NAG Library Function Document

nag_zuncsd (f08rnc)

1 Purpose

nag_zuncsd (f08rnc) computes the CS decomposition of a complex $m$ by $m$ unitary matrix $X$, partitioned into a $2 \times 2$ array of submatrices.

2 Specification

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

void nag_zuncsd (Nag_OrderType order, Nag_ComputeUType jobu1,
    Nag_ComputeUType jobu2, Nag_ComputeVTType jobv1t,
    Nag_ComputeVTType jobv2t, Nag_SignsType signs, Integer m, Integer p,
    Integer q, Complex x11[], Integer pdx11, Complex x12[], Integer pdx12,
    Complex x21[], Integer pdx21, Complex x22[], Integer pdx22,
    double theta[], Complex u1[], Integer pdu1, Complex u2[], Integer pdu2,
    Complex v1t[], Integer pdv1t, Complex v2t[], Integer pdv2t,
    NagError *fail)
```

3 Description

The $m$ by $m$ unitary matrix $X$ is partitioned as

$$
X = \begin{pmatrix}
    X_{11} & X_{12} \\
    X_{21} & X_{22}
\end{pmatrix}
$$

where $X_{11}$ is a $p$ by $q$ submatrix and the dimensions of the other submatrices $X_{12}, X_{21}$ and $X_{22}$ are such that $X$ remains $m$ by $m$.

The CS decomposition of $X$ is $X = U \Sigma_p V^T$ where $U$, $V$ and $\Sigma_p$ are $m$ by $m$ matrices, such that

$$
U = \begin{pmatrix}
    U_1 & 0 \\
    0 & U_2
\end{pmatrix}
$$

is a unitary matrix containing the $p$ by $p$ unitary matrix $U_1$ and the $(m - p)$ by $(m - p)$ unitary matrix $U_2$;

$$
V = \begin{pmatrix}
    V_1 & 0 \\
    0 & V_2
\end{pmatrix}
$$

is a unitary matrix containing the $q$ by $q$ unitary matrix $V_1$ and the $(m - q)$ by $(m - q)$ unitary matrix $V_2$; and

$$
\Sigma_p = \begin{pmatrix}
    I_{11} & 0 & 0 & 0 \\
    C & 0 & 0 & -S \\
    0 & 0 & 0 & I_{12} \\
    0 & S & C & 0 \\
    0 & I_{21} & 0 & 0
\end{pmatrix}
$$

contains the $r$ by $r$ non-negative diagonal submatrices $C$ and $S$ satisfying $C^2 + S^2 = I$, where $r = \min(p, m - p, q, m - q)$ and the top left partition is $p$ by $q$.

The identity matrix $I_{11}$ is of order $\min(p, q) - r$ and vanishes if $\min(p, q) = r$. 

Mark 25

The identity matrix $I_{12}$ is of order $\min(p, m - q) - r$ and vanishes if $\min(p, m - q) = r$.
The identity matrix $I_{21}$ is of order $\min(m, p - q) - r$ and vanishes if $\min(m, p - q) = r$.
The identity matrix $I_{22}$ is of order $\min(m, p - m, q) - r$ and vanishes if $\min(m, p - m, q) = r$.
In each of the four cases $r = p, q, m - p, m - q$ at least two of the identity matrices vanish.
The indicated zeros represent augmentations by additional rows or columns (but not both) to the square diagonal matrices formed by $I_{ij}$ and $C$ or $S$.
$\Sigma_p$ does not need to be stored in full; it is sufficient to return only the values $\theta_i$ for $i = 1, 2, \ldots, r$ where $C_{ii} = \cos(\theta_i)$ and $S_{ii} = \sin(\theta_i)$.
The algorithm used to perform the complete CS decomposition is described fully in Sutton (2009) including discussions of the stability and accuracy of the algorithm.

4 References

5 Arguments
1: $\text{order} \quad \text{Nag\_OrderType}$
   $\text{Input}$
   $\text{On entry:}$ the $\text{order}$ 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 $\text{order} = \text{Nag\_RowMajor}$. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.
   $\text{Constraint:}$ $\text{order} = \text{Nag\_RowMajor}$ or $\text{Nag\_ColMajor}$.
2: $\text{jobu1} \quad \text{Nag\_ComputeUType}$
   $\text{Input}$
   $\text{On entry:}$ if $\text{jobu1} = \text{Nag\_AllU}$, $U_1$ is computed;
   if $\text{jobu1} = \text{Nag\_NotU}$, $U_1$ is not computed.
   $\text{Constraint:}$ $\text{jobu1} = \text{Nag\_AllU}$ or $\text{Nag\_NotU}$.
3: $\text{jobu2} \quad \text{Nag\_ComputeUType}$
   $\text{Input}$
   $\text{On entry:}$ if $\text{jobu2} = \text{Nag\_AllU}$, $U_2$ is computed;
   if $\text{jobu2} = \text{Nag\_NotU}$, $U_2$ is not computed.
   $\text{Constraint:}$ $\text{jobu2} = \text{Nag\_AllU}$ or $\text{Nag\_NotU}$.
4: $\text{jobvt1} \quad \text{Nag\_ComputeVTType}$
   $\text{Input}$
   $\text{On entry:}$ if $\text{jobvt1} = \text{Nag\_AllVT}$, $V_1^T$ is computed;
   if $\text{jobvt1} = \text{Nag\_NotVT}$, $V_1^T$ is not computed.
   $\text{Constraint:}$ $\text{jobvt1} = \text{Nag\_AllVT}$ or $\text{Nag\_NotVT}$.
5: \textbf{jobv2t} – Nag_ComputeVTType \hspace{1cm} \textit{Input}

\textit{On entry:}
- if \textbf{jobv2t} = Nag_AllVT, \( V_2^T \) is computed;
- if \textbf{jobv2t} = Nag_NotVT, \( V_2^T \) is not computed.

\textit{Constraint:} \textbf{jobv2t} = Nag_AllVT or Nag_NotVT.

6: \textbf{signs} – Nag_SignsType \hspace{1cm} \textit{Input}

\textit{On entry:}
- if \textbf{signs} = Nag_LowerMinus, the lower-left block is made nonpositive (the other convention);
- if \textbf{signs} = Nag_UpperMinus, the upper-right block is made nonpositive (the default convention).

\textit{Constraint:} \textbf{signs} = Nag_LowerMinus or Nag_UpperMinus.

7: \textbf{m} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \( m \), the number of rows and columns in the unitary matrix \( X \).

\textit{Constraint:} \( m \geq 0 \).

8: \textbf{p} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \( p \), the number of rows in \( X_{11} \) and \( X_{12} \).

\textit{Constraint:} \( 0 \leq p \leq m \).

9: \textbf{q} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \( q \), the number of columns in \( X_{11} \) and \( X_{21} \).

\textit{Constraint:} \( 0 \leq q \leq m \).

10: \textbf{x11[dim]} – Complex \hspace{1cm} \textit{Input/Output}

\textit{Note:} the dimension, \( \text{dim} \), of the array \textbf{x11} must be at least

\[
\max(1, \text{pdx11} \times p) \quad \text{when} \quad \text{order} = \text{Nag_RowMajor};
\]
\[
\max(1, \text{pdx11} \times q) \quad \text{when} \quad \text{order} = \text{Nag_ColMajor}.
\]

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

\[
x_{11}(j-1) \times \text{pdx11} + i - 1 \quad \text{when} \quad \text{order} = \text{Nag_ColMajor};
\]
\[
x_{11}(i-1) \times \text{pdx11} + j - 1 \quad \text{when} \quad \text{order} = \text{Nag_RowMajor}.
\]

\textit{On entry:} the upper left partition of the unitary matrix \( X \) whose CSD is desired.

\textit{On exit:} contains details of the unitary matrix used in a simultaneous bidiagonalization process.

11: \textbf{pdx11} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \textbf{order}) in the array \textbf{x11}.

\textit{Constraints:}
- if \textbf{order} = Nag_RowMajor, \( \text{pdx11} \geq \max(1, q) \);
- if \textbf{order} = Nag_ColMajor, \( \text{pdx11} \geq \max(1, p) \).
12: \( x12[\text{dim}] \) – Complex

**Note:** the dimension, \( \text{dim} \), of the array \( x12 \) must be at least
\[
\max(1, \text{pdx12} \times p) \text{ when order = } \text{Nag\_RowMajor};
\]
\[
\max(1, \text{pdx12} \times (m - q)) \text{ when order = } \text{Nag\_ColMajor}.
\]

The \((i,j)\)th element of the matrix is stored in
\[
x12[(j - 1) \times \text{pdx12} + i - 1] \text{ when order = } \text{Nag\_ColMajor};
\]
\[
x12[(i - 1) \times \text{pdx12} + j - 1] \text{ when order = } \text{Nag\_RowMajor}.
\]

*On entry:* the upper right partition of the unitary matrix \( X \) whose CSD is desired.

*On exit:* contains details of the unitary matrix used in a simultaneous bidiagonalization process.

13: \( \text{pdx12} \) – Integer

**Input**

*On entry:* the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( x12 \).

**Constraints:**
\[
\text{if order = Nag\_RowMajor, } \text{pdx12} \geq \max(1, m - q);
\]
\[
\text{if order = Nag\_ColMajor, } \text{pdx12} \geq \max(1, p).
\]

14: \( x21[\text{dim}] \) – Complex

**Input/Output**

**Note:** the dimension, \( \text{dim} \), of the array \( x21 \) must be at least
\[
\max(1, \text{pdx21} \times (m - p)) \text{ when order = } \text{Nag\_RowMajor};
\]
\[
\max(1, \text{pdx21} \times q) \text{ when order = } \text{Nag\_ColMajor}.
\]

The \((i,j)\)th element of the matrix is stored in
\[
x21[(j - 1) \times \text{pdx21} + i - 1] \text{ when order = } \text{Nag\_ColMajor};
\]
\[
x21[(i - 1) \times \text{pdx21} + j - 1] \text{ when order = } \text{Nag\_RowMajor}.
\]

*On entry:* the lower left partition of the unitary matrix \( X \) whose CSD is desired.

*On exit:* contains details of the unitary matrix used in a simultaneous bidiagonalization process.

15: \( \text{pdx21} \) – Integer

**Input**

*On entry:* the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( x21 \).

**Constraints:**
\[
\text{if order = Nag\_RowMajor, } \text{pdx21} \geq \max(1, q);
\]
\[
\text{if order = Nag\_ColMajor, } \text{pdx21} \geq \max(1, m - p).
\]

16: \( x22[\text{dim}] \) – Complex

**Input/Output**

**Note:** the dimension, \( \text{dim} \), of the array \( x22 \) must be at least
\[
\max(1, \text{pdx22} \times (m - p)) \text{ when order = } \text{Nag\_RowMajor};
\]
\[
\max(1, \text{pdx22} \times (m - q)) \text{ when order = } \text{Nag\_ColMajor}.
\]

The \((i,j)\)th element of the matrix is stored in
\[
x22[(j - 1) \times \text{pdx22} + i - 1] \text{ when order = } \text{Nag\_ColMajor};
\]
\[
x22[(i - 1) \times \text{pdx22} + j - 1] \text{ when order = } \text{Nag\_RowMajor}.
\]

*On entry:* the lower right partition of the unitary matrix \( X \) CSD is desired.

*On exit:* contains details of the unitary matrix used in a simultaneous bidiagonalization process.
17:  \texttt{pdx22} – Integer  \hspace{2cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) in the array \texttt{x22}.

\textit{Constraints:}

if \texttt{order} = Nag\_RowMajor, \texttt{pdx22} \geq \max(1, m - q);

if \texttt{order} = Nag\_ColMajor, \texttt{pdx22} \geq \max(1, m - p).

18:  \texttt{theta[dim]} – double  \hspace{2cm} \textit{Output}

\textit{Note:} the dimension, \texttt{dim}, of the array \texttt{theta} must be at least \min(p, m - p, q, m - q).

\textit{On exit:} the values \(\theta_i\) for \(i = 1, 2, \ldots, r\) where \(r = \min(p, m - p, q, m - q)\). The diagonal submatrices \(C\) and \(S\) of \(\Sigma_p\) are constructed from these values as

\[ C = \text{diag}(\cos(\texttt{theta}[0]), \ldots, \cos(\texttt{theta}[r - 1])) \]  
\[ S = \text{diag}(\sin(\texttt{theta}[0]), \ldots, \sin(\texttt{theta}[r - 1])). \]

19:  \texttt{u1[dim]} – Complex  \hspace{2cm} \textit{Output}

\textit{Note:} the dimension, \texttt{dim}, of the array \texttt{u1} must be at least

\[ \max(1, \texttt{pdu1} \times p) \] when \texttt{jobu1} = Nag\_AllU;

otherwise \texttt{u1} may be \texttt{NULL}.

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

\[ \texttt{u1}[(j - 1) \times \texttt{pdu1} + i - 1] \] when \texttt{order} = Nag\_ColMajor;

\[ \texttt{u1}[(i - 1) \times \texttt{pdu1} + j - 1] \] when \texttt{order} = Nag\_RowMajor.

\textit{On exit:} if \texttt{jobu1} = Nag\_AllU, \texttt{u1} contains the \(p\) by \(p\) unitary matrix \(U_1\).

20:  \texttt{pdu1} – Integer  \hspace{2cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) in the array \texttt{u1}.

\textit{Constraint:} if \texttt{jobu1} = Nag\_AllU, \texttt{pdu1} \geq \max(1, p)

21:  \texttt{u2[dim]} – Complex  \hspace{2cm} \textit{Output}

\textit{Note:} the dimension, \texttt{dim}, of the array \texttt{u2} must be at least

\[ \max(1, \texttt{pdu2} \times (m - p)) \] when \texttt{jobu2} = Nag\_AllU;

otherwise \texttt{u2} may be \texttt{NULL}.

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

\[ \texttt{u2}[(j - 1) \times \texttt{pdu2} + i - 1] \] when \texttt{order} = Nag\_ColMajor;

\[ \texttt{u2}[(i - 1) \times \texttt{pdu2} + j - 1] \] when \texttt{order} = Nag\_RowMajor.

\textit{On exit:} if \texttt{jobu2} = Nag\_AllU, \texttt{u2} contains the \(m - p\) by \(m - p\) unitary matrix \(U_2\).

22:  \texttt{pdu2} – Integer  \hspace{2cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) in the array \texttt{u2}.

\textit{Constraint:} if \texttt{jobu2} = Nag\_AllU, \texttt{pdu2} \geq \max(1, m - p)

23:  \texttt{v1t[dim]} – Complex  \hspace{2cm} \textit{Output}

\textit{Note:} the dimension, \texttt{dim}, of the array \texttt{v1t} must be at least

\[ \max(1, \texttt{pdv1t} \times q) \] when \texttt{jobv1t} = Nag\_AllVT;

otherwise \texttt{v1t} may be \texttt{NULL}.
The \((i,j)\)th element of the matrix is stored in
\[
\begin{align*}
\text{v1t}[(j-1) \times \text{pdv1t} + i - 1] & \quad \text{when order = Nag\_ColMajor;} \\
\text{v1t}[(i-1) \times \text{pdv1t} + j - 1] & \quad \text{when order = Nag\_RowMajor.}
\end{align*}
\]

On exit: if jobv1t = Nag\_AllVT, v1t contains the \(q\) by \(q\) unitary matrix \(V_1^H\).

24: \textbf{pdv1t} – Integer  
\textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) in the array \texttt{v1t}.

\textit{Constraint:} if jobv1t = Nag\_AllVT, pdv1t \(\geq \max(1, q)\)

25: \textbf{v2t}[\text{dim}] – Complex  
\textit{Output}

\textit{Note:} the dimension, \textit{dim}, of the array \textit{v2t} must be at least
\[
\max(1, \text{pdv2t} \times (m - q)) \quad \text{when jobv2t = Nag\_AllVT;}
\]
otherwise \textit{v2t} may be NULL.

The \((i,j)\)th element of the matrix is stored in
\[
\begin{align*}
\text{v2t}[(j-1) \times \text{pdv2t} + i - 1] & \quad \text{when order = Nag\_ColMajor;} \\
\text{v2t}[(i-1) \times \text{pdv2t} + j - 1] & \quad \text{when order = Nag\_RowMajor.}
\end{align*}
\]

On exit: if jobv2t = Nag\_AllVT, v2t contains the \(m - q\) by \(m - q\) unitary matrix \(V_2^H\).

26: \textbf{pdv2t} – Integer  
\textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) in the array \texttt{v2t}.

\textit{Constraint:} if jobv2t = Nag\_AllVT, pdv2t \(\geq \max(1, m - q)\)

27: \textbf{fail} – NagError *  
\textit{Input/Output}

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

6 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 \langle value \rangle had an illegal value.

\textbf{NE\_CONVERGENCE}

The Jacobi-type procedure failed to converge during an internal reduction to bidiagonal-block form. The process requires convergence to \(\min(p, m - p, q, m - q)\) values, the value of fail.errnum gives the number of converged values.

\textbf{NE\_ENUM\_INT\_2}

On entry, jobu1 = \langle value \rangle, pdu1 = \langle value \rangle and p = \langle value \rangle.

Constraint: if jobu1 = Nag\_AllU, pdu1 \(\geq \max(1, p)\).

On entry, jobu1 = \langle value \rangle, pdu1 = \langle value \rangle, p = \langle value \rangle.

Constraint: if jobu1 = Nag\_AllU, pdu1 \(\geq p\).

On entry, jobv1t = \langle value \rangle, pdv1t = \langle value \rangle and q = \langle value \rangle.

Constraint: if jobv1t = Nag\_AllVT, pdv1t \(\geq \max(1, q)\).

f08rnc.6  
Mark 25
On entry, jobv1t = (value), pdv1t = (value), q = (value).
Constraint: if jobv1t = Nag_AllVT, pdv1t \geq q.

**NE_ENUM_INT_3**

On entry, jobu2 = (value), pdu2 = (value), m = (value) and p = (value).
Constraint: if jobu2 = Nag_AllU, pdu2 \geq \max(1, m - p).

On entry, jobu2 = (value), pdu2 = (value), m = (value) and p = (value).
Constraint: if jobu2 = Nag_AllU, pdu2 \geq m - p.

On entry, jobv2t = (value), pdv2t = (value), m = (value) and q = (value).
Constraint: if jobv2t = Nag_AllVT, pdv2t \geq \max(1, m - q).

On entry, jobv2t = (value), pdv2t = (value), m = (value) and q = (value).
Constraint: if jobv2t = Nag_AllVT, pdv2t \geq m - q.

On entry, order = (value), pdx11 = (value), p = (value) and q = (value).
Constraint: if order = Nag_RowMajor, pdx11 \geq \max(1, p);
if order = Nag_ColMajor, pdx11 \geq \max(1, q).

On entry, order = (value), pdx11 = (value), p = (value) and q = (value).
Constraint: if order = Nag_RowMajor, pdx11 \geq \max(1, q);
if order = Nag_ColMajor, pdx11 \geq \max(1, p).

**NE_ENUM_INT_4**

On entry, order = (value), pdx12 = (value), m = (value), p = (value) and q = (value).
Constraint: if order = Nag_RowMajor, pdx12 \geq \max(1, m - q);
if order = Nag_ColMajor, pdx12 \geq \max(1, p).

On entry, order = (value), pdx12 = (value), m = (value), p = (value) and q = (value).
Constraint: if order = Nag_RowMajor, pdx12 \geq \max(1, p);
if order = Nag_ColMajor, pdx12 \geq \max(1, m - q).

On entry, order = (value), pdx21 = (value), m = (value), p = (value) and q = (value).
Constraint: if order = Nag_RowMajor, pdx21 \geq \max(1, m - p);
if order = Nag_ColMajor, pdx21 \geq \max(1, q).

On entry, order = (value), pdx21 = (value), m = (value), p = (value) and q = (value).
Constraint: if order = Nag_RowMajor, pdx21 \geq \max(1, q);
if order = Nag_ColMajor, pdx21 \geq \max(1, m - p).

On entry, order = (value), pdx22 = (value), m = (value), p = (value) and q = (value).
Constraint: if order = Nag_RowMajor, pdx22 \geq \max(1, m - p);
if order = Nag_ColMajor, pdx22 \geq \max(1, m - q).

On entry, order = (value), pdx22 = (value), m = (value), p = (value) and q = (value).
Constraint: if order = Nag_RowMajor, pdx22 \geq \max(1, m - q);
if order = Nag_ColMajor, pdx22 \geq \max(1, m - p).

**NE_INT**

On entry, m = (value).
Constraint: m \geq 0.

**NE_INT_2**

On entry, m = (value) and p = (value).
Constraint: 0 \leq p \leq m.

On entry, m = (value) and q = (value).
Constraint: 0 \leq q \leq m.
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 $CS$ decomposition is nearly the exact $CS$ decomposition for the nearby matrix $(X + E)$, where

$$\|E\|_2 = O(\epsilon),$$

and $\epsilon$ is the machine precision.

8 Parallelism and Performance
nag_zuncsd (f08rnc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zuncsd (f08rnc) 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 total number of floating-point operations required to perform the full $CS$ decomposition is approximately $2m^3$.

The real analogue of this function is nag_dorcsd (f08rac).

10 Example
This example finds the full $CS$ decomposition of a unitary 6 by 6 matrix $X$ (see Section 10.2) partitioned in 3 by 3 blocks.

The decomposition is performed both on submatrices of the unitary matrix $X$ and on separated partition matrices. Code is also provided to perform a recombining check if required.

10.1 Program Text
/* nag_zuncsd (f08rnc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 24, 2013. */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>

/* nag_zuncsd (f08rnc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 24, 2013. */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
```c
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer exit_status = 0;
    Integer pdx, pdu, pdv, pdx11, pdx12, pdx21, pdx22, pdw;
    Integer i, j, m, p, q, n11, n12, n21, n22, r;
    Integer recombine = 0, reprint = 0;
    Complex cone = {1.0, 0.0}, czero = {0.0, 0.0};
    /* Arrays */
    Complex *u = 0, *u1 = 0, *u2 = 0, *v = 0, *v1t = 0, *v2t = 0, *w = 0,
        *x = 0, *x11 = 0, *x12 = 0, *x21 = 0, *x22 = 0;
    double *theta = 0;
    /* Nag Types */
    Nag_OrderType order;
    NagError fail;
    #ifdef NAG_COLUMN_MAJOR
    #define X(I,J) x[(J-1)*pdx + I-1]
    #define U(I,J) u[(J-1)*pdu + I-1]
    #define V(I,J) v[(J-1)*pdv + I-1]
    #define W(I,J) w[(J-1)*pdw + I-1]
    #define X11(I,J) x11[(J-1)*pdx11 + I-1]
    #define X12(I,J) x12[(J-1)*pdx12 + I-1]
    #define X21(I,J) x21[(J-1)*pdx21 + I-1]
    #define X22(I,J) x22[(J-1)*pdx22 + I-1]
    order = Nag_ColMajor;
    #else
    #define X(I,J) x[(I-1)*pdx + J-1]
    #define U(I,J) u[(I-1)*pdu + J-1]
    #define V(I,J) v[(I-1)*pdv + J-1]
    #define W(I,J) w[(I-1)*pdw + J-1]
    #define X11(I,J) x11[(I-1)*pdx11 + J-1]
    #define X12(I,J) x12[(I-1)*pdx12 + J-1]
    #define X21(I,J) x21[(I-1)*pdx21 + J-1]
    #define X22(I,J) x22[(I-1)*pdx22 + J-1]
    order = Nag_RowMajor;
    #endif
    INIT_FAIL(fail);
    printf("nag_zuncsd (f08rnc) Example Program Results\n\n");
    fflush(stdout);
    /* 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, &p, &q);
    #else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\n] ", &m, &p, &q);
    #endif
    r = MIN(MIN(p,q),MIN(m-p,m-q));
    if (!((x = NAG_ALLOC(m*m, Complex))||
        (u = NAG_ALLOC(m*m, Complex))||
        (v = NAG_ALLOC(m*m, Complex))||
        (w = NAG_ALLOC(m*m, Complex))||
        (theta = NAG_ALLOC(r, double))||
        (x11 = NAG_ALLOC(p*q, Complex))||
        (x12 = NAG_ALLOC(p*(m-q), Complex))||
        (x21 = NAG_ALLOC((m-p)*q, Complex))||
        (x22 = NAG_ALLOC((m-p)*(m-q), Complex))||
        (ul = NAG_ALLOC(p*p, Complex))||
        // Add more conditions if necessary
    )
    else
        /* Add error handling code here */
        fail = Nag_Fail;
    // Continue with the program
```
(u2 = NAG_ALLOC((m-p)*(m-p), Complex))
(v1t = NAG_ALLOC(q*q, Complex))
(v2t = NAG_ALLOC((m-q)*(m-q), Complex))

{ printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}
pdx = m;
pdu = m;
pdv = m;
pdw = m;
#endif NAG_COLUMN_MAJOR
pdx11 = p;
pdx12 = p;
pdx21 = m-p;
pdx22 = m-p;
#else
pdx11 = q;
pdx12 = m-q;
pdx21 = q;
pdx22 = m-q;
#endif

/* Read (by column) and print orthogonal X from data file
  * (as, say, generated by a generalized singular value decomposition).
  */
  for ( i=1; i<=m; i++) {
    for (j=1;j<=m; j++)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf ) ", &X(j, i).re, &X(j, i).im );
#else
      scanf(" ( %lf , %lf ) ", &X(j, i).re, &X(j, i).im );
#endif
#ifdef _WIN32
      scanf_s("%*\[^
\] ");
#else
      scanf("%*\[^
\] ");
#endif
  }
/* Store partitions of X in separate matrices */
  for (j=1;j<=p; j++) {
    for (i=1;i<=q; i++)
      X11(j, i) = X(j, i);
    for (i=1;i<=m-q; i++)
      X12(j, i) = X(j, i + q);
  }
  for (j=1;j<=m-p; j++) {
    for (i=1;i<=q; i++)
      X21(j, i) = X(j + p, i);
    for (i=1;i<=m-q; i++)
      X22(j, i) = X(j + p, i + q);
  }
  /* nag_gen_complx_mat_print_comp (x04dbc).
     * Print least-squares solutions.
     */
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m,
                             m, x, pdx, Nag_BracketForm, "%7.4f",
                             "Unitary matrix X", Nag_IntegerLabels,
                             0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n\n", fail.message);
    exit_status = 1;
    goto END;
  }
  printf("\n");
  fflush(stdout);
  /* nag_zuncsd (f08rnc).
   * Compute the complete CS factorization of X:
   * X11 is stored in X(1:p, 1:q), X12 is stored in X(1:p, q+1:m)
   * X21 is stored in X(p+1:m, 1:q), X22 is stored in X(p+1:m, q+1:m)
   * U1 is stored in U(1:p, 1:p), U2 is stored in U(p+1:m, p+1:m)

f08rnc.10 Mark 25
NAG Library Manual
* V1 is stored in V(1:q, 1:q), V2 is stored in V(q+1:m, q+1:m) */

/* This is how you might pass partitions as sub-matrices */

nag_zuncsd(order, Nag_AllU, Nag_AllU, Nag_AllVT, Nag_AllVT, Nag_UpperMinus,
        m, p, q, x, pdx, &x(1,q+1), pdx, &x(p+1,1), pdx, &x(p+1,q+1), pdx,
        theta, u, pdu, &U(p+1,p+1), pdu, v, pdv, &V(q+1,q+1), pdv, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zuncsd (f08rnc).
        \n\s\n", fail.message);
    exit_status = 2;
    goto END;
}

/* Print Theta using matrix printing routine
* nag_gen_real_mat_print (x04cac).
* Note: U1, U2, V1T, V2T not printed since these may differ by a sign
* change in columns of U1, U2 and corresponding rows of V1T, V2T. */

printf(" Component of CS factorization of X:\n\n\n");
fflush(stdout);
nag_gen_real_mat_print(Nag_ColMajor, Nag_GeneralMatrix, Nag_NonUnitDiag, r,
        1, theta, r, " Theta", 0, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).
        \n\s\n", fail.message);
    exit_status = 3;
    goto END;
}

printf("\n");
fflush(stdout);

/* And this is how you might pass partitions as separate matrices. */

nag_zuncsd(order, Nag_AllU, Nag_AllU, Nag_AllVT, Nag_AllVT, Nag_UpperMinus,
        m, p, q, x11, pdx11, x12, pdx12, x21, pdx21, x22, pdx22, theta,
        u1, p, u2, m-p, v1t, q, v2t, m-q, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zuncsd (f08rnc).
        \n\s\n", fail.message);
    exit_status = 4;
    goto END;
}

if (reprint != 0) {
    printf("Component of CS factorization of X using separate matrices:\n\n");
    fflush(stdout);
nag_gen_real_mat_print(Nag_ColMajor, Nag_GeneralMatrix, Nag_NonUnitDiag, r,
        1, theta, r, " Theta", 0, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_gen_real_mat_print (x04cac).
            \n\s\n", fail.message);
        exit_status = 5;
        goto END;
    }
    printf("\n");
    fflush(stdout);
}

if (recombine != 0) {
    /* Recombining should return the original matrix.
     * Assemble Sigma_p into X */
    for (i=1; i<=m; i++) {
        for (j=1; j<=m; j++) {
            X(i,j) = czero;
        }
    }
n11 = MIN(p,q)-r;
n12 = MIN(p,m-q)-r;
n21 = MIN(m-p,q)-r;
n22 = MIN(m-p,m-q)-r;
    /* top half */
    for (j=1; j<=n11; j++) {
        X(j,j) = cone;
    }
}
for (j=1; j<=r; j++) {
    X(j+n11,j+n11).re = cos(theta[j-1]);
    X(j+n11,j+n11).im = 0.0;
    X(j+n11,j+n11+r+n21+n22).re = -sin(theta[j-1]);
    X(j+n11,j+n11+r+n21+n22).im = 0.0;
}
for (j=1; j<=n12; j++) {
    X(j+n11+r,j+n11+r+n21+n22+r).re = -1.0;
    X(j+n11+r,j+n11+r+n21+n22+r).im = 0.0;
} /* bottom half */
for (j=1; j<=n22; j++) {
    X(p+j,q+j) = cone;
}
for (j=1; j<=r; j++) {
    X(p+n22+j,j+n11).re = sin(theta[j-1]);
    X(p+n22+j,j+n11).im = 0.0;
    X(p+n22+j,j+r+n21+n22).re = cos(theta[j-1]);
    X(p+n22+j,j+r+n21+n22).im = 0.0;
}
for (j=1; j<=n21; j++) {
    X(p+n22+r+j,n11+r+j) = cone;
}
/* multiply U * Sigma_p into w */
  nag_zgemm(order, Nag_NoTrans, Nag_NoTrans, m, m, m, cone,
    &U(1,1), pdu, &X(1,1), pdx, czero, &W(1,1), pdw, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgemm (f16zac).\n\n", fail.message);
    exit_status = 6;
    goto END;
  }
  /* form U * Sigma_p * V^T into u */
  nag_zgemm(order, Nag_NoTrans, Nag_NoTrans, m, m, m, cone,
    &W(1,1), pdw, &V(1,1), pdv, czero, &U(1,1), pdu, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgemm (f16zac).\n\n", fail.message);
    exit_status = 7;
    goto END;
  }
  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print least-squares solutions.
   */
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
    m, m, &U(1,1), pdu, Nag_BracketForm, "%7.4f",
    " U * Sigma_p * V^T", Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n\n", fail.message);
    exit_status = 8;
    goto END;
  }
  printf("\n");
  fflush(stdout);
}

END:
NAG_FREE(x);
NAG_FREE(u);
NAG_FREE(v);
NAG_FREE(w);
NAG_FREE(theta);
NAG_FREE(x11);
NAG_FREE(x12);
NAG_FREE(x21);
NAG_FREE(x22);
NAG_FREE(u1);
NAG_FREE(u2);
NAG_FREE(v1t);
NAG_FREE(v2t);
return exit_status;
}

10.2 Program Data

nag_zuncsd (f08rnc) Example Program Data

6 2 3 : m, p, q

( -1.3038e-02, -3.2595e-01)
( 4.2764e-01, -6.2582e-01)
( -3.2595e-01, 1.6428e-01)
( 1.5906e-01, -5.2151e-03)
( -1.7210e-01, -1.3038e-02)
( -2.6336e-01, -2.4772e-01) : column 1 of unitary matrix X

( -1.4039e-01, -2.6167e-01)
( 8.6298e-02, -3.8174e-02)
( 3.8163e-01, -1.8219e-01)
( -2.8207e-01, 1.9732e-01)
( -5.0942e-01, -5.0319e-01)
( -1.0861e-01, 2.8474e-01) : column 2 of unitary matrix X

( 2.5177e-01, -7.9789e-01)
( -3.2188e-01, 1.6112e-01)
( 1.3231e-01, -1.4565e-02)
( 2.1598e-01, 1.8813e-01)
( 3.6488e-02, 2.0316e-01)
( 1.0906e-01, -1.2712e-01) : column 3 of unitary matrix X

( -5.0956e-02, -2.1750e-01)
( 1.1979e-01, 1.6319e-01)
( -5.0671e-01, 1.8615e-01)
( -4.0163e-01, 2.6787e-01)
( 1.9271e-01, 1.5574e-01)
( -8.8159e-02, 5.6169e-01) : column 4 of unitary matrix X

( -4.5947e-02, 1.4052e-04)
( -8.0311e-02, -4.3605e-01)
( 5.9714e-02, -5.8974e-01)
( -4.6443e-02, 3.0864e-01)
( 5.7843e-01, -1.2439e-01)
( 1.5763e-02, 4.7130e-02) : column 5 of unitary matrix X

( -5.2773e-02, -2.2492e-01)
( -3.8117e-02, -2.1907e-01)
( -1.3850e-01, -9.0941e-02)
( -3.7354e-01, -5.5148e-01)
( -1.8815e-02, -5.5686e-02)
( 6.5007e-01, 4.9173e-03) : column 6 of unitary matrix X

10.3 Program Results

nag_zuncsd (f08rnc) Example Program Results

Unitary matrix X

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-0.0130, -0.3260)</td>
<td>(-0.1404, -0.2617)</td>
<td>(0.2518, -0.7979)</td>
<td>(-0.0510, -0.2175)</td>
</tr>
<tr>
<td>2</td>
<td>(0.4276, -0.6258)</td>
<td>(0.0863, -0.0382)</td>
<td>(-0.3219, 0.1611)</td>
<td>(0.1198, 0.1632)</td>
</tr>
<tr>
<td>3</td>
<td>(-0.3260, 0.1643)</td>
<td>(0.3816, -0.1822)</td>
<td>(0.1323, -0.0146)</td>
<td>(0.5067, 0.1862)</td>
</tr>
<tr>
<td>4</td>
<td>(0.1591, -0.0052)</td>
<td>(-0.2821, 0.1973)</td>
<td>(0.2160, 0.1881)</td>
<td>(-0.4016, 0.2679)</td>
</tr>
<tr>
<td>5</td>
<td>(0.1721, 0.0130)</td>
<td>(-0.5094, -0.5032)</td>
<td>(0.0365, 0.2032)</td>
<td>(0.1927, 0.1557)</td>
</tr>
<tr>
<td>6</td>
<td>(-0.2634, -0.2477)</td>
<td>(-0.1086, 0.2847)</td>
<td>(0.1091, -0.1271)</td>
<td>(-0.0882, 0.5617)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-0.0459, 0.0001)</td>
<td>(-0.0528, -0.2249)</td>
</tr>
<tr>
<td>2</td>
<td>(-0.0803, -0.4360)</td>
<td>(-0.0381, -0.2191)</td>
</tr>
</tbody>
</table>
Component of CS factorization of X:

<table>
<thead>
<tr>
<th>Theta</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.3146</td>
<td>0.5760</td>
</tr>
</tbody>
</table>