NAG Library Function Document

nag_dorcsd (f08rac)

1 Purpose

nag_dorcsd (f08rac) computes the CS decomposition of a real $m$ by $m$ orthogonal matrix $X$, partitioned into a 2 by 2 array of submatrices.

2 Specification

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

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

3 Description

The $m$ by $m$ orthogonal 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 an orthogonal matrix containing the $p$ by $p$ orthogonal matrix $U_1$ and the $(m-p)$ by $(m-p)$ orthogonal matrix $U_2$;

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

is an orthogonal matrix containing the $q$ by $q$ orthogonal matrix $V_1$ and the $(m-q)$ by $(m-q)$ orthogonal matrix $V_2$; and

$$\Sigma_p = \begin{pmatrix} I_{11} & 0 & 0 & 0 \\ C & 0 & 0 & -S \\ 0 & 0 & 0 & 0 \\ 0 & S & C & 0 \\ 0 & I_{21} & 0 & 0 \\ I_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{12} \end{pmatrix}$$

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 $\min(p, q) = r$. 
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.

### 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: **jobu1** – Nag_ComputeUType

   *Input*

   *On entry:*

   - if jobu1 = Nag_AllU, $U_1$ is computed;
   - if jobu1 = Nag_NotU, $U_1$ is not computed.

   *Constraint:* jobu1 = Nag_AllU or Nag_NotU.

3: **jobu2** – Nag_ComputeUType

   *Input*

   *On entry:*

   - if jobu2 = Nag_AllU, $U_2$ is computed;
   - if jobu2 = Nag_NotU, $U_2$ is not computed.

   *Constraint:* jobu2 = Nag_AllU or Nag_NotU.

4: **jobv1t** – Nag_ComputeVTTType

   *Input*

   *On entry:*

   - if jobv1t = Nag_AllVT, $V_1^T$ is computed;
   - if jobv1t = Nag_NotVT, $V_1^T$ is not computed.

   *Constraint:* jobv1t = Nag_AllVT or Nag_NotVT.
5: \textbf{jobv2t} – Nag_ComputeVTType  
\textit{Input}

\textit{On entry:}

\begin{itemize}
  \item if \texttt{jobv2t} = Nag_AllVT, \( V_2^T \) is computed;
  \item if \texttt{jobv2t} = Nag_NotVT, \( V_2^T \) is not computed.
\end{itemize}

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

6: \textbf{signs} – Nag_SignsType  
\textit{Input}

\textit{On entry:}

\begin{itemize}
  \item if \texttt{signs} = Nag_LowerMinus, the lower-left block is made nonpositive (the other convention);
  \item if \texttt{signs} = Nag_UpperMinus, the upper-right block is made nonpositive (the default convention).
\end{itemize}

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

7: \textbf{m} – Integer  
\textit{Input}

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

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

8: \textbf{p} – Integer  
\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  
\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[\textit{dim}]} – double  
\textit{Input/Output}

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

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

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

\begin{itemize}
  \item \textbf{x11}[(j - 1) \times \text{pdx11} + i - 1] \text{ when } \text{order} = \text{Nag_ColMajor};
  \item \textbf{x11}[(i - 1) \times \text{pdx11} + j - 1] \text{ when } \text{order} = \text{Nag_RowMajor}.
\end{itemize}

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

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

11: \textbf{pdx11} – Integer  
\textit{Input}

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

\textit{Constraints:}

\begin{itemize}
  \item if \texttt{order} = Nag_RowMajor, \( \text{pdx11} \geq \max(1, q) \);
  \item if \texttt{order} = Nag_ColMajor, \( \text{pdx11} \geq \max(1, p) \).
\end{itemize}
12: \( \mathbf{x12}[\text{dim}] \) – double

**Input/Output**

Note: the dimension, \( \text{dim} \), of the array \( \mathbf{x12} \) must be at least

\[
\max(1, \mathbf{pdx12} \times p) \text{ when } \text{order} = \text{Nag\_RowMajor}; \\
\max(1, \mathbf{pdx12} \times (m - q)) \text{ when } \text{order} = \text{Nag\_ColMajor}.
\]

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

\[
\begin{align*}
\mathbf{x12}(j - 1) & \times \mathbf{pdx12} + i - 1 \text{ when } \text{order} = \text{Nag\_ColMajor}; \\
\mathbf{x12}(i - 1) & \times \mathbf{pdx12} + j - 1 \text{ when } \text{order} = \text{Nag\_RowMajor}.
\end{align*}
\]

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

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

13: \( \mathbf{pdx12} \) – Integer

**Input**

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

Constraints:

\[
\begin{align*}
\text{if } \text{order} = \text{Nag\_RowMajor}, & \quad \mathbf{pdx12} \geq \max(1, m - q); \\
\text{if } \text{order} = \text{Nag\_ColMajor}, & \quad \mathbf{pdx12} \geq \max(1, p).
\end{align*}
\]

14: \( \mathbf{x21}[\text{dim}] \) – double

**Input/Output**

Note: the dimension, \( \text{dim} \), of the array \( \mathbf{x21} \) must be at least

\[
\max(1, \mathbf{pdx21} \times (m - p)) \text{ when } \text{order} = \text{Nag\_RowMajor}; \\
\max(1, \mathbf{pdx21} \times q) \text{ when } \text{order} = \text{Nag\_ColMajor}.
\]

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

\[
\begin{align*}
\mathbf{x21}(j - 1) & \times \mathbf{pdx21} + i - 1 \text{ when } \text{order} = \text{Nag\_ColMajor}; \\
\mathbf{x21}(i - 1) & \times \mathbf{pdx21} + j - 1 \text{ when } \text{order} = \text{Nag\_RowMajor}.
\end{align*}
\]

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

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

15: \( \mathbf{pdx21} \) – Integer

**Input**

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

Constraints:

\[
\begin{align*}
\text{if } \text{order} = \text{Nag\_RowMajor}, & \quad \mathbf{pdx21} \geq \max(1, q); \\
\text{if } \text{order} = \text{Nag\_ColMajor}, & \quad \mathbf{pdx21} \geq \max(1, m - p).
\end{align*}
\]

16: \( \mathbf{x22}[\text{dim}] \) – double

**Input/Output**

Note: the dimension, \( \text{dim} \), of the array \( \mathbf{x22} \) must be at least

\[
\max(1, \mathbf{pdx22} \times (m - p)) \text{ when } \text{order} = \text{Nag\_RowMajor}; \\
\max(1, \mathbf{pdx22} \times (m - q)) \text{ when } \text{order} = \text{Nag\_ColMajor}.
\]

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

\[
\begin{align*}
\mathbf{x22}(j - 1) & \times \mathbf{pdx22} + i - 1 \text{ when } \text{order} = \text{Nag\_ColMajor}; \\
\mathbf{x22}(i - 1) & \times \mathbf{pdx22} + j - 1 \text{ when } \text{order} = \text{Nag\_RowMajor}.
\end{align*}
\]

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

On exit: contains details of the orthogonal matrix used in a simultaneous bidiagonalization process.
17: **pdx22** – Integer  
*Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **x22**.

*Constraints:*  
if **order** = Nag_RowMajor, **pdx22** ≥ max(1, m – q);  
if **order** = Nag-ColMajor, **pdx22** ≥ max(1, m – p).

18: **theta[**dim]**] – double  
*Output*  
*Note:* the dimension, **dim**, of the array **theta** must be at least min(p, m – p, q, m – q).

*On exit:* the values θi for i = 1, 2, ..., r where r = min(p, m – p, q, m – q). The diagonal submatrices C and S of Σp are constructed from these values as  
C = diag(cos(theta[0]), ..., cos(theta[r - 1])) and  
S = diag(sin(theta[0]), ..., sin(theta[r - 1])).

19: **u1[**dim]**] – double  
*Output*  
*Note:* the dimension, **dim**, of the array **u1** must be at least max(1, pdv1t × q) when **jobv1t** = Nag_AllVT; otherwise **u1** may be NULL.

The (i, j)th element of the matrix is stored in  
\[ u1[(j - 1) \times pdv1t + i - 1] \] when **order** = Nag_ColMajor;  
\[ u1[(i - 1) \times pdv1t + j - 1] \] when **order** = Nag_RowMajor.

*On exit:* if **jobu1** = Nag_AllU, **u1** contains the p by p orthogonal matrix U.

20: **pdu1** – Integer  
*Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **u1**.

*Constraint:* if **jobu1** = Nag_AllU, **pdu1** ≥ max(1, p).

21: **u2[**dim]**] – double  
*Output*  
*Note:* the dimension, **dim**, of the array **u2** must be at least max(1, pdv2 × (m – p)) when **jobu2** = Nag_AllU; otherwise **u2** may be NULL.

The (i, j)th element of the matrix is stored in  
\[ u2[(j - 1) \times pdv2 + i - 1] \] when **order** = Nag_ColMajor;  
\[ u2[(i - 1) \times pdv2 + j - 1] \] when **order** = Nag_RowMajor.

*On exit:* if **jobu2** = Nag_AllU, **u2** contains the m – p by m – p orthogonal matrix U.

22: **pdu2** – Integer  
*Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **u2**.

*Constraint:* if **jobu2** = Nag_AllU, **pdu2** ≥ max(1, m – p).

23: **v1t[**dim]**] – double  
*Output*  
*Note:* the dimension, **dim**, of the array **v1t** must be at least max(1, pdv1t × q) when **jobv1t** = Nag_AllVT; otherwise **v1t** may be NULL.
The \((i,j)\)th element of the matrix is stored in
\[ v_{1t}(j-1) \times pdv_{1t} + i - 1 \] when \(\text{order} = \text{Nag\_ColMajor} \);
\[ v_{1t}(i-1) \times pdv_{1t} + j - 1 \] when \(\text{order} = \text{Nag\_RowMajor} \).

On exit: if \(\text{jobv}_{1t} = \text{Nag\_AllVT} \), \(v_{1t}\) contains the \(q\) by \(q\) orthogonal matrix \(V_1^T\).

24: \(pdv_{1t}\) – Integer
\[ \text{Input} \]
\[ \text{On entry:} \text{the stride separating row or column elements (depending on the value of } \text{order} \text{) in the array } v_{1t}. \]
\[ \text{Constraint: if } \text{jobv}_{1t} = \text{Nag\_AllVT}, pdv_{1t} \geq \max(1, q) \]

25: \(v_{2t}[\text{dim}]\) – double
\[ \text{Output} \]
\[ \text{Note: the dimension, } \text{dim}, \text{ of the array } v_{2t} \text{ must be at least} \]
\[ \max(1, pdv_{2t} \times (m - q)) \text{ when } \text{jobv}_{2t} = \text{Nag\_AllVT}; \]
\[ \text{otherwise } v_{2t} \text{ may be NULL.} \]

The \((i,j)\)th element of the matrix is stored in
\[ v_{2t}(j-1) \times pdv_{2t} + i - 1 \] when \(\text{order} = \text{Nag\_ColMajor} \);
\[ v_{2t}(i-1) \times pdv_{2t} + j - 1 \] when \(\text{order} = \text{Nag\_RowMajor} \).

On exit: if \(\text{jobv}_{2t} = \text{Nag\_AllVT} \), \(v_{2t}\) contains the \(m - q\) by \(m - q\) orthogonal matrix \(V_2^T\).

26: \(pdv_{2t}\) – Integer
\[ \text{Input} \]
\[ \text{On entry:} \text{the stride separating row or column elements (depending on the value of } \text{order} \text{) in the array } v_{2t}. \]
\[ \text{Constraint: if } \text{jobv}_{2t} = \text{Nag\_AllVT}, pdv_{2t} \geq \max(1, m - q) \]

27: \(\text{fail} = \text{Nag\_Error\_*} \)
\[ \text{Input/Output} \]
\[ \text{The NAG error argument (see Section 3.6 in the Essential Introduction).} \]

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 \(\langle\text{value}\rangle\) 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 \(\text{fail.errnum}\) gives the number of converged values.

\textbf{NE\_ENUM\_INT\_2}

On entry, \(\text{jobu} = \langle\text{value}\rangle\), \(pdu = \langle\text{value}\rangle\) and \(p = \langle\text{value}\rangle\).
Constraint: if \(\text{jobu} = \text{Nag\_AllU}, pdu \geq \max(1, p)\).

On entry, \(\text{jobu} = \langle\text{value}\rangle\), \(pdu = \langle\text{value}\rangle\), \(p = \langle\text{value}\rangle\).
Constraint: if \(\text{jobu} = \text{Nag\_AllU}, pdu \geq p\).

On entry, \(\text{jobv}_{1t} = \langle\text{value}\rangle\), \(pdv_{1t} = \langle\text{value}\rangle\) and \(q = \langle\text{value}\rangle\).
Constraint: if \(\text{jobv}_{1t} = \text{Nag\_AllVT}, pdv_{1t} \geq \max(1, q)\).
On entry, jobv1t = ⟨value⟩, pdv1t = ⟨value⟩, q = ⟨value⟩.
Constraint: if jobv1t = Nag_AllVT, pdv1t ≥ q.

NE_ENUM_INT_3

On entry, jobu2 = ⟨value⟩, pdu2 = ⟨value⟩, m = ⟨value⟩ and p = ⟨value⟩.
Constraint: if jobu2 = Nag_AllU, pdu2 ≥ max(1, m − p).
On entry, jobu2 = ⟨value⟩, pdu2 = ⟨value⟩, m = ⟨value⟩ and p = ⟨value⟩.
Constraint: if jobu2 = Nag_AllU, pdu2 ≥ m − p.
On entry, jobv2t = ⟨value⟩, pdv2t = ⟨value⟩, m = ⟨value⟩ and q = ⟨value⟩.
Constraint: if jobv2t = Nag_AllVT, pdv2t ≥ max(1, m − q).
On entry, jobv2t = ⟨value⟩, pdv2t = ⟨value⟩, m = ⟨value⟩ and q = ⟨value⟩.
Constraint: if jobv2t = Nag_AllVT, pdv2t ≥ m − q.
On entry, order = ⟨value⟩, pdx11 = ⟨value⟩, p = ⟨value⟩ and q = ⟨value⟩.
Constraint: if order = Nag_RowMajor, pdx11 ≥ max(1, p);
if order = Nag_ColMajor, pdx11 ≥ max(1, q).
On entry, order = ⟨value⟩, pdx11 = ⟨value⟩, p = ⟨value⟩ and q = ⟨value⟩.
Constraint: if order = Nag_RowMajor, pdx11 ≥ max(1, q);
if order = Nag_ColMajor, pdx11 ≥ 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 ≥ max(1, m − q);
if order = Nag_ColMajor, pdx12 ≥ max(1, p).
On entry, order = ⟨value⟩, pdx12 = ⟨value⟩, m = ⟨value⟩, p = ⟨value⟩ and q = ⟨value⟩.
Constraint: if order = Nag_RowMajor, pdx12 ≥ max(1, p);
if order = Nag_ColMajor, pdx12 ≥ max(1, m − q).
On entry, order = ⟨value⟩, pdx21 = ⟨value⟩, m = ⟨value⟩, p = ⟨value⟩ and q = ⟨value⟩.
Constraint: if order = Nag_RowMajor, pdx21 ≥ max(1, m − p);
if order = Nag_ColMajor, pdx21 ≥ max(1, q).
On entry, order = ⟨value⟩, pdx21 = ⟨value⟩, m = ⟨value⟩, p = ⟨value⟩ and q = ⟨value⟩.
Constraint: if order = Nag_RowMajor, pdx21 ≥ max(1, q);
if order = Nag_ColMajor, pdx21 ≥ max(1, m − p).
On entry, order = ⟨value⟩, pdx22 = ⟨value⟩, m = ⟨value⟩, p = ⟨value⟩ and q = ⟨value⟩.
Constraint: if order = Nag_RowMajor, pdx22 ≥ max(1, m − p);
if order = Nag_ColMajor, pdx22 ≥ max(1, m − q).
On entry, order = ⟨value⟩, pdx22 = ⟨value⟩, m = ⟨value⟩, p = ⟨value⟩ and q = ⟨value⟩.
Constraint: if order = Nag_RowMajor, pdx22 ≥ max(1, m − q);
if order = Nag_ColMajor, pdx22 ≥ max(1, m − p).

NE_INT

On entry, m = ⟨value⟩.
Constraint: m ≥ 0.

NE_INT_2

On entry, m = ⟨value⟩ and p = ⟨value⟩.
Constraint: 0 ≤ p ≤ m.
On entry, m = ⟨value⟩ and q = ⟨value⟩.
Constraint: 0 ≤ q ≤ m.
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 \textit{machine precision}.

8 Parallelism and Performance

\texttt{nag_dorcsd} (f08rac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\texttt{nag_dorcsd} (f08rac) 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 complex analogue of this function is \texttt{nag_zuncsd} (f08rnc).

10 Example

This example finds the full $CS$ decomposition of

$$
X = \begin{pmatrix}
-0.13484 & 0.52524 & -0.20924 & 0.81373 \\
0.67420 & -0.52213 & -0.38886 & 0.34874 \\
0.26968 & 0.52757 & -0.65782 & -0.46499 \\
0.67420 & 0.41615 & 0.61014 & 0.00000
\end{pmatrix}
$$

partitioned in 2 by 2 blocks.

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

/* nag_dorcsd (f08rac) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 24, 2013.
*/

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#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, pdu1, pdu2, pdv1t;
    Integer pdv2t, pdw;
    Integer i, j, m, p, q, n11, n12, n21, n22, r;
    Integer recombine = 1, reprint = 0;
    double alpha, beta;
    /* Arrays */
    double *theta = 0, *u = 0, *u1 = 0, *u2 = 0, *v = 0, *v1t = 0, *w = 0,
    *v2t = 0, *x = 0, *x1l = 0, *x12 = 0, *x2l = 0, *x22 = 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_dorcsd (f08rac) Example Program Results\n\n");
    fflush(stdout);

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

    /* Scan the data file */
    #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, double))||
    !(u = NAG_ALLOC(m*m, double))||
    !(v = NAG_ALLOC(m*m, double))||
    !(w = NAG_ALLOC(m*m, double))||
    !(theta = NAG_ALLOC(r, double))||
    !(x11 = NAG_ALLOC(p*q, double))||
    !(x12 = NAG_ALLOC(p*(m-q), double))||
    !(x21 = NAG_ALLOC((m-p)*q, double))||
    !(x22 = NAG_ALLOC((m-p)*(m-q), double))||
    !(u1 = NAG_ALLOC(p*p, double))||
    !(u2 = NAG_ALLOC((m-p)*(m-p), double))||
    !(v1t = NAG_ALLOC(q*q, double))||
    !(v2t = NAG_ALLOC((m-q)*(m-q), double))))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

pdx = m; pdu = m; pdv = m; pdw = m;

define NAG_COLUMN_MAJOR
    pdx11 = p; pdx12 = p; pdx21 = m-p; pdx22 = m-p;
define NAG_COLUMN_MAJOR
    else
    pdx11 = q; pdx12 = m-q; pdx21 = q; pdx22 = m-q;
define NAG_COLUMN_MAJOR
    endif
    /* Read 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++)
define _WIN32
    scanf_s("%lf", &X(i, j));
define _WIN32
    else
    scanf("%lf", &X(i, j));
define _WIN32
    endif
    /* nag_gen_real_mat_print (x04cac).
    */
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, m,
            &X(1,1), pdx, " Orthogonal matrix X", 0, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_gen_real_mat_print (x04cac).\n");
        exit_status = 1;
        goto END;
    }
    printf("\n");
    fflush(stdout);
    /* nag_dorcsd (f08rac).
    * 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)
    * V1 is stored in V(1:q, 1:q), V2 is stored in V(q+1:m, q+1:m)
    */
    for (j=1;j<=p; j++) {
        for (i=1;i<=m; 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<=m; i++) { X21(j, i) = X(j + p, i);
            for (i=1;i<=m-q; i++) X22(j, i) = X(j + p, i + q);
        }
    }
    for ( i=1; i<=m; i++)
    for (j=1;j<=m; j++) {
    U(i,j) = 0.0;
    V(i,j) = 0.0;
}
/* This is how you might pass partitions as sub-matrices */
nag_dorcsd(order, Nag_AllU, Nag_AllU, Nag_AllVT, Nag_AllVT, Nag_UpperMinus,
m, p, q, &X(1,1), pdx, &X(l, q+1), pdx, &X(p+1, l), pdx, &X(p+1, q+1),
pdx, theta, &U(1,1), pdu, &U(p+1, p+1), pdu, &V(1,1), pdv,
&V(q+1, q+1), pdv, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dorcsd (f08rac).");
    exit_status = 2;
    goto END;
}
/* Print Theta, U1, U2, V1T, V2T
 * using matrix printing routine nag_gen_real_mat_print (x04cac).
 */
printf("Components of CS factorization of X:\n");
fflush(stdout);
nag_gen_real_mat_print(Nag_ColMajor, Nag_GeneralMatrix, Nag_NonUnitDiag,
r, 1, theta, r, " Theta", 0, &fail);
printf("\n");
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
p, p, &U(1,1), pdu, " U1", 0, &fail);
printf("\n");
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
m-p, m-p, &U(p+1, p+1), pdu, " U2", 0, &fail);
printf("\n");
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
q, q, &V(1,1), pdv, " V1T", 0, &fail);
printf("\n");
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
m-q, m-q, &V(q+1, q+1), pdv, " V2T", 0, &fail);
printf("\n");
fflush(stdout);
/* And this is how you might pass partitions as separate matrices. */
nag_dorcsd(order, Nag_AllU, Nag_AllU, Nag_AllVT, Nag_AllVT, Nag_UpperMinus,
m, p, q,
x11, pdx11, x12, pdx12, x21, pdx21, x22, pdx22, theta,
u1, pdu1, u2, pdu2, v1t, pdv1t, v2t, pdv2t, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error second from nag_dorcsd (f08rac).");
    exit_status = 3;
    goto END;
}
/* Print Theta, U1, U2, V1T, V2T
 * using matrix printing routine nag_gen_real_mat_print (x04cac).
 */
if (reprint != 0) {
    printf("Components of CS factorization of X:\n");
    nag_gen_real_mat_print(Nag_ColMajor, Nag_GeneralMatrix, Nag_NonUnitDiag,
r, 1, theta, r, " Theta", 0, &fail);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
p, p, &U(1,1), pdu, " U1", 0, &fail);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
m-p, m-p, &U(p+1, p+1), pdu, " U2", 0, &fail);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
q, q, &V(1,1), pdv, " V1T", 0, &fail);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
m-q, m-q, &V(q+1, q+1), pdv, " V2T", 0, &fail);
}
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) = 0.0;
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) = 1.0;
for (j=1; j<r; j++) {
    X(j+n11,j+n11) = cos(theta[j-1]);
    X(j+n11,j+n11+r+n21+n22) = -sin(theta[j-1]);
}
for (j=1; j<=n12; j++) X(j+n11+r,j+n11+r+n21+n22+r) = -1.0;

/* bottom half */
for (j=1; j<=n22; j++) X(p+j,q+j) = 1.0;
for (j=1; j<r; j++) {
    X(p+n22+j,j+n11) = sin(theta[j-1]);
    X(p+n22+j,j+r+n21+n22) = cos(theta[j-1]);
}
for (j=1; j<=n21; j++) X(p+n22+r+j,n11+r+j) = 1.0;

alpha = 1.0;
beta = 0.0;

/* multiply U * Sigma_p into w */
nag_dgemm(order, Nag_NoTrans, Nag_NoTrans, m, m, m, alpha,
        &U(1,1), pdu, &X(1,1), pdx, beta, &W(1,1), pdw, &fail);
/* form U * Sigma_p * V^T into u */
nag_dgemm(order, Nag_NoTrans, Nag_NoTrans, m, m, m, alpha,
        &W(1,1), pdw, &V(1,1), pdv, beta, &U(1,1), pdu, &fail);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
m, m, &U(1,1), pdu, " U * Sigma_p * V^T", 0, &fail);

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_dorcsd (f08rac) Example Program Data

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>m</td>
<td>p</td>
<td>q</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>0.7576</td>
<td>0.3697</td>
<td>0.3838</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>-0.7576</td>
<td>-0.3697</td>
<td>0.3838</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>-0.4077</td>
<td>-0.1552</td>
<td>-0.1129</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>-0.0488</td>
<td>0.7240</td>
<td>-0.6730</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>-0.2287</td>
<td>0.0088</td>
<td>0.2235</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>0.4530</td>
<td>0.5612</td>
<td>0.5806</td>
</tr>
</tbody>
</table>

: orthogonal matrix X

10.3 Program Results

nag_dorcsd (f08rac) Example Program Results

Orthogonal matrix X

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>-0.7576</td>
<td>0.3697</td>
<td>0.3838</td>
<td>0.2126</td>
</tr>
<tr>
<td>2</td>
<td>-0.4077</td>
<td>-0.1552</td>
<td>-0.1129</td>
<td>0.2676</td>
</tr>
<tr>
<td>3</td>
<td>-0.0488</td>
<td>0.7240</td>
<td>-0.6730</td>
<td>-0.1301</td>
</tr>
<tr>
<td>4</td>
<td>-0.2287</td>
<td>0.0088</td>
<td>0.2235</td>
<td>-0.9235</td>
</tr>
<tr>
<td>5</td>
<td>0.4530</td>
<td>0.5612</td>
<td>0.5806</td>
<td>0.1162</td>
</tr>
</tbody>
</table>
**Components of CS factorization of X:**

- **Theta**
  - 1: 0.1811
  - 2: 0.8255

- **U1**
  - 1: -0.8249, -0.3370, -0.4538
  - 2: -0.2042, -0.5710, 0.7952
  - 3: -0.5271, 0.7486, 0.4022

- **U2**
  - 1: -0.9802, -0.1982
  - 2: -0.1982, 0.9802

- **V1T**
  - 1: 0.7461, -0.6658
  - 2: 0.6658, 0.7461

- **V2T**
  - 1: -0.3397, 0.8967, -0.2837
  - 2: 0.7738, 0.4379, 0.4576
  - 3: 0.5346, -0.0640, -0.8427

- **U * Sigma_p * V^T**
  - 1: -0.7576, 0.3697, 0.3838, 0.2126, -0.3112
  - 2: -0.4077, -0.1551, -0.1129, 0.2677, 0.8517
  - 3: -0.0488, 0.7240, -0.6730, -0.1300, 0.0602
  - 4: -0.2287, 0.0088, 0.2235, -0.9234, 0.2120
  - 5: 0.4530, 0.5612, 0.5806, 0.1162, 0.3595