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

```c
#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,

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

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

   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: `ip` – Integer
   
   *Input*
   
   On entry: `m`, the number of predictor variables in the fitted model.

   *Constraint:* `ip > 1`.

3: `my` – Integer
   
   *Input*
   
   On entry: `r`, the number of response variables.

   *Constraint:* `my ≥ 1`.

4: `maxfac` – Integer
   
   *Input*
   
   On entry: `k`, the number of factors available in the PLS model.

   *Constraint:* `1 ≤ maxfac ≤ ip`.

5: `nfact` – Integer
   
   *Input*
   
   On entry: `l`, the number of factors to include in the calculation of parameter estimates.

   *Constraint:* `1 ≤ nfact ≤ maxfac`.

6: `p[dim]` – const double
   
   *Input*
   
   Note: the dimension, `dim`, of the array `p` must be at least
   
   \[
   \max(1, \text{pdp} \times \text{maxfac}) \text{ when } \text{order} = \text{Nag-ColMajor}; \\
   \max(1, \text{ip} \times \text{pdp}) \text{ when } \text{order} = \text{Nag-RowMajor}.
   \]

   The \((i,j)\)th element of the matrix \(P\) is stored in
   
   \[
   p[(j - 1) \times \text{pdp} + i - 1] \text{ when } \text{order} = \text{Nag-ColMajor}; \\
   p[(i - 1) \times \text{pdp} + j - 1] \text{ when } \text{order} = \text{Nag-RowMajor}.
   \]

   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:*
   
   \[
   \text{if } \text{order} = \text{Nag-ColMajor}, \text{ pdp} ≥ \text{ip}; \\
   \text{if } \text{order} = \text{Nag-RowMajor}, \text{ pdp} ≥ \text{maxfac}.
   \]

8: `c[dim]` – const double
   
   *Input*
   
   Note: the dimension, `dim`, of the array `c` must be at least
   
   \[
   \max(1, \text{pdc} \times \text{maxfac}) \text{ when } \text{order} = \text{Nag-ColMajor}; \\
   \max(1, \text{my} \times \text{pdc}) \text{ when } \text{order} = \text{Nag-RowMajor}.
   \]

   The \((i,j)\)th element of the matrix \(C\) is stored in
   
   \[
   c[(j - 1) \times \text{pdc} + i - 1] \text{ when } \text{order} = \text{Nag-ColMajor}; \\
   c[(i - 1) \times \text{pdc} + j - 1] \text{ when } \text{order} = \text{Nag-RowMajor}.
   \]

   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, pdc ≥ my;
if order = Nag_RowMajor, pdc ≥ maxfac.

10: w[dim] – const double

Note: the dimension, dim, of the array w must be at least
max(1, pdw × maxfac) when order = Nag_ColMajor;
max(1, ip × 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

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 ≥ ip;
if order = Nag_RowMajor, pdw ≥ maxfac.

12: rcond – double

On entry: singular values of \(P^TW\) 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

Note: the dimension, dim, of the array b must be at least
max(1, pdb × my) when order = Nag_ColMajor;
max(1, ip × 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 \(i = 1, 2, \ldots, ip\) and \(j = 1, 2, \ldots, my\).

14: pdb – Integer

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 ≥ ip;
if order = Nag_RowMajor, pdb ≥ my.

15: orig – Nag_EstimatesOption

On entry: indicates how parameter estimates are calculated.

orig = Nag_EstimatesStand
Parameter estimates for the centered, and possibly, scaled data.
Parameter estimates for the original data.

Constraint: \( \text{orig} = \text{Nag\_EstimatesStand or Nag\_EstimatesOrig} \).

16: \( \text{xbar[ip]} \) – const double
   \( \text{Input} \)
   \( \text{On entry: if orig = Nag\_EstimatesOrig, mean values of predictor variables in the model; otherwise xbar is not referenced.} \)

17: \( \text{ybar[my]} \) – const double
   \( \text{Input} \)
   \( \text{On entry: if orig = Nag\_EstimatesOrig, mean value of each response variable in the model; otherwise ybar is not referenced.} \)

18: \( \text{iscale} \) – Nag\_ScalePredictor
   \( \text{Input} \)
   \( \text{On entry: if orig = Nag\_EstimatesOrig, iscale must take the value supplied to either nag\_pls\_orth\_scores\_svd (g02lac) or nag\_pls\_orth\_scores\_wold (g02lbc); otherwise iscale is not referenced.} \)
   \( \text{Constraint: if orig = Nag\_EstimatesOrig, iscale = Nag\_PredNoScale, Nag\_PredStdScale or Nag\_PredUserScale.} \)

19: \( \text{xstd[ip]} \) – const double
   \( \text{Input} \)
   \( \text{On entry: if orig = Nag\_EstimatesOrig and iscale \neq Nag\_PredNoScale, the scalings of predictor variables in the model as returned from either nag\_pls\_orth\_scores\_svd (g02lac) or nag\_pls\_orth\_scores\_wold (g02lbc); otherwise xstd is not referenced.} \)

20: \( \text{ystd[my]} \) – const double
   \( \text{Input} \)
   \( \text{On entry: if orig = Nag\_EstimatesOrig and iscale \neq Nag\_PredNoScale, the scalings of response variables as returned from either nag\_pls\_orth\_scores\_svd (g02lac) or nag\_pls\_orth\_scores\_wold (g02lbc); otherwise ystd is not referenced.} \)

21: \( \text{ob[dim]} \) – double
   \( \text{Output} \)
   \( \text{Note: the dimension, dim, of the array ob must be at least} \)
   \( \text{pdob} \times \text{my when orig = Nag\_EstimatesOrig and order = Nag\_ColMajor;} \)
   \( \max(1, (\text{ip} + 1) \times \text{pdob}) \) \( \text{when orig = Nag\_EstimatesOrig and order = Nag\_RowMajor;} \)
   \( 1 \) \( \text{otherwise.} \)

Where \( \text{OB}(i, j) \) appears in this document, it refers to the array element
\( \text{ob}((j - 1) \times \text{pdob} + i - 1) \) \( \text{when order = Nag\_ColMajor;} \)
\( \text{ob}((i - 1) \times \text{pdob} + j - 1) \) \( \text{when order = Nag\_RowMajor.} \)

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

22: \( \text{pdob} \) – Integer
   \( \text{Input} \)
   \( \text{On entry: the stride separating row or column elements (depending on the value of order) in the array ob.} \)
   \( \text{Constraints:} \)
   \( \text{if order = Nag\_ColMajor,} \)
   \( \text{if orig = Nag\_EstimatesOrig, pdob} \geq \text{ip} + 1; \)
   \( \text{otherwise pdob} \geq 1; \)
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: ycv[dim] – const double
   Input
   Note: the dimension, dim, of the array ycv must be at least my when vipopt ≠ 0.
   Where YCV(i, j) appears in this document, it refers to the array element
   ycv[(j - 1) × pdycv + i - 1] when order = Nag_ColMajor;
   ycv[(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 jth response variable explained by the first i factors, for i = 1, 2, ..., nfact and j = 1, 2, ..., my; otherwise ycv 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 ycv.

   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 ith 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 ith predictor variable in the model for the jth 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 ≠ 0, pdvip ≥ ip;
    if order = Nag_RowMajor, pdvip ≥ vipopt.

28:  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.
See Section 3.2.1.2 in the Essential Introduction for further information.

**NE_BAD_PARAM**

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

**NE_ENUM_INT**

On entry, iscale = ⟨value⟩.
Constraint: if orig = Nag_EstimatesOrig, iscale = Nag_PredNoScale or Nag_PredStdScale.

On entry, orig = ⟨value⟩ and my = ⟨value⟩.
Constraint: my > 0.

**NE_ENUM_INT_2**

On entry, orig = ⟨value⟩, pdb = ⟨value⟩, my = ⟨value⟩.
Constraint: if orig = Nag_EstimatesOrig, pdb ≥ my;
otherwise pdb ≥ 1.

**NE_INT**

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

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

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

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

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

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

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

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

On entry, pdycv = ⟨value⟩.
Constraint: pdycv > 0.
On entry, \(\text{maxfac} = \langle\text{value}\rangle\) and \(\text{ip} = \langle\text{value}\rangle\).
Constraint: \(1 \leq \text{maxfac} \leq \text{ip}\).

On entry, \(\text{nfact} = \langle\text{value}\rangle\) and \(\text{maxfac} = \langle\text{value}\rangle\).
Constraint: \(1 \leq \text{nfact} \leq \text{maxfac}\).

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

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

On entry, \(\text{pdc} = \langle\text{value}\rangle\) and \(\text{maxfac} = \langle\text{value}\rangle\).
Constraint: \(\text{pdc} \geq \text{maxfac}\).

On entry, \(\text{pdc} = \langle\text{value}\rangle\) and \(\text{my} = \langle\text{value}\rangle\).
Constraint: \(\text{pdc} \geq \text{my}\).

On entry, \(\text{pdob} = \langle\text{value}\rangle\) and \(\text{ip} = \langle\text{value}\rangle\).
Constraint: if \(\text{orig} = \text{Nag_EstimatesOrig}\), \(\text{pdob} \geq \text{ip} + 1\).

On entry, \(\text{pdp} = \langle\text{value}\rangle\) and \(\text{ip} = \langle\text{value}\rangle\).
Constraint: \(\text{pdp} \geq \text{ip}\).

On entry, \(\text{pdp} = \langle\text{value}\rangle\) and \(\text{maxfac} = \langle\text{value}\rangle\).
Constraint: \(\text{pdp} \geq \text{maxfac}\).

On entry, \(\text{pdvip} = \langle\text{value}\rangle\) and \(\text{ip} = \langle\text{value}\rangle\).
Constraint: if \(\text{vipopt} \neq 0\), \(\text{pdvip} \geq \text{ip}\).

On entry, \(\text{pdvip} = \langle\text{value}\rangle\) and \(\text{vipopt} = \langle\text{value}\rangle\).
Constraint: \(\text{pdvip} \geq \text{vipopt}\).

On entry, \(\text{pdw} = \langle\text{value}\rangle\) and \(\text{ip} = \langle\text{value}\rangle\).
Constraint: \(\text{pdw} \geq \text{ip}\).

On entry, \(\text{pdw} = \langle\text{value}\rangle\) and \(\text{maxfac} = \langle\text{value}\rangle\).
Constraint: \(\text{pdw} \geq \text{maxfac}\).

On entry, \(\text{pdycv} = \langle\text{value}\rangle\) and \(\text{nfact} = \langle\text{value}\rangle\).
Constraint: if \(\text{vipopt} \neq 0\), \(\text{pdycv} \geq \text{nfact}\).

On entry, \(\text{vipopt} = \langle\text{value}\rangle\) and \(\text{my} = \langle\text{value}\rangle\).
Constraint: \(\text{my} > 0\).

On entry, \(\text{vipopt} = \langle\text{value}\rangle\) and \(\text{my} = \langle\text{value}\rangle\).
Constraint: \(\text{vipopt} = 0, 1, \text{or my}\).

On entry, \(\text{pdycv} = \langle\text{value}\rangle\), \(\text{vipopt} = \langle\text{value}\rangle\) and \(\text{my} = \langle\text{value}\rangle\).
Constraint: if \(\text{vipopt} \neq 0\), \(\text{pdycv} \geq \text{my}\).

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.

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 calculations are based on the singular value decomposition of $P^TW$.

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

```c
/* nag_pls_orth_scores_fit (g02lcc) Example Program. */
* * Copyright 2014 Numerical Algorithms Group. *
* * Mark 9, 2009. */
#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 exit_status = 0;
    Integer i, ip, ip1, j, maxfac, my, nfact, vipopt;
    Integer pdb, pdc, pdob, pdp, pdvip, pdw, pdycv;
    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;
    char siscale[40], sorig[40];
    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
        
```
```c
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,
    _countof(sorig), siscale, _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);
#endif

pdb = ip;
#define B(I, J) b[(J-1)*pdb + I-1]
pdc = my;
#define C(I, J) c[(J-1)*pdc + I-1]
pdob = ip+1;
#define OB(I, J) ob[(J-1)*pdob + I-1]
pdp = ip;
#define P(I, J) p[(J-1)*pdp + I-1]
#define VIP(I, J) vip[(J-1)*pdvip + I-1]
#define W(I, J) w[(J-1)*pdw + I-1]
#define YCV(I, J) ycv[(J-1)*pdycv + I-1]
order = Nag_ColMajor;
#else
pdb = my;
#define B(I, J) b[(I-1)*pdb + J-1]
pdc = maxfac;
#define C(I, J) c[(I-1)*pdc + J-1]
pdob = my;
#define OB(I, J) ob[(I-1)*pdob + J-1]
pdp = maxfac;
#define P(I, J) p[(I-1)*pdp + J-1]
#define VIP(I, J) vip[(I-1)*pdvip + J-1]
#define W(I, J) w[(I-1)*pdw + J-1]
#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+1):my),
           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)) ||
    !(ystd = NAG_ALLOC(my, double)) ||
    !(ybar = NAG_ALLOC(my, double)))
{
  printf("Allocation failure\n");
  exit_status = -1;
goto END;
}
/* Read P*/
for (i = 1; i <= ip; i++)
```

Mark 25 g02lcc.9
for (j = 1; j <= maxfac; j++)
#endif
#endif
#define _WIN32
scanf_s("%lf ", &P(i, j));
#else
scanf("%lf ", &P(i, j));
#define _WIN32
#endif
#define _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif
/* Read C*/
for (i = 1; i <= my; i++)
{
for (j = 1; j <= maxfac; j++)
#endif
#endif
#define _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif
/* Read W*/
for (i = 1; i <= ip; i++)
{
for (j = 1; j <= maxfac; j++)
#endif
#endif
#define _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif
/* Read YCV*/
for (i = 1; i <= maxfac; i++)
{
for (j = 1; j <= my; j++)
#endif
#endif
#define _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif
/* Read means*/
if (orig == Nag_EstimatesOrig)
{
for (j = 0; j < ip; j++)
#endif
#endif
#define _WIN32
scanf_s("%lf ", &xbar[j]);
#else
scanf("%lf ", &xbar[j]);
#endif
#endif
#define _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif
#endif
for (j = 0; j < my; j++)
#endif
#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++)
#endif
#ifdef _WIN32
scanf_s("%lf ", &xstd[j]);
#else
scanf("%lf ", &xstd[j]);
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
}
/* Calculate predictions*/
rcond = -1.00e0;
#endif
#ifdef _WIN32
scanf_s("%lf ", &ystd[j]);
#else
scanf("%lf ", &ystd[j]);
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
if (iscale != Nag_PredNoScale)
{
  for (j = 0; j < ip; j++)
#endif
#ifdef _WIN32
scanf_s("%lf ", &xstd[j]);
#else
scanf("%lf ", &xstd[j]);
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
}
/* Calculate predictions*/
rcond = -1.00e0;
#endif
#ifdef _WIN32
scanf_s("%lf ", &ystd[j]);
#else
scanf("%lf ", &ystd[j]);
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
}
/* Calculate predictions*/
rcond = -1.00e0;
#endif
#ifdef _WIN32
scanf_s("%lf ", &ystd[j]);
#else
scanf("%lf ", &ystd[j]);
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
}
/* Calculate predictions*/
rcond = -1.00e0;
#endif
#ifdef _WIN32
scanf_s("%lf ", &ystd[j]);
#else
scanf("%lf ", &ystd[j]);
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
if (iscale != Nag_PredNoScale)
{
  for (j = 0; j < ip; j++)
#endif
#ifdef _WIN32
scanf_s("%lf ", &xstd[j]);
#else
scanf("%lf ", &xstd[j]);
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
}
/* Calculate predictions*/
rcond = -1.00e0;
#endif
#ifdef _WIN32
scanf_s("%lf ", &ystd[j]);
#else
scanf("%lf ", &ystd[j]);
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
}
/* Calculate predictions*/
rcond = -1.00e0;
#endif
#ifdef _WIN32
scanf_s("%lf ", &ystd[j]);
#else
scanf("%lf ", &ystd[j]);
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
}
/* Calculate predictions*/
rcond = -1.00e0;
#endif
#ifdef _WIN32
scanf_s("%lf ", &ystd[j]);
#else
scanf("%lf ", &ystd[j]);
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
#endif
#ifdef _WIN32
scanf_s("%*[\n "");
#else
scanf("%*[\n "");
#endif
}
{ /* 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, pdb, "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_orths_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
10.3 Program Results

nag_pls_orth_scores_fit (g02lcc) Example Program Results

B
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  -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   0.6111
<table>
<thead>
<tr>
<th>#</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.3182</td>
</tr>
<tr>
<td>3</td>
<td>0.7513</td>
</tr>
<tr>
<td>4</td>
<td>0.5048</td>
</tr>
<tr>
<td>5</td>
<td>0.2712</td>
</tr>
<tr>
<td>6</td>
<td>0.3593</td>
</tr>
<tr>
<td>7</td>
<td>1.5777</td>
</tr>
<tr>
<td>8</td>
<td>2.4348</td>
</tr>
<tr>
<td>9</td>
<td>1.1322</td>
</tr>
<tr>
<td>10</td>
<td>1.2226</td>
</tr>
<tr>
<td>11</td>
<td>1.1799</td>
</tr>
<tr>
<td>12</td>
<td>0.8840</td>
</tr>
<tr>
<td>13</td>
<td>0.2129</td>
</tr>
<tr>
<td>14</td>
<td>0.2129</td>
</tr>
<tr>
<td>15</td>
<td>0.2129</td>
</tr>
</tbody>
</table>