# 主成分回帰 PCR: Principal Component Regression

青木繁伸

2020 年 3 月 17 日

## 1　目的

特異値分解により主成分回帰を行う。

R の **pls** パッケージに含まれる pcr() を **Python** に翻訳・修正したものである。

R の **pls** の情報

```
Package:            pls
Title:              Partial Least Squares and Principal Component Regression
Version:            2.7-0
Date:               2018-08-20
Authors@R:          c(person("Bjørn-Helge", "Mevik", role = c("aut", "cre"), email =
                    "b-h@mevik.net"), person("Ron", "Wehrens", role = "aut"), person("Kristian
                    Hovde", "Liland", role = "aut"), person("Paul", "Hiemstra", role = "ctb"))
Author:             Bjørn-Helge Mevik [aut, cre], Ron Wehrens [aut], Kristian Hovde Liland [aut],
                    Paul Hiemstra [ctb]
Maintainer:         Bjørn-Helge Mevik <b-h@mevik.net>
Encoding:           UTF-8
LazyData:           yes
Description:        Multivariate regression methods Partial Least Squares Regression (PLSR),
                    Principal Component Regression (PCR) and Canonical Powered Partial Least
                    Squares (CPPLS).
Depends:            R (>= 2.10)
Imports:            grDevices, graphics, methods, stats
Suggests:           MASS, parallel, Rmpi, testthat, RUnit
License:            GPL-2
URL:                http://mevik.net/work/software/pls.html, https://github.com/bhmevik/pls
BugReports:         https://github.com/bhmevik/pls/issues
NeedsCompilation:   no
Packaged:           2018-08-20 18:36:26 UTC; bhm
Repository:         CRAN
Date/Publication:   2018-08-21 05:10:11 UTC
Built:              R 3.5.0; ; 2018-08-22 14:03:28 UTC; unix
```

参考文献

Martens, H., Næs, T. (1989) *Multivariate calibration*. Chichester: Wiley.

Seasholtz, M. B. and Kowalski, B. R. (1992) The effect of mean centering on prediction in multivariate calibration. *Journal of Chemometrics*, **6**(2), 103 − 111.

# 2 使用法

```
import sys
sys.path.append("statlib")
from multi import pcr

pcr(x, y, ncomp=None, center=True, verbose=True)
```

予測値，スコアのプロット

```
import sys
sys.path.append("statlib")

from multi import pls_plot
pls_plot(obj, type="p", ncomp=None, ny=1, ax1=1, ax2=2,
     txt=None, color="blue", color2= "red", alpha=0.3)
```

新しいデータに対する予測値を求める

```
import sys
sys.path.append("statlib")

from multi import pls_predict
pls_predict(obj, newX=None, ncomp=None, comps=None, type="response", verbose=True):
```

## 2.1 引数

| | |
|---|---|
| x | 独立変数データ行列 |
| y | 従属変数データ行列（2 列以上でもよい） |
| ncomp | 使用する主成分の個数（指定しなくてもよい） |
| center | デフォルトで各変数を中心化する True |
| verbose | 必要最小限のプリント出力をする |
| obj | pls 回帰プログラムの戻り値 |
| type | 実測値と予測値の対比プロットの場合（デフォルト）は "p"，スコアの二次元配置をプロットする場合は "s" を指定する |
| ncomp | 回帰に使うコンポーネント数（デフォルトは最大数） |
| ny | 複数の従属変数を分析した場合は何番目の従属変数かを表す（デフォルトは 1） |
| ax1 | 横軸にとるスコアの番号（デフォルトは 1） |
| ax2 | 縦軸にとるスコアの番号（デフォルトは 2） |

| | |
|---|---|
| color | ドットの描画色（デフォルトは青） |
| txt | ドットに添えるテキスト（デフォルトは None） |
| color2 | ドットに添えるテキストの描画色（デフォルトは赤） |
| alpha | アルファチャネル（デフォルトは 0.3） |
| newX | 新しいデータセット（独立変数データ行列のみ）<br>デフォルト（None）の場合は分析に用いたデータセットが仮定される |
| ncomp | 使用する合成変数（主成分）の数。表示されるのはそれぞれの合成変数に対して。 |
| comps | 使用する合成変数（主成分）のセット〔デフォルトの場合は None)。表示されるのは総合した 1 個のモデルにおける値。 |
| type | type="response" の場合（デフォルト）は予測値を返す。type="score" の場合は comps で指定された合成変数（主成分得点）の値を返す。 |

### 2.2 戻り値の名前

| | |
|---|---|
| "coefficients" | ncomp までの主成分得点に対する回帰係数 |
| "scores" | 主成分得点（各主成分得点の不偏分散が固有値に等しい） |
| "loadings" | 平均値 0 に調整された独立変数に掛けて主成分得点を求めるときの重み（固有ベクトル） |
| "yLoadings" | 従属変数の負荷量 |
| "projection" | 独立変数データ行列を主成分得点（スコア）に変換するための射影子<br>pcr() の場合は loadings と同じ |
| "xMeans" | 独立変数データの平均値ベクトル |
| "yMeans" | 従属変数データの平均値ベクトル |
| "fittedValues" | 予測値 |
| "residuals" | 残差 |
| "xVariances" | 各主成分で説明される独立変数の分散 |
| "xTotalVariance" | 独立変数の全分散（xVariances の合計） |

## 3 使用例

### 3.1 3 変数を使って 1 変数を予測する例

使用例を使って，pcr() と pca() + mreg() の関係について説明しよう。
iris データセットの iris["sl"] を iris["sw"]，iris["pl"]，iris["pw"] で予測する。

```
import pandas as pd

df = pd.read_csv("../Python/data/iris.csv")
x = df.loc[:, ["sw", "pl", "pw"]]
y = pd.DataFrame(df.loc[:, "sl"])
```

pcr() の戻り値を a に代入しておく。分析に使用される主成分の個数は独立変数の個数に等しい。

```
import sys
sys.path.append("statlib")
from multi import pcr

a = pcr(x, y, verbose=False)
```

主成分分析の主成分得点も第 3 主成分まで求めておく。

以下に示す主成分分析プログラムは，分析に使用するデータの中心化（平均値を 0 にする）は行うが，尺度化（分散を 1 に基準化する）は行わない。

```
import numpy as np
from scipy.linalg import svd

def pca(dat, npca=None, verbose=True):
    nr, nc = dat.shape
    dat -= dat.mean(axis=0)
    u, s, v = svd(dat, full_matrices=False)
    eval = s**2 / (nr - 1)
    if npca is None or npca > nc or npca < 1 or npca != int(npca):
        npca = np.sum(eval >= 1)
    s = s[:npca]
    fl = v[:npca, :].T
    fs = u[:, :npca] * s
    return fl, eval, fs

fl, eval, fs = pca(x, npca=3)
```

fl は主成分得点を計算するための固有ベクトルで，元のデータ行列に掛けることで主成分得点になる。

```
factor_score = np.array(x) @ fl
print(factor_score[:10, :])
```

```
[[-2.59236698 -0.18279054  0.03171688]
 [-2.54255916  0.31061706 -0.03206491]
 [-2.65421811  0.11739671 -0.04614231]
 [-2.4607849   0.20779286  0.02028156]
 [-2.60232854 -0.28147206  0.04447323]
 [-2.27992642 -0.62123657  0.01963468]
 [-2.54386575 -0.09975496 -0.07197846]
 [-2.49066959 -0.08825171  0.05855063]
 [-2.5325976   0.40929858 -0.04482127]
 [-2.49932457  0.2234388   0.11122054]]
```

これは pca() の戻り値 fs である。

```
print(fs[:10, :]) # 主成分得点
```

```
[[-2.59236698 -0.18279054  0.03171688]
```

```
 [-2.54255916  0.31061706 -0.03206491]
 [-2.65421811  0.11739671 -0.04614231]
 [-2.4607849   0.20779286  0.02028156]
 [-2.60232854 -0.28147206  0.04447323]
 [-2.27992642 -0.62123657  0.01963468]
 [-2.54386575 -0.09975496 -0.07197846]
 [-2.49066959 -0.08825171  0.05855063]
 [-2.5325976   0.40929858 -0.04482127]
 [-2.49932457  0.2234388   0.11122054]]
```

主成分得点は pcr() の戻り値の "scores" に対応する。

```
print(a["scores"][:10, :])
```

```
[[-2.59236698 -0.18279054  0.03171688]
 [-2.54255916  0.31061706 -0.03206491]
 [-2.65421811  0.11739671 -0.04614231]
 [-2.4607849   0.20779286  0.02028156]
 [-2.60232854 -0.28147206  0.04447323]
 [-2.27992642 -0.62123657  0.01963468]
 [-2.54386575 -0.09975496 -0.07197846]
 [-2.49066959 -0.08825171  0.05855063]
 [-2.5325976   0.40929858 -0.04482127]
 [-2.49932457  0.2234388   0.11122054]]
```

次に，主成分得点を用いて y を予測する（重回帰分析）。

```
from scipy.linalg import solve

def mreg(x, y):
    dat = np.hstack((x, y))
    r = np.corrcoef(dat, rowvar=False)
    beta = solve(r[:-1, :-1], r[:-1, -1])
    SS = dat.var(axis=0, ddof=0) * dat.shape[0]
    b = beta / np.sqrt(SS[:-1] / SS[-1])
    means = dat.mean(axis=0)
    b0 = means[-1] - sum(b * means[:-1])
    fittedvalues = x @ b + b0
    return b0, b, fittedvalues
```

主成分を用いて重回帰分析をしたときの予測値は fittedvalues で，これは pcr() の戻り値の "fittedValues" に対応する。

第 1 主成分だけを用いる場合，以下の 4 通りの計算方法の結果が一致する。

```
b0, b, fittedvalues = mreg(fs[:, :1], y) # 第 1 主成分だけを使う
print(fittedvalues[:10])
```

```
[4.88097434 4.89946439 4.85801347 4.92982128 4.87727633 4.996961
 4.89897935 4.91872725 4.9031624  4.91551428]
```

```
print(a["fittedValues"][0, :10, 0]) # 第 1 主成分だけを使った結果
```

```
[4.88097434 4.89946439 4.85801347 4.92982128 4.87727633 4.996961
 4.89897935 4.91872725 4.9031624  4.91551428]
```

```
fitted_values = a["scores"][:, 0] *  b + b0 # 主成分得点で予測値
print(fitted_values[:10])
```

```
[4.88097434 4.89946439 4.85801347 4.92982128 4.87727633 4.996961
 4.89897935 4.91872725 4.9031624  4.91551428]
```

```
fitted_values2 = np.array(x) @ fl[:, 0] * b + b0
print(fitted_values2[:10])
```

```
[4.88097434 4.89946439 4.85801347 4.92982128 4.87727633 4.996961
 4.89897935 4.91872725 4.9031624  4.91551428]
```

上の最後の 2 つの計算を比較することにより，pcr() の "coefficients" は，pca() の固有ベクトル（重み） fl と mreg() の偏回帰係数 b の積になっていることがわかる。

```
print(fl[:, 0] * b)
```

```
[-0.0369801   0.34054899  0.14307001]
```

```
print(a["coefficients"][0, :, 0])
```

```
[-0.0369801   0.34054899  0.14307001]
```

第 2 主成分までを用いる場合，以下の 4 通りの計算方法の結果が一致する。

```
b0, b, fittedvalues = mreg(fs[:, :2], y) # 第 2 主成分までを使う
print(fittedvalues[:10])
```

```
[4.9878275  4.71788817 4.78938733 4.80835261 5.04181537 5.36011487
 4.95729273 4.97031621 4.6639003  4.78489952]
```

```
print(a["fittedValues"][1, :10, 0]) # 第 2 主成分までを使った結果
```

```
[4.9878275  4.71788817 4.78938733 4.80835261 5.04181537 5.36011487
 4.95729273 4.97031621 4.6639003  4.78489952]
```

```
fitted_values = a["scores"][:, :2] @  b + b0 # 主成分得点で予測値
print(fitted_values[:10])
```

```
[4.9878275  4.71788817 4.78938733 4.80835261 5.04181537 5.36011487
 4.95729273 4.97031621 4.6639003  4.78489952]
```

```
fitted_values2 = np.array(x) @ fl[:, :2] @ b + b0
print(fitted_values2[:10])
```

```
[4.9878275  4.71788817 4.78938733 4.80835261 5.04181537 5.36011487
 4.95729273 4.97031621 4.6639003  4.78489952]
```

上の最後の 2 つの計算を比較することにより，pcr() の "coefficients" は，pca() の固有ベクトル（重み） fl と mreg() の偏回帰係数 b の積になっていることがわかる。

```
print(fl[:, :2] @ b)
```

```
[0.53987867 0.36476572 0.2345309 ]
```

```
print(a["coefficients"][1, :, 0])
```

```
[0.53987867 0.36476572 0.2345309 ]
```

第 3 主成分までを用いる場合，以下の 4 通りの計算方法の結果が一致する。

```
b0, b, fittedvalues = mreg(fs[:, :3], y) # 第 3 主成分までを使う
print(fittedvalues[:10])
```

```
[5.01541576 4.68999718 4.74925142 4.82599409 5.08049948 5.37719368
 4.89468378 5.02124524 4.62491347 4.88164236]
```

```
print(a["fittedValues"][2, :10, 0]) # 第 3 主成分までを使った結果
```

```
[5.01541576 4.68999718 4.74925142 4.82599409 5.08049948 5.37719368
 4.89468378 5.02124524 4.62491347 4.88164236]
```

```
fitted_values = a["scores"][:, :3] @  b + b0 # 主成分得点で予測値
print(fitted_values[:10])
```

```
[5.01541576 4.68999718 4.74925142 4.82599409 5.08049948 5.37719368
 4.89468378 5.02124524 4.62491347 4.88164236]
```

```
fitted_values2 = np.array(x) @ fl @ b + b0
print(fitted_values2[:10])
```

```
[5.01541576 4.68999718 4.74925142 4.82599409 5.08049948 5.37719368
 4.89468378 5.02124524 4.62491347 4.88164236]
```

上の最後の 2 つの計算を比較することにより，pcr() の "coefficients" は，pca() の固有ベクトル（重み） fl と mreg() の偏回帰係数 b の積になっていることがわかる。

```
print(fl @ b)
```

```
[ 0.65083716  0.70913196 -0.55648266]
```

```
print(a["coefficients"][2, :, 0])
```

```
[ 0.65083716  0.70913196 -0.55648266]
```

pca() の戻り値の fl と pcr() の戻り値 "loadings" が一致する。

```
print(fl)
```

```
[[-0.09961563 -0.98681521  0.12756357]
 [ 0.91735823 -0.04142686  0.39590111]
 [ 0.38539668 -0.15645943 -0.90938982]]
```

```
print(a["loadings"])
```

```
[[-0.09961563 -0.98681521  0.12756357]
 [ 0.91735823 -0.04142686  0.39590111]
 [ 0.38539668 -0.15645943 -0.90938982]]
```

pca() での，それぞれの主成分得点の不偏分散は，pcr() の戻り値 eval である。

```
print(fs.var(axis=0, ddof=1))
```

```
[3.6963811  0.15685442 0.03402802]
```

```
print(eval)
```

```
[3.6963811  0.15685442 0.03402802]
```

```
print(np.sum(eval))
```

```
3.8872635346756175
```

```
print(eval/np.sum(eval)) # 割合
```

```
[0.95089542 0.04035086 0.00875372]
```

```
print(np.cumsum(eval)/np.sum(eval)) # 累積割合
```

```
[0.95089542 0.99124628 1.        ]
```

pls() での，それぞれの成分により説明される独立変数の分散は戻り値 "xVariances" である。全体の分散 "xTotalVariance" に対する割合は eval の場合と同じになっている。

```
xvar = a["xVariances"]
totvar = a["xTotalVariance"]
print(xvar)
```

```
[550.76078393  23.37130811   5.07017463]
```

```
print(xvar/totvar) # 割合
```

```
[0.95089542 0.04035086 0.00875372]
```

```
print(np.cumsum(xvar)/totvar) # 累積割合
```

```
[0.95089542 0.99124628 1.        ]
```

np.sum(eval) と "xTotalVariance" の関係は，

```
np.sum(eval) * (len(y)-1) == a["xTotalVariance"]
```

```
print(np.sum(eval) * (len(y)-1))
print(a["xTotalVariance"])
```

```
579.202266666667
579.2022666666667
```

yLoadings は，全ての主成分を用いて回帰分析を行ったときの偏回帰係数である。

```
print(a["yLoadings"][0])
```

```
[ 0.37122792 -0.58456616  0.86982891]
```

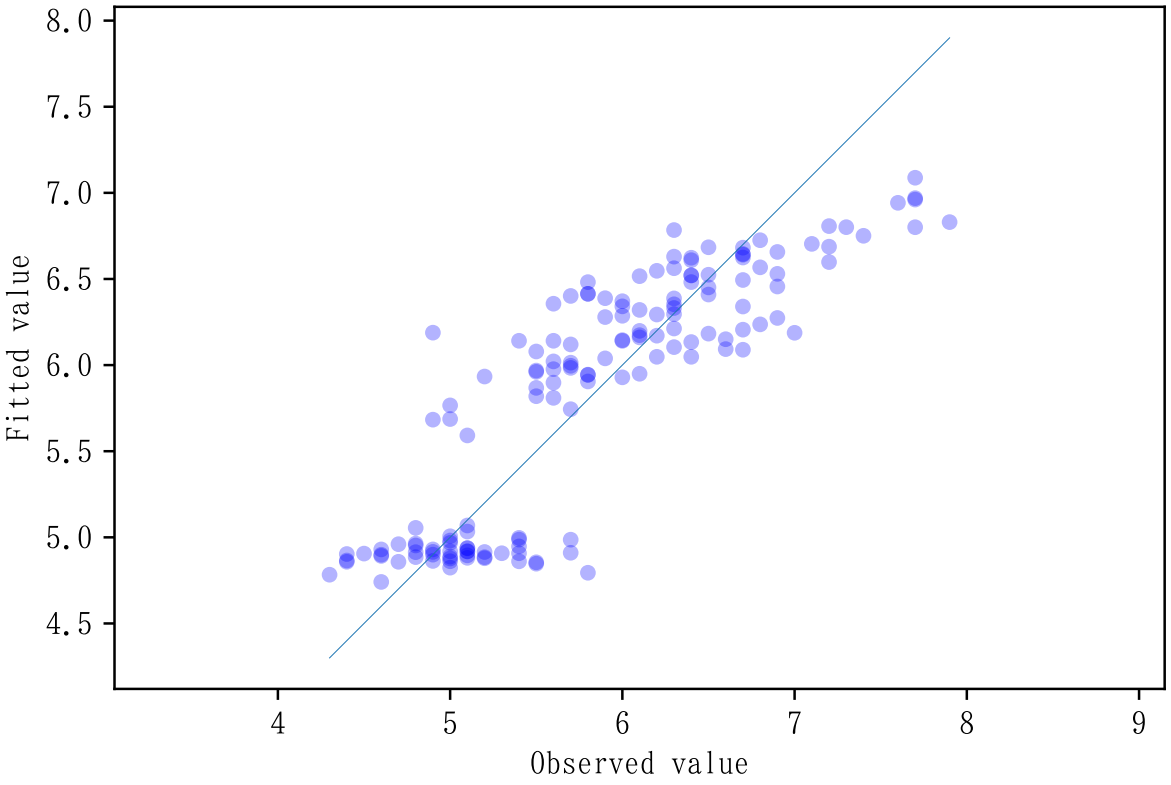
```
print(b) # 3 主成分を使用した場合の偏回帰係数
```

```
[ 0.37122792 -0.58456616  0.86982891]
```

実測値と予測値の関係図
合成変数 1 個を使う場合

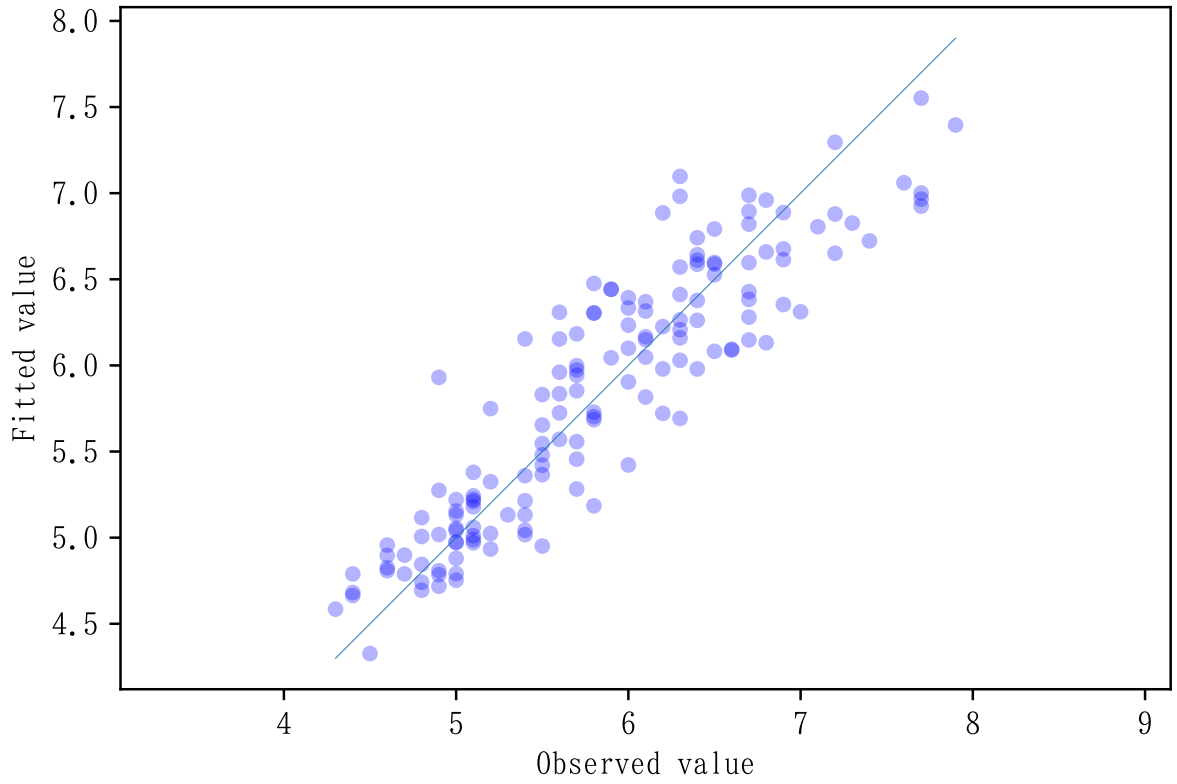
```
import sys
sys.path.append("statlib")
from multi import pls_plot

pls_plot(a, ncomp=1)
```



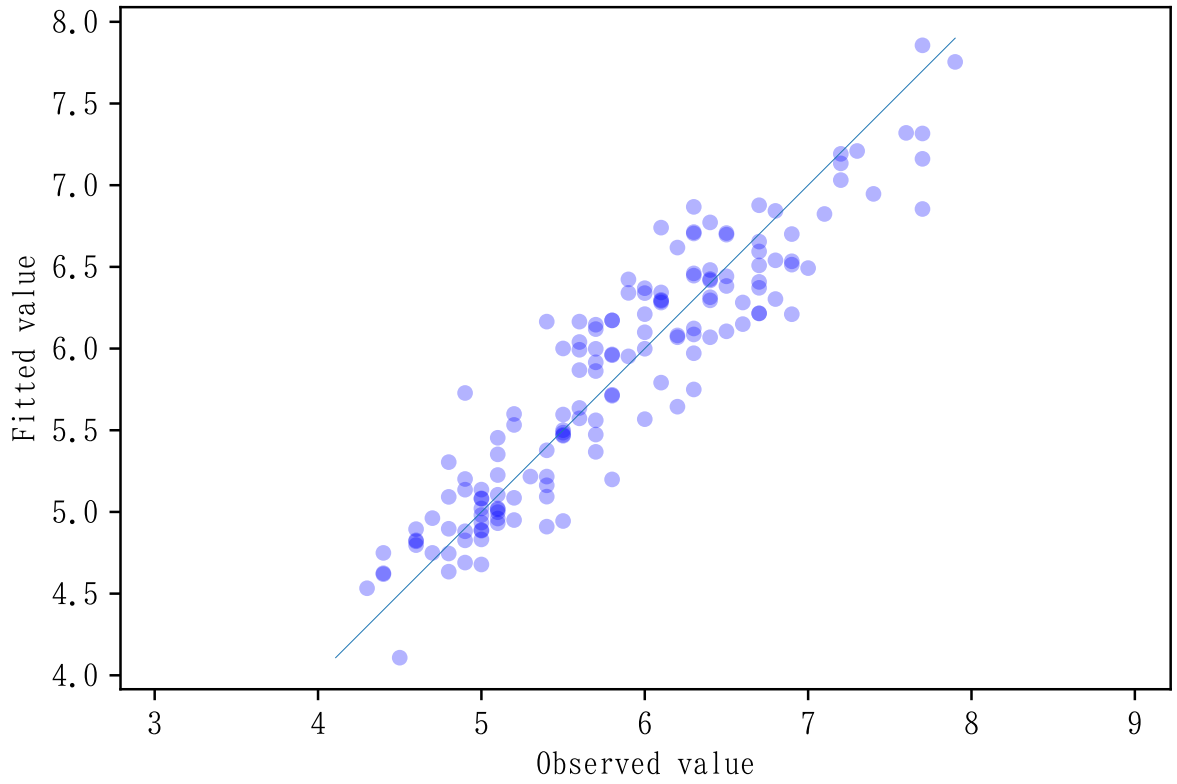

合成変数 2 個を使う場合

```
pls_plot(a, ncomp=2)
```

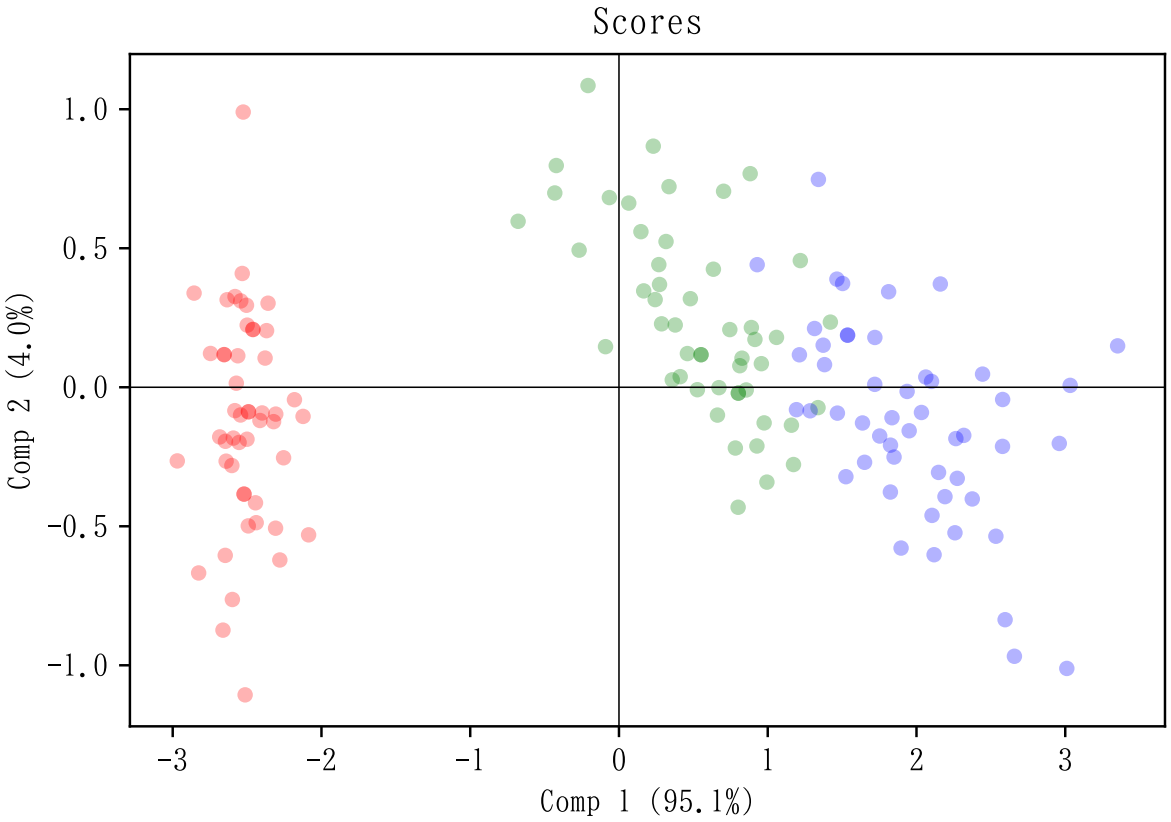



合成変数 3 個を使う場合

```
pls_plot(a, ncomp=3)
```




```
import numpy as np
color = np.hstack((np.repeat("red", 50), np.repeat("green", 50), np.
    repeat("blue", 50)))
pls_plot(a, type="s", ncomp=[1, 2], color=color)
```




### 3.2　2 変数を使って 2 変数を予測する例

予測変数は 2 個以上あっても構わない。

`iris["sl"]`, `iris[""sw]` を使って，`iris["pl"]`　と `iris["pw"]` を予測する。

```
df = pd.read_csv("../Python/data/iris.csv")
x = df.iloc[:, :2]
y = df.iloc[:, 2:4]
a = pcr(x, y)
```

```
***** Coefficients

1 comp
          pl        pw
sl  1.875806  0.758497
sw -0.159413 -0.064460

2 comps
          pl        pw
sl  1.775593  0.723292
sw -1.338623 -0.478721
```

```
***** Scores

          Comp1     Comp2
Obj1   -0.778148  0.378133
Obj2   -0.935090 -0.137007
Obj3   -1.151308  0.045339
Obj4   -1.242481 -0.062770
Obj5   -0.886256  0.469306
...          ...       ...
Obj146  0.858445  0.015414
Obj147  0.502221 -0.516662
Obj148  0.659163 -0.001522
Obj149  0.326369  0.371638
Obj150  0.061318 -0.052329

[150 rows x 2 columns]

***** Loadings

       Comp1     Comp2
sl  0.996408  0.084678
sw -0.084678  0.996408

                 Comp1  Comp2
SS loadings        1.0    1.0
Prop. var.         0.5    0.5
Cumu. prop. var.   0.5    1.0

***** y loadings

       Comp1     Comp2
pl  1.882568 -1.183461
pw  0.761231 -0.415755

                    Comp1     Comp2
SS loadings      4.123534  1.573433
Prop. var.       2.061767  0.786716
Cumu. prop. var. 2.061767  2.848483

Projection
```

```
       Comp1     Comp2
sl  0.996408  0.084678
sw -0.084678  0.996408

***** Fitted values

1 comp
              pl        pw
Obj1    2.293084  0.606983
Obj2    1.997629  0.487513
Obj3    1.590586  0.322922
Obj4    1.418946  0.253518
Obj5    2.089562  0.524687
...          ...       ...
Obj146  5.374080  1.852808
Obj147  4.703464  1.581639
Obj148  4.998919  1.701109
Obj149  4.372412  1.447776
Obj150  3.873435  1.246011

[150 rows x 2 columns]

2 comps
              pl        pw
Obj1    1.845579  0.449772
Obj2    2.159772  0.544475
Obj3    1.536929  0.304072
Obj4    1.493232  0.279615
Obj5    1.534157  0.329571
...          ...       ...
Obj146  5.355839  1.846400
Obj147  5.314913  1.796444
Obj148  5.000720  1.701742
Obj149  3.932593  1.293266
Obj150  3.935365  1.267767

[150 rows x 2 columns]

***** Residuals

1 comp
              pl        pw
```

```
Obj1   -0.893084 -0.406983
Obj2   -0.597629 -0.287513
Obj3   -0.290586 -0.122922
Obj4    0.081054 -0.053518
Obj5   -0.689562 -0.324687
...          ...       ...
Obj146 -0.174080  0.447192
Obj147  0.296536  0.318361
Obj148  0.201081  0.298891
Obj149  1.027588  0.852224
Obj150  1.226565  0.553989

[150 rows x 2 columns]

2 comps
              pl        pw
Obj1   -0.445579 -0.249772
Obj2   -0.759772 -0.344475
Obj3   -0.236929 -0.104072
Obj4    0.006768 -0.079615
Obj5   -0.134157 -0.129571
...          ...       ...
Obj146 -0.155839  0.453600
Obj147 -0.314913  0.103556
Obj148  0.199280  0.298258
Obj149  1.467407  1.006734
Obj150  1.164635  0.532233

[150 rows x 2 columns]

***** x means
sl  5.843333
sw  3.057333

***** y means
pl  3.758000
pw  1.199333

***** Explained variances of x by each component
Comp1  102.705656
Comp2   27.769611
```

```
***** Total variance of x
130.47527
```

## 3.3 新しいデータセットに対する予測

pcr() による分析結果を obj に代入する。

```
import numpy as np
import pandas as pd

df = pd.read_csv("../Python/data/pls.csv")
x = df.loc[:, ["X1", "X2", "X3", "X4", "X5"]]
y = df.loc[:, ["Y1", "Y2", "Y3"]]

import sys
sys.path.append("statlib")
from multi import pcr, pls_predict
obj = pcr(x, y, verbose=False)
```

新しいデータセット（$n = 4$）

```
newX = [[71.5, 39.5, 71.6, 46.1, 41.9],
        [38.4, 40.9, 47.2, 50.9, 53.9],
        [49.8, 54.2, 41.7, 27.0, 72.8],
        [53.0, 42.3, 71.9, 64.7, 65.2]]
```

ncomp を省略すると，理論的に可能な合成変数の数までのそれぞれについて予測値を求める。

この場合は合成変数を 1 個だけ，2 個まで，...，5 個までに対して予測値を求める。1 〜 3 列は，Y1，Y2，Y3 の 3 変数に対する予測値である。行数 4 は $n = 4$ に対応する。4 行列が comp=5 個表示される。

```
a = pls_predict(obj, newX=newX)
```

```
***** Predicted values
[[[55.21263704 53.81925873 71.51296719]
  [48.60031245 48.97412077 37.88470191]
  [44.446208   45.93022767 16.75819416]
  [53.01331219 52.20771771 60.32787208]]

 [[59.48482524 54.10640473 72.07011901]
  [51.00379753 49.13566589 38.19814926]
  [48.6782008  46.21467203 17.31010394]
  [60.19241627 52.69024586 61.26412552]]

 [[62.14191508 53.82680336 76.93587137]
  [49.29524495 49.31545418 35.06939027]
  [50.92632191 45.97810579 21.42693899]
```

```
 [59.34810382 52.77909152 59.71799214]]

[[62.17840879 53.92697114 76.9349182 ]
 [49.2974586  49.32153019 35.06933245]
 [50.54271457 44.92518207 21.43695843]
 [61.14824788 57.72011899 59.6709742 ]]

[[59.81989093 52.70381925 78.30231213]
 [47.55825186 48.41955949 36.07766941]
 [53.11102888 46.25713657 19.94793116]
 [64.74794061 59.58695707 57.58398641]]]
```

4 個までの合成変数を使った場合（ncomp=4）の予測値を求める。

```
a = pls_predict(obj, newX=newX, ncomp=4)
```

```
***** Predicted values
[[[62.17840879 53.92697114 76.9349182 ]
  [49.2974586  49.32153019 35.06933245]
  [50.54271457 44.92518207 21.43695843]
  [61.14824788 57.72011899 59.6709742 ]]]
```

1 個だけの合成変数を使った場合，2 個までの合成変数を使った場合（ncomp=4）の予測値を求める。

```
a = pls_predict(obj, newX=newX, ncomp=[1,2])
```

```
***** Predicted values
[[[55.21263704 53.81925873 71.51296719]
  [48.60031245 48.97412077 37.88470191]
  [44.446208   45.93022767 16.75819416]
  [53.01331219 52.20771771 60.32787208]]

 [[59.48482524 54.10640473 72.07011901]
  [51.00379753 49.13566589 38.19814926]
  [48.6782008  46.21467203 17.31010394]
  [60.19241627 52.69024586 61.26412552]]]
```

1，2，3，4 番目の合成変数を使った場合の予測値を求める。これは ncomp=4 を指定したときと同じ結果になる。

```
a = pls_predict(obj, newX=newX, comps=[1,2,3,4])
```

```
***** Predicted values
[[62.17840879 53.92697114 76.9349182 ]
 [49.2974586  49.32153019 35.06933245]
 [50.54271457 44.92518207 21.43695843]
```

```
 [61.14824788 57.72011899 59.6709742 ]]
```

1，4 番目の合成変数を使った場合の予測値を求める。

```
a = pls_predict(obj, newX=newX, comps=[1,4])
```

```
***** Predicted values
[[55.24913075 53.9194265  71.51201401]
 [48.6025261  48.98019678 37.88464409]
 [44.06260066 44.87730395 16.76821359]
 [54.81345625 57.14874518 60.28085414]]
```

これ以降はスコア（主成分得点）を求める（type="score"）。
5 個の主成分得点を求める。

```
a = pls_predict(obj, newX=newX, type="score")
```

```
***** Predicted scores
[[-2.50490324e+01  1.19094641e+01 -1.74312790e+01 -3.17387493e-01
   6.84254736e+00]
 [ 6.72002471e+00  6.70013069e+00  1.12085998e+01 -1.92521844e-02
   5.04579789e+00]
 [ 2.66785098e+01  1.17974125e+01 -1.47483257e+01  3.33625078e+00
  -7.45121017e+00]
 [-1.44823284e+01  2.00129953e+01  5.53893425e+00 -1.56559363e+01
  -1.04434519e+01]]
```

4 番目の主成分得点を求める。

```
a = pls_predict(obj, newX=newX, ncomp=4, type="score")
```

```
***** Predicted scores
[[ -0.31738749]
 [ -0.01925218]
 [  3.33625078]
 [-15.65593625]]
```

1 番目，2 番目の主成分得点を求める。

```
a = pls_predict(obj, newX=newX, ncomp=[1,2], type="score")
```

```
***** Predicted scores
[[-25.04903244  11.90946414]
 [  6.72002471   6.70013069]
 [ 26.67850982  11.7974125 ]
 [-14.48232842  20.01299533]]
```

1 番目，2 番目，3 番目，4 番目の主成分得点を求める。

```
a = pls_predict(obj, newX=newX, comps=[1,2,3,4], type="score")
```

```
***** Predicted scores
[[-2.50490324e+01  1.19094641e+01 -1.74312790e+01 -3.17387493e-01]
 [ 6.72002471e+00  6.70013069e+00  1.12085998e+01 -1.92521844e-02]
 [ 2.66785098e+01  1.17974125e+01 -1.47483257e+01  3.33625078e+00]
 [-1.44823284e+01  2.00129953e+01  5.53893425e+00 -1.56559363e+01]]
```

1 番目，4 番目の主成分得点を求める。

```
a = pls_predict(obj, newX=newX, comps=[1,4], type="score")
```

```
***** Predicted scores
[[-2.50490324e+01 -3.17387493e-01]
 [ 6.72002471e+00 -1.92521844e-02]
 [ 2.66785098e+01  3.33625078e+00]
 [-1.44823284e+01 -1.56559363e+01]]
```