, 0.96) (-0.91, 2.06) (-0.05, 0.41)
(-0.98, 1.98) (-1.20, 0.19) (-0.66, 0.42) (-0.81, 0.56)
( 0.62, -0.46) ( 1.01, 0.02) ( 0.63, -0.17) ( 1.11, 0.60)
( 0.37, 0.38) ( 0.19, -0.54) (-0.98, -0.36) ( 0.22, -0.20)
( 0.83, 0.51) ( 0.20, 0.01) (-0.17, -0.46) ( 1.47, 1.59)
( 1.08, -0.28) ( 0.20, -0.12) (-0.07, 1.23) ( 0.26, 0.26) : matrix A

( 1.00, 0.00) ( 0.00, 0.00) (-1.00, 0.00) ( 0.00, 0.00)
( 0.00, 0.00) ( 1.00, 0.00) ( 0.00, 0.00) (-1.00, 0.00) : matrix B

10.3 Program Results

nag_ztgsja (f08ysc) Example Program Results

Number of infinite generalized singular values (k): 2
Number of finite generalized singular values (l): 2
Effective Numerical rank of (A^H B^HT)^H (k+l): 4

Finite generalized singular values:
Number of cycles of the Kogbetliantz method: 2