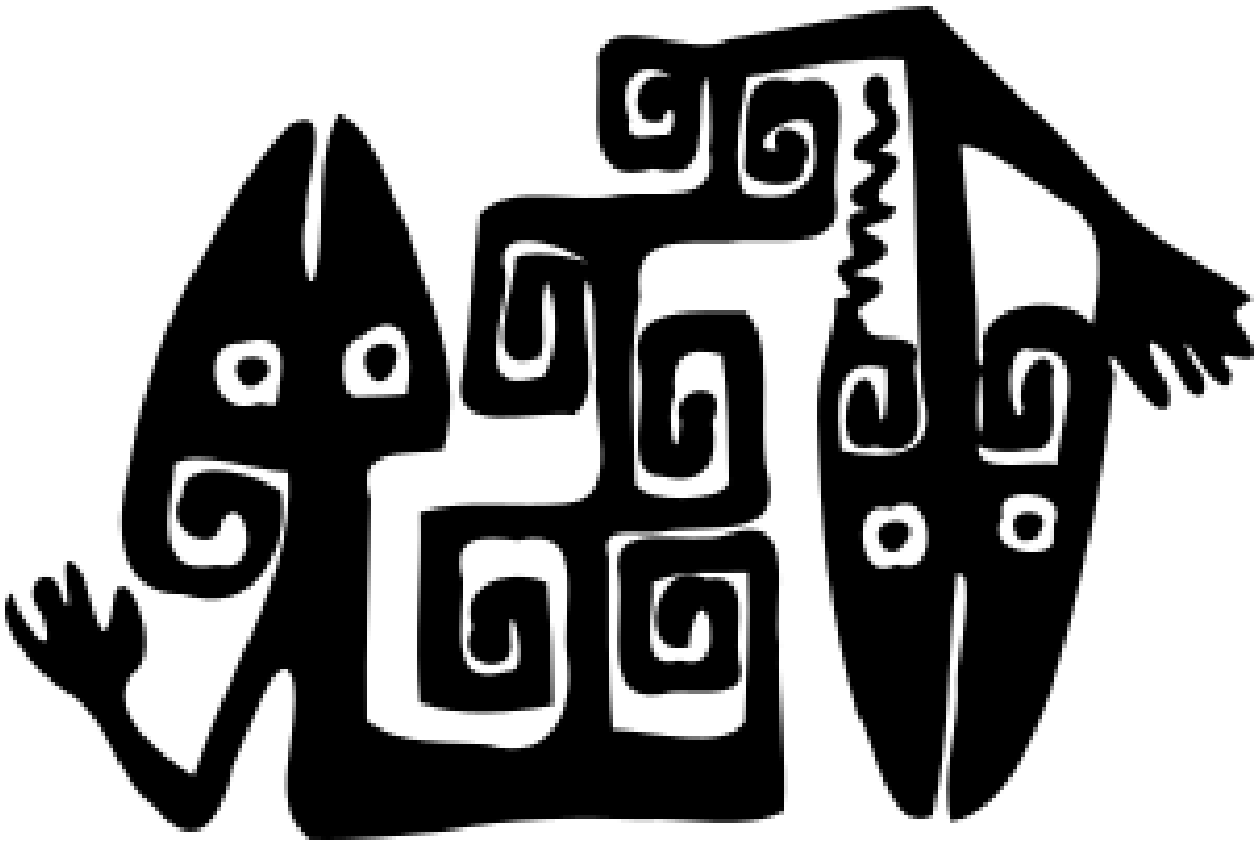
Hoggorm Documentation

Release 0.11.1

Oliver Tomic

Oct 30, 2018

Content



CHAPTER 1

Quickstart

hoggorm is a Python package for explorative multivariate statistics in Python. It contains

- PCA (principal component analysis)
- PCR (principal component regression)
- PLSR (partial least squares regression)
  - PLSR1 for single variable responses
  - PLSR2 for multivariate responses
- matrix corrlation coefficients RV and RV2.

Unlike scikit-learn, whis is an excellent Python machine learning package focusing on classification and predicition, hoggorm rather aims at understanding and interpretation of the variance in the data. hoggorm also contains tools for prediction.

Note: Results computed with the hoggorm package can be visualised using plotting functions implemented in the complementary hoggormplot package.

1.1 Requirements

Make sure that Python 3.5 or higher is installed. A convenient way to install Python and many useful packages for scientific computing is to use the Anaconda distribution.

- numpy

1.2 Installation and updates

1.2.1 Installation

Install hoggorm easily from the command line from the PyPI - the Python Packaging Index.

```
pip install hoggorm
```

1.2.2 Upgrading

To upgrade hoggorm from a previously installed older version execute the following from the command line:

```
pip install --upgrade hoggorm
```

If you need more information on how to install Python packages using pip, please see the pip documentation.

1.3 Documentation

- Documentation at Read the Docs
- Jupyter notebooks with examples of how to use hoggorm
  - for PCA
  - for PCR (coming soon)
  - for PLSR1 (coming soon)
  - for PLSR2 (coming soon)
  - for matrix correlation ceoefficitents RV and RV2 (coming soon)

More examples in Jupyter notebooks are provided at hoggormExamples GitHub repository.

1.4 Example

```
import hoggorm as ho

# Compute PCA model with
# - 5 components
# - standardised/scaled variables
# - KFold cross validation with 4 folds
model = ho.nipalsPCA(arrX=myData, numComp=5, Xstand=True, cvType=["Kfold", 4])

# Extract results from PCA model
scores = model.X_scores()
loadings = model.X_loadings()
cumulativeCalibratedExplainedVariance_allVariables = model.X_cumCalExplVar_indVar()
cumulativeValidatedExplainedVariance_total = model.X_cumValExplVar()
```

1.5 hoggorm repository on GitHub

The source code is available at the hoggorm GitHub repository.

CHAPTER 2

Principal Component Analysis (PCA)

The **nipalsPCA** class carries out principal component analysis. It analyses one data array and looks for systematic variance in the data using principal components (PC's). See below for a description of the methods in **nipalsPCA** as well as some examples of how to use it.

class hoggorm.pca.**nipalsPCA**(*arrX*, *numComp=None*, *Xstand=False*, *cvType=None*)

This class carries out Principal Component Analysis using the NIPALS algorithm.

Parameters

- **arrX** (*numpy array*) – A numpy array containing the data
- **numComp** (*int, optional*) – An integer that defines how many components are to be computed
- **Xstand** (*boolean, optional*) – Defines whether variables in `arrX` are to be standardised/scaled or centered

  False [columns of `arrX` are mean centred (default)] `Xstand = False`

  True [columns of `arrX` are mean centred and devided by their own standard deviation] `Xstand = True`
- **cvType** (*list, optional*) – The list defines cross validation settings when computing the PCA model. Note if *cvType* is not provided, cross validation will not be performed and as such cross validation results will not be available. Choose cross validation type from the following:

  loo [leave one out / a.k.a. full cross validation (default)] `cvType = ["loo"]`

  KFold [leave out one fold or segment] `cvType = ["KFold", numFolds]`

  numFolds: int

  Number of folds or segments

  lolo [leave one label out] `cvType = ["lolo", lablesList]`

  lablesList: list

Sequence of lables. Must be same lenght as number of rows in `arrX`. Leaves out objects with same lable.

Returns A class that contains the PCA model and computational results

Return type class

EXAMPLES

First import the hoggorm package.

```
>>> import hoggorm as ho
```

Import your data into a numpy array.

```
>>> myData
array([[ 5.7291665,  3.416667 ,  3.175    ,  2.6166668,  6.2208333],
       [ 6.0749993,  2.7416666,  3.6333339,  3.3833334,  6.1708336],
       [ 6.1166663,  3.4916666,  3.5208333,  2.7125003,  6.1625004],
       ...,
       [ 6.3333335,  2.3166668,  4.1249995,  4.3541665,  6.7500005],
       [ 5.8250003,  4.8291669,  1.4958333,  1.0958334,  6.0999999],
       [ 5.6499996,  4.6624999,  1.9291668,  1.0749999,  6.0249996]])
>>> np.shape(myData)
(14, 5)
```

Examples of how to compute a PCA model using different settings for the input parameters.

```
>>> model = ho.nipalsPCA(arrX=myData, numComp=5, Xstand=False)
>>> model = ho.nipalsPCA(arrX=myData)
>>> model = ho.nipalsPCA(arrX=myData, numComp=3)
>>> model = ho.nipalsPCA(arrX=myData, Xstand=True)
>>> model = ho.nipalsPCA(arrX=myData, cvType=["loo"])
>>> model = ho.nipalsPCA(arrX=myData, cvType=["KFold", 4])
>>> model = ho.nipalsPCA(arrX=myData, cvType=["lolo", [1, 2, 3, 4, 5, 6, 7, 1, 2,␣
↪3, 4, 5, 6, 7]])
```

Examples of how to extract results from the PCA model.

```
>>> scores = model.X_scores()
>>> loadings = model.X_loadings()
>>> cumulativeCalibratedExplainedVariance_allVariables = model.X_cumCalExplVar_
↪indVar()
```

X_MSECV()
Returns an array holding MSECV across all variables in X acquired through cross validation after each computed component. First row is MSECV for zero components, second row for component 1, third row for component 2, etc.

X_MSECV_indVar()
Returns an arrary holding MSECV for each variable in X acquired through cross validation. First row is MSECV for zero components, second row for component 1, etc.

X_MSEE()
Returns an array holding MSEE across all variables in X acquired through calibration after each computed component. First row is MSEE for zero components, second row for component 1, third row for component 2, etc.

X_MSEE_indVar()
Returns an array holding MSEE for each variable in array X acquired through calibration after each computed component. First row holds MSEE for zero components, second row for component 1, third row for component 2, etc.

X_PRESSCV()
Returns an array holding PRESSCV across all variables in X acquired through cross validation after each computed component. First row is PRESSEV for zero components, second row for component 1, third row for component 2, etc.

X_PRESSCV_indVar()
Returns array holding PRESSEV for each individual variable in X acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row for component 1, third row for component 2, etc.

X_PRESSE()
Returns array holding PRESSE across all variables in X acquired through calibration after each computed component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

X_PRESSE_indVar()
Returns array holding PRESSE for each individual variable in X acquired through calibration after each computed component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

X_RMSECV()
Returns an array holding RMSECV across all variables in X acquired through cross validation after each computed component. First row is RMSECV for zero components, second row for component 1, third row for component 2, etc.

X_RMSECV_indVar()
Returns an arrary holding RMSECV for each variable in X acquired through cross validation after each computed component. First row is RMSECV for zero components, second row for component 1, third row for component 2, etc.

X_RMSEE()
Returns an array holding RMSEE across all variables in X acquired through calibration after each computed component. First row is RMSEE for zero components, second row for component 1, third row for component 2, etc.

X_RMSEE_indVar()
Returns an array holding RMSEE for each variable in array X acquired through calibration after each components. First row holds RMSEE for zero components, second row for component 1, third row for component 2, etc.

X_calExplVar()
Returns a list holding the calibrated explained variance for each component. First number in list is for component 1, second number for component 2, etc.

X_corrLoadings()
Returns array holding correlation loadings of array X. First column holds correlation loadings for component 1, second column holds correlation loadings for component 2, etc.

X_cumCalExplVar()
Returns a list holding the cumulative validated explained variance for array X after each component. First number represents zero components, second number represents component 1, etc.

X_cumCalExplVar_indVar()
Returns an array holding the cumulative calibrated explained variance for each variable in X after each

component. First row represents zero components, second row represents one component, third row represents two components, etc. Columns represent variables.

X_cumValExplVar()
Returns a list holding the cumulative validated explained variance for array X after each component.

X_cumValExplVar_indVar()
Returns an array holding the cumulative validated explained variance for each variable in X after each component. First row represents zero components, second row represents component 1, third row for compnent 2, etc. Columns represent variables.

X_loadings()
Returns array holding loadings P of array X. Rows represent variables and columns represent components. First column holds loadings for component 1, second column holds scores for component 2, etc.

X_means()
Returns array holding the column means of input array X.

X_predCal()
Returns a dictionary holding the predicted arrays Xhat from calibration after each computed component. Dictionary key represents order of component.

X_predVal()
Returns a dictionary holding the predicted arrays Xhat from validation after each computed component. Dictionary key represents order of component.

X_residuals()
Returns a dictionary holding arrays of residuals for array X after each computed component. Dictionary key represents order of component.

X_scores()
Returns array holding scores T. First column holds scores for component 1, second column holds scores for component 2, etc.

X_scores_predict(*Xnew*, *numComp=None*)
Returns array of X scores from new X data using the exsisting model. Rows represent objects and columns represent components.

X_valExplVar()
Returns a list holding the validated explained variance for X after each component. First number in list is for component 1, second number for component 2, third number for component 3, etc.

__init__(*arrX*, *numComp=None*, *Xstand=False*, *cvType=None*)
On initialisation check how arrX and arrY are to be pre-processed (Xstand and Ystand are either True or False). Then check whether number of components chosen by user is OK.

corrLoadingsEllipses()
Returns a dictionary hodling coordinates of ellipses that represent 50% and 100% expl. variance in correlation loadings plot. The coordinates are stored in arrays.

cvTrainAndTestData()
Returns a list consisting of dictionaries holding training and test sets.

modelSettings()
Returns a dictionary holding the settings under which NIPALS PCA was run.

CHAPTER 3

Principal Component Regression (PCR)

The **nipalsPCR** class carries out principal component regression. It analyses two data arrays and finds common systematic variance between the two arrays. See below for a description of the methods in **nipalsPCR** as well as some examples of how to use it.

class hoggorm.pcr.**nipalsPCR**(*arrX*, *arrY*, *numComp=None*, *Xstand=False*, *Ystand=False*, *cvType=None*)

This class carries out Principal Component Regression for two arrays using NIPALS algorithm.

Parameters

- **arrX** (*numpy array*) – This is X in the PCR model. Number and order of objects (rows) must match those of `arrY`.
- **arrY** (*numpy array*) – This is Y in the PCR model. Number and order of objects (rows) must match those of `arrX`.
- **numComp** (*int, optional*) – An integer that defines how many components are to be computed. If not provided, the maximum possible number of components is used.
- **Xstand** (*boolean, optional*) – Defines whether variables in `arrX` are to be standardised/scaled or centered.

  False [columns of `arrX` are mean centred (default)] `Xstand = False`

  True [columns of `arrX` are mean centred and devided by their own standard deviation] `Xstand = True`
- **Ystand** (*boolean, optional*) – Defines whether variables in `arrY` are to be standardised/scaled or centered.

  False [columns of `arrY` are mean centred (default)] `Ystand = False`

  True [columns of `arrY` are mean centred and devided by their own standard deviation] `Ystand = True`
- **cvType** (*list, optional*) – The list defines cross validation settings when computing the PCA model. Note if *cvType* is not provided, cross validation will not be performed and

as such cross validation results will not be available. Choose cross validation type from the following:

loo [leave one out / a.k.a. full cross validation (default)] `cvType = ["loo"]`

KFold [leave out one fold or segment] `cvType = ["KFold", numFolds]`

numFolds: int

Number of folds or segments

- **lolo** (*leave one label out*) – `cvType = ["lolo", labelsList]`

labelsList: list

Sequence of lables. Must be same lenght as number of rows in `arrX` and `arrY`. Leaves out objects with same lable.

Returns A class that contains the PCR model and computational results

Return type class

EXAMPLES

First import the hoggormpackage

```
>>> import hoggorm as ho
```

Import your data into a numpy array.

```
>>> np.shape(my_X_data)
(14, 292)
>>> np.shape(my_Y_data)
(14, 5)
```

Examples of how to compute a PCR model using different settings for the input parameters.

```
>>> model = ho.nipalsPCR(arrX=my_X_data, arrY=my_Y_data, numComp=5)
>>> model = ho.nipalsPCR(arrX=my_X_data, arrY=my_Y_data)
>>> model = ho.nipalsPCR(arrX=my_X_data, arrY=my_Y_data, numComp=3, Ystand=True)
>>> model = ho.nipalsPCR(arrX=my_X_data, arrY=my_Y_data, Xstand=False,␣
↪Ystand=True)
>>> model = ho.nipalsPCR(arrX=my_X_data, arrY=my_Y_data, cvType=["loo"])
>>> model = ho.nipalsPCR(arrX=my_X_data, arrY=my_Y_data, cvType=["KFold", 7])
>>> model = ho.nipalsPCR(arrX=my_X_data, arrY=my_Y_data, cvType=["lolo", [1, 2, 3,
↪ 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7]])
```

Examples of how to extract results from the PCR model.

```
>>> X_scores = model.X_scores()
>>> X_loadings = model.X_loadings()
>>> Y_loadings = model.Y_loadings()
>>> X_cumulativeCalibratedExplainedVariance_allVariables = model.X_cumCalExplVar_
↪indVar()
>>> Y_cumulativeValidatedExplainedVariance_total = model.Y_cumCalExplVar()
```

X_MSECV()

Returns an array holding MSECV across all variables in X acquired through cross validation after each computed component. First row is MSECV for zero components, second row for component 1, third row for component 2, etc.

X_MSECV_indVar()
Returns an arrary holding MSECV for each variable in X acquired through cross validation. First row is MSECV for zero components, second row for component 1, etc.

X_MSEE()
Returns an array holding MSEE across all variables in X acquired through calibration after each computed component. First row is MSEE for zero components, second row for component 1, third row for component 2, etc.

X_MSEE_indVar()
Returns an array holding MSEE for each variable in array X acquired through calibration after each computed component. First row holds MSEE for zero components, second row for component 1, third row for component 2, etc.

X_PRESSCV()
Returns an array holding PRESSCV across all variables in X acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row for component 1, third row for component 2, etc.

X_PRESSCV_indVar()
Returns array holding PRESSCV for each individual variable in X acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row for component 1, third row for component 2, etc.

X_PRESSE()
Returns array holding PRESSE across all variables in X acquired through calibration after each computed component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

X_PRESSE_indVar()
Returns array holding PRESSE for each individual variable in X acquired through calibration after each computed component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

X_RMSECV()
Returns an array holding RMSECV across all variables in X acquired through cross validation after each computed component. First row is RMSECV for zero components, second row for component 1, third row for component 2, etc.

X_RMSECV_indVar()
Returns an arrary holding RMSECV for each variable in X acquired through cross validation after each computed component. First row is RMSECV for zero components, second row for component 1, third row for component 2, etc.

X_RMSEE()
Returns an array holding RMSEE across all variables in X acquired through calibration after each computed component. First row is RMSEE for zero components, second row for component 1, third row for component 2, etc.

X_RMSEE_indVar()
Returns an array holding RMSEE for each variable in array X acquired through calibration after each component. First row holds RMSEE for zero components, second row for component 1, third row for component 2, etc.

X_calExplVar()
Returns a list holding the calibrated explained variance for each component. First number in list is for component 1, second number for component 2, etc.

X_corrLoadings()

Returns array holding correlation loadings of array X. First column holds correlation loadings for component 1, second column holds correlation loadings for component 2, etc.

X_cumCalExplVar()
Returns a list holding the cumulative calibrated explained variance for array X after each component.

X_cumCalExplVar_indVar()
Returns an array holding the cumulative calibrated explained variance for each variable in X after each component. First row represents zero components, second row represents one component, third row represents two components, etc. Columns represent variables.

X_cumValExplVar()
Returns a list holding the cumulative validated explained variance for array X after each component. First number represents zero components, second number represents component 1, etc.

X_cumValExplVar_indVar()
Returns an array holding the cumulative validated explained variance for each variable in X after each component. First row represents zero components, second row represents component 1, third row for compnent 2, etc. Columns represent variables.

X_loadings()
Returns array holding loadings of array X. Rows represent variables and columns represent components. First column holds loadings for component 1, second column holds scores for component 2, etc.

X_means()
Returns array holding column means of array X.

X_predCal()
Returns a dictionary holding the predicted arrays Xhat from calibration after each computed component. Dictionary key represents order of component.

X_predVal()
Returns dictionary holding arrays of predicted Xhat after each component from validation. Dictionary key represents order of component.

X_residuals()
Returns a dictionary holding the residual arrays for array X after each computed component. Dictionary key represents order of component.

X_scores()
Returns array holding scores of array X. First column holds scores for component 1, second column holds scores for component 2, etc.

X_scores_predict(*Xnew*, *numComp=None*)
Returns array of X scores from new X data using the exsisting model. Rows represent objects and columns represent components.

X_valExplVar()
Returns a list holding the validated explained variance for X after each component. First number in list is for component 1, second number for component 2, third number for component 3, etc.

Y_MSECV()
Returns an array holding MSECV across all variables in Y acquired through cross validation after each computed component. First row is MSECV for zero components, second row component 1, third row for component 2, etc.

Y_MSECV_indVar()
Returns an array holding MSECV of each variable in array Y acquired through cross validation after each computed component. First row is MSECV for zero components, second row component 1, third row for component 2, etc.

Y_MSEE()
Returns an array holding MSEE across all variables in Y acquired through calibration after each computed component. First row is MSEE for zero components, second row for component 1, third row for component 2, etc.

Y_MSEE_indVar()
Returns an array holding MSEE for each variable in array Y acquired through calibration after each computed component. First row holds MSEE for zero components, second row for component 1, third row for component 2, etc.

Y_PRESSCV()
Returns an array holding PRESSCV across all variables in Y acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row component 1, third row for component 2, etc.

Y_PRESSCV_indVar()
Returns an array holding PRESSCV of each variable in array Y acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row component 1, third row for component 2, etc.

Y_PRESSE()
Returns array holding PRESSE across all variables in Y acquired through calibration after each computed component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

Y_PRESSE_indVar()
Returns array holding PRESSE for each individual variable in Y acquired through calibration after each component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

Y_RMSECV()
Returns an array holding RMSECV across all variables in Y acquired through cross validation after each computed component. First row is RMSECV for zero components, second row component 1, third row for component 2, etc.

Y_RMSECV_indVar()
Returns an array holding RMSECV for each variable in array Y acquired through cross validation after each computed component. First row is RMSECV for zero components, second row component 1, third row for component 2, etc.

Y_RMSEE()
Returns an array holding RMSEE across all variables in Y acquired through calibration after each computed component. First row is RMSEE for zero components, second row for component 1, third row for component 2, etc.

Y_RMSEE_indVar()
Returns an array holding RMSEE for each variable in array Y acquired through calibration after each component. First row holds RMSEE for zero components, second row for component 1, third row for component 2, etc.

Y_calExplVar()
Returns a list holding the calibrated explained variance for each component. First number in list is for component 1, second number for component 2, etc.

Y_corrLoadings()
Returns array holding correlation loadings of array X. First column holds correlation loadings for component 1, second column holds correlation loadings for component 2, etc.

Y_cumCalExplVar()
Returns a list holding the cumulative calibrated explained variance for array X after each component. First

number represents zero components, second number represents component 1, etc.

Y_cumCalExplVar_indVar()
Returns an array holding the cumulative calibrated explained variance for each variable in Y after each component. First row represents zero components, second row represents one component, third row represents two components, etc. Columns represent variables.

Y_cumValExplVar()
Returns a list holding the cumulative validated explained variance for array X after each component. First number represents zero components, second number represents component 1, etc.

Y_cumValExplVar_indVar()
Returns an array holding the cumulative validated explained variance for each variable in Y after each component. First row represents zero components, second row represents component 1, third row for compnent 2, etc. Columns represent variables.

Y_loadings()
Returns an array holding loadings C of array Y. Rows represent variables and columns represent components. First column for component 1, second columns for component 2, etc.

Y_means()
Returns array holding means of columns in array Y.

Y_predCal()
Returns dictionary holding arrays of predicted Yhat after each component from calibration. Dictionary key represents order of components.

Y_predVal()
Returns dictionary holding arrays of predicted Yhat after each component from validation. Dictionary key represents order of component.

Y_predict(*Xnew*, *numComp=1*)
Return predicted Yhat from new measurements X.

Y_residuals()
Returns a dictionary holding residuals F of array Y after each component. Dictionary key represents order of component.

Y_valExplVar()
Returns a list holding the validated explained variance for Y after each component. First number in list is for component 1, second number for component 2, third number for component 3, etc.

__init__(*arrX*, *arrY*, *numComp=None*, *Xstand=False*, *Ystand=False*, *cvType=None*)
On initialisation check how arrX and arrY are to be pre-processed (parameters Xstand and Ystand are either True or False). Then check whether number of components chosen by user is OK.

corrLoadingsEllipses()
Returns coordinates for the ellipses that represent 50% and 100% expl. variance in correlation loadings plot.

cvTrainAndTestData()
Returns a list consisting of dictionaries holding training and test sets.

modelSettings()
Returns a dictionary holding the settings under which NIPALS PCR was run.

regressionCoefficients(*numComp=1*)
Returns regression coefficients from the fitted model using all available samples and a chosen number of components.

CHAPTER 4

Partial Least Squares Regression (PLSR)

4.1 PLSR1

class hoggorm.plsr1.**nipalsPLS1**(*arrX, vecy, numComp=3, Xstand=False, Ystand=False, cvType=['loo']*)

This class carries out partial least squares regression (PLSR) for two arrays using NIPALS algorithm. The y array is univariate, which is why PLS1 is applied.

Parameters

- **arrX** (*numpy array*) – This is X in the PLS1 model. Number and order of objects (rows) must match those of `arrY`.
- **vecy** (*numpy array*) – This is y in the PLS1 model. Number and order of objects (rows) must match those of `arrX`.
- **numComp** (*int, optional*) – An integer that defines how many components are to be computed. If not provided, the maximum possible number of components is used.
- **Xstand** (*boolean, optional*) – Defines whether variables in `arrX` are to be standardised/scaled or centered.

  False [columns of `arrX` are mean centred (default)] `Xstand = False`

  True [columns of `arrX` are mean centred and devided by their own standard deviation] `Xstand = True`
- **Ystand** (*boolean, optional*) – Defines whether `vecy` is to be standardised/scaled or centered.

  False [`vecy` is to be mean centred (default)] `Ystand = False`

  True [`vecy` is to be mean centred and devided by its own standard deviation] `Ystand = True`
- **cvType** (*list, optional*) – The list defines cross validation settings when computing the PCA model. Note if *cvType* is not provided, cross validation will not be performed and

as such cross validation results will not be available. Choose cross validation type from the following:

loo [leave one out / a.k.a. full cross validation (default)] `cvType = ["loo"]`

KFold [leave out one fold or segment] `cvType = ["KFold", numFolds]`

numFolds: int

Number of folds or segments

- **lolo** (*leave one label out*) – `cvType = ["lolo", labelsList]`

labelsList: list

Sequence of lables. Must be same lenght as number of rows in `arrX` and `arrY`. Leaves out objects with same lable.

Returns A class that contains the PLS1 model and computational results

Return type class

EXAMPLES

First import the hoggormpackage

```
>>> import hoggorm as ho
```

Import your data into a numpy array.

```
>>> np.shape(my_X_data)
(14, 292)
>>> np.shape(my_y_data)
(14, 1)
```

Examples of how to compute a PLS1 model using different settings for the input parameters.

```
>>> model = ho.nipalsPLS1(arrX=my_X_data, vecy=my_y_data, numComp=5)
>>> model = ho.nipalsPLS1(arrX=my_X_data, vecy=my_y_data)
>>> model = ho.nipalsPLS1(arrX=my_X_data, vecy=my_y_data, numComp=3, Ystand=True)
>>> model = ho.nipalsPLS1(arrX=my_X_data, vecy=my_y_data, Xstand=False,␣
↪Ystand=True)
>>> model = ho.nipalsPLS1(arrX=my_X_data, vecy=my_y_data, cvType=["loo"])
>>> model = ho.nipalsPLS1(arrX=my_X_data, vecy=my_y_data, cvType=["KFold", 7])
>>> model = ho.nipalsPLS1(arrX=my_X_data, vecy=my_y_data, cvType=["lolo", [1, 2,␣
↪3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7]]])
```

Examples of how to extract results from the PCR model.

```
>>> X_scores = model.X_scores()
>>> X_loadings = model.X_loadings()
>>> y_loadings = model.Y_loadings()
>>> X_cumulativeCalibratedExplainedVariance_allVariables = model.X_cumCalExplVar_
↪indVar()
>>> Y_cumulativeValidatedExplainedVariance_total = model.Y_cumCalExplVar()
```

X_MSECV()

Returns an array holding MSECV across all variables in X acquired through cross validation after each computed component. First row is MSECV for zero components, second row for component 1, third row for component 2, etc.

X_MSECV_indVar()
Returns an arrary holding MSECV for each variable in X acquired through cross validation. First row is MSECV for zero components, second row for component 1, etc.

X_MSEE()
Returns an array holding MSEE across all variables in X acquired through calibration after each computed component. First row is MSEE for zero components, second row for component 1, third row for component 2, etc.

X_MSEE_indVar()
Returns an array holding MSEE for each variable in array X acquired through calibration after each computed component. First row holds MSEE for zero components, second row for component 1, third row for component 2, etc.

X_PRESSCV()
Returns an array holding PRESSCV across all variables in X acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row for component 1, third row for component 2, etc.

X_PRESSCV_indVar()
Returns array holding PRESSCV for each individual variable in X acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row for component 1, third row for component 2, etc.

X_PRESSE()
Returns array holding PRESSE across all variables in X acquired through calibration after each computed component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

X_PRESSE_indVar()
Returns array holding PRESSE for each individual variable in X acquired through calibration after each computed component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

X_RMSECV()
Returns an array holding RMSECV across all variables in X acquired through cross validation after each computed component. First row is RMSECV for zero components, second row for component 1, third row for component 2, etc.

X_RMSECV_indVar()
Returns an arrary holding RMSECV for each variable in X acquired through cross validation after each computed component. First row is RMSECV for zero components, second row for component 1, third row for component 2, etc.

X_RMSEE()
Returns an array holding RMSEE across all variables in X acquired through calibration after each computed component. First row is RMSEE for zero components, second row for component 1, third row for component 2, etc.

X_RMSEE_indVar()
Returns an array holding RMSEE for each variable in array X acquired through calibration after each component. First row holds RMSEE for zero components, second row for component 1, third row for component 2, etc.

X_calExplVar()
Returns a list holding the calibrated explained variance for each component. First number in list is for component 1, second number for component 2, etc.

X_corrLoadings()

Returns array holding correlation loadings of array X. First column holds correlation loadings for component 1, second column holds correlation loadings for component 2, etc.

X_cumCalExplVar()
Returns a list holding the cumulative calibrated explained variance for array X after each component.

X_cumCalExplVar_indVar()
Returns an array holding the cumulative calibrated explained variance for each variable in X after each component. First row represents zero components, second row represents one component, third row represents two components, etc. Columns represent variables.

X_cumValExplVar()
Returns a list holding the cumulative validated explained variance for array X after each component. First number represents zero components, second number represents component 1, etc.

X_cumValExplVar_indVar()
Returns an array holding the cumulative validated explained variance for each variable in X after each component. First row represents zero components, second row represents component 1, third row for compnent 2, etc. Columns represent variables.

X_loadingWeights()
Returns an array holding X loadings weights.

X_loadings()
Returns array holding loadings of array X. Rows represent variables and columns represent components. First column holds loadings for component 1, second column holds scores for component 2, etc.

X_means()
Returns array holding the column means of X.

X_predCal()
Returns a dictionary holding the predicted arrays Xhat from calibration after each computed component. Dictionary key represents order of component.

X_predVal()
Returns dictionary holding arrays of predicted Xhat after each component from validation. Dictionary key represents order of component.

X_residuals()
Returns a dictionary holding the residual arrays for array X after each computed component. Dictionary key represents order of component.

X_scores()
Returns array holding scores of array X. First column holds scores for component 1, second column holds scores for component 2, etc.

X_scores_predict(*Xnew*, *numComp=None*)
Returns array of X scores from new X data using the exsisting model. Rows represent objects and columns represent components.

X_valExplVar()
Returns a list holding the validated explained variance for X after each component. First number in list is for component 1, second number for component 2, third number for component 3, etc.

Y_MSECV()
Returns an array holding MSECV of vector y acquired through cross validation after each computed component. First row is MSECV for zero components, second row component 1, third row for component 2, etc.

Y_MSEE()
Returns an array holding MSEE of vector y acquired through calibration after each component. First row

holds MSEE for zero components, second row component 1, third row for component 2, etc.

Y_PRESSCV()
Returns an array holding PRESSECV for Y acquired through cross validation after each computed component. First row is PRESSECV for zero components, second row component 1, third row for component 2, etc.

Y_PRESSE()
Returns an array holding PRESSE for y acquired through calibration after each computed component. First row is PRESSE for zero components, second row component 1, third row for component 2, etc.

Y_RMSECV()
Returns an array holding RMSECV for vector y acquired through cross validation after each computed component. First row is RMSECV for zero components, second row component 1, third row for component 2, etc.

Y_RMSEE()
Returns an array holding RMSEE of vector y acquired through calibration after each computed component. First row is RMSEE for zero components, second row component 1, third row for component 2, etc.

Y_calExplVar()
Returns list holding calibrated explained variance for each component in vector y.

Y_corrLoadings()
Returns an array holding correlation loadings of vector y. Columns represent components. First column for component 1, second columns for component 2, etc.

Y_cumCalExplVar()
Returns a list holding the calibrated explained variance for each component. First number represent zero components, second number one component, etc.

Y_cumValExplVar()
Returns list holding cumulative validated explained variance in vector y.

Y_loadings()
Returns an array holding loadings of vector y. Columns represent components. First column for component 1, second columns for component 2, etc.

Y_means()
Returns an array holding the mean of vector y.

Y_predCal()
Returns dictionary holding arrays of predicted yhat after each component from calibration. Dictionary key represents order of components.

Y_predVal()
Returns dictionary holding arrays of predicted yhat after each component from validation. Dictionary key represents order of component.

Y_predict(*Xnew*, *numComp=1*)
Return predicted yhat from new measurements X.

Y_residuals()
Returns list of arrays holding residuals of vector y after each component.

Y_scores()
Returns scores of array Y (NOT IMPLEMENTED)

Y_valExplVar()
Returns list holding validated explained variance for each component in vector y.

__init__(*arrX, vecy, numComp=3, Xstand=False, Ystand=False, cvType=['loo']*)
On initialisation check how X and y are to be pre-processed (which mode is used). Then check whether number of PC's chosen by user is OK. Then run NIPALS PLS1 algorithm.

corrLoadingsEllipses()
Returns coordinates of ellipses that represent 50% and 100% expl. variance in correlation loadings plot.

cvTrainAndTestData()
Returns a list consisting of dictionaries holding training and test sets.

modelSettings()
Returns a dictionary holding settings under which PLS1 was run.

regressionCoefficients(*numComp=1*)
Returns regression coefficients from the fitted model using all available samples and a chosen number of components.

4.2 PLSR2

class hoggorm.plsr2.**nipalsPLS2**(*arrX*, *arrY*, *numComp=None*, *Xstand=False*, *Ystand=False*, *cvType=None*)
This class carries out partial least squares regression (PLSR) for two arrays using NIPALS algorithm. The Y array is multivariate, which is why PLS2 is applied.

Parameters

- **arrX** (*numpy array*) – This is X in the PCR model. Number and order of objects (rows) must match those of `arrY`.
- **arrY** (*numpy array*) – This is Y in the PCR model. Number and order of objects (rows) must match those of `arrX`.
- **numComp** (*int, optional*) – An integer that defines how many components are to be computed. If not provided, the maximum possible number of components is used.
- **Xstand** (*boolean, optional*) – Defines whether variables in `arrX` are to be standardised/scaled or centered.

  False [columns of `arrX` are mean centred (default)] `Xstand = False`

  True [columns of `arrX` are mean centred and devided by their own standard deviation] `Xstand = True`
- **Ystand** (*boolean, optional*) – Defines whether variables in `arrY` are to be standardised/scaled or centered.

  False [columns of `arrY` are mean centred (default)] `Ystand = False`

  True [columns of `arrY` are mean centred and devided by their own standard deviation] `Ystand = True`
- **cvType** (*list, optional*) – The list defines cross validation settings when computing the PCA model. Note if *cvType* is not provided, cross validation will not be performed and as such cross validation results will not be available. Choose cross validation type from the following:

  loo [leave one out / a.k.a. full cross validation (default)] `cvType = ["loo"]`

  KFold [leave out one fold or segment] `cvType = ["KFold", numFolds]`

  numFolds: int

Number of folds or segments

- **lolo**(*leave one label out*) – cvType = ["lolo", labelsList]

  labelsList: list

  Sequence of lables. Must be same lenght as number of rows in `arrX` and `arrY`. Leaves out objects with same lable.

Returns A class that contains the PLS2 model and computational results

Return type class

EXAMPLES

First import the hoggormpackage

```
>>> import hoggorm as ho
```

Import your data into a numpy array.

```
>>> np.shape(my_X_data)
(14, 292)
>>> np.shape(my_Y_data)
(14, 5)
```

Examples of how to compute a PLS2 model using different settings for the input parameters.

```
>>> model = ho.nipalsPLS2(arrX=my_X_data, arrY=my_Y_data, numComp=5)
>>> model = ho.nipalsPLS2(arrX=my_X_data, arrY=my_Y_data)
>>> model = ho.nipalsPLS2(arrX=my_X_data, arrY=my_Y_data, numComp=3, Ystand=True)
>>> model = ho.nipalsPLS2(arrX=my_X_data, arrY=my_Y_data, Xstand=False,␣
↪Ystand=True)
>>> model = ho.nipalsPLS2(arrX=my_X_data, arrY=my_Y_data, cvType=["loo"])
>>> model = ho.nipalsPLS2(arrX=my_X_data, arrY=my_Y_data, cvType=["KFold", 7])
>>> model = ho.nipalsPLS2(arrX=my_X_data, arrY=my_Y_data, cvType=["lolo", [1, 2,␣
↪3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7]])
```

Examples of how to extract results from the PLS2 model.

```
>>> X_scores = model.X_scores()
>>> X_loadings = model.X_loadings()
>>> Y_loadings = model.Y_loadings()
>>> X_cumulativeCalibratedExplainedVariance_allVariables = model.X_cumCalExplVar_
↪indVar()
>>> Y_cumulativeValidatedExplainedVariance_total = model.Y_cumCalExplVar()
```

X_MSECV()
Returns an array holding MSECV across all variables in X acquired through cross validation after each computed component. First row is MSECV for zero components, second row for component 1, third row for component 2, etc.

X_MSECV_indVar()
Returns an arrary holding MSECV for each variable in X acquired through cross validation. First row is MSECV for zero components, second row for component 1, etc.

X_MSEE()
Returns an array holding MSEE across all variables in X acquired through calibration after each computed

component. First row is MSEE for zero components, second row for component 1, third row for component 2, etc.

X_MSEE_indVar()
Returns an array holding MSEE for each variable in array X acquired through calibration after each computed component. First row holds MSEE for zero components, second row for component 1, third row for component 2, etc.

X_PRESSCV()
Returns an array holding PRESSCV across all variables in X acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row for component 1, third row for component 2, etc.

X_PRESSCV_indVar()
Returns array holding PRESSCV for each individual variable in X acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row for component 1, third row for component 2, etc.

X_PRESSE()
Returns array holding PRESSE across all variables in X acquired through calibration after each computed component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

X_PRESSE_indVar()
Returns array holding PRESSE for each individual variable in X acquired through calibration after each computed component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

X_RMSECV()
Returns an array holding RMSECV across all variables in X acquired through cross validation after each computed component. First row is RMSECV for zero components, second row for component 1, third row for component 2, etc.

X_RMSECV_indVar()
Returns an arrary holding RMSECV for each variable in X acquired through cross validation after each computed component. First row is RMSECV for zero components, second row for component 1, third row for component 2, etc.

X_RMSEE()
Returns an array holding RMSEE across all variables in X acquired through calibration after each computed component. First row is RMSEE for zero components, second row for component 1, third row for component 2, etc.

X_RMSEE_indVar()
Returns an array holding RMSEE for each variable in array X acquired through calibration after each component. First row holds RMSEE for zero components, second row for component 1, third row for component 2, etc.

X_calExplVar()
Returns a list holding the calibrated explained variance for each component. First number in list is for component 1, second number for component 2, etc.

X_corrLoadings()
Returns array holding correlation loadings of array X. First column holds correlation loadings for component 1, second column holds correlation loadings for component 2, etc.

X_cumCalExplVar()
Returns a list holding the cumulative calibrated explained variance for array X after each component.

X_cumCalExplVar_indVar()
Returns an array holding the cumulative calibrated explained variance for each variable in X after each component. First row represents zero components, second row represents one component, third row represents two components, etc. Columns represent variables.

X_cumValExplVar()
Returns a list holding the cumulative validated explained variance for array X after each component. First number represents zero components, second number represents component 1, etc.

X_cumValExplVar_indVar()
Returns an array holding the cumulative validated explained variance for each variable in X after each component. First row represents zero components, second row represents component 1, third row for compnent 2, etc. Columns represent variables.

X_loadingWeights()
Returns an array holding loadings weights of array X.

X_loadings()
Returns array holding loadings of array X. Rows represent variables and columns represent components. First column holds loadings for component 1, second column holds scores for component 2, etc.

X_means()
Returns a vector holding the column means of X.

X_predCal()
Returns a dictionary holding the predicted arrays Xhat from calibration after each computed component. Dictionary key represents order of component.

X_predVal()
Returns dictionary holding arrays of predicted Xhat after each component from validation. Dictionary key represents order of component.

X_residuals()
Returns a dictionary holding the residual arrays for array X after each computed component. Dictionary key represents order of component.

X_scores()
Returns array holding scores of array X. First column holds scores for component 1, second column holds scores for component 2, etc.

X_scores_predict(*Xnew*, *numComp=None*)
Returns array of X scores from new X data using the exsisting model. Rows represent objects and columns represent components.

X_valExplVar()
Returns a list holding the validated explained variance for X after each component. First number in list is for component 1, second number for component 2, third number for component 3, etc.

Y_MSECV()
Returns an array holding MSECV across all variables in Y acquired through cross validation after each computed component. First row is MSECV for zero components, second row component 1, third row for component 2, etc.

Y_MSECV_indVar()
Returns an array holding MSECV of each variable in array Y acquired through cross validation after each computed component. First row is MSECV for zero components, second row component 1, third row for component 2, etc.

Y_MSEE()
Returns an array holding MSEE across all variables in Y acquired through calibration after each computed

component. First row is MSEE for zero components, second row for component 1, third row for component 2, etc.

Y_MSEE_indVar()
Returns an array holding MSEE for each variable in array Y acquired through calibration after each computed component. First row holds MSEE for zero components, second row for component 1, third row for component 2, etc.

Y_PRESSCV()
Returns an array holding PRESSCV across all variables in Y acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row component 1, third row for component 2, etc.

Y_PRESSCV_indVar()
Returns an array holding PRESSCV of each variable in array Y acquired through cross validation after each computed component. First row is PRESSCV for zero components, second row component 1, third row for component 2, etc.

Y_PRESSE()
Returns array holding PRESSE across all variables in Y acquired through calibration after each computed component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

Y_PRESSE_indVar()
Returns array holding PRESSE for each individual variable in Y acquired through calibration after each component. First row is PRESSE for zero components, second row for component 1, third row for component 2, etc.

Y_RMSECV()
Returns an array holding RMSECV across all variables in Y acquired through cross validation after each computed component. First row is RMSECV for zero components, second row component 1, third row for component 2, etc.

Y_RMSECV_indVar()
Returns an array holding RMSECV for each variable in array Y acquired through cross validation after each computed component. First row is RMSECV for zero components, second row component 1, third row for component 2, etc.

Y_RMSEE()
Returns an array holding RMSEE across all variables in Y acquired through calibration after each computed component. First row is RMSEE for zero components, second row for component 1, third row for component 2, etc.

Y_RMSEE_indVar()
Returns an array holding RMSEE for each variable in array Y acquired through calibration after each component. First row holds RMSEE for zero components, second row for component 1, third row for component 2, etc.

Y_calExplVar()
Returns a list holding the calibrated explained variance for each component. First number in list is for component 1, second number for component 2, etc.

Y_corrLoadings()
Returns array holding correlation loadings of array X. First column holds correlation loadings for component 1, second column holds correlation loadings for component 2, etc.

Y_cumCalExplVar()
Returns a list holding the cumulative calibrated explained variance for array X after each component. First number represents zero components, second number represents component 1, etc.

Y_cumCalExplVar_indVar()
Returns an array holding the cumulative calibrated explained variance for each variable in Y after each component. First row represents zero components, second row represents one component, third row represents two components, etc. Columns represent variables.

Y_cumValExplVar()
Returns a list holding the cumulative validated explained variance for array X after each component. First number represents zero components, second number represents component 1, etc.

Y_cumValExplVar_indVar()
Returns an array holding the cumulative validated explained variance for each variable in Y after each component. First row represents zero components, second row represents component 1, third row for compnent 2, etc. Columns represent variables.

Y_loadings()
Returns an array holding loadings C of array Y. Rows represent variables and columns represent components. First column for component 1, second columns for component 2, etc.

Y_means()
Returns a vector holding the column means of array Y.

Y_predCal()
Returns dictionary holding arrays of predicted Yhat after each component from calibration. Dictionary key represents order of components.

Y_predVal()
Returns dictionary holding arrays of predicted Yhat after each component from validation. Dictionary key represents order of component.

Y_predict(*Xnew*, *numComp=1*)
Return predicted Yhat from new measurements X.

Y_residuals()
Returns a dictionary holding residuals F of array Y after each component. Dictionary key represents order of component.

Y_scores()
Returns an array holding loadings C of array Y. Rows represent variables and columns represent components. First column for component 1, second columns for component 2, etc.

Y_valExplVar()
Returns a list holding the validated explained variance for Y after each component. First number in list is for component 1, second number for component 2, third number for component 3, etc.

__init__(*arrX*, *arrY*, *numComp=None*, *Xstand=False*, *Ystand=False*, *cvType=None*)
On initialisation check whether number of PC's chosen by user is given and smaller than maximum number of PC's possible.Then check how X and Y are to be pre-processed (whether 'Xstand' and 'Ystand' are used). Then run NIPALS PLS2 algorithm.

corrLoadingsEllipses()
Returns the coordinates of ellipses that represent 50% and 100% expl. variance in correlation loadings plot.

cvTrainAndTestData()
Returns a list consisting of dictionaries holding training and test sets.

modelSettings()
Returns a dictionary holding settings under which PLS2 was run.

regressionCoefficients(*numComp=1*)
Returns regression coefficients from the fitted model using all available samples and a chosen number of

components.

scoresRegressionCoeffs()
Returns a one dimensional array holding regression coefficients between scores of array X and Y.

CHAPTER 5

Matrix correlation coefficient methods

This module provides statistical tools for computation of matrix correlation coefficients (MCC). The MCCs provide information on to what degree multivariate data contained in two data arrays are correlated.

hoggorm.mat_corr_coeff.**RV2coeff**(*dataList*)

This function computes the RV matrix correlation coefficients between pairs of arrays. The number and order of objects (rows) for the two arrays must match. The number of variables in each array may vary. The RV2 coefficient is a modified version of the RV coefficient with values -1 <= RV2 <= 1. RV2 is independent of object and variable size.

Reference: Matrix correlations for high-dimensional data - the modified RV-coefficient

Parameters **dataList** (*list*) – A list holding an arbitrary number of numpy arrays for which the RV coefficient will be computed.

Returns A list holding an arbitrary number of numpy arrays for which the RV coefficient will be computed.

Return type numpy array

Examples

```
>>> import hoggorm as ho
>>> import numpy as np
>>>
>>> # Generate some random data. Note that number of rows must match across arrays
>>> arr1 = np.random.rand(50, 100)
>>> arr2 = np.random.rand(50, 20)
>>> arr3 = np.random.rand(50, 500)
>>>
>>> # Center the data before computation of RV coefficients
>>> arr1_cent = arr1 - np.mean(arr1, axis=0)
>>> arr2_cent = arr2 - np.mean(arr2, axis=0)
>>> arr3_cent = arr3 - np.mean(arr3, axis=0)
>>>
```

(continues on next page)

(continued from previous page)

```
>>> # Compute RV matrix correlation coefficients on mean centered data
>>> rv_results = ho.RVcoeff([arr1_cent, arr2_cent, arr3_cent])
>>> array([[ 1.        , -0.00563174,  0.04028299],
           [-0.00563174,  1.        ,  0.08733739],
           [ 0.04028299,  0.08733739,  1.        ]])
>>>
>>> # Get RV for arr1_cent and arr2_cent
>>> rv_results[0, 1]
    -0.00563174
>>>
>>> # or
>>> rv_results[1, 0]
    -0.00563174
>>>
>>> # Get RV for arr2_cent and arr3_cent
>>> rv_results[1, 2]
    0.08733739
>>>
>>> # or
>>> rv_results[2, 1]
    0.08733739
```

hoggorm.mat_corr_coeff.**RVcoeff**(*dataList*)

This function computes the RV matrix correlation coefficients between pairs of arrays. The number and order of objects (rows) for the two arrays must match. The number of variables in each array may vary.

Reference: The STATIS method

Parameters **dataList** (*list*) – A list holding numpy arrays for which the RV coefficient will be computed.

Returns A numpy array holding RV coefficients for pairs of numpy arrays. The diagonal in the result array holds ones, since RV is computed on identical arrays, i.e. first array in `dataList` against frist array in

Return type numpy array

Examples

```
>>> import hoggorm as ho
>>> import numpy as np
>>>
>>> # Generate some random data. Note that number of rows must match across arrays
>>> arr1 = np.random.rand(50, 100)
>>> arr2 = np.random.rand(50, 20)
>>> arr3 = np.random.rand(50, 500)
>>>
>>> # Center the data before computation of RV coefficients
>>> arr1_cent = arr1 - np.mean(arr1, axis=0)
>>> arr2_cent = arr2 - np.mean(arr2, axis=0)
>>> arr3_cent = arr3 - np.mean(arr3, axis=0)
>>>
>>> # Compute RV matrix correlation coefficients on mean centered data
>>> rv_results = ho.RVcoeff([arr1_cent, arr2_cent, arr3_cent])
>>> array([[ 1.        ,  0.41751839,  0.77769025],
           [ 0.41751839,  1.        ,  0.51194496],
```

(continues on next page)

(continued from previous page)

```
           [ 0.77769025,  0.51194496,  1.        ]])
>>>
>>> # Get RV for arr1_cent and arr2_cent
>>> rv_results[0, 1]
    0.41751838661314689
>>>
>>> # or
>>> rv_results[1, 0]
    0.41751838661314689
>>>
>>> # Get RV for arr2_cent and arr3_cent
>>> rv_results[1, 2]
    0.51194496245209853
>>>
>>> # or
>>> rv_results[2, 1]
    0.51194496245209853
```

class hoggorm.mat_corr_coeff.**SMI**(*X1*, *X2*, ***kargs*)

Similarity of Matrices Index (SMI)

A similarity index for comparing coupled data matrices. A two-step process starts with extraction of stable subspaces using Principal Component Analysis or some other method yielding two orthonormal bases. These bases are compared using Orthogonal Projection (OP / ordinary least squares) or Procrustes Rotation (PR). The result is a similarity measure that can be adjusted to various data sets and contexts and which includes explorative plotting and permutation based testing of matrix subspace equality.

Reference: A similarity index for comparing coupled matrices

Parameters

- **X1** (*numpy array*) – first matrix to be compared.
- **X2** (*numpy array*) – second matrix to be compared.
- **ncomp1** (*int, optional*) – maximum number of subspace components from the first matrix.
- **ncomp2** (*int, optional*) – maximum number of subspace components from the second matrix.
- **projection** (*list, optional*) – type of projection to apply, defaults to "Orthogonal", alternatively "Procrustes".
- **Scores1** (*numpy array, optional*) – user supplied score-matrix to replace singular value decomposition of first matrix.
- **Scores2** (*numpy array, optional*) – user supplied score-matrix to replace singular value decomposition of second matrix.

Returns

Return type An SMI object containing all combinations of components.

EXAMPLES

```
>>> import numpy as np
>>> import hoggorm as ho
```

```
>>> X1 = ho.center(np.random.rand(100,300))
>>> U, s, V = np.linalg.svd(X1, 0)
>>> X2 = np.dot(np.dot(np.delete(U, 2,1), np.diag(np.delete(s,2))), np.delete(V,2,
↪0))
```

```
>>> smiOP = ho.SMI(X1, X2, ncomp1 = 10, ncomp2 = 10)
>>> smiPR = ho.SMI(X1, X2, ncomp1 = 10, ncomp2 = 10, projection = "Procrustes")
>>> smiCustom = ho.SMI(X1, X2, ncomp1 = 10, ncomp2 = 10, Scores1 = U)
```

```
>>> print(smiOP.smi)
>>> print(smiOP.significance())
>>> print(smiPR.significance(B = 100))
```

significance(***kargs*)

Significance estimation for Similarity of Matrices Index (SMI)

For each combination of components significance is estimated by sampling from a null distribution of no similarity, i.e. when the rows of one matrix is permuted B times and corresponding SMI values are computed. If the vector replicates is included, replicates will be kept together through permutations.

Parameters

- **integer** (*B*) – number of permutations, default = 10000.
- **replicates** (*numpy array*) – integer vector of replicates (must be balanced).

Returns

Return type An array containing P-values for all combinations of components.

CHAPTER 6

Utililty classes and functions

There are number of functions and classes that might be useful for working with data outside the hoggorm package. They are provided here for convenience.

6.1 Functions in hoggorm.statTools module

The hoggorm.statTools module provides some functions that can be useful when working with multivariate data sets.

`hoggorm.statTools.`**`center`**(*arr*, *axis=0*)

This function centers an array column-wise or row-wise.

Parameters **`arrX`** (*numpy array*) – A numpy array containing the data

Returns Mean centered data.

Return type numpy array

Examples

```
>>> import hoggorm as ho
>>> # Column centering of array
>>> centData = ho.center(data, axis=0)
```

```
>>> # Row centering of array
>>> centData = ho.center(data, axis=1)
```

`hoggorm.statTools.`**`matrixRank`**(*arr*, *tol=1e-08*)

Computes the rank of an array/matrix, i.e. number of linearly independent variables. This is not the same as numpy.rank() which only returns the number of ways (2-way, 3-way, etc) an array/matrix has.

Parameters **`arrX`** (*numpy array*) – A numpy array containing the data

Returns Rank of matrix.

Return type scalar

Examples

```
>>> import hoggorm as ho
>>>
>>> # Get the rank of the data
>>> ho.matrixRank(myData)
>>> 8
```

hoggorm.statTools.**ortho**(*arr1*, *arr2*)

This function orthogonalises arr1 with respect to arr2. The function then returns orthogonalised array arr1_orth.

Parameters

- **arr1** (*numpy array*) – A numpy array containing some data
- **arr2** (*numpy array*) – A numpy array containing some data

Returns A numpy array holding orthogonalised numpy array `arr1`.

Return type numpy array

Examples

some examples

hoggorm.statTools.**standardise**(*arr*, *mode=0*)

This function standardises the input array either column-wise (mode = 0) or row-wise (mode = 1).

Parameters

- **arrX** (*numpy array*) – A numpy array containing the data
- **selection** (*int*) – An integer indicating whether standardisation should happen column wise or row wise.

Returns Standardised data.

Return type numpy array

Examples

```
>>> import hoggorm as ho
>>> # Standardise array column-wise
>>> standData = ho.standardise(data, mode=0)
```

```
>>> # Standardise array row-wise
>>> standData = ho.standarise(data, mode=1)
```

6.2 Cross validation classes in hoggorm.cross_val module

hoggorm classes PCA, PLSR and PCR use a number classes for computation of the models which are found in the hoggorm.cross_val module.

The cross validation classes in this module are used inside the multivariate statistical methods and may be called upon using the `cvType` input parameter for these methods. They are not intended to be used outside the multivariate statistical methods, even though it is possible. They are shown here to illustrate how the different cross validation options work.

The code in this module is based on the cross_val.py module from scikt-learn 0.4. It is adapted to work with hoggorm.

Authors:

Alexandre Gramfort <alexandre.gramfort@inria.fr>

Gael Varoquaux <gael.varoquaux@normalesup.org>

License: BSD Style.

class hoggorm.cross_val.**KFold**(*n*, *k*)

K-Folds cross validation iterator: Provides train/test indexes to split data in train test sets

__init__(*n*, *k*)

K-Folds cross validation iterator: Provides train/test indexes to split data in train test sets

Parameters

- **n** (*int*) – Total number of elements
- **k** (*int*) – number of folds

Examples

```
>>> import hoggorm as ho
>>> X = [[1, 2], [3, 4], [1, 2], [3, 4]]
>>> y = [1, 2, 3, 4]
>>> kf = ho.KFold(4, k=2)
>>> for train_index, test_index in kf:
...     print "TRAIN:", train_index, "TEST:", test_index
...     X_train, X_test, y_train, y_test = cross_val.split(train_index, test_
↪index, X, y)
TRAIN: [False False  True  True] TEST: [ True  True False False]
TRAIN: [ True  True False False] TEST: [False False  True  True]
```

Notes

All the folds have size trunc(n/k), the last one has the complementary

class hoggorm.cross_val.**LeaveOneLabelOut**(*labels*)

Leave-One-Label_Out cross-validation iterator: Provides train/test indexes to split data in train test sets

__init__(*labels*)

Leave-One-Label_Out cross validation: Provides train/test indexes to split data in train test sets

Parameters **labels** (*list*) – List of labels

Examples

```
>>> import hoggorm as ho
>>> X = [[1, 2], [3, 4], [5, 6], [7, 8]]
>>> y = [1, 2, 1, 2]
>>> labels = [1, 1, 2, 2]
>>> lolo = ho.LeaveOneLabelOut(labels)
>>> for train_index, test_index in lol:
...     print "TRAIN:", train_index, "TEST:", test_index
...     X_train, X_test, y_train, y_test = cross_val.split(train_index,
↪     test_index, X, y)
...     print X_train, X_test, y_train, y_test
TRAIN: [False False  True  True] TEST: [ True  True False False]
[[5 6]
[7 8]] [[1 2]
[3 4]] [1 2] [1 2]
TRAIN: [ True  True False False] TEST: [False False  True  True]
[[1 2]
[3 4]] [[5 6]
[7 8]] [1 2] [1 2]
```

class hoggorm.cross_val.**LeaveOneOut**(*n*)

Leave-One-Out cross validation iterator: Provides train/test indexes to split data in train test sets

__init__(*n*)

Leave-One-Out cross validation iterator: Provides train/test indexes to split data in train test sets

Parameters **n** (*int*) – Total number of elements

Examples

```
>>> import hoggorm as ho
>>> X = [[1, 2], [3, 4]]
>>> y = [1, 2]
>>> loo = ho.LeaveOneOut(2)
>>> for train_index, test_index in loo:
...     print "TRAIN:", train_index, "TEST:", test_index
...     X_train, X_test, y_train, y_test = cross_val.split(train_index, test_
↪index, X, y)
...     print X_train, X_test, y_train, y_test
TRAIN: [False  True] TEST: [ True False]
[[3 4]] [[1 2]] [2] [1]
TRAIN: [ True False] TEST: [False  True]
[[1 2]] [[3 4]] [1] [2]
```

class hoggorm.cross_val.**LeavePOut**(*n*, *p*)

Leave-P-Out cross validation iterator: Provides train/test indexes to split data in train test sets

__init__(*n*, *p*)

Leave-P-Out cross validation iterator: Provides train/test indexes to split data in train test sets

Parameters

- **n** (*int*) – Total number of elements
- **p** (*int*) – Size test sets

Examples

```
>>> import hoggorm as ho
>>> X = [[1, 2], [3, 4], [5, 6], [7, 8]]
>>> y = [1, 2, 3, 4]
>>> lpo = ho.LeavePOut(4, 2)
>>> for train_index, test_index in lpo:
...     print "TRAIN:", train_index, "TEST:", test_index
...     X_train, X_test, y_train, y_test = cross_val.split(train_index, test_
↪index, X, y)
TRAIN: [False False  True  True] TEST: [ True  True False False]
TRAIN: [False  True False  True] TEST: [ True False  True False]
TRAIN: [False  True  True False] TEST: [ True False False  True]
TRAIN: [ True False False  True] TEST: [False  True  True False]
TRAIN: [ True False  True False] TEST: [False  True False  True]
TRAIN: [ True  True False False] TEST: [False False  True  True]
```

hoggorm.cross_val.**split**(*train_indexes*, *test_indexes*, **args*)

For each arg return a train and test subsets defined by indexes provided in train_indexes and test_indexes

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

Symbols

C

H

K

L

M

N

O

R

S

X

Y