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

```c
#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)

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

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

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

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

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

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

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

\[
R = \begin{pmatrix}
k & l \\
l & 0 & R_{11} \\
0 & S & 0 \\
0 & 0 & I
\end{pmatrix},
\]

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

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

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

\[
R = \begin{pmatrix}
k & m - k & k + l - m \\
m - k & 0 & R_{11} \\
0 & R_{22} & R_{13} \\
0 & 0 & R_{23}
\end{pmatrix},
\]

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 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


### 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_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:* $m \geq 0$.

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

- $\max(1, pda \times n)$ 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
\[
\begin{align*}
\text{a}[(j - 1) \times \text{pda} + i - 1] & \quad \text{when order} = \text{Nag}\_\text{ColMajor}; \\
\text{a}[(i - 1) \times \text{pda} + j - 1] & \quad \text{when order} = \text{Nag}\_\text{RowMajor}.
\end{align*}
\]

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: \pda – Integer

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

Constraints:
\[
\begin{align*}
\text{if order} &= \text{Nag}\_\text{ColMajor}, \quad \text{pda} \geq \max(1, m); \\
\text{if order} &= \text{Nag}\_\text{RowMajor}, \quad \text{pda} \geq \max(1, n).
\end{align*}
\]

12: \( b[dim] \) – double

Input/Output

Note: the dimension, \( dim \), of the array \( b \) must be at least
\[
\max(1, \text{pdb} \times \text{n}) \quad \text{when order} = \text{Nag}\_\text{ColMajor}; \\
\max(1, \text{p} \times \text{pdb}) \quad \text{when order} = \text{Nag}\_\text{RowMajor}.
\]

Where \( B(i, j) \) appears in this document, it refers to the array element
\[
\begin{align*}
\text{b}[(j - 1) \times \text{pdb} + i - 1] & \quad \text{when order} = \text{Nag}\_\text{ColMajor}; \\
\text{b}[(i - 1) \times \text{pdb} + j - 1] & \quad \text{when order} = \text{Nag}\_\text{RowMajor}.
\end{align*}
\]

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: \pdb – Integer

Input

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

Constraints:
\[
\begin{align*}
\text{if order} &= \text{Nag}\_\text{ColMajor}, \quad \text{pdb} \geq \max(1, p); \\
\text{if order} &= \text{Nag}\_\text{RowMajor}, \quad \text{pdb} \geq \max(1, n).
\end{align*}
\]

14: \( \text{tola} \) – double

Input

15: \( \text{tolb} \) – 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}\_zggsyvp (f08vsc), say
\[
\begin{align*}
\text{tola} &= \max(\text{m}, \text{n}) \| A \| \epsilon, \\
\text{tolb} &= \max(\text{p}, \text{n}) \| B \| \epsilon,
\end{align*}
\]

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

16: \( \alpha[n] \) – double

Output

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

On exit: \textbf{alpha} and \textbf{beta} contain the generalized singular value pairs of \textbf{A} and \textbf{B};
\[ \alpha[i] = 1, \quad \text{for } i = 0, 1, \ldots, k - 1, \text{ and} \]
\[ \beta[i] = 0, \quad \text{for } i = 0, 1, \ldots, k - 1, \text{ and} \]
if \( m - k - l \geq 0, \) \( \alpha[i] = \alpha_i, \quad \beta[i] = \beta_i, \) for \( i = k, \ldots, k + l - 1, \text{ or} \]
if \( m - k - l < 0, \) \( \alpha[i] = \alpha_i, \quad \beta[i] = \beta_i, \) for \( i = k, \ldots, m - 1 \) and \( \alpha[i] = 0, \beta[i] = 1, \) for \( i = k + l, \ldots, n - 1. \)

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

18:  \textbf{u[dim]} – double  \textit{Input/Output}

\textbf{Note:} the dimension, \textit{dim}, of the array \textbf{u} must be at least
\[ \max(1, \text{pdu} \times m) \] when \textbf{jobu} = \texttt{Nag	extunderscore AllU} or \texttt{Nag	extunderscore InitU};
\[ 1 \] otherwise.

The \((i, j)\)th element of the matrix \textit{U} is stored in
\[ u[(j - 1) \times \text{pdu} + i - 1] \text{ when } \text{order} = \texttt{Nag	extunderscore ColMajor}; \]
\[ u[(i - 1) \times \text{pdu} + j - 1] \text{ when } \text{order} = \texttt{Nag	extunderscore RowMajor}. \]

On entry: if \textbf{jobu} = \texttt{Nag	extunderscore AllU}, \textbf{u} must contain an \( m \) by \( m \) matrix \( U_1 \) (usually the orthogonal matrix returned by nag_dggsvp (f08vec)).

On exit: if \textbf{jobu} = \texttt{Nag	extunderscore AllU}, \textbf{u} contains the product \( U_1U. \)

If \textbf{jobu} = \texttt{Nag	extunderscore InitU}, \textbf{u} contains the orthogonal matrix \( U. \)

If \textbf{jobu} = \texttt{Nag	extunderscore NotU}, \textbf{u} is not referenced.

19:  \textbf{pdu} – Integer  \textit{Input}

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

Constraints:
\[ \text{if } \text{jobu} = \texttt{Nag	extunderscore AllU} \text{ or } \texttt{Nag	extunderscore InitU}, \quad \text{pdu} \geq \max(1, m); \]
\[ \text{otherwise } \text{pdu} \geq 1. \]

20:  \textbf{v[dim]} – double  \textit{Input/Output}

\textbf{Note:} the dimension, \textit{dim}, of the array \textbf{v} must be at least
\[ \max(1, \text{pdv} \times p) \] when \textbf{jobv} = \texttt{Nag	extunderscore ComputeV} or \texttt{Nag	extunderscore InitV};
\[ 1 \] otherwise.

The \((i, j)\)th element of the matrix \textit{V} is stored in
\[ v[(j - 1) \times \text{pdv} + i - 1] \text{ when } \text{order} = \texttt{Nag	extunderscore ColMajor}; \]
\[ v[(i - 1) \times \text{pdv} + j - 1] \text{ when } \text{order} = \texttt{Nag	extunderscore RowMajor}. \]

On entry: if \textbf{jobv} = \texttt{Nag	extunderscore ComputeV}, \textbf{v} must contain an \( p \) by \( p \) matrix \( V_1 \) (usually the orthogonal matrix returned by nag_dggsvp (f08vec)).

On exit: if \textbf{jobv} = \texttt{Nag	extunderscore InitV}, \textbf{v} contains the orthogonal matrix \( V. \)

If \textbf{jobv} = \texttt{Nag	extunderscore ComputeV}, \textbf{v} contains the product \( V_1V. \)

If \textbf{jobv} = \texttt{Nag	extunderscore NotV}, \textbf{v} is not referenced.

21:  \textbf{pdv} – Integer  \textit{Input}

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

Mark 25
Constraints:
if \texttt{jobv} = \texttt{Nag\_ComputeV} or \texttt{Nag\_InitV}, \texttt{pdv} \geq \max(1, p); otherwise \texttt{pdv} \geq 1.

22: \texttt{q[dim]} – double 
\textbf{Input/Output}

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

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

\textbf{On entry:} if \texttt{jobq} = \texttt{Nag\_ComputeQ}, \texttt{q} must contain an \(n\) by \(n\) matrix \(Q_1\) (usually the orthogonal matrix returned by \texttt{nag\_dggsvp (f08vec)}).

\textbf{On exit:} if \texttt{jobq} = \texttt{Nag\_InitQ}, \texttt{q} contains the orthogonal matrix \(Q\).
If \texttt{jobq} = \texttt{Nag\_ComputeQ}, \texttt{q} contains the product \(Q_1Q\).
If \texttt{jobq} = \texttt{Nag\_NotQ}, \texttt{q} is not referenced.

23: \texttt{pdq} – Integer 
\textbf{Input}

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

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

24: \texttt{ncycle} – Integer * 
\textbf{Output}

\textbf{On exit:} the number of cycles required for convergence.

25: \texttt{fail} – \texttt{NagError} * 
\textbf{Input/Output}

\textbf{The NAG error argument (see Section 3.6 in the Essential Introduction)}.

6 \textbf{Error Indicators and Warnings}

\textbf{NE\_ALLOC\_FAIL}
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE\_BAD\_PARAM}
On entry, argument \texttt{(value)} had an illegal value.

\textbf{NE\_CONVERGENCE}
The procedure does not converge after 40 cycles.

\textbf{NE\_ENUM\_INT\_2}
On entry, \texttt{jobq} = \texttt{(value)}, \texttt{pdq} = \texttt{(value)} and \texttt{n} = \texttt{(value)}.
Constraint: if \texttt{jobq} = \texttt{Nag\_ComputeQ} or \texttt{Nag\_InitQ}, \texttt{pdq} \geq \max(1, n); otherwise \texttt{pdq} \geq 1.
On entry, \( \text{jobu} = \langle \text{value} \rangle, \ \text{pdu} = \langle \text{value} \rangle \) and \( \text{m} = \langle \text{value} \rangle \).
Constraint: if \( \text{jobu} = \text{Nag}_\text{AllU} \) or \( \text{Nag}_\text{InitU} \), \( \text{pdu} \geq \max(1, \text{m}) \); otherwise \( \text{pdu} \geq 1 \).

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

\begin{itemize}
  \item [**NE_INT**]
  On entry, \( \text{m} = \langle \text{value} \rangle \).
  Constraint: \( \text{m} \geq 0 \).

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

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

  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: \( \text{pdv} > 0 \).
\end{itemize}

\begin{itemize}
  \item [**NE_INT_2**]
  On entry, \( \text{pda} = \langle \text{value} \rangle \) and \( \text{m} = \langle \text{value} \rangle \).
  Constraint: \( \text{pda} \geq \max(1, \text{m}) \).

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

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

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

\begin{itemize}
  \item [**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.
\end{itemize}

\begin{itemize}
  \item [**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.
\end{itemize}
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 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_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

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 5 & 6 \\ 7 & 8 & 8 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -2 & -3 & 3 \\ -4 & 6 & 5 \end{pmatrix}.$$
\*\* 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]
#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_dtgsja (f08yec) Example Program Results\n\n");

/* Skip heading in data file */
#ifndef _WIN32
scanf_s("%*[\n");
#else
scanf("%*[\n");
#endif
#ifndef _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;
}
#ifndef _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);
#ifndef _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);
#ifndef _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);
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);
#ifndef NAG_COLUMN_MAJOR
f08 – Least-squares and Eigenvalue Problems (LAPACK)

Mark 25

f08yec

f08yec.9
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

nag_dge_norm(order, Nag_ProbeniusNorm, m, n, a, pda, &norma, &fail);
nag_dge_norm(order, Nag_ProbeniusNorm, p, n, b, pdb, &normb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_norm (f16rac).\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*(Q**T), B = V*D2*(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\n", k);
printf("Number of finite generalized singular values (l): %5\n", l);
printf("Effective Numerical rank of (A**T B**T)**T (k+l): %5\n", irank);
printf("\nNumber of cycles of the Kogbetliantz method: %12\n", ncycle);

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

/* Print the orthogonal matrices */
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",
"Non singular upper triangular matrix R", intlab, NULL, intlab, NULL, 80, 0, NULL, &fail);
    printf("\n");
}
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%\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