NAG Library Function Document

nag_pls_orth_scores_wold (g02lbc)

1 Purpose

nag_pls_orth_scores_wold (g02lbc) fits an orthogonal scores partial least squares (PLS) regression by using Wold's iterative method.

2 Specification

```
#include <nag.h>
#include <nagg02.h>

void nag_pls_orth_scores_wold (Nag_OrderType order, Integer n, Integer mx,
     const double x[], Integer pdx, const Integer isx[], Integer ip, Integer my,
     const double y[], Integer pdy, double xbar[], double ybar[],
     Nag_ScalePredictor iscale, double xstd[], double ystd[], Integer maxfac,
     Integer maxit, double tau, double xres[], Integer pdxres, double yres[],
     Integer pdyres, double w[], Integer pdw, double p[], Integer pdp, double t[],
     Integer pdt, double c[], Integer pdc, double u[], Integer pdu, double xcv[],
     double ycv[], Integer pdycv, NagError *fail)
```

3 Description

Let X_1 be the mean-centred n by m data matrix X of n observations on m predictor variables. Let Y_1 be the mean-centred n by r data matrix Y of n observations on r response variables.

The first of the k factors PLS methods extract from the data predicts both X_1 and Y_1 by regressing on a t_1 column vector of n scores:

$$\begin{aligned} \hat{X}_1 &= t_1 p_1^{\mathrm{T}} \\ \hat{Y}_1 &= t_1 c_1^{\mathrm{T}}, \quad \text{with } t_1^{\mathrm{T}} t_1 = 1, \end{aligned}$$

where the column vectors of m x-loadings p_1 and r y-loadings c_1 are calculated in the least squares sense:

$$\begin{aligned} p_1^{\mathrm{T}} &= t_1^{\mathrm{T}} X_1 \\ c_1^{\mathrm{T}} &= t_1^{\mathrm{T}} Y_1. \end{aligned}$$

The x-score vector $t_1 = X_1 w_1$ is the linear combination of predictor data X_1 that has maximum covariance with the y-scores $u_1 = Y_1 c_1$, where the x-weights vector w_1 is the normalised first left singular vector of $X_1^{\mathrm{T}} Y_1$.

The method extracts subsequent PLS factors by repeating the above process with the residual matrices:

$$\begin{aligned} X_i &= X_{i-1} - \hat{X}_{i-1} \\ Y_i &= Y_{i-1} - \hat{Y}_{i-1}, \quad i = 2, 3, \ldots, k, \end{aligned}$$

and with orthogonal scores:

$$t_i^{\mathrm{T}} t_j = 0, \quad j = 1, 2, \ldots, i-1.$$

Optionally, in addition to being mean-centred, the data matrices X_1 and Y_1 may be scaled by standard deviations of the variables. If data are supplied mean-centred, the calculations are not affected within numerical accuracy.

4 References

Wold H (1966) Estimation of principal components and related models by iterative least squares *In: Multivariate Analysis* (ed P R Krishnaiah) 391–420 Academic Press NY

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: **n** – Integer *Input*

On entry: n, the number of observations.

Constraint: $\mathbf{n} > 1$.

3: **mx** – Integer *Input*

On entry: the number of predictor variables.

Constraint: $\mathbf{mx} > 1$.

4: **x**[*dim*] – const double *Input*

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

$\max(1, \mathbf{pdx} \times \mathbf{mx})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdx})$ when **order** = Nag_RowMajor.

Where $\mathbf{X}(i, j)$ appears in this document, it refers to the array element

$\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$ when **order** = Nag_RowMajor.

On entry: $\mathbf{X}(i, j)$ must contain the ith observation on the jth predictor variable, for $i = 1, 2, \ldots, \mathbf{n}$ and $j = 1, 2, \ldots, \mathbf{mx}$.

5: **pdx** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **x**.

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pdx} \geq \mathbf{n}$;
if **order** = Nag_RowMajor, $\mathbf{pdx} \geq \mathbf{mx}$.

6: **isx**[**mx**] – const Integer *Input*

On entry: indicates which predictor variables are to be included in the model.

$\mathbf{isx}[j-1] = 1$
The jth predictor variable (with variates in the jth column of X) is included in the model.

$\mathbf{isx}[j-1] = 0$
Otherwise.

Constraint: the sum of elements in **isx** must equal **ip**.

7: **ip** – Integer *Input*

On entry: m, the number of predictor variables in the model.

Constraint: $1 < \mathbf{ip} \leq \mathbf{mx}$.

8: **my** – Integer *Input*

On entry: r, the number of response variables.

Constraint: $\mathbf{my} \geq 1$.

9: **y**[*dim*] – const double *Input*

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

$\max(1, \mathbf{pdy} \times \mathbf{my})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdy})$ when **order** = Nag_RowMajor.

Where $\mathbf{Y}(i, j)$ appears in this document, it refers to the array element

$\mathbf{y}[(j-1) \times \mathbf{pdy} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{y}[(i-1) \times \mathbf{pdy} + j - 1]$ when **order** = Nag_RowMajor.

On entry: $\mathbf{Y}(i, j)$ must contain the ith observation for the jth response variable, for $i = 1, 2, \ldots, \mathbf{n}$ and $j = 1, 2, \ldots, \mathbf{my}$.

10: **pdy** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **y**.

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pdy} \geq \mathbf{n}$;
if **order** = Nag_RowMajor, $\mathbf{pdy} \geq \mathbf{my}$.

11: **xbar**[**ip**] – double *Output*

On exit: mean values of predictor variables in the model.

12: **ybar**[**my**] – double *Output*

On exit: the mean value of each response variable.

13: **iscale** – Nag_ScalePredictor *Input*

On entry: indicates how predictor variables are scaled.

iscale = Nag_PredStdScale
Data are scaled by the standard deviation of variables.

iscale = Nag_PredUserScale
Data are scaled by user-supplied scalings.

iscale = Nag_PredNoScale
No scaling.

Constraint: **iscale** = Nag_PredNoScale, Nag_PredStdScale or Nag_PredUserScale.

14: **xstd**[**ip**] – double *Input/Output*

On entry: if **iscale** = Nag_PredUserScale, **xstd**$[j-1]$ must contain the user-supplied scaling for the jth predictor variable in the model, for $j = 1, 2, \ldots, \mathbf{ip}$. Otherwise **xstd** need not be set.

On exit: if **iscale** = Nag_PredStdScale, standard deviations of predictor variables in the model. Otherwise **xstd** is not changed.

15: **ystd**[**my**] – double *Input/Output*

On entry: if **iscale** = Nag_PredUserScale, **ystd**$[j-1]$ must contain the user-supplied scaling for the jth response variable in the model, for $j = 1, 2, \ldots,$ **my**. Otherwise **ystd** need not be set.

On exit: if **iscale** = Nag_PredStdScale, the standard deviation of each response variable. Otherwise **ystd** is not changed.

16: **maxfac** – Integer *Input*

On entry: k, the number of latent variables to calculate.

Constraint: $1 \leq$ **maxfac** $\leq$ **ip**.

17: **maxit** – Integer *Input*

On entry: if **my** $= 1$, **maxit** is not referenced; otherwise the maximum number of iterations used to calculate the x-weights.

Suggested value: **maxit** $= 200$.

Constraint: if **my** > 1, **maxit** > 1.

18: **tau** – double *Input*

On entry: if **my** $= 1$, **tau** is not referenced; otherwise the iterative procedure used to calculate the x-weights will halt if the Euclidean distance between two subsequent estimates is less than or equal to **tau**.

Suggested value: **tau** $= 1.0\text{e}{-4}$.

Constraint: if **my** > 1, **tau** > 0.0.

19: **xres**[*dim*] – double *Output*

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

$\max(1, \textbf{pdxres} \times \textbf{ip})$ when **order** = Nag_ColMajor;
$\max(1, \textbf{n} \times \textbf{pdxres})$ when **order** = Nag_RowMajor.

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

xres$[(j-1) \times$ **pdxres** $+ i - 1]$ when **order** = Nag_ColMajor;

xres$[(i-1) \times$ **pdxres** $+ j - 1]$ when **order** = Nag_RowMajor.

On exit: the predictor variables' residual matrix X_k.

20: **pdxres** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **xres**.

Constraints:

if **order** = Nag_ColMajor, **pdxres** $\geq$ **n**;
if **order** = Nag_RowMajor, **pdxres** $\geq$ **ip**.

21: **yres**[*dim*] – double *Output*

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

$\max(1, \textbf{pdyres} \times \textbf{my})$ when **order** = Nag_ColMajor;
$\max(1, \textbf{n} \times \textbf{pdyres})$ when **order** = Nag_RowMajor.

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

yres$[(j-1) \times$ **pdyres** $+ i - 1]$ when **order** = Nag_ColMajor;

yres$[(i-1) \times$ **pdyres** $+ j - 1]$ when **order** = Nag_RowMajor.

On exit: the residuals for each response variable, Y_k.

22: **pdyres** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **yres**.

Constraints:

if **order** = Nag_ColMajor, **pdyres** $\geq$ **n**;
if **order** = Nag_RowMajor, **pdyres** $\geq$ **my**.

23: **w**[*dim*] – double *Output*

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

$\max(1, \textbf{pdw} \times \textbf{maxfac})$ when **order** = Nag_ColMajor;
$\max(1, \textbf{ip} \times \textbf{pdw})$ when **order** = Nag_RowMajor.

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

w$[(j-1) \times$ **pdw** $+ i - 1]$ when **order** = Nag_ColMajor;

w$[(i-1) \times$ **pdw** $+ j - 1]$ when **order** = Nag_RowMajor.

On exit: the jth column of W contains the x-weights w_j, for $j = 1, 2, \ldots,$ **maxfac**.

24: **pdw** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **w**.

Constraints:

if **order** = Nag_ColMajor, **pdw** $\geq$ **ip**;
if **order** = Nag_RowMajor, **pdw** $\geq$ **maxfac**.

25: **p**[*dim*] – double *Output*

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

$\max(1, \textbf{pdp} \times \textbf{maxfac})$ when **order** = Nag_ColMajor;
$\max(1, \textbf{ip} \times \textbf{pdp})$ when **order** = Nag_RowMajor.

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

p$[(j-1) \times$ **pdp** $+ i - 1]$ when **order** = Nag_ColMajor;

p$[(i-1) \times$ **pdp** $+ j - 1]$ when **order** = Nag_RowMajor.

On exit: the jth column of P contains the x-loadings p_j, for $j = 1, 2, \ldots,$ **maxfac**.

26: **pdp** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **p**.

Constraints:

if **order** = Nag_ColMajor, **pdp** $\geq$ **ip**;
if **order** = Nag_RowMajor, **pdp** $\geq$ **maxfac**.

27: **t**[*dim*] – double *Output*

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

$\max(1, \textbf{pdt} \times \textbf{maxfac})$ when **order** = Nag_ColMajor;
$\max(1, \textbf{n} \times \textbf{pdt})$ when **order** = Nag_RowMajor.

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

t$[(j-1) \times$ **pdt** $+ i - 1]$ when **order** = Nag_ColMajor;

t$[(i-1) \times$ **pdt** $+ j - 1]$ when **order** = Nag_RowMajor.

On exit: the jth column of T contains the x-scores t_j, for $j = 1, 2, \ldots,$ **maxfac**.

28: **pdt** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **t**.

Constraints:

if **order** = Nag_ColMajor, **pdt** $\geq$ **n**;
if **order** = Nag_RowMajor, **pdt** $\geq$ **maxfac**.

29: **c**[*dim*] – double *Output*

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

max(1, **pdc** $\times$ **maxfac**) when **order** = Nag_ColMajor;
max(1, **my** $\times$ **pdc**) when **order** = Nag_RowMajor.

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

c$[(j-1) \times$ **pdc** $+ i - 1]$ when **order** = Nag_ColMajor;

c$[(i-1) \times$ **pdc** $+ j - 1]$ when **order** = Nag_RowMajor.

On exit: the jth column of C contains the y-loadings c_j, for $j = 1, 2, \ldots,$ **maxfac**.

30: **pdc** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **c**.

Constraints:

if **order** = Nag_ColMajor, **pdc** $\geq$ **my**;
if **order** = Nag_RowMajor, **pdc** $\geq$ **maxfac**.

31: **u**[*dim*] – double *Output*

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

max(1, **pdu** $\times$ **maxfac**) when **order** = Nag_ColMajor;
max(1, **n** $\times$ **pdu**) when **order** = Nag_RowMajor.

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

u$[(j-1) \times$ **pdu** $+ i - 1]$ when **order** = Nag_ColMajor;

u$[(i-1) \times$ **pdu** $+ j - 1]$ when **order** = Nag_RowMajor.

On exit: the jth column of U contains the y-scores u_j, for $j = 1, 2, \ldots,$ **maxfac**.

32: **pdu** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **u**.

Constraints:

if **order** = Nag_ColMajor, **pdu** $\geq$ **n**;
if **order** = Nag_RowMajor, **pdu** $\geq$ **maxfac**.

33: **xcv**[**maxfac**] – double *Output*

On exit: **xcv**$[j-1]$ contains the cumulative percentage of variance in the predictor variables explained by the first j factors, for $j = 1, 2, \ldots,$ **maxfac**.

34: **ycv**[*dim*] – double *Output*

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

$\max(1, \mathbf{pdycv} \times \mathbf{my})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{maxfac} \times \mathbf{pdycv})$ when **order** = Nag_RowMajor.

Where $\mathbf{YCV}(i, j)$ appears in this document, it refers to the array element

$\mathbf{ycv}[(j-1) \times \mathbf{pdycv} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{ycv}[(i-1) \times \mathbf{pdycv} + j - 1]$ when **order** = Nag_RowMajor.

On exit: $\mathbf{YCV}(i, j)$ is the cumulative percentage of variance of the jth response variable explained by the first i factors, for $i = 1, 2, \ldots,$ **maxfac** and $j = 1, 2, \ldots,$ **my**.

35: **pdycv** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **ycv**.

Constraints:

if **order** = Nag_ColMajor, **pdycv** $\geq$ **maxfac**;
if **order** = Nag_RowMajor, **pdycv** $\geq$ **my**.

36: **fail** – NagError * *Input/Output*

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **mx** = ⟨*value*⟩.
Constraint: **mx** > 1.

On entry, **my** = ⟨*value*⟩.
Constraint: **my** ≥ 1.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** > 1.

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

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

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

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

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

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

On entry, **pdxres** $= \langle value \rangle$.
Constraint: **pdxres** > 0.

On entry, **pdy** $= \langle value \rangle$.
Constraint: **pdy** > 0.

On entry, **pdycv** $= \langle value \rangle$.
Constraint: **pdycv** > 0.

On entry, **pdyres** $= \langle value \rangle$.
Constraint: **pdyres** > 0.

NE_INT_2

On entry, **ip** $= \langle value \rangle$, **mx** $= \langle value \rangle$.
Constraint: $1 <$ **ip** $\leq$ **mx**.

On entry, **maxfac** $= \langle value \rangle$, **ip** $= \langle value \rangle$.
Constraint: $1 \leq$ **maxfac** $\leq$ **ip**.

On entry, **maxit** $= \langle value \rangle$.
Constraint: if **my** > 1, **maxit** > 1.

On entry, **pdc** $= \langle value \rangle$, **maxfac** $= \langle value \rangle$.
Constraint: **pdc** $\geq$ **maxfac**.

On entry, **pdc** $= \langle value \rangle$, **my** $= \langle value \rangle$.
Constraint: **pdc** $\geq$ **my**.

On entry, **pdp** $= \langle value \rangle$, **ip** $= \langle value \rangle$.
Constraint: **pdp** $\geq$ **ip**.

On entry, **pdp** $= \langle value \rangle$, **maxfac** $= \langle value \rangle$.
Constraint: **pdp** $\geq$ **maxfac**.

On entry, **pdt** $= \langle value \rangle$, **maxfac** $= \langle value \rangle$.
Constraint: **pdt** $\geq$ **maxfac**.

On entry, **pdt** $= \langle value \rangle$, **n** $= \langle value \rangle$.
Constraint: **pdt** $\geq$ **n**.

On entry, **pdu** $= \langle value \rangle$, **maxfac** $= \langle value \rangle$.
Constraint: **pdu** $\geq$ **maxfac**.

On entry, **pdu** $= \langle value \rangle$, **n** $= \langle value \rangle$.
Constraint: **pdu** $\geq$ **n**.

On entry, **pdw** $= \langle value \rangle$, **ip** $= \langle value \rangle$.
Constraint: **pdw** $\geq$ **ip**.

On entry, **pdw** $= \langle value \rangle$, **maxfac** $= \langle value \rangle$.
Constraint: **pdw** $\geq$ **maxfac**.

On entry, **pdx** $= \langle value \rangle$, **mx** $= \langle value \rangle$.
Constraint: **pdx** $\geq$ **mx**.

On entry, **pdx** $= \langle value \rangle$, **n** $= \langle value \rangle$.
Constraint: **pdx** $\geq$ **n**.

On entry, **pdxres** $= \langle value \rangle$, **ip** $= \langle value \rangle$.
Constraint: **pdxres** $\geq$ **ip**.

On entry, **pdxres** $= \langle value \rangle$, **n** $= \langle value \rangle$.
Constraint: **pdxres** $\geq$ **n**.

On entry, **pdy** $= \langle value \rangle$, **my** $= \langle value \rangle$.
Constraint: **pdy** $\geq$ **my**.

On entry, **pdy** $= \langle value \rangle$, **n** $= \langle value \rangle$.
Constraint: **pdy** $\geq$ **n**.

On entry, **pdycv** = ⟨*value*⟩, **maxfac** = ⟨*value*⟩.
Constraint: **pdycv** ≥ **maxfac**.

On entry, **pdycv** = ⟨*value*⟩, **my** = ⟨*value*⟩.
Constraint: **pdycv** ≥ **my**.

On entry, **pdyres** = ⟨*value*⟩, **my** = ⟨*value*⟩.
Constraint: **pdyres** ≥ **my**.

On entry, **pdyres** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: **pdyres** < **n**.

NE_INT_ARG_CONS

On entry, **ip** is not equal to the sum of **isx** elements: **ip** = ⟨*value*⟩, sum(**isx**) = ⟨*value*⟩.

NE_INT_ARRAY_VAL_1_OR_2

On entry, element ⟨*value*⟩ of **isx** is invalid.

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.

NE_REAL

On entry, **tau** = ⟨*value*⟩.
Constraint: if **my** > 1, **tau** > 0.0.

7 Accuracy

In general, the iterative method used in the calculations is less accurate (but faster) than the singular value decomposition approach adopted by nag_pls_orth_scores_svd (g02lac).

8 Further Comments

nag_pls_orth_scores_wold (g02lbc) allocates internally $(n + r)$ elements of double storage.

9 Example

This example reads in data from an experiment to measure the biological activity in a chemical compound, and a PLS model is estimated.

9.1 Program Text

```
/* nag_pls_orth_scores_wold (g02lbc) Example Program.
 *
 * Copyright 2008, Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>
#include <nagx04.h>

int main(int argc, char *argv[])
{
  FILE               *fpin, *fpout;
  char               *outfile = 0;
  /*Integer scalar and array declarations */
```

```
  Integer            exit_status = 0;
  Integer            i, ip, j, maxfac, maxit, mx, my, n;
  Integer            pdc, pdp, pdt, pdu, pdw, pdx, pdxres, pdy, pdycv, pdyres;
  Integer            *isx = 0;
  /*Double scalar and array declarations */
  double             tau;
  double             *c = 0, *p = 0, *t = 0, *u = 0, *w = 0, *x = 0, *xbar = 0;
  double             *xcv = 0, *xres = 0, *xstd = 0, *y = 0, *ybar = 0;
  double             *ycv = 0, *yres = 0, *ystd = 0;
  /*Character scalar and array declarations */
  char               siscale[18];
  /*NAG Types */
  Nag_OrderType      order;
  Nag_ScalePredictor iscale;
  NagError           fail;

  INIT_FAIL(fail);

  /* Check for command-line IO options */
  fpin = nag_example_file_io(argc, argv, "-data", NULL);
  fpout = nag_example_file_io(argc, argv, "-results", NULL);
  (void) nag_example_file_io(argc, argv, "-nag_write", &outfile);

  fprintf(fpout,
          "%s\n", "nag_pls_orth_scores_wold (g02lbc) Example Program Results");
  /* Skip header in data file.*/
  fscanf(fpin, "%*[^\n] ");
  /* Read data values.*/
  fscanf(fpin, "%ld%ld%ld%s %ld%*[^\n] ",
         &n, &mx, &my, siscale, &maxfac);
  iscale = (Nag_ScalePredictor) nag_enum_name_to_value(siscale);

  if (!(isx = NAG_ALLOC(mx, Integer)))
    {
      fprintf(fpout, "Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  for (j = 0; j < mx; j++)
    fscanf(fpin, "%ld ", &isx[j]);
  fscanf(fpin, "%*[^\n] ");
  ip = 0;
  for (j = 0; j < mx; j++)
    {
      if (isx[j] == 1)
        ip = ip+1;
    }
    #ifdef NAG_COLUMN_MAJOR
  pdc = my;
    #define C(I, J)    c[(J-1)*pdc + I-1]
  pdp = ip;
    #define P(I, J)    p[(J-1)*pdp + I-1]
  pdt = n;
    #define T(I, J)    t[(J-1)*pdt + I-1]
  pdu = n;
    #define U(I, J)    u[(J-1)*pdu + I-1]
  pdw = ip;
    #define W(I, J)    w[(J-1)*pdw + I-1]
  pdx = n;
    #define X(I, J)    x[(J-1)*pdx + I-1]
  pdxres = n;
    #define XRES(I, J) xres[(J-1)*pdxres + I-1]
  pdy = n;
    #define Y(I, J)    y[(J-1)*pdy + I-1]
  pdycv = maxfac;
    #define YCV(I, J)  ycv[(J-1)*pdycv + I-1]
  pdyres = n;
    #define YRES(I, J) yres[(J-1)*pdyres + I-1]
  order = Nag_ColMajor;
    #else
  pdc = maxfac;
```

```
  #define C(I, J)    c[(I-1)*pdc + J-1]
pdp = maxfac;
  #define P(I, J)    p[(I-1)*pdp + J-1]
pdt = maxfac;
  #define T(I, J)    t[(I-1)*pdt + J-1]
pdu = maxfac;
  #define U(I, J)    u[(I-1)*pdu + J-1]
pdw = maxfac;
  #define W(I, J)    w[(I-1)*pdw + J-1]
pdx = mx;
  #define X(I, J)    x[(I-1)*pdx + J-1]
pdxres = ip;
  #define XRES(I, J) xres[(I-1)*pdxres + J-1]
pdy = my;
  #define Y(I, J)    y[(I-1)*pdy + J-1]
pdycv = my;
  #define YCV(I, J)  ycv[(I-1)*pdycv + J-1]
pdyres = my;
  #define YRES(I, J) yres[(I-1)*pdyres + J-1]
order = Nag_RowMajor;
  #endif
if (!(c = NAG_ALLOC(pdc*(order == Nag_RowMajor?my:maxfac), double)) ||
    !(p = NAG_ALLOC(pdp*(order == Nag_RowMajor?ip:maxfac), double)) ||
    !(t = NAG_ALLOC(pdt*(order == Nag_RowMajor?n:maxfac), double)) ||
    !(u = NAG_ALLOC(pdu*(order == Nag_RowMajor?n:maxfac), double)) ||
    !(w = NAG_ALLOC(pdw*(order == Nag_RowMajor?ip:maxfac), double)) ||
    !(x = NAG_ALLOC(pdx*(order == Nag_RowMajor?n:mx), double)) ||
    !(xbar = NAG_ALLOC(ip, double)) ||
    !(xcv = NAG_ALLOC(maxfac, double)) ||
    !(xres = NAG_ALLOC(pdxres*(order == Nag_RowMajor?n:ip), double)) ||
    !(xstd = NAG_ALLOC(ip, double)) ||
    !(y = NAG_ALLOC(pdy*(order == Nag_RowMajor?n:my), double)) ||
    !(ybar = NAG_ALLOC(my, double)) ||
    !(ycv = NAG_ALLOC(pdycv*(order == Nag_RowMajor?maxfac:my),
                      double)) ||
    !(yres = NAG_ALLOC(pdyres*(order == Nag_RowMajor?n:my), double)) ||
    !(ystd = NAG_ALLOC(my, double)))
  {
    fprintf(fpout, "Allocation failure\n");
    exit_status = -1;
    goto END;
  }
maxit = 200;
tau = 1.00e-4;
/* Read data values.*/
for (i = 1; i <= n; i++)
  {
    for (j = 1; j <= mx; j++)
      fscanf(fpin, "%lf ", &X(i, j));
    for (j = 1; j <= my; j++)
      fscanf(fpin, "%lf ", &Y(i, j));
  }
fscanf(fpin, "%*[^\n] ");
/* Fit a PLS model.*/
/*
 * nag_pls_orth_scores_wold (g02lbc)
 * Partial least-squares
 */
nag_pls_orth_scores_wold(order, n, mx, x, pdx, isx, ip, my, y, pdy, xbar,
                         ybar, iscale, xstd, ystd, maxfac, maxit, tau,
                         xres, pdxres, yres, pdyres, w, pdw, p, pdp, t,
                         pdt, c, pdc, u, pdu, xcv, ycv, pdycv, &fail);
if (fail.code != NE_NOERROR)
  {
    fprintf(fpout, "Error from nag_pls_orth_scores_wold (g02lbc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }
/*
 * nag_gen_real_mat_print (x04cac)
```

```
   * Print real general matrix (easy-to-use)
   */
  if (outfile) fclose(fpout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, ip,
                         maxfac, p, pdp, "x-loadings, P", outfile, &fail);
  if (outfile && !(fpout = fopen(outfile, "a")))
    {
      exit_status = 2;
      goto END;
    }
  if (fail.code != NE_NOERROR)
    {
      fprintf(fpout, "Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  /*
   * nag_gen_real_mat_print (x04cac)
   * Print real general matrix (easy-to-use)
   */
  if (outfile) fclose(fpout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                         maxfac, t, pdt, "x-scores, T", outfile, &fail);
  if (outfile && !(fpout = fopen(outfile, "a")))
    {
      exit_status = 2;
      goto END;
    }
  if (fail.code != NE_NOERROR)
    {
      fprintf(fpout, "Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  /*
   * nag_gen_real_mat_print (x04cac)
   * Print real general matrix (easy-to-use)
   */
  if (outfile) fclose(fpout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, my,
                         maxfac, c, pdc, "y-loadings, C", outfile, &fail);
  if (outfile && !(fpout = fopen(outfile, "a")))
    {
      exit_status = 2;
      goto END;
    }
  if (fail.code != NE_NOERROR)
    {
      fprintf(fpout, "Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  /*
   * nag_gen_real_mat_print (x04cac)
   * Print real general matrix (easy-to-use)
   */
  if (outfile) fclose(fpout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                         maxfac, u, pdu, "y-scores, U", outfile, &fail);
  if (outfile && !(fpout = fopen(outfile, "a")))
    {
      exit_status = 2;
      goto END;
    }
  if (fail.code != NE_NOERROR)
    {
      fprintf(fpout, "Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
```

```
      exit_status = 1;
      goto END;
    }
  fprintf(fpout, "\n");
  fprintf(fpout, "%s\n", "Explained Variance");
  fprintf(fpout, "%12s   %21s\n", "Model effects", "Dependent variable(s)");
  for (i = 1; i <= maxfac; i++)
    {
      fprintf(fpout, "%12.6f", xcv[i-1]);
      for (j = 1; j <= my; j++)
        fprintf(fpout, "  %12.6f%s", YCV(i, j), j%9?" ":"\n");
      fprintf(fpout, "\n");
    }

 END:
  if (fpin != stdin) fclose(fpin);
  if (fpout != stdout) fclose(fpout);
  if (c) NAG_FREE(c);
  if (p) NAG_FREE(p);
  if (t) NAG_FREE(t);
  if (u) NAG_FREE(u);
  if (w) NAG_FREE(w);
  if (x) NAG_FREE(x);
  if (xbar) NAG_FREE(xbar);
  if (xcv) NAG_FREE(xcv);
  if (xres) NAG_FREE(xres);
  if (xstd) NAG_FREE(xstd);
  if (y) NAG_FREE(y);
  if (ybar) NAG_FREE(ybar);
  if (ycv) NAG_FREE(ycv);
  if (yres) NAG_FREE(yres);
  if (ystd) NAG_FREE(ystd);
  if (isx) NAG_FREE(isx);

  return exit_status;
}
```

9.2 Program Data

```
nag_pls_orth_scores_wold (g02lbc) Example Program Data
15 15 1 Nag_PredStdScale 4 : n, mx, my, iscale, maxfac
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 : isx
-2.6931 -2.5271 -1.2871  3.0777  0.3891 -0.0701  1.9607 -1.6324  0.5746
1.9607 -1.6324  0.574 2.8369  1.4092 -3.1398 0.00
-2.6931 -2.5271 -1.2871  3.0777  0.3891 -0.0701  1.9607 -1.6324  0.5746
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 0.28
-2.6931 -2.5271 -1.2871  3.0777  0.3891 -0.0701  0.0744 -1.7333  0.0902
1.9607 -1.6324  0.5746  2.8369  1.4092 -3.1398 0.20
-2.6931 -2.5271 -1.2871  3.0777  0.3891 -0.0701  0.0744 -1.7333  0.0902
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 0.51
-2.6931 -2.5271 -1.2871  2.8369  1.4092 -3.1398  0.0744 -1.7333  0.0902
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 0.11
-2.6931 -2.5271 -1.2871  3.0777  0.3891 -0.0701 -4.7548  3.6521  0.8524
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 2.73
-2.6931 -2.5271 -1.2871  3.0777  0.3891 -0.0701  0.0744 -1.7333  0.0902
0.0744 -1.7333  0.0902 -1.2201  0.8829  2.2253 0.18
-2.6931 -2.5271 -1.2871  3.0777  0.3891 -0.0701  2.4064  1.7438  1.1057
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 1.53
-2.6931 -2.5271 -1.2871  0.0744 -1.7333  0.0902  0.0744 -1.7333  0.0902
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 -0.10
 2.2261 -5.3648  0.3049  3.0777  0.3891 -0.0701  0.0744 -1.7333  0.0902
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 -0.52
-4.1921 -1.0285 -0.9801  3.0777  0.3891 -0.0701  0.0744 -1.7333  0.0902
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 0.40
-4.9217  1.2977  0.4473  3.0777  0.3891 -0.0701  0.0744 -1.7333  0.0902
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 0.30
-2.6931 -2.5271 -1.2871  3.0777  0.3891 -0.0701  2.2261 -5.3648  0.3049
```

```
2.2261 -5.3648  0.3049  2.8369  1.4092 -3.1398 -1.00
-2.6931 -2.5271 -1.2871  3.0777  0.3891 -0.0701 -4.9217  1.2977  0.4473
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 1.57
-2.6931 -2.5271 -1.2871  3.0777  0.3891 -0.0701 -4.1921 -1.0285 -0.9801
0.0744 -1.7333  0.0902  2.8369  1.4092 -3.1398 0.59 : End of observations
```

9.3 Program Results

```
nag_pls_orth_scores_wold (g02lbc) Example Program Results
 x-loadings, P

             1          2          3          4
 1     -0.6708    -1.0047     0.6505     0.6169
 2      0.4943     0.1355    -0.9010    -0.2388
 3     -0.4167    -1.9983    -0.5538     0.8474
 4      0.3930     1.2441    -0.6967    -0.4336
 5      0.3267     0.5838    -1.4088    -0.6323
 6      0.0145     0.9607     1.6594     0.5361
 7     -2.4471     0.3532    -1.1321    -1.3554
 8      3.5198     0.6005     0.2191     0.0380
 9      1.0973     2.0635    -0.4074    -0.3522
10     -2.4466     2.5640    -0.4806     0.3819
11      2.2732    -1.3110    -0.7686    -1.8959
12     -1.7987     2.4088    -0.9475    -0.4727
13      0.3629     0.2241    -2.6332     2.3739
14      0.3629     0.2241    -2.6332     2.3739
15     -0.3629    -0.2241     2.6332    -2.3739
 x-scores, T

          1        2        3        4
 1  -0.1896   0.3898  -0.2502  -0.2479
 2   0.0201  -0.0013  -0.1726  -0.2042
 3  -0.1889   0.3141  -0.1727  -0.1350
 4   0.0210  -0.0773  -0.0950  -0.0912
 5  -0.0090  -0.2649  -0.4195  -0.1327
 6   0.5479   0.2843   0.1914   0.2727
 7  -0.0937  -0.0579   0.6799  -0.6129
 8   0.2500   0.2033  -0.1046  -0.1014
 9  -0.1005  -0.2992   0.2131   0.1223
10  -0.1810  -0.4427   0.0559   0.2114
11   0.0497  -0.0762  -0.1526  -0.0771
12   0.0173  -0.2517  -0.2104   0.1044
13  -0.6002   0.3596   0.1876   0.4812
14   0.3796   0.1338   0.1410   0.1999
15   0.0773  -0.2139   0.1085   0.2106
 y-loadings, C

            1          2          3          4
 1      3.5425     1.0475     0.2548     0.1866
 y-scores, U

             1          2          3          4
 1     -1.7670     0.1812    -0.0600    -0.0320
 2     -0.6724    -0.2735    -0.0662    -0.0402
 3     -0.9852     0.4097     0.0158     0.0198
 4      0.2267    -0.0107     0.0180     0.0177
 5     -1.3370    -0.3619    -0.0173     0.0073
 6      8.9056     0.6000     0.0701     0.0422
 7     -1.0634     0.0332     0.0235    -0.0151
 8      4.2143     0.3184     0.0232     0.0219
 9     -2.1580    -0.2652     0.0153     0.0011
10     -3.7999    -0.4520     0.0082     0.0034
11     -0.2033    -0.2446    -0.0392    -0.0214
12     -0.5942    -0.2398     0.0089     0.0165
13     -5.6764     0.5487     0.0375     0.0185
14      4.3707    -0.1161    -0.0639    -0.0535
15      0.5395    -0.1274     0.0261     0.0139

Explained Variance
```

```
Model effects   Dependent variable(s)
   16.902124     89.638060
   29.674338     97.476270
   44.332404     97.939839
   56.172041     98.188474
```
