Principal Component Analysis

import library

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib import cm
```

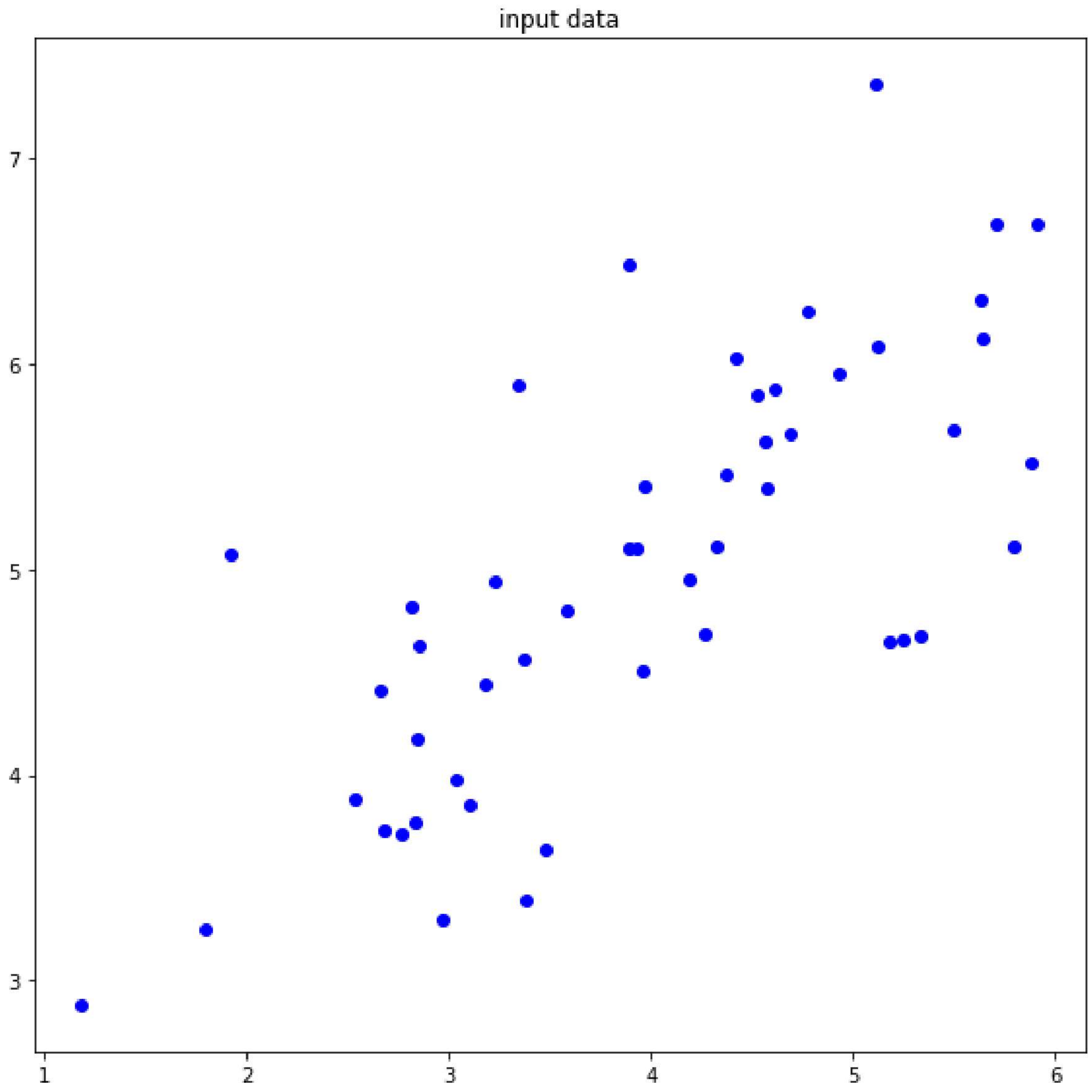
load data

In [ ]:
```
fname_data  = 'assignment_12_data.txt'
feature0    = np.genfromtxt(fname_data, delimiter=',')

number_data     = np.size(feature0, 0)
number_feature  = np.size(feature0, 1)

print('number of data : {}'.format(number_data))
print('number of feature : {}'.format(number_feature))
```

```
number of data : 50
number of feature : 2
```

plot the input data

In [ ]:
```
plt.figure(figsize=(8,8))
plt.title('input data')

x0 = feature0[:,0]
y0 = feature0[:,1]

plt.scatter(x0, y0, color='blue')

plt.tight_layout()
plt.show()
```



Normalization (Z-scoring)

- shape of feature = $n \times m$ where n is the number of data and m is the dimension of features

In [ ]:
```
def normalize(feature):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    x = feature[:, 0]
    y = feature[:, 1]
    x_tild = (x - np.mean(x)) / np.std(x)
    y_tild = (y - np.mean(y)) / np.std(y)
    feature_normalize = np.array([x_tild, y_tild]).T

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return feature_normalize
```

In [ ]:
```
feature = normalize(feature0)

x   = feature[:, 0]
```

```
y   = feature[:, 1]

min_x = np.min(x)
min_y = np.min(y)

max_x = np.max(x)
max_y = np.max(y)
```

compute covariance matrix

- shape of feature = $n \times m$ where n is the number of data and m is the dimension of features

In [ ]:
```
def compute_covariance(feature):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    n = np.size(feature, 0)
    Sigma = np.matmul(feature.T, feature) / n

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return Sigma
```

compute principal components

- `np.linalg.eig`
- `argsort()`
- return the eigenvalues and the eigenvectors in a decreasing order according to the eigenvalues

In [ ]:
```
def compute_principal_component(feature):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    Sigma = compute_covariance(feature)
    eigValue, eigVector = np.linalg.eig(Sigma)
    arg = eigValue.argsort()[::-1]

    principal_component_1 = eigValue[arg[0]] * eigVector[:, arg[0]]
    principal_component_2 = eigValue[arg[1]] * eigVector[:, arg[1]]

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return (principal_component_1, principal_component_2)
```

compute the projection of point onto the axis

- `np.matmul`

- `np.dot`
- shape of feature = $n \times m$ where n is the number of data and m is the dimension of features
- shape of vector = $m \times 1$ where m is the dimension of features

In [ ]:
```
def compute_projection_onto_line(feature, vector):

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    projection = np.matmul(feature, vector).reshape(-1,1) * (vector / (np.linalg.no

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return projection
```

compute the principal components and the projection of feature

In [ ]:
```
(principal_component_1, principal_component_2) = compute_principal_component(feature

projection1 = compute_projection_onto_line(feature, principal_component_1)
projection2 = compute_projection_onto_line(feature, principal_component_2)
```

---

functions for presenting the results

In [ ]:
```
def function_result_01():

    plt.figure(figsize=(8,8))
    plt.title('data normalized by z-scoring')
    plt.scatter(x, y, color='blue')

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
def function_result_02():

    plt.figure(figsize=(8,8))
    plt.title('principal components')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
```

```
    plt.scatter(x, y, color='blue')
    plt.annotate('',xy=(principal_component_1), xytext = (0,0), arrowprops=dict(face
    plt.annotate('',xy=(principal_component_2), xytext = (0,0), arrowprops=dict(face

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
def function_result_03():

    plt.figure(figsize=(8,8))
    plt.title('first principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    m = principal_component_1[1] / principal_component_1[0]
    extend_x = np.arange(min_x - 0.5, max_x + 0.5)

    plt.scatter(x, y, color='blue')
    plt.plot(extend_x, m*extend_x, '-', color='red')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
def function_result_04():

    plt.figure(figsize=(8,8))
    plt.title('second principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    m = principal_component_2[1] / principal_component_2[0]
    extend_x = np.arange(min_x - 0.5, max_x + 0.5)

    plt.scatter(x, y, color='blue')
    plt.plot(extend_x, m*extend_x, '-', color='red')


    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
def function_result_05():
```

```
    plt.figure(figsize=(8,8))
    plt.title('projection onto the first principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    m = principal_component_1[1] / principal_component_1[0]
    extend_x = np.arange(min_x - 0.5, max_x + 0.5)

    plt.scatter(x, y, color='blue')
    plt.plot(extend_x, m*extend_x, '-', color='red')

    plt.scatter(projection1[:, 0], projection1[:, 1], color='green')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
def function_result_06():

    plt.figure(figsize=(8,8))
    plt.title('projection onto the second principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    m = principal_component_2[1] / principal_component_2[0]
    extend_x = np.arange(min_x - 0.5, max_x + 0.5)

    plt.scatter(x, y, color='blue')
    plt.plot(extend_x, m*extend_x, '-', color='red')


    plt.scatter(projection2[:, 0], projection2[:, 1], color='green')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
def function_result_07():

    plt.figure(figsize=(8,8))
    plt.title('projection onto the first principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    m = principal_component_1[1] / principal_component_1[0]
    extend_x = np.arange(min_x - 0.5, max_x + 0.5)
```

```
    plt.scatter(x, y, color='blue')
    plt.plot(extend_x, m*extend_x, '-', color='red')


    plt.scatter(projection1[:, 0], projection1[:, 1], color='green')

    for i in range(np.size(feature, 0)) :
        proj_x = np.array([feature[i,0], projection1[i,0]])
        proj_y = np.array([feature[i,1], projection1[i,1]])
        plt.plot(proj_x, proj_y, '-', color='gray')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

In [ ]:

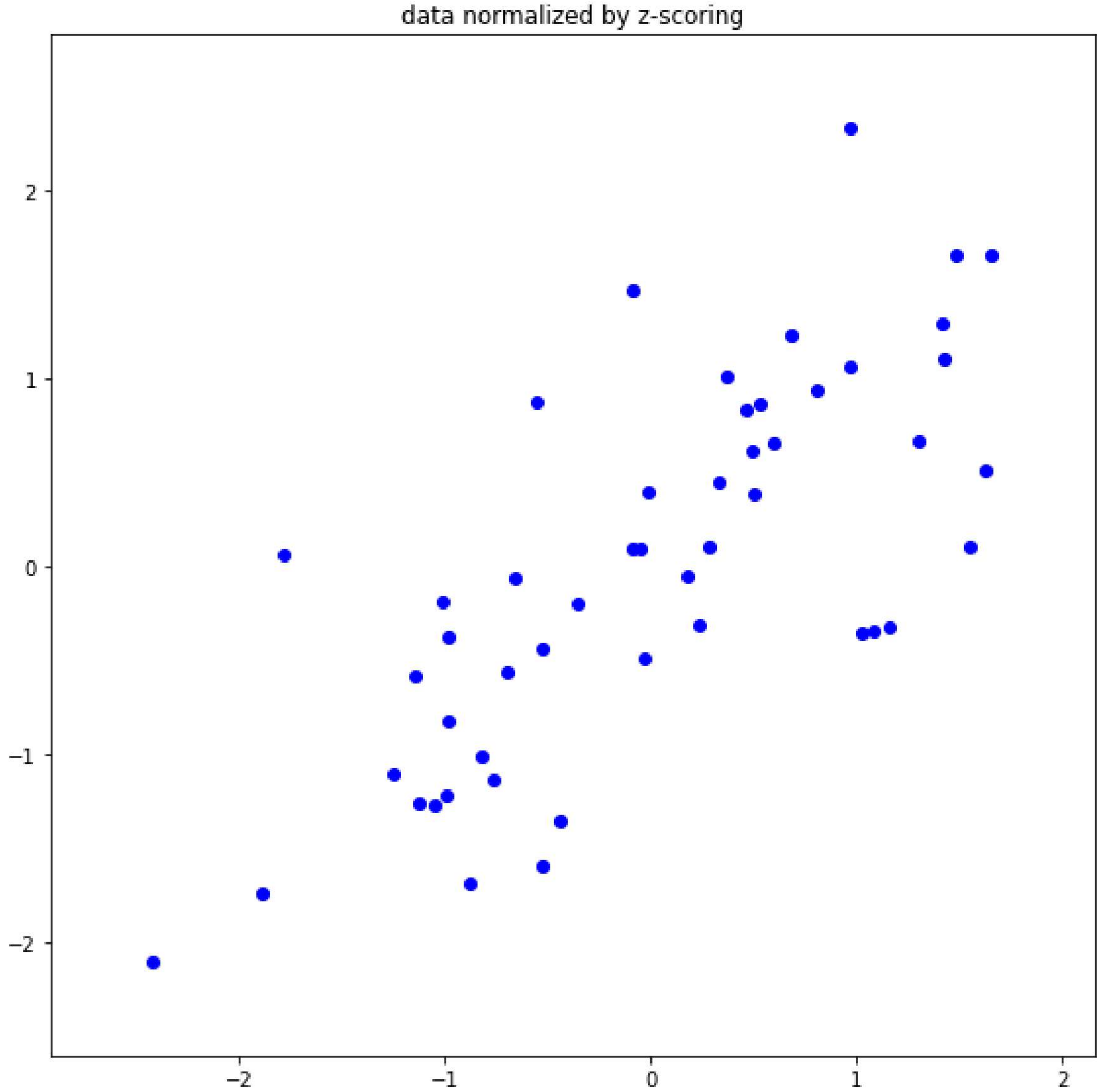
```
def function_result_08():

    plt.figure(figsize=(8,8))
    plt.title('projection to the second principle axis')

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    m = principal_component_2[1] / principal_component_2[0]
    extend_x = np.arange(min_x - 0.5, max_x + 0.5)

    plt.scatter(x, y, color='blue')
    plt.plot(extend_x, m*extend_x, '-', color='red')


    plt.scatter(projection2[:, 0], projection2[:, 1], color='green')

    for i in range(np.size(feature, 0)) :
        proj_x = np.array([feature[i,0], projection2[i,0]])
        proj_y = np.array([feature[i,1], projection2[i,1]])
        plt.plot(proj_x, proj_y, '-', color='gray')

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    plt.xlim(min_x - 0.5, max_x + 0.5)
    plt.ylim(min_y - 0.5, max_y + 0.5)

    plt.tight_layout()
    plt.show()
```

---

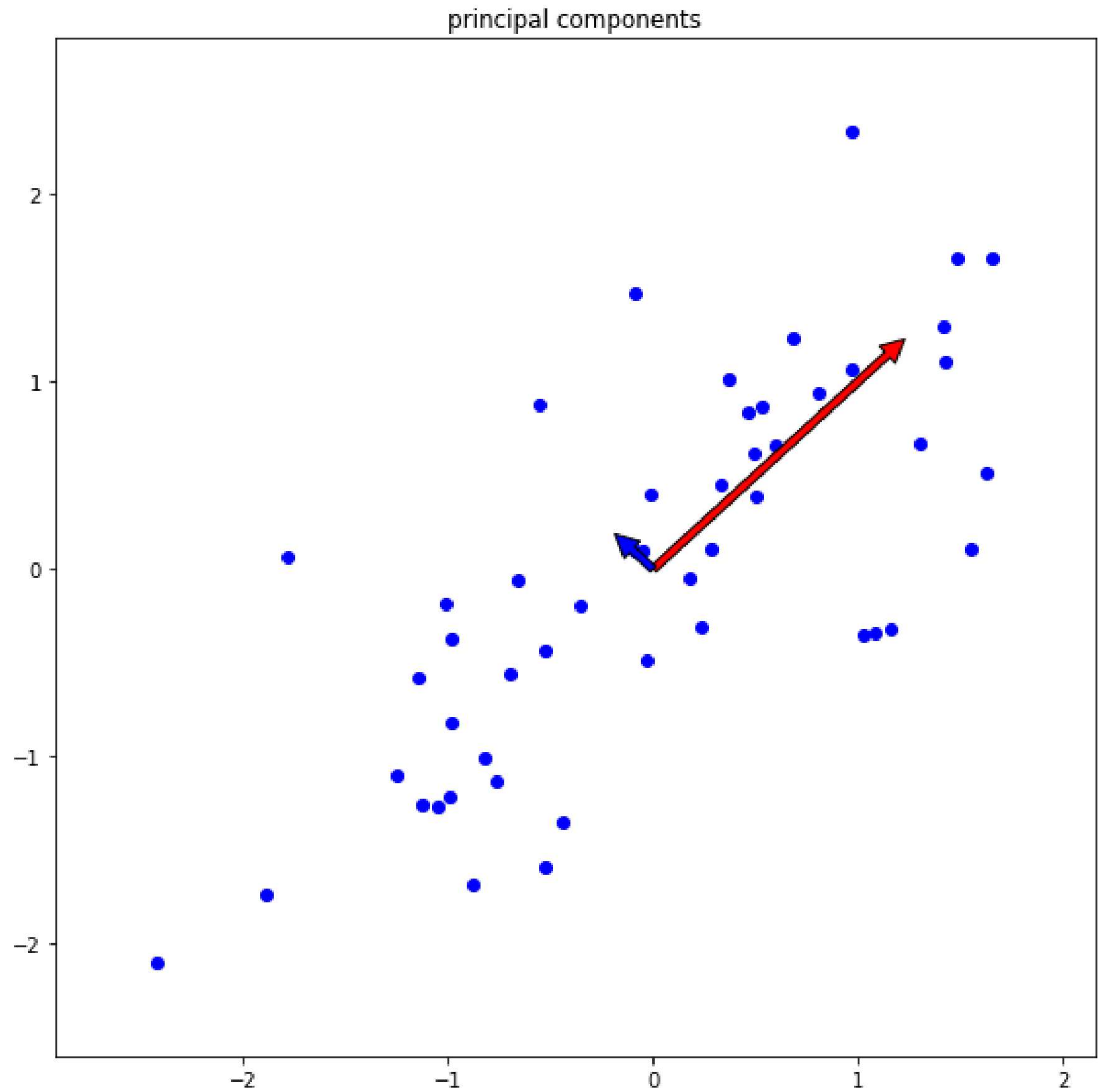
results

In [ ]:
```
number_result = 8

for i in range(number_result):
    title = '## [RESULT {:02d}]'.format(i+1)
    name_function = 'function_result_{:02d}()'.format(i+1)

    print('**************************************************')
    print(title)
    print('**************************************************')
    eval(name_function)
```

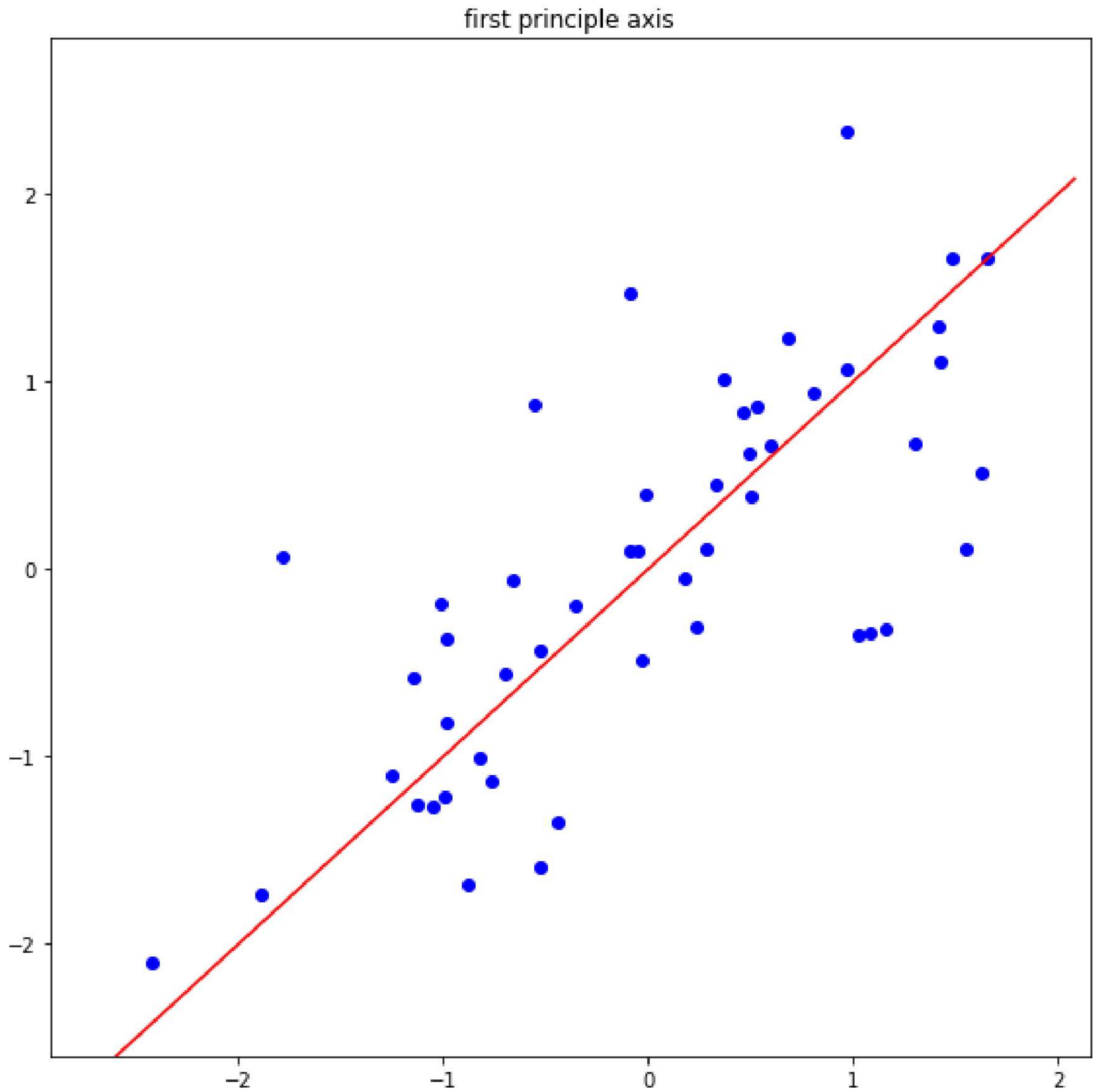
```
**************************************************
## [RESULT 01]
**************************************************
```



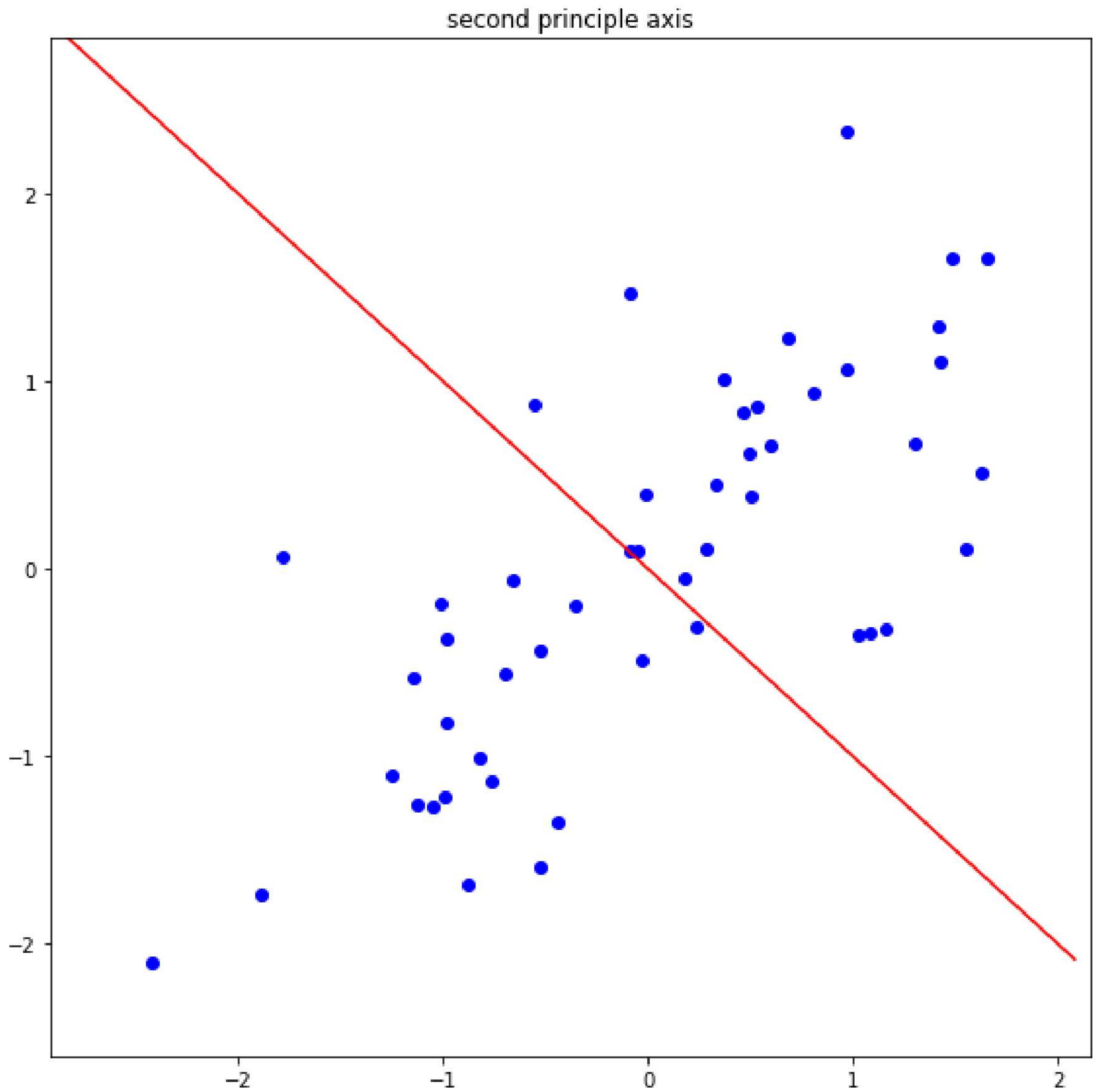

```
**************************************************
## [RESULT 02]
**************************************************
```



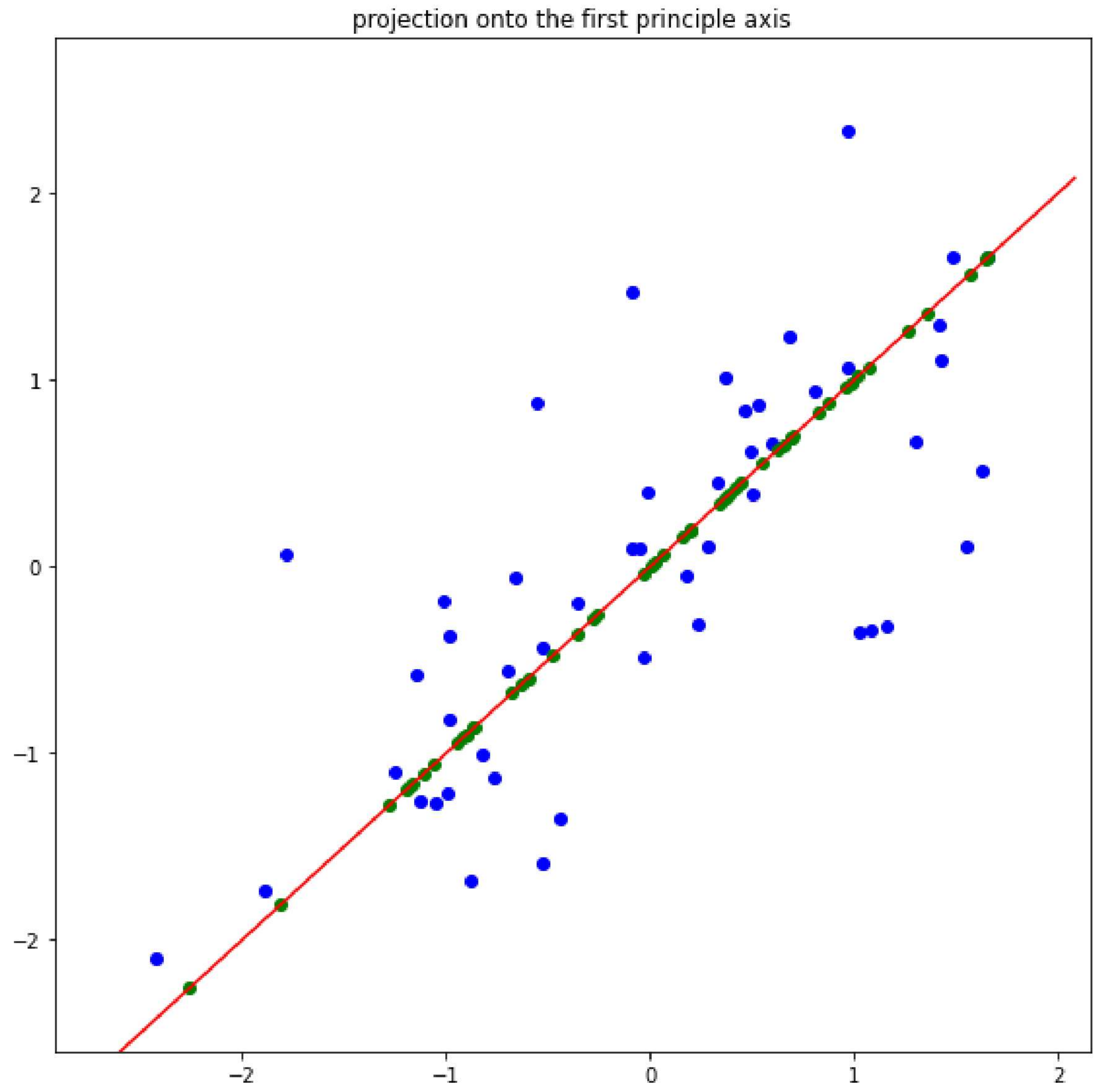

```
**************************************************
## [RESULT 03]
**************************************************
```



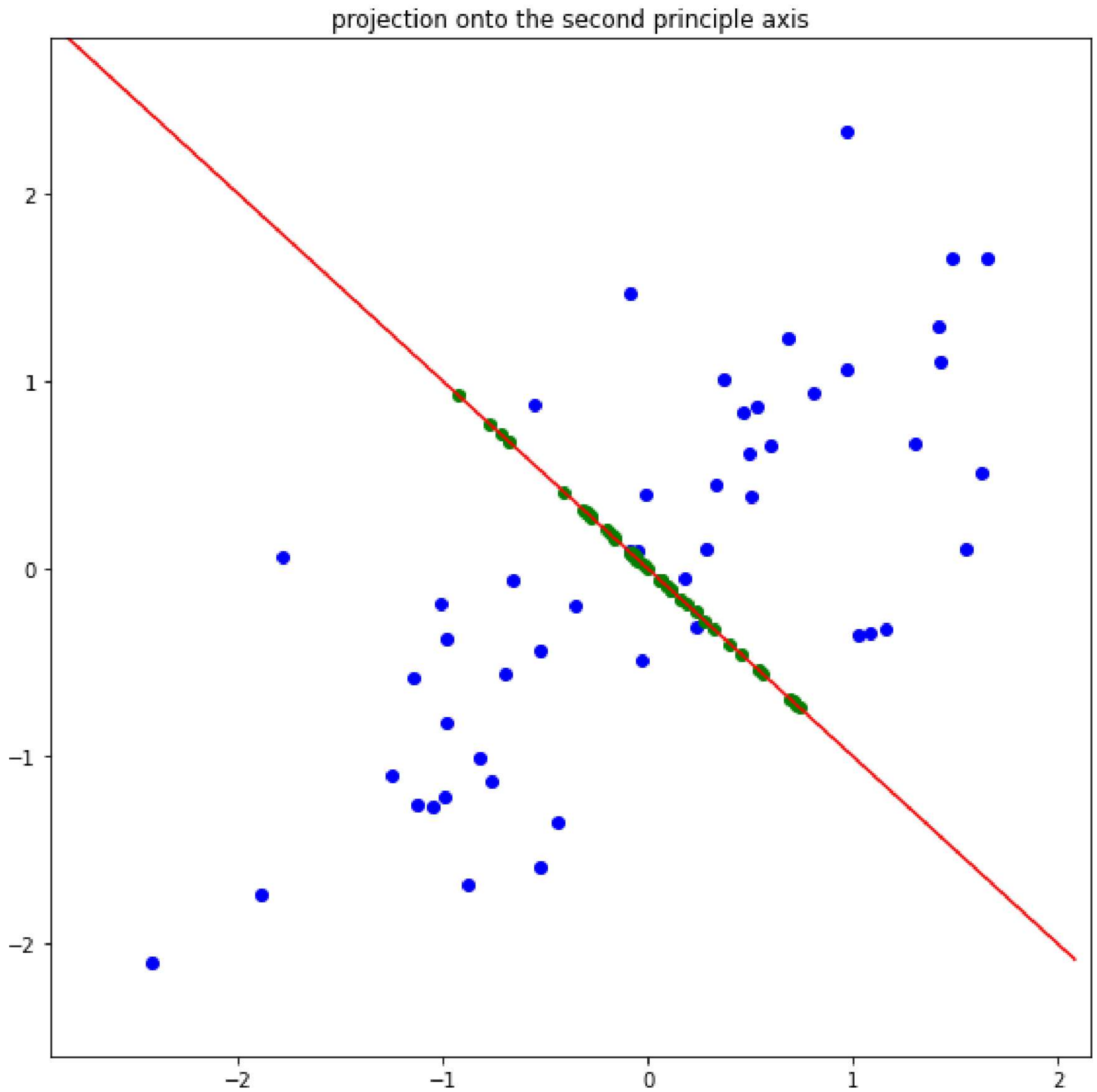

```
************************************************
## [RESULT 04]
************************************************
```



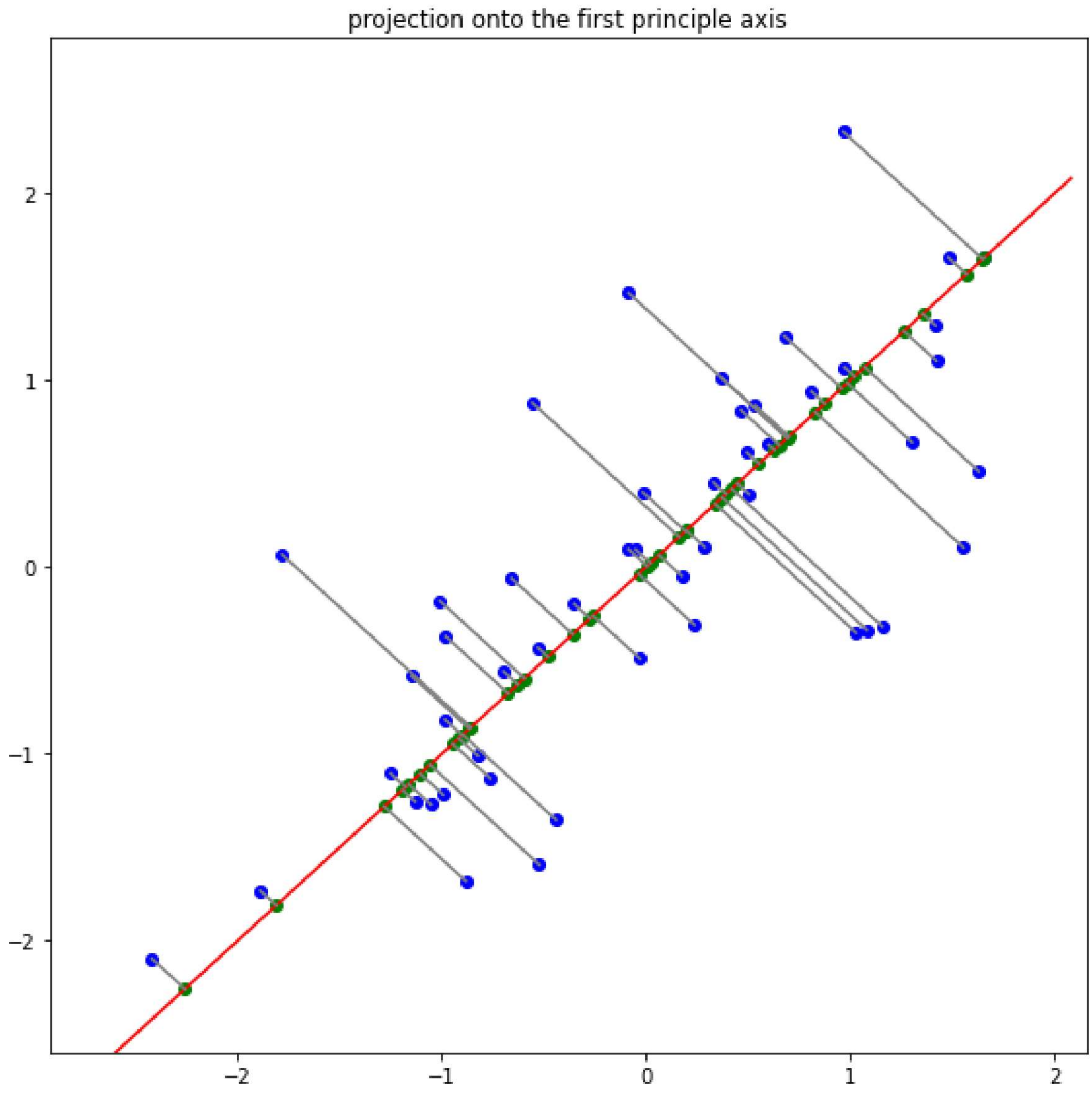
****
[RESULT 05]
****



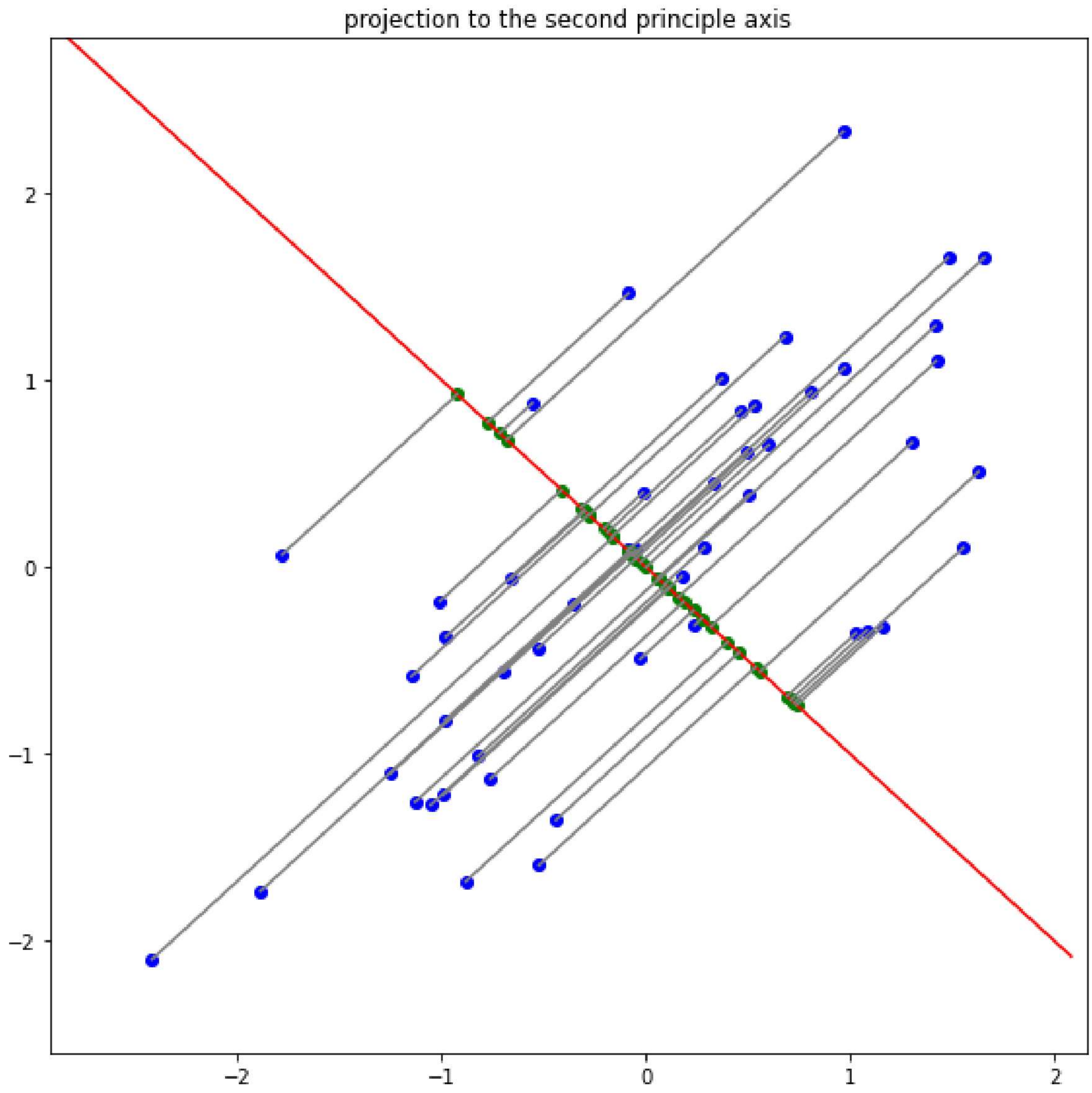

```
**************************************************
## [RESULT 06]
**************************************************
```



****
[RESULT 07]
****




```
************************************************
## [RESULT 08]
************************************************
```


projection to the second principle axis
2
1
0
−1
−2
−2
−1
0
1
2