

NAG Library Function Document

nag_pls_orth_scores_pred (g02ldc)

1 Purpose

nag_pls_orth_scores_pred (g02ldc) calculates predictions given the output from an orthogonal scores PLS regression (nag_pls_orth_scores_svd (g02lac) or nag_pls_orth_scores_wold (g02lbc)) and nag_pls_orth_scores_fit (g02lcc).

2 Specification

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

```
void nag_pls_orth_scores_pred (Nag_OrderType order, Integer ip, Integer my,
    Nag_EstimatesOption orig, const double xbar[], const double ybar[],
    Nag_ScalePredictor iscale, const double xstd[], const double ystd[],
    const double b[], Integer pdb, Integer n, Integer mz, const Integer isz[],
    const double z[], Integer pdz, double yhat[], Integer pdyhat, NagError *fail)
```

3 Description

nag_pls_orth_scores_pred (g02ldc) calculates the predictions \hat{Y} of a PLS model given a set Z of test data and a set B of parameter estimates as returned by nag_pls_orth_scores_fit (g02lcc).

If nag_pls_orth_scores_fit (g02lcc) returns parameter estimates for the original data scale, no further information is required.

If nag_pls_orth_scores_fit (g02lcc) returns parameter estimates for the centred, and possibly scaled, data, further information is required. The means of variables in the fitted model must be supplied. In the case of a PLS model fitted by using scaled data, the standard deviations of variables in the fitted model must also be supplied. These means and standard deviations are those returned by either nag_pls_orth_scores_svd (g02lac) and nag_pls_orth_scores_wold (g02lbc).

4 References

None.

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: **ip** – Integer *Input*
On entry: the number of predictor variables in the fitted model. **ip** must take the same value as that supplied to nag_pls_orth_scores_svd (g02lac) or nag_pls_orth_scores_wold (g02lbc) to fit the model.
Constraint: **ip** > 1.

- 3: **my** – Integer *Input*
On entry: the number of response variables in the fitted model. **my** must take the same value as that supplied to `nag_pls_orth_scores_svd` (g02lac) or `nag_pls_orth_scores_wold` (g02lbc) to fit the model.
Constraint: **my** ≥ 1 .
- 4: **orig** – Nag_EstimatesOption *Input*
On entry: indicates how parameter estimates are supplied.
orig = Nag_EstimatesOrig
Parameter estimates are for the original data.
orig = Nag_EstimatesStand
Parameter estimates are for the centred, and possibly scaled, data.
Constraint: **orig** = Nag_EstimatesStand or Nag_EstimatesOrig.
- 5: **xbar[ip]** – const double *Input*
On entry: if **orig** = Nag_EstimatesStand, **xbar** must contain mean values of predictor variables in the model; otherwise **xbar** is not referenced.
- 6: **ybar[my]** – const double *Input*
On entry: if **orig** = Nag_EstimatesStand, **ybar** must contain the mean value of each response variable in the model; otherwise **ybar** is not referenced.
- 7: **iscale** – Nag_ScalePredictor *Input*
On entry: if **orig** = Nag_EstimatesStand, **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_EstimatesStand, **iscale** = Nag_PredNoScale, Nag_PredStdScale or Nag_PredUserScale.
- 8: **xstd[ip]** – const double *Input*
On entry: if **orig** = Nag_EstimatesStand and **iscale** \neq Nag_PredNoScale, **xstd** must contain 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.
- 9: **ystd[my]** – const double *Input*
On entry: if **orig** = Nag_EstimatesStand and **iscale** \neq Nag_PredNoScale, **ystd** must contain 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.
- 10: **b[dim]** – const double *Input*
Note: the dimension, *dim*, of the array **b** must be at least
 $\max(1, \mathbf{pdb} \times \mathbf{my})$ when **order** = Nag_ColMajor;
 $\max(1, \times \mathbf{pdb})$ when **order** = Nag_RowMajor and **iporig** = Nag_EstimatesStand;
 $\max(1, \times \mathbf{pdb})$ when **order** = Nag_RowMajor and $1 + \mathbf{iporig} = \text{Nag_EstimatesOrig}$.
The (*i*, *j*)th element of the matrix *B* is stored in
 $\mathbf{b}[(j - 1) \times \mathbf{pdb} + i - 1]$ when **order** = Nag_ColMajor;
 $\mathbf{b}[(i - 1) \times \mathbf{pdb} + j - 1]$ when **order** = Nag_RowMajor.
On entry: if **orig** = Nag_EstimatesStand, **b** must contain the parameter estimate for the centred, and possibly scaled, data as returned by `nag_pls_orth_scores_fit` (g02lcc); otherwise **b** must contain the parameter estimates for the original data as returned by `nag_pls_orth_scores_fit` (g02lcc).

- 11: **pdb** – Integer *Input*
On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **b**.
Constraints:
 if **order** = Nag_ColMajor,
 if **orig** = Nag_EstimatesStand, **pdb** ≥ **ip**;
 if **orig** = Nag_EstimatesOrig, **pdb** ≥ 1 + **ip**;
 if **order** = Nag_RowMajor, **pdb** ≥ **my**.
- 12: **n** – Integer *Input*
On entry: *n*, the number of observations in the test data *Z*.
Constraint: **n** ≥ 1.
- 13: **mz** – Integer *Input*
On entry: the number of available predictor variables in the test data.
Constraint: **mz** ≥ **ip**.
- 14: **isz[mz]** – const Integer *Input*
On entry: indicates which predictor variables are to be included in the model. Predictor variables included from **z** must be in the same order as those included in the fitted model.
 If **isz**[*j* – 1] = 1, the *j*th predictor variable is included in the model, for *j* = 1, 2, . . . , **mz**, otherwise **isz**[*j* – 1] = 0.
Constraints:

$$\mathbf{isz}[j - 1] = 0 \text{ or } 1, \text{ for } j = 1, 2, \dots, \mathbf{mz};$$

$$\sum_j \mathbf{isz}[j - 1] = \mathbf{ip}.$$
- 15: **z[dim]** – const double *Input*
Note: the dimension, *dim*, of the array **z** must be at least
 $\max(1, \mathbf{pdz} \times \mathbf{mz})$ when **order** = Nag_ColMajor;
 $\max(1, \mathbf{n} \times \mathbf{pdz})$ when **order** = Nag_RowMajor.
 Where **Z**(*i*, *j*) appears in this document, it refers to the array element
 $\mathbf{z}[(j - 1) \times \mathbf{pdz} + i - 1]$ when **order** = Nag_ColMajor;
 $\mathbf{z}[(i - 1) \times \mathbf{pdz} + j - 1]$ when **order** = Nag_RowMajor.
On entry: **Z**(*i*, *j*) contains the *i*th observation on the *j*th available predictor variable, for *i* = 1, 2, . . . , **n** and *j* = 1, 2, . . . , **mz**.
- 16: **pdz** – Integer *Input*
On entry: the stride separating matrix row or column elements (depending on the value of **order**) in **z**.
Constraints:
 if **order** = Nag_ColMajor, **pdz** ≥ **n**;
 if **order** = Nag_RowMajor, **pdz** ≥ **mz**.
- 17: **yhat[dim]** – double *Output*
Note: the dimension, *dim*, of the array **yhat** must be at least
 $\max(1, \mathbf{pdyhat} \times \mathbf{my})$ when **order** = Nag_ColMajor;
 $\max(1, \mathbf{n} \times \mathbf{pdyhat})$ when **order** = Nag_RowMajor.

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

$$\begin{aligned} & \mathbf{yhat}[(j-1) \times \mathbf{pdyhat} + i - 1] \text{ when } \mathbf{order} = \mathbf{Nag_ColMajor}; \\ & \mathbf{yhat}[(i-1) \times \mathbf{pdyhat} + j - 1] \text{ when } \mathbf{order} = \mathbf{Nag_RowMajor}. \end{aligned}$$

On exit: $\mathbf{YHAT}(i, j)$ contains the i th predicted value of the j th y -variable in the model.

18: **pdyhat** – Integer *Input*

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

Constraints:

$$\begin{aligned} & \text{if } \mathbf{order} = \mathbf{Nag_ColMajor}, \mathbf{pdyhat} \geq \mathbf{n}; \\ & \text{if } \mathbf{order} = \mathbf{Nag_RowMajor}, \mathbf{pdyhat} \geq \mathbf{my}. \end{aligned}$$

19: **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 $\langle value \rangle$ had an illegal value.

NE_ENUM_CHARACTER

On entry, **iscale** = $\langle value \rangle$.

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

NE_INT

On entry, **ip** = $\langle value \rangle$.

Constraint: **ip** > 1.

On entry, **my** = $\langle value \rangle$.

Constraint: **my** ≥ 1.

On entry, **n** = $\langle value \rangle$.

Constraint: **n** ≥ 1.

On entry, **pdb** = $\langle value \rangle$.

Constraint: **pdb** > 0.

On entry, **pdyhat** = $\langle value \rangle$.

Constraint: **pdyhat** > 0.

On entry, **pdz** = $\langle value \rangle$.

Constraint: **pdz** > 0.

NE_INT_2

On entry, **mz** = $\langle value \rangle$, **ip** = $\langle value \rangle$.

Constraint: **mz** ≥ **ip**.

On entry, **pdb** = $\langle value \rangle$, **ip** + 1 = $\langle value \rangle$.

Constraint: if **orig** = Nag_EstimatesOrig, **pdb** ≥ 1 + **ip**.

On entry, **pdb** = $\langle value \rangle$, **ip** = $\langle value \rangle$.

Constraint: if **orig** = Nag_EstimatesStand, **pdb** ≥ **ip**.

On entry, **pdb** = $\langle value \rangle$, **my** = $\langle value \rangle$.

Constraint: **pdb** \geq **my**.

On entry, **pdihat** = $\langle value \rangle$, **my** = $\langle value \rangle$.

Constraint: **pdihat** \geq **my**.

On entry, **pdihat** = $\langle value \rangle$, **n** = $\langle value \rangle$.

Constraint: **pdihat** \geq **n**.

On entry, **pdz** = $\langle value \rangle$, **mz** = $\langle value \rangle$.

Constraint: **pdz** \geq **mz**.

On entry, **pdz** = $\langle value \rangle$, **n** = $\langle value \rangle$.

Constraint: **pdz** \geq **n**.

NE_INT_ARG_CONS

On entry, the number of elements of **isz** equal to 1 is not **ip**.

NE_INT_ARRAY_VAL_1_OR_2

On entry, **isz**[$j - 1$] = $\langle value \rangle$, $j = \langle value \rangle$.

Constraint: **isz**[$j - 1$] = 0 or 1.

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.

7 Accuracy

Not applicable.

8 Further Comments

nag_pls_orth_scores_pred (g02ldc) allocates internally $3 \times \mathbf{ip} + \mathbf{my}$ elements of double storage.

9 Example

This example reads in parameter estimates for a fitted PLS model and prediction data, and the PLS model predictions are calculated.

9.1 Program Text

```

/* nag_pls_orth_scores_pred (g02ldc) 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, l, my, mz, n;

```

```

Integer          pdb, pdyhat, pdz;
Integer          *isz = 0;
/*Double scalar and array declarations */
double          *b = 0, *xbar = 0, *xstd = 0, *ybar = 0, *yhat = 0;
double          *ystd = 0, *z = 0;
/*Character scalar and array declarations */
char            siscale[18], sorig[24];
/*NAG Types */
Nag_OrderType   order;
Nag_ScalePredictor iscale;
Nag_EstimatesOption orig;
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_pred (g02ldc) Example Program Results");
/* Skip header in data file.*/
fscanf(fpin, "%*[\n] ");
/* Read data values.*/
fscanf(fpin, "%ld%ld%s %s %ld%ld%*[\n] ",
        &ip, &my, sorig, siscale, &n, &mz);
orig = (Nag_EstimatesOption) nag_enum_name_to_value(sorig);
iscale = (Nag_ScalePredictor) nag_enum_name_to_value(siscale);

#ifdef NAG_COLUMN_MAJOR
pdb = ((orig == Nag_EstimatesStand)?ip:l+ip);
#define B(I, J)    b[(J-1)*pdb + I-1]
pdyhat = n;
#define YHAT(I, J) yhat[(J-1)*pdyhat + I-1]
pdz = n;
#define Z(I, J)    z[(J-1)*pdz + I-1]
order = Nag_ColMajor;
#else
pdb = my;
#define B(I, J)    b[(I-1)*pdb + J-1]
pdyhat = my;
#define YHAT(I, J) yhat[(I-1)*pdyhat + J-1]
pdz = mz;
#define Z(I, J)    z[(I-1)*pdz + J-1]
order = Nag_RowMajor;
#endif
if (!(b = NAG_ALLOC(pdb*(order == Nag_RowMajor?(l+ip):my), double)) ||
    !(xbar = NAG_ALLOC(ip, double)) ||
    !(xstd = NAG_ALLOC(ip, double)) ||
    !(ybar = NAG_ALLOC(my, double)) ||
    !(yhat = NAG_ALLOC(pdyhat*(order == Nag_RowMajor?n:my), double)) ||
    !(ystd = NAG_ALLOC(my, double)) ||
    !(z = NAG_ALLOC(pdz*(order == Nag_RowMajor?n:mz), double)) ||
    !(isz = NAG_ALLOC(mz, Integer)))
{
    fprintf(fpout, "Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read prediction x-data*/
for (i = 1; i <= n; i++)
{
    for (j = 1; j <= mz; j++)
        fscanf(fpin, "%lf ", &Z(i, j));
}
fscanf(fpin, "%*[\n] ");
/* Read elements of isz*/
for (j = 0; j < mz; j++)
    fscanf(fpin, "%ld ", &isz[j]);
fscanf(fpin, "%*[\n] ");

```

```

/* Read parameter estimates*/
l = ip;
if (orig != Nag_EstimatesStand)
{
    l = l+1;
}
for (j = 1; j <= l; j++)
{
    for (i = 1; i <= my; i++)
        fscanf(fpin, "%lf ", &B(j, i));
}
fscanf(fpin, "%*[\n] ");
/* Read means*/
if (orig == Nag_EstimatesStand)
{
    for (j = 0; j < ip; j++)
        fscanf(fpin, "%lf ", &xbar[j]);
    fscanf(fpin, "%*[\n] ");
    for (l = 0; l < my; l++)
        fscanf(fpin, "%lf ", &ybar[l]);
    fscanf(fpin, "%*[\n] ");
}
/* Read scalings*/
if ((orig == Nag_EstimatesStand) && (iscale != Nag_PredNoScale))
{
    for (j = 0; j < ip; j++)
        fscanf(fpin, "%lf ", &xstd[j]);
    fscanf(fpin, "%*[\n] ");
    for (l = 0; l < my; l++)
        fscanf(fpin, "%lf ", &ystd[l]);
    fscanf(fpin, "%*[\n] ");
}
/* Calculate predictions*/
/*
 * nag_pls_orth_scores_pred (g02ldc)
 * Partial least-squares
 */
nag_pls_orth_scores_pred(order, ip, my, orig, xbar, ybar, iscale, xstd,
                        ystd, b, pdb, n, mz, isz, z, pdz, yhat, pdyhat,
                        &fail);
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_pls_orth_scores_pred (g02ldc).\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, my,
                        yhat, pdyhat, "YHAT", 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;
}

END:
if (fpin != stdin) fclose(fpin);
if (fpout != stdout) fclose(fpout);
if (b) NAG_FREE(b);

```

```

    if (xbar) NAG_FREE(xbar);
    if (xstd) NAG_FREE(xstd);
    if (ybar) NAG_FREE(ybar);
    if (yhat) NAG_FREE(yhat);
    if (ystd) NAG_FREE(ystd);
    if (z) NAG_FREE(z);
    if (isz) NAG_FREE(isz);

    return exit_status;
}

```

9.2 Program Data

```

nag_pls_orth_scores_pred (g02ldc) Example Program Data
15 1 Nag_EstimatesStand Nag_PredStdScale 15 15 : ip, my, orig, iscale, n, mz
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701
 1.9607 -1.6324 0.5746 1.9607 -1.6324 0.5740
 2.8369 1.4092 -3.1398
-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
-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
-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
-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
-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
-2.6931 -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
-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
-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
 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
-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
-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
-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
-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
-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 : z
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 : Elements of isz
-0.1383 0.0572 -0.1906 0.1238 0.0591 0.0936
-0.2842 0.4713 0.2661 -0.0914 0.1226 -0.0488
 0.0332 0.0332 -0.0332 : b
-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
```

9.3 Program Results

```
nag_pls_orth_scores_pred (g02ldc) Example Program Results
YHAT
```

```
1
1      0.2132
2      0.5152
3      0.1437
4      0.4459
5      0.1716
6      2.4809
7      0.0964
8      1.4475
9      -0.1546
10     -0.5492
11     0.5393
12     0.2686
13     -1.1332
14     1.7975
15     0.4973
```
