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 means and 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

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

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.
my – Integer

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.

orig – Nag_EstimatesOption

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.

xbar[ip] – const double

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

ybar[my] – const double

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

iscale – Nag_ScalePredictor

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.

xstd[ip] – const double

On entry: if orig = Nag_EstimatesStand and iscale ≠ 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.

ystd[my] – const double

On entry: if orig = Nag_EstimatesStand and iscale ≠ 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.

b[dim] – const 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 and orig = Nag_EstimatesStand;

max(1, 1 + ip × pdb) when order = Nag_RowMajor and orig = Nag_EstimatesOrig.

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

b[(j - 1) × pdb + i - 1] when order = Nag_ColMajor;

b[(i - 1) × 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 row or column elements (depending on the value of `order`) in the array `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:*  
   - `isz[j - 1]` = 0 or 1, for `j` = 1, 2, …, `mz`;  
   - \[ \sum_j isz[j - 1] = ip. \]

15: **z[dim]** – const double  
   *Input*  
   *Note:* the dimension, `dim`, of the array `z` must be at least  
   - `max(1, pdz * mz)` when `order` = Nag_ColMajor;  
   - `max(1, n * pdz)` when `order` = Nag_RowMajor.  
   Where `Z(i, j)` appears in this document, it refers to the array element  
   - `z[(j - 1) * pdz + i - 1]` when `order` = Nag_ColMajor;  
   - `z[(i - 1) * 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 row or column elements (depending on the value of `order`) in the array `z`.  
   *Constraints:*  
   - if `order` = Nag_ColMajor, `pdz` ≥ `n`;  
   - if `order` = Nag_RowMajor, `pdz` ≥ `mz`.  

Mark 25
17: \texttt{yhat}[\texttt{dim}] \textendash \textit{double} \quad \textit{Output}

\textbf{Note:} the dimension, \texttt{dim}, of the array \texttt{yhat} must be at least
\begin{align*}
\max(1, \texttt{pdyhat} \times \texttt{my}) & \quad \text{when } \texttt{order} = \text{Nag\_ColMajor}; \\
\max(1, \texttt{n} \times \texttt{pdyhat}) & \quad \text{when } \texttt{order} = \text{Nag\_RowMajor}.
\end{align*}

Where \texttt{YHAT}(i, j) appears in this document, it refers to the array element
\begin{align*}
\texttt{yhat}[(j - 1) \times \texttt{pdyhat} + i - 1] & \quad \text{when } \texttt{order} = \text{Nag\_ColMajor}; \\
\texttt{yhat}[(i - 1) \times \texttt{pdyhat} + j - 1] & \quad \text{when } \texttt{order} = \text{Nag\_RowMajor}.
\end{align*}

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

18: \texttt{pdyhat} \textendash \textit{Integer} \quad \textit{Input}

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

\textbf{Constraints:}
\begin{align*}
\text{if } \texttt{order} = \text{Nag\_ColMajor}, \texttt{pdyhat} & \geq \texttt{n}; \\
\text{if } \texttt{order} = \text{Nag\_RowMajor}, \texttt{pdyhat} & \geq \texttt{my}.
\end{align*}

19: \texttt{fail} \textendash \textit{NagError *} \quad \textit{Input/Output}

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

6 \quad \textbf{Error Indicators and Warnings}

\texttt{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

\texttt{NE\_BAD\_PARAM}

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

\texttt{NE\_ENUM\_CHARACTER}

On entry, \texttt{iscale} = \langle value \rangle.
Constraint: if \texttt{orig} = \text{Nag\_EstimatesStand}, \texttt{iscale} = \text{Nag\_PredNoScale}, \text{Nag\_PredStdScale} or \text{Nag\_PredUserScale}.

\texttt{NE\_INT}

On entry, \texttt{ip} = \langle value \rangle.
Constraint: \texttt{ip} > 1.

On entry, \texttt{my} = \langle value \rangle.
Constraint: \texttt{my} \geq 1.

On entry, \texttt{n} = \langle value \rangle.
Constraint: \texttt{n} \geq 1.

On entry, \texttt{pdb} = \langle value \rangle.
Constraint: \texttt{pdb} > 0.

On entry, \texttt{pdyhat} = \langle value \rangle.
Constraint: \texttt{pdyhat} > 0.

On entry, \texttt{pdz} = \langle value \rangle.
Constraint: \texttt{pdz} > 0.
NE_INT_2
On entry, \( mz = \langle \text{value} \rangle \) and \( ip = \langle \text{value} \rangle \).
Constraint: \( mz \geq ip \).
On entry, \( pdb = \langle \text{value} \rangle \) and \( ip + 1 = \langle \text{value} \rangle \).
Constraint: if \( \text{orig} = \text{Nag}_{\text{EstimatesOrig}} \), \( pdb \geq 1 + ip \).
On entry, \( pdb = \langle \text{value} \rangle \) and \( ip = \langle \text{value} \rangle \).
Constraint: if \( \text{orig} = \text{Nag}_{\text{EstimatesStand}} \), \( pdb \geq ip \).
On entry, \( pdb = \langle \text{value} \rangle \) and \( my = \langle \text{value} \rangle \).
Constraint: \( pdb \geq my \).
On entry, \( pdyhat = \langle \text{value} \rangle \) and \( my = \langle \text{value} \rangle \).
Constraint: \( pdyhat \geq my \).
On entry, \( pdyhat = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( pdyhat \geq n \).
On entry, \( pdz = \langle \text{value} \rangle \) and \( mz = \langle \text{value} \rangle \).
Constraint: \( pdz \geq mz \).
On entry, \( pdz = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
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, \( isz[j-1] = \langle \text{value} \rangle \), \( j = \langle \text{value} \rangle \).
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.
An unexpected error has been triggered by this function. Please contact NAG. See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE
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
Not applicable.

8 Parallelism and Performance
\text{g02ldc} \text{ is not threaded by NAG in any implementation.}
\text{g02ldc} \text{ 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_pred (g02ldc)` allocates internally $3 \times \text{ip} + \text{my}$ elements of double storage.

10 Example

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

10.1 Program Text

```c
/* nag_pls_orth_scores_pred (g02ldc) Example Program. */
* Copyright 2014 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(void)
{
    /*Integer scalar and array declarations */
    Integer exit_status = 0;
    Integer i, ip, j, i, 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[40], sorig[40];
    /*NAG Types */
    Nag_OrderType order;
    Nag_ScalePredictor iscale;
    Nag_EstimatesOption orig;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_pls_orth_scores_pred (g02ldc) 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%39s %39s %NAG_IFMT%NAG_IFMT%[*\n] ",
            &ip, &my, sorig, _countof(sorig), siscale, _countof(siscale), &n, &mz);
    #else
        scanf("%NAG_IFMT%NAG_IFMT%39s %39s %NAG_IFMT%NAG_IFMT%[*\n] ",
            &ip, &my, sorig, isiscale, &n, &mz);
    #endif
    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:1+ip);
        define B(I, J) b[(J-1)*pdb + I-1]
        pdyhat = n;
        define YHAT(I, J) yhat[(J-1)*pdyhat + I-1]
    #endif
```
pdz = n;
  #define Z(I, J) z[(I-1)*pdz + J-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[(J-1)*pdz + I-1]
#endif
if (!(b = NAG_ALLOC(pdb*(order == Nag_RowMajor?(1+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)))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}
/* Read prediction x-data*/
for (i = 1; i <= n; i++)
  {for (j = 1; j <= mz; j++)
#define _WIN32
  scanf_s("%lf ", &Z(i, j));
#define "%lf ", &Z(i, j));
#else
  scanf("%lf ", &Z(i, j));
#endif
}  
#endif _WIN32
  scanf_s("%*[\n] ");
#else
  scanf("%*[\n] ");
#endif /* Read elements of isz*/
for (j = 0; j < mz; j++)
#define _WIN32
  scanf_s("%NAG_IFMT" , &isz[j]);
#else
  scanf("%NAG_IFMT" , &isz[j]);
#endif
#else _WIN32
  scanf_s("%*[\n] ");
#else
  scanf("%*[\n] ");
#endif /* Read parameter estimates*/
l = ip;
if (orig != Nag_EstimatesStand)
  {l = l+1;
  for (j = 1; j <= l; j++)
    {for (i = 1; i <= my; i++)
#define _WIN32
    scanf_s("%lf ", &B(j, i));
#else
    scanf("%lf ", &B(j, i));
#endif
}  
#endif _WIN32
  scanf_s("%*[\n] ");
#else
  scanf("%*[\n] ");
#endif
/* Read means*/
if (orig == Nag_EstimatesStand)
{
    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 (l = 0; l < my; l++)
        #ifdef _WIN32
            scanf_s("%lf ", &ybar[l]);
        #else
            scanf("%lf ", &ybar[l]);
        #endif
        #ifdef _WIN32
            scanf_s("%*[\n] ");
        #else
            scanf("%*[\n] ");
        #endif
}
/* Read scalings*/
if ((orig == Nag_EstimatesStand) && (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 (l = 0; l < my; l++)
        #ifdef _WIN32
            scanf_s("%lf ", &ystd[l]);
        #else
            scanf("%lf ", &ystd[l]);
        #endif
        #ifdef _WIN32
            scanf_s("%*[\n] ");
        #else
            scanf("%*[\n] ");
        #endif
}
/* 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, pdz, yhat, pdyhat, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_pls_orth_scores_pred (g02ldc).\n\n\n", fail.message);
    exit_status = 1;
    goto END;
}
/*
 * nag_gen_real_mat_print (x04cac)
 * Print real general matrix (easy-to-use)
flush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, my, 
yhat, pdyhat, "YHAT", 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(xbar);
NAG_FREE(xstd);
NAG_FREE(ybar);
NAG_FREE(yhat);
NAG_FREE(ystd);
NAG_FREE(z);
NAG_FREE(isz);
return exit_status;
}
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.8369</td>
<td>1.4092</td>
<td>-3.1398</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-2.6931</td>
<td>-2.5721</td>
<td>-1.2871</td>
<td>3.0777</td>
<td>0.3891</td>
<td>-0.0701</td>
</tr>
<tr>
<td>3</td>
<td>-4.1921</td>
<td>-1.0285</td>
<td>-0.9801</td>
<td>0.0744</td>
<td>-1.7333</td>
<td>0.0902</td>
</tr>
<tr>
<td>4</td>
<td>2.8369</td>
<td>1.4092</td>
<td>-3.1398</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Elements of isz**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.1383</td>
<td>0.0572</td>
<td>-0.1906</td>
<td>0.1238</td>
<td>0.0591</td>
<td>0.0936</td>
</tr>
<tr>
<td>2</td>
<td>-0.2842</td>
<td>0.4713</td>
<td>0.2661</td>
<td>-0.0914</td>
<td>0.1226</td>
<td>-0.0488</td>
</tr>
<tr>
<td>3</td>
<td>0.0332</td>
<td>0.0332</td>
<td>-0.0332</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-2.6137</td>
<td>-2.3614</td>
<td>-1.0449</td>
<td>2.8614</td>
<td>0.3156</td>
<td>-0.2641</td>
</tr>
<tr>
<td>5</td>
<td>-0.3146</td>
<td>-1.1221</td>
<td>0.2401</td>
<td>0.4694</td>
<td>-1.9619</td>
<td>0.1691</td>
</tr>
<tr>
<td>6</td>
<td>2.5664</td>
<td>1.3741</td>
<td>-2.7821</td>
<td>xbar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.4520</td>
<td></td>
<td>ybar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1.4956</td>
<td>1.3233</td>
<td>0.5829</td>
<td>0.7735</td>
<td>0.6247</td>
<td>0.7966</td>
</tr>
<tr>
<td>9</td>
<td>2.4113</td>
<td>2.0421</td>
<td>0.4678</td>
<td>0.8197</td>
<td>0.9420</td>
<td>0.1735</td>
</tr>
<tr>
<td>10</td>
<td>1.0475</td>
<td>0.1359</td>
<td>1.3853</td>
<td>xstd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.9062</td>
<td></td>
<td>ystd</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 10.3 Program Results

The program results show the predicted scores (YHAT) from the NAG Library function `g02ldc`.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2132</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.5152</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.1437</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.4459</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.1716</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2.4809</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.0964</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1.4475</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-0.1546</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>-0.5492</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.5393</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.2686</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>-1.1332</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1.7975</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.4973</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>