

NAG Library Function Document

nag_pls_orth_scores_fit (g02lcc)

1 Purpose

nag_pls_orth_scores_fit (g02lcc) calculates parameter estimates for a given number of factors given the output from an orthogonal scores PLS regression (nag_pls_orth_scores_svd (g02lac) or nag_pls_orth_scores_wold (g02lbc)).

2 Specification

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

void nag_pls_orth_scores_fit (Nag_OrderType order, Integer ip, Integer my,
    Integer maxfac, Integer nfact, const double p[], Integer pdp,
    const double c[], Integer pdc, const double w[], Integer pdw,
    double rcond, double b[], Integer pdb, Nag_EstimatesOption orig,
    const double xbar[], const double ybar[], Nag_ScalePredictor iscale,
    const double xstd[], const double ystd[], double ob[], Integer pdob,
    Integer vipopt, const double ycv[], Integer pdycv, double vip[],
    Integer pdvip, NagError *fail)
```

3 Description

The parameter estimates B for a l -factor orthogonal scores PLS model with m predictor variables and r response variables are given by,

$$B = W(P^T W)^{-1} C^T, \quad B \in \mathbb{R}^{m \times r},$$

where W is the m by k ($\geq l$) matrix of x -weights; P is the m by k matrix of x -loadings; and C is the r by k matrix of y -loadings for a fitted PLS model.

The parameter estimates B are for centred, and possibly scaled, predictor data X_1 and response data Y_1 . Parameter estimates may also be given for the predictor data X and response data Y .

Optionally, nag_pls_orth_scores_fit (g02lcc) will calculate variable influence on projection (VIP) statistics, see Wold (1994).

4 References

Wold S (1994) PLS for multivariate linear modelling QSAR: chemometric methods in molecular design *Methods and Principles in Medicinal Chemistry* (ed van de Waterbeemd H) Verlag-Chemie

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 2.3.1.3 in How to Use the NAG Library and its Documentation for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

- 2: **ip** – Integer *Input*
On entry: m , the number of predictor variables in the fitted model.
Constraint: $\mathbf{ip} > 1$.
- 3: **my** – Integer *Input*
On entry: r , the number of response variables.
Constraint: $\mathbf{my} \geq 1$.
- 4: **maxfac** – Integer *Input*
On entry: k , the number of factors available in the PLS model.
Constraint: $1 \leq \mathbf{maxfac} \leq \mathbf{ip}$.
- 5: **nfact** – Integer *Input*
On entry: l , the number of factors to include in the calculation of parameter estimates.
Constraint: $1 \leq \mathbf{nfact} \leq \mathbf{maxfac}$.
- 6: **p**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **p** must be at least
 $\max(1, \mathbf{pdp} \times \mathbf{maxfac})$ when **order** = Nag_ColMajor;
 $\max(1, \mathbf{ip} \times \mathbf{pdp})$ when **order** = Nag_RowMajor.
The (i, j)th element of the matrix P is stored in
 $\mathbf{p}[(j-1) \times \mathbf{pdp} + i - 1]$ when **order** = Nag_ColMajor;
 $\mathbf{p}[(i-1) \times \mathbf{pdp} + j - 1]$ when **order** = Nag_RowMajor.
On entry: x -loadings as returned from nag_pls_orth_scores_svd (g02lac) and nag_pls_orth_scores_wold (g02lbc).
- 7: **pdp** – Integer *Input*
On entry: the stride separating row or column elements (depending on the value of **order**) in the array **p**.
Constraints:
if **order** = Nag_ColMajor, $\mathbf{pdp} \geq \mathbf{ip}$;
if **order** = Nag_RowMajor, $\mathbf{pdp} \geq \mathbf{maxfac}$.
- 8: **c**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **c** must be at least
 $\max(1, \mathbf{pdc} \times \mathbf{maxfac})$ when **order** = Nag_ColMajor;
 $\max(1, \mathbf{my} \times \mathbf{pdc})$ when **order** = Nag_RowMajor.
The (i, j)th element of the matrix C is stored in
 $\mathbf{c}[(j-1) \times \mathbf{pdc} + i - 1]$ when **order** = Nag_ColMajor;
 $\mathbf{c}[(i-1) \times \mathbf{pdc} + j - 1]$ when **order** = Nag_RowMajor.
On entry: y -loadings as returned from nag_pls_orth_scores_svd (g02lac) and nag_pls_orth_scores_wold (g02lbc).
- 9: **pdc** – Integer *Input*
On entry: the stride separating row or column elements (depending on the value of **order**) in the array **c**.

Constraints:

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

10: **w**[*dim*] – const double *Input*

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

max(1, **pdw** \times **maxfac**) when **order** = Nag_ColMajor;
max(1, **ip** \times **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 entry: *x*-weights as returned from nag_pls_orth_scores_svd (g02lac) and nag_pls_orth_scores_wold (g02lbc).

11: **pdw** – Integer *Input*

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

Constraints:

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

12: **rcond** – double *Input*

On entry: singular values of $P^T W$ less than **rcond** times the maximum singular value are treated as zero when calculating parameter estimates. If **rcond** is negative, a value of 0.005 is used.

13: **b**[*dim*] – double *Output*

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

max(1, **pdb** \times **my**) when **order** = Nag_ColMajor;
max(1, **ip** \times **pdb**) when **order** = Nag_RowMajor.

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

b[(*j* – 1) \times **pdb** + *i* – 1] when **order** = Nag_ColMajor;
b[(*i* – 1) \times **pdb** + *j* – 1] when **order** = Nag_RowMajor.

On exit: **B**(*i*, *j*) contains the parameter estimate for the *i*th predictor variable in the model for the *j*th response variable, for *i* = 1, 2, ..., **ip** and *j* = 1, 2, ..., **my**.

14: **pdb** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, **pdb** \geq **ip**;
if **order** = Nag_RowMajor, **pdb** \geq **my**.

15: **orig** – Nag_EstimatesOption *Input*

On entry: indicates how parameter estimates are calculated.

orig = Nag_EstimatesStand

Parameter estimates for the centred, and possibly, scaled data.

orig = Nag_EstimatesOrig

Parameter estimates for the original data.

Constraint: **orig** = Nag_EstimatesStand or Nag_EstimatesOrig.

16: **xbar[ip]** – const double *Input*

On entry: if **orig** = Nag_EstimatesOrig, mean values of predictor variables in the model; otherwise **xbar** is not referenced.

17: **ybar[my]** – const double *Input*

On entry: if **orig** = Nag_EstimatesOrig, mean value of each response variable in the model; otherwise **ybar** is not referenced.

18: **iscale** – Nag_ScalePredictor *Input*

On entry: if **orig** = Nag_EstimatesOrig, **iscale** must take the value supplied to either nag_pls_orth_scores_svd (g02lac) or nag_pls_orth_scores_wold (g02lbc); otherwise **iscale** is not referenced.

Constraint: if **orig** = Nag_EstimatesOrig, **iscale** = Nag_PredNoScale, Nag_PredStdScale or Nag_PredUserScale.

19: **xstd[ip]** – const double *Input*

On entry: if **orig** = Nag_EstimatesOrig and **iscale** \neq Nag_PredNoScale, the scalings of predictor variables in the model as returned from either nag_pls_orth_scores_svd (g02lac) or nag_pls_orth_scores_wold (g02lbc); otherwise **xstd** is not referenced.

20: **ystd[my]** – const double *Input*

On entry: if **orig** = Nag_EstimatesOrig and **iscale** \neq Nag_PredNoScale, the scalings of response variables as returned from either nag_pls_orth_scores_svd (g02lac) or nag_pls_orth_scores_wold (g02lbc); otherwise **ystd** is not referenced.

21: **ob[dim]** – double *Output*

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

pdob \times **my** when **orig** = Nag_EstimatesOrig and **order** = Nag_ColMajor;
 $\max(1, (\mathbf{ip} + 1) \times \mathbf{pdob})$ when **orig** = Nag_EstimatesOrig and **order** = Nag_RowMajor;
 1 otherwise.

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

ob[(*j* – 1) \times **pdob** + *i* – 1] when **order** = Nag_ColMajor;
ob[(*i* – 1) \times **pdob** + *j* – 1] when **order** = Nag_RowMajor.

On exit: if **orig** = Nag_EstimatesOrig, **OB**(1, *j*) contains the intercept value for the *j*th response variable, and **OB**(*i* + 1, *j*) contains the parameter estimate on the original scale for the *i*th predictor variable in the model, for *i* = 1, 2, ..., **ip** and *j* = 1, 2, ..., **my**. Otherwise **ob** is not referenced.

22: **pdob** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor,
 if **orig** = Nag_EstimatesOrig, **pdob** \geq **ip** + 1;
 otherwise **pdob** \geq 1.;

if **order** = Nag_RowMajor,
 if **orig** = Nag_EstimatesOrig, **pdob** ≥ **my**;
 otherwise **pdob** ≥ 1..

23: **vipopt** – Integer *Input*

On entry: a flag that determines variable influence on projections (VIP) options.

vipopt = 0
 VIP are not calculated.

vipopt = 1
 VIP are calculated for predictor variables using the mean explained variance in responses.

vipopt = **my**
 VIP are calculated for predictor variables for each response variable in the model.

Note that setting **vipopt** = **my** when **my** = 1 gives the same result as setting **vipopt** = 1 directly.

Constraint: **vipopt** = 0, 1, or **my**.

24: **yvc**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **yvc** must be at least **my** when **vipopt** ≠ 0.

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

yvc[(*j* – 1) × **pdycv** + *i* – 1] when **order** = Nag_ColMajor;
yvc[(*i* – 1) × **pdycv** + *j* – 1] when **order** = Nag_RowMajor.

On entry: if **vipopt** ≠ 0, **YCV**(*i*, *j*) is the cumulative percentage of variance of the *j*th response variable explained by the first *i* factors, for *i* = 1, 2, ..., **nfact** and *j* = 1, 2, ..., **my**; otherwise **yvc** is not referenced.

25: **pdycv** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, if **vipopt** ≠ 0, **pdycv** ≥ **nfact**;
 if **order** = Nag_RowMajor,
 if **vipopt** ≠ 0, **pdycv** ≥ **my**..

26: **vip**[*dim*] – double *Output*

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

max(1, **pdvip** × **vipopt**) when **order** = Nag_ColMajor;
 max(1, **ip** × **pdvip**) when **order** = Nag_RowMajor and **vipopt** ≠ 0.

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

vip[(*j* – 1) × **pdvip** + *i* – 1] when **order** = Nag_ColMajor;
vip[(*i* – 1) × **pdvip** + *j* – 1] when **order** = Nag_RowMajor.

On exit: if **vipopt** = 1, **VIP**(*i*, 1) contains the VIP statistic for the *i*th predictor variable in the model for all response variables, for *i* = 1, 2, ..., **ip**.

If **vipopt** = **my**, **VIP**(*i*, *j*) contains the VIP statistic for the *i*th predictor variable in the model for the *j*th response variable, for *i* = 1, 2, ..., **ip** and *j* = 1, 2, ..., **my**.

Otherwise **vip** is not referenced.

- 27: **pdvip** – Integer *Input*
On entry: the stride separating row or column elements (depending on the value of **order**) in the array **vip**.
Constraints:
 if **order** = Nag_ColMajor, if **vipopt** \neq 0, **pdvip** \geq **ip**;
 if **order** = Nag_RowMajor, **pdvip** \geq **vipopt**.
- 28: **fail** – NagError * *Input/Output*
 The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
 See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_ENUM_INT

On entry, **iscale** = $\langle value \rangle$.
 Constraint: if **orig** = Nag_EstimatesOrig, **iscale** = Nag_PredNoScale or Nag_PredStdScale.
 On entry, **orig** = $\langle value \rangle$ and **my** = $\langle value \rangle$.
 Constraint: **my** $>$ 0.

NE_ENUM_INT_2

On entry, **orig** = $\langle value \rangle$, **pdob** = $\langle value \rangle$, **my** = $\langle value \rangle$.
 Constraint: if **orig** = Nag_EstimatesOrig, **pdob** \geq **my**;
 otherwise **pdob** \geq 1.

NE_INT

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

On entry, **my** = $\langle value \rangle$.
 Constraint: **my** \geq 1.

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

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

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

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

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

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

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

NE_INT_2

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

On entry, **nfact** = $\langle value \rangle$ and **maxfac** = $\langle value \rangle$.
 Constraint: $1 \leq \mathbf{nfact} \leq \mathbf{maxfac}$.

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

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

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

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

On entry, **pdob** = $\langle value \rangle$ and **ip** = $\langle value \rangle$.
 Constraint: if **orig** = Nag_EstimatesOrig, **pdob** \geq **ip** + 1.

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

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

On entry, **pdvip** = $\langle value \rangle$ and **ip** = $\langle value \rangle$.
 Constraint: if **vipopt** \neq 0, **pdvip** \geq **ip**.

On entry, **pdvip** = $\langle value \rangle$ and **vipopt** = $\langle value \rangle$.
 Constraint: **pdvip** \geq **vipopt**.

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

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

On entry, **pdycv** = $\langle value \rangle$ and **nfact** = $\langle value \rangle$.
 Constraint: if **vipopt** \neq 0, **pdycv** \geq **nfact**.

On entry, **vipopt** = $\langle value \rangle$ and **my** = $\langle value \rangle$.
 Constraint: **my** > 0.

On entry, **vipopt** = $\langle value \rangle$ and **my** = $\langle value \rangle$.
 Constraint: **vipopt** = 0, 1, or **my**.

NE_INT_3

On entry, **pdycv** = $\langle value \rangle$, **vipopt** = $\langle value \rangle$ and **my** = $\langle value \rangle$.
 Constraint:
 if **vipopt** \neq 0, **pdycv** \geq **my**.

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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

The calculations are based on the singular value decomposition of $P^T W$.

8 Parallelism and Performance

`nag_pls_orth_scores_fit` (g02lcc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

`nag_pls_orth_scores_fit` (g02lcc) 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

`nag_pls_orth_scores_fit` (g02lcc) allocates internally $l(l+r+4) + \max(2l, r)$ elements of double storage.

10 Example

This example reads in details of a PLS model, and a set of parameter estimates are calculated along with their VIP statistics.

10.1 Program Text

```

/* nag_pls_orth_scores_fit (g02lcc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>
#include <nagx04.h>

int main(void)
{
    /*Integer scalar and array declarations */
    Integer exit_status = 0;
    Integer i, ip, ip1, j, maxfac, my, nfact, vipopt;
    Integer pdb, pdc, pdob, pdp, pdvip, pdw, pdycv;
    /*Double scalar and array declarations */
    double rcond;
    double *b = 0, *c = 0, *ob = 0, *p = 0, *vip = 0, *w = 0;
    double *xbar = 0, *xstd = 0, *ybar = 0, *ycv = 0, *ystd = 0;
    /*Character scalar and array declarations */
    char siscale[40], sorig[40];
    /*NAG Types */

```



```

Nag_OrderType order;
Nag_ScalePredictor iscale;
Nag_EstimatesOption orig;
NagError fail;

INIT_FAIL(fail);

printf("nag_pls_orth_scores_fit (g02lcc) Example Program Results\n");
/* Skip header in data file */
#ifdef _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif
/* Read data values */
#ifdef _WIN32
scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%39s "
"%39s %" NAG_IFMT "%*[\n] ", &ip, &my, &maxfac, &nfact, sorig,
(unsigned)_countof(sorig), siscale,
(unsigned)_countof(siscale), &vipopt);
#else
scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%39s "
"%39s %" NAG_IFMT "%*[\n] ", &ip, &my, &maxfac, &nfact, sorig,
siscale, &vipopt);
#endif
orig = (Nag_EstimatesOption) nag_enum_name_to_value(sorig);
iscale = (Nag_ScalePredictor) nag_enum_name_to_value(siscale);

#ifdef NAG_COLUMN_MAJOR
pdb = ip;
pdc = my;
#define C(I, J) c[(J-1)*pdc + I-1]
pdob = ip + 1;
pdp = ip;
#define P(I, J) p[(J-1)*pdp + I-1]
pdvip = ip;
pdw = ip;
#define W(I, J) w[(J-1)*pdw + I-1]
pdycv = maxfac;
#define YCV(I, J) ycv[(J-1)*pdycv + I-1]
order = Nag_ColMajor;
#else
pdb = my;
pdc = maxfac;
#define C(I, J) c[(I-1)*pdc + J-1]
pdob = my;
pdp = maxfac;
#define P(I, J) p[(I-1)*pdp + J-1]
pdvip = vipopt;
pdw = maxfac;
#define W(I, J) w[(I-1)*pdw + J-1]
pdycv = my;
#define YCV(I, J) ycv[(I-1)*pdycv + J-1]
order = Nag_RowMajor;
#endif
if (!(b = NAG_ALLOC(pdb * (order == Nag_RowMajor ? ip : my), double)) ||
!(c = NAG_ALLOC(pdc * (order == Nag_RowMajor ? my : maxfac), double)) ||
!(ob = NAG_ALLOC(pdob * (order == Nag_RowMajor ? (ip + 1) : my),
double)) ||
!(p = NAG_ALLOC(pdp * (order == Nag_RowMajor ? ip : maxfac), double)) ||
!(vip = NAG_ALLOC(pdvip * (order == Nag_RowMajor ? ip : vipopt),
double)) ||
!(w = NAG_ALLOC(pdw * (order == Nag_RowMajor ? ip : maxfac), double)) ||
!(xbar = NAG_ALLOC(ip, double)) ||
!(xstd = NAG_ALLOC(ip, double)) ||
!(ybar = NAG_ALLOC(my, double)) ||
!(ycv = NAG_ALLOC(pdycv * (order == Nag_RowMajor ? maxfac : my),
double)) || !(ystd = NAG_ALLOC(my, double)))
{
printf("Allocation failure\n");
exit_status = -1;
}

```

```

        goto END;
    }
    /* Read P */
    for (i = 1; i <= ip; i++) {
        for (j = 1; j <= maxfac; j++)
#ifdef _WIN32
            scanf_s("%lf ", &P(i, j));
#else
            scanf("%lf ", &P(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    /* Read C */
    for (i = 1; i <= my; i++) {
        for (j = 1; j <= maxfac; j++)
#ifdef _WIN32
            scanf_s("%lf ", &C(i, j));
#else
            scanf("%lf ", &C(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    /* Read W */
    for (i = 1; i <= ip; i++) {
        for (j = 1; j <= maxfac; j++)
#ifdef _WIN32
            scanf_s("%lf ", &W(i, j));
#else
            scanf("%lf ", &W(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    /* Read YCV */
    for (i = 1; i <= maxfac; i++) {
        for (j = 1; j <= my; j++)
#ifdef _WIN32
            scanf_s("%lf ", &YCV(i, j));
#else
            scanf("%lf ", &YCV(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    /* Read means */
    if (orig == Nag_EstimatesOrig) {
        for (j = 0; j < ip; j++)
#ifdef _WIN32
            scanf_s("%lf ", &xbar[j]);
#else
            scanf("%lf ", &xbar[j]);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

```

```

    for (j = 0; j < my; j++)
#ifdef _WIN32
        scanf_s("%lf ", &ybar[j]);
#else
        scanf("%lf ", &ybar[j]);
#endif
#ifdef _WIN32
        scanf_s("%*[\n] ");
#else
        scanf("%*[\n] ");
#endif
        if (iscale != Nag_PredNoScale) {
            for (j = 0; j < ip; j++)
#ifdef _WIN32
                scanf_s("%lf ", &xstd[j]);
#else
                scanf("%lf ", &xstd[j]);
#endif
#ifdef _WIN32
                scanf_s("%*[\n] ");
#else
                scanf("%*[\n] ");
#endif
            for (j = 0; j < my; j++)
#ifdef _WIN32
                scanf_s("%lf ", &ystd[j]);
#else
                scanf("%lf ", &ystd[j]);
#endif
#ifdef _WIN32
                scanf_s("%*[\n] ");
#else
                scanf("%*[\n] ");
#endif
        }
    }
    /* Calculate predictions */
    rcond = -1.00e0;
    ip1 = ip + 1;
    /*
    * nag_pls_orth_scores_fit (g02lcc)
    * Partial least squares
    */
    nag_pls_orth_scores_fit(order, ip, my, maxfac, nfact, p, pdp, c, pdc, w,
                           pdw, rcond, b, pdb, orig, xbar, ybar, iscale, xstd,
                           ystd, ob, pdob, vipopt, ycv, pdycv, vip,
                           pdvip, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_pls_orth_scores_fit (g02lcc).\n%s\n",
              fail.message);
        exit_status = 1;
        goto END;
    }
    /*
    * nag_gen_real_mat_print (x04cac)
    * Print real general matrix (easy-to-use)
    */
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, ip, my,
                           b, pdb, "B ", 0, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    if (orig == Nag_EstimatesOrig) {
        /*
        * nag_gen_real_mat_print (x04cac)
        * Print real general matrix (easy-to-use)
        */
        fflush(stdout);

```

```

nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, ip1,
                        my, ob, pdob, "OB", 0, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}
}
if (vipopt != 0) {
    /*
    * nag_gen_real_mat_print (x04cac)
    * Print real general matrix (easy-to-use)
    */
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, ip,
                          vipopt, vip, pdvip, "VIP", 0, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
        exit_status = 1;
        goto END;
    }
}
}
END:
NAG_FREE(b);
NAG_FREE(c);
NAG_FREE(ob);
NAG_FREE(p);
NAG_FREE(vip);
NAG_FREE(w);
NAG_FREE(xbar);
NAG_FREE(xstd);
NAG_FREE(ybar);
NAG_FREE(ycv);
NAG_FREE(ystd);

return exit_status;
}

```

10.2 Program Data

```

nag_pls_orth_scores_fit (g02lcc) Example Program Data
15 1 4 2 Nag_EstimatesOrig Nag_PredStdScale 1      : model parameters
-0.6708    -1.0047     0.6505     0.6169
 0.4943     0.1355    -0.9010    -0.2388
-0.4167    -1.9983    -0.5538     0.8474
 0.3930     1.2441    -0.6967    -0.4336
 0.3267     0.5838    -1.4088    -0.6323
 0.0145     0.9607     1.6594     0.5361
-2.4471     0.3532    -1.1321    -1.3554
 3.5198     0.6005     0.2191     0.0380
 1.0973     2.0635    -0.4074    -0.3522
-2.4466     2.5640    -0.4806     0.3819
 2.2732    -1.3110    -0.7686    -1.8959
-1.7987     2.4088    -0.9475    -0.4727
 0.3629     0.2241    -2.6332     2.3739
 0.3629     0.2241    -2.6332     2.3739
-0.3629    -0.2241     2.6332    -2.3739      : p
 3.5425     1.0475     0.2548     0.1866      : c
-1.5764E-01 -1.5935E-01  1.7774E-01  5.4029E-02
 8.5680E-02 -1.5240E-04 -1.2179E-01  1.0989E-01
-1.6931E-01 -3.7431E-01  9.4348E-02  3.1878E-01
 1.2153E-01  2.0589E-01 -1.8144E-01 -4.4610E-02
 7.1133E-02  5.5884E-02 -2.6916E-01  5.4912E-02
 6.5188E-02  2.4170E-01  2.3365E-01 -1.8849E-01
-4.2481E-01 -1.8798E-03 -3.2413E-01 -1.1600E-01
 6.5370E-01  1.6725E-01  2.1908E-01  2.5461E-01
 2.8504E-01  3.6549E-01 -1.9244E-01 -1.5430E-01

```

```

-2.9341E-01   5.0464E-01  -1.0952E-02   1.3881E-01
 2.9829E-01  -3.6979E-01  -4.9942E-01  -4.9355E-01
-2.0313E-01   4.1952E-01  -2.5684E-01  -7.5647E-02
 5.6905E-02  -2.3197E-02  -3.0503E-01   3.9673E-01
 5.6905E-02  -2.3197E-02  -3.0503E-01   3.9673E-01
-5.6905E-02   2.3197E-02   3.0503E-01  -3.9673E-01 : w
89.638060
97.476270
97.939839
98.188474 : ycv
-2.6137 -2.3614 -1.0449  2.8614  0.3156 -0.2641
-0.3146 -1.1221  0.2401  0.4694 -1.9619  0.1691
 2.5664  1.3741 -2.7821 : xbar
 0.4520 : ybar
 1.4956  1.3233  0.5829  0.7735  0.6247  0.7966
 2.4113  2.0421  0.4678  0.8197  0.9420  0.1735
 1.0475  0.1359  1.3853 : xstd
 0.9062 : ystd

```

10.3 Program Results

nag_pls_orth_scores_fit (g02lcc) Example Program Results

```

B
  1
  1 -0.1383
  2  0.0572
  3 -0.1906
  4  0.1238
  5  0.0591
  6  0.0936
  7 -0.2842
  8  0.4713
  9  0.2661
 10 -0.0914
 11  0.1226
 12 -0.0488
 13  0.0332
 14  0.0332
 15 -0.0332
OB
  1
  1 -0.4374
  2 -0.0838
  3  0.0392
  4 -0.2964
  5  0.1451
  6  0.0857
  7  0.1065
  8 -0.1068
  9  0.2091
 10  0.5155
 11 -0.1011
 12  0.1180
 13 -0.2548
 14  0.0287
 15  0.2214
 16 -0.0217
VIP
  1
  1  0.6111
  2  0.3182
  3  0.7513
  4  0.5048
  5  0.2712
  6  0.3593
  7  1.5777
  8  2.4348
  9  1.1322
 10  1.2226

```

11	1.1799
12	0.8840
13	0.2129
14	0.2129
15	0.2129
