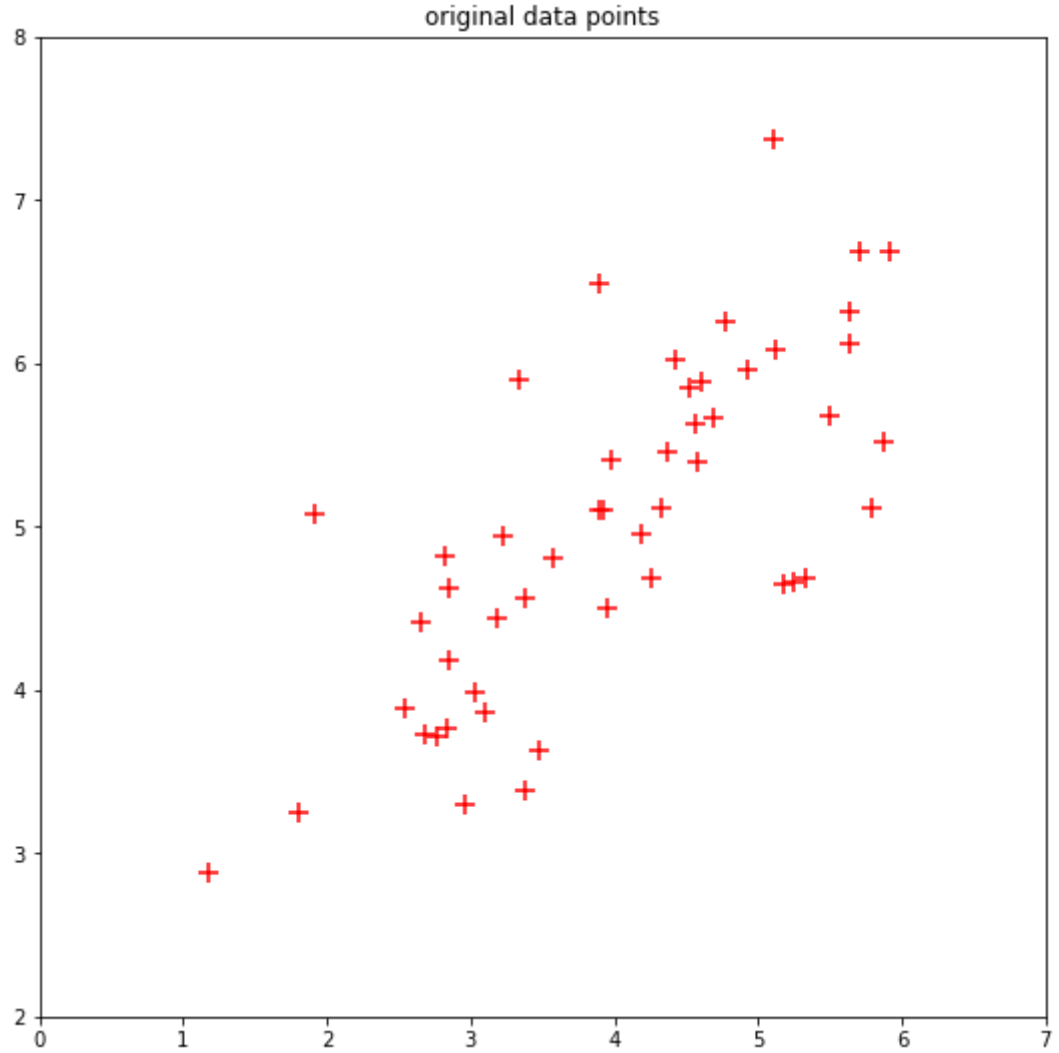
In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import copy
import time
import tensorflow as tf
```

Load the data from the file

In [22]:

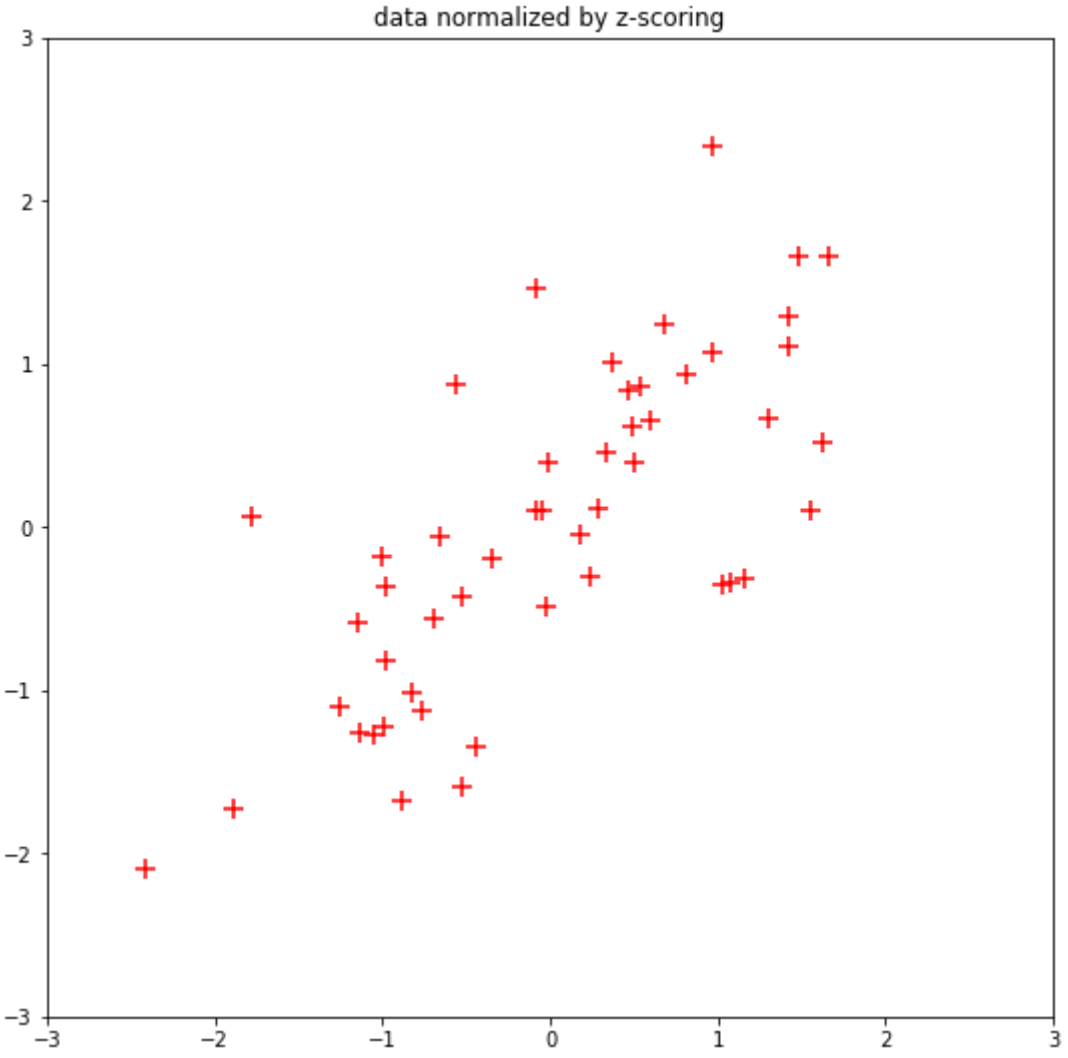
```
data = np.loadtxt('data-pca.txt', delimiter=',')

x = data[:,0]
y = data[:,1]
n = data.shape[0]

plt.figure(1,figsize=(9,9))
plt.scatter(x, y, s=100, c='r', marker='+', label='data')
plt.xlim(0, 7)
plt.ylim(2, 8)
plt.title('original data points')
plt.show()
print(type(x), x)
```




```
<class 'numpy.ndarray'> [3.38156 4.52788 2.65568 2.76523 2.84656 3.89067 3.47581 5.91
13  3.92889
 4.56184 4.57407 4.37173 4.19169 5.24409 2.83584 5.63527 4.68633 2.85051
 5.11016 5.18256 5.70733 3.57968 5.63938 4.26347 2.53652 3.22383 4.92949
 5.79296 2.81685 3.88882 3.34323 5.87973 3.10392 5.33151 3.37543 4.77668
 2.67575 5.50028 1.7971  4.32251 4.421   3.1793  3.03354 4.60935 2.96379
 3.97176 1.18023 1.91895 3.95525 5.11795]
```

In [17]:

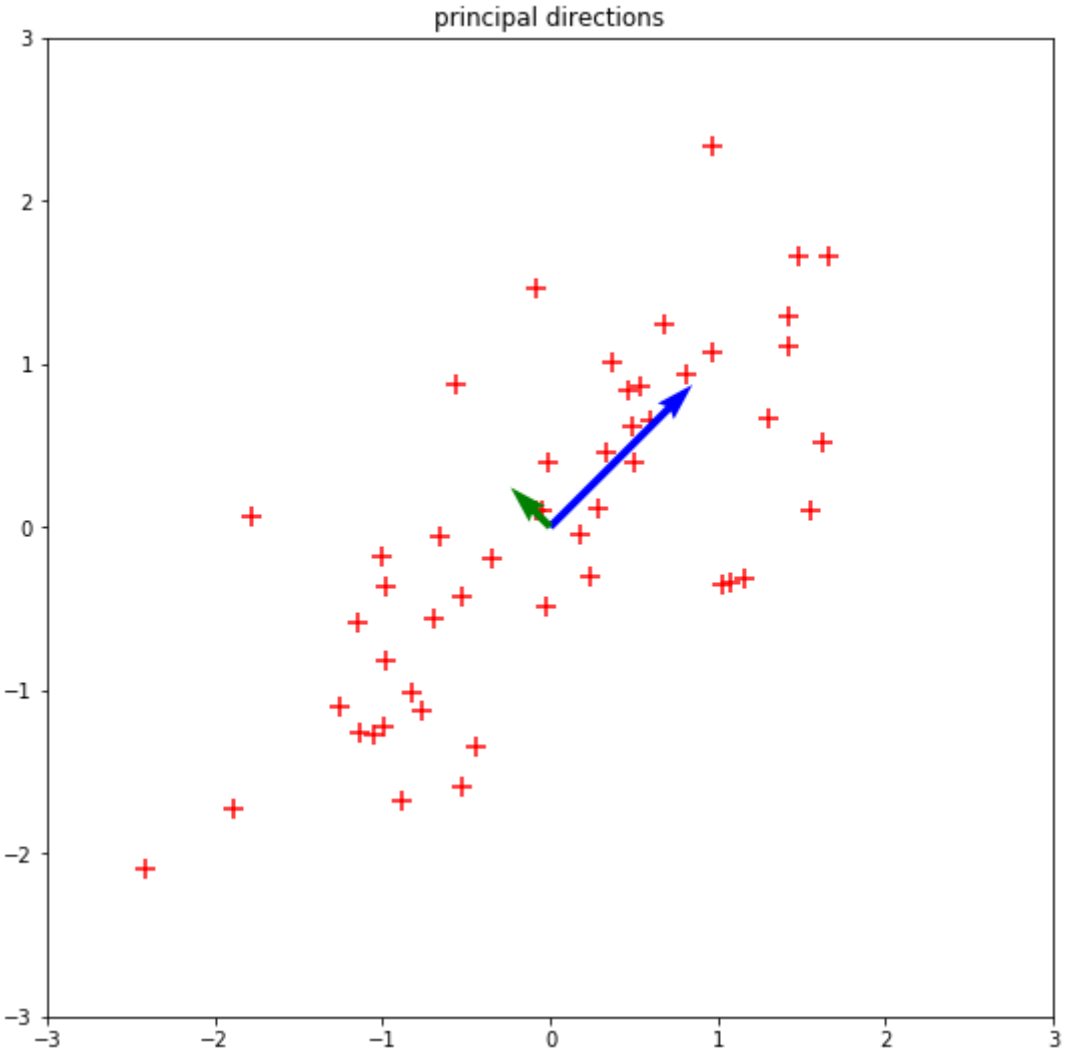
```
xn, yn = normalize_data(x, y)

plt.figure(2,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('data normalized by z-scoring')
plt.show()
```



In [86]:

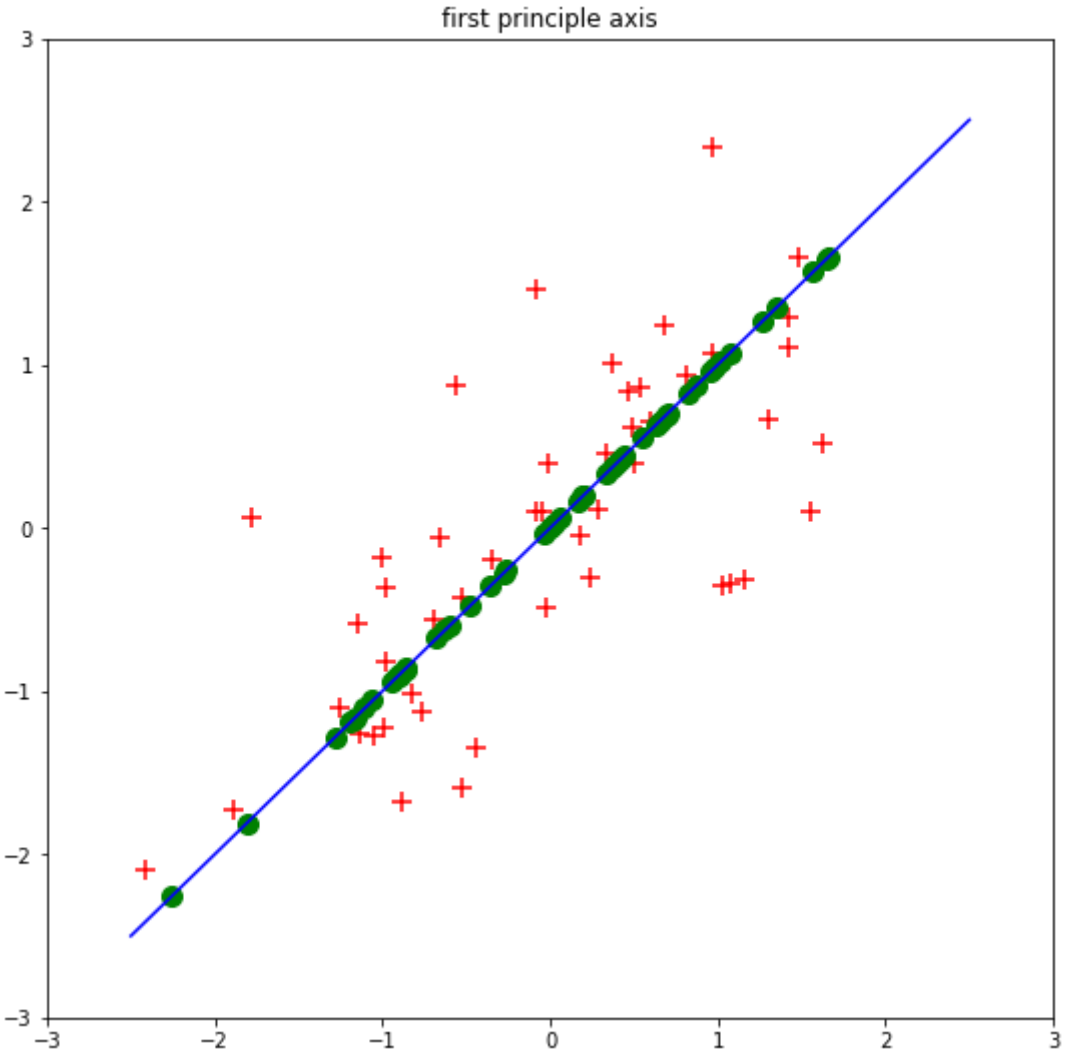
```
data_n = np.zeros((n, 2))

data_n[:,0] = xn
data_n[:,1] = yn
#print(data_n)

covar = compute_covariance(xn, yn)
print(covar)

dir = compute_principal_direction(covar)
print(dir)

vec1 = dir[:,0]
vec2 = dir[:,1]
print(vec1, vec2)
print(vec1.shape)

plt.figure(3,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.quiver(0, 0, vec1[0], vec1[1], color='b', scale=5, pivot='tail')
plt.quiver(0, 0, vec2[0], vec2[1], color='g', pivot='tail')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('principal directions')
plt.show()
```

```
[[1.02040816 0.75054082]
 [0.75054082 1.02040816]]
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
[0.70710678 0.70710678] [-0.70710678  0.70710678]
(2,)
```


principal directions
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

In [205]:

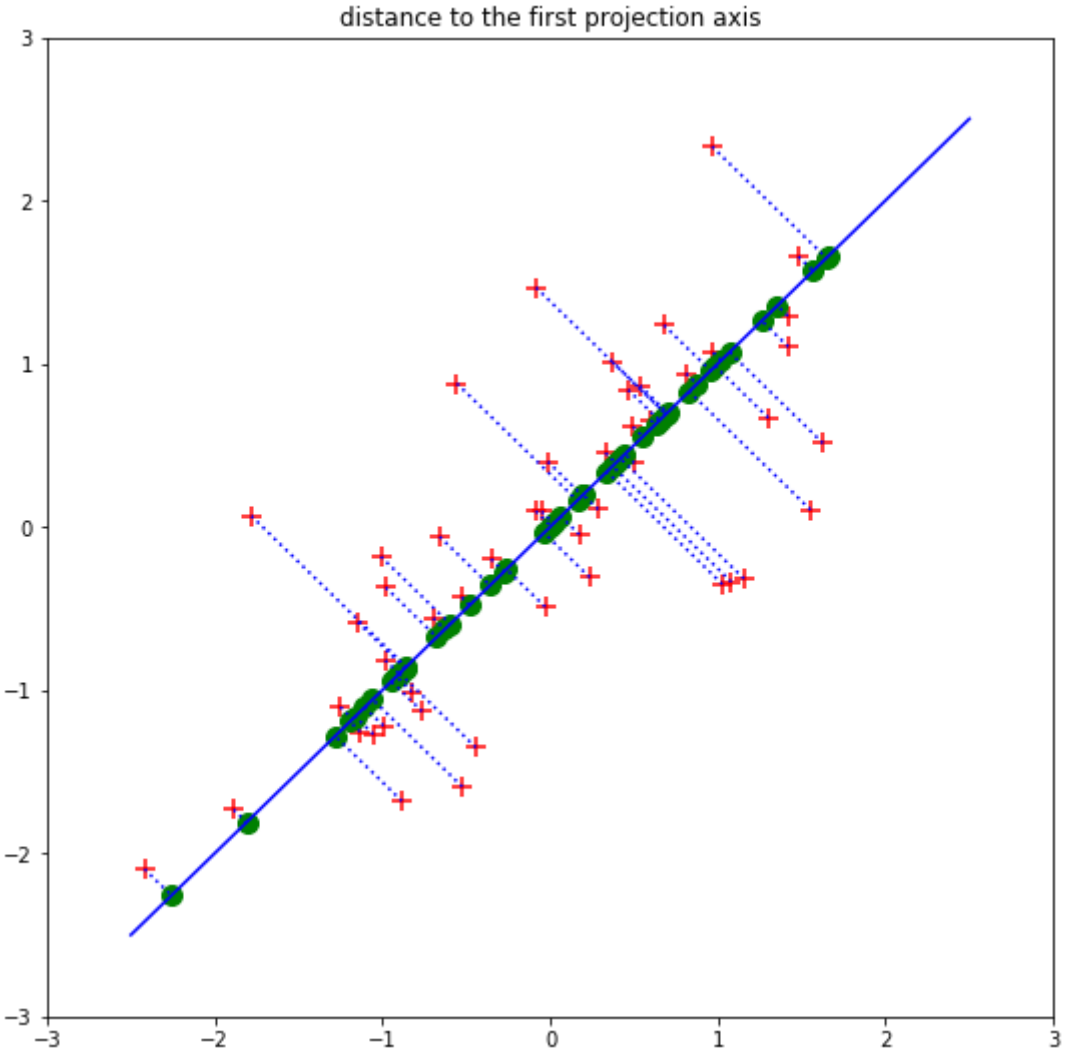
```
proj_1st = compute_projection(data_n, vec1)

plt.figure(7,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_1st, y_1st, c='b')
plt.scatter(proj_1st[0], proj_1st[1], s=100, c='g')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('first principle axis')
plt.show()
```



In [201]:

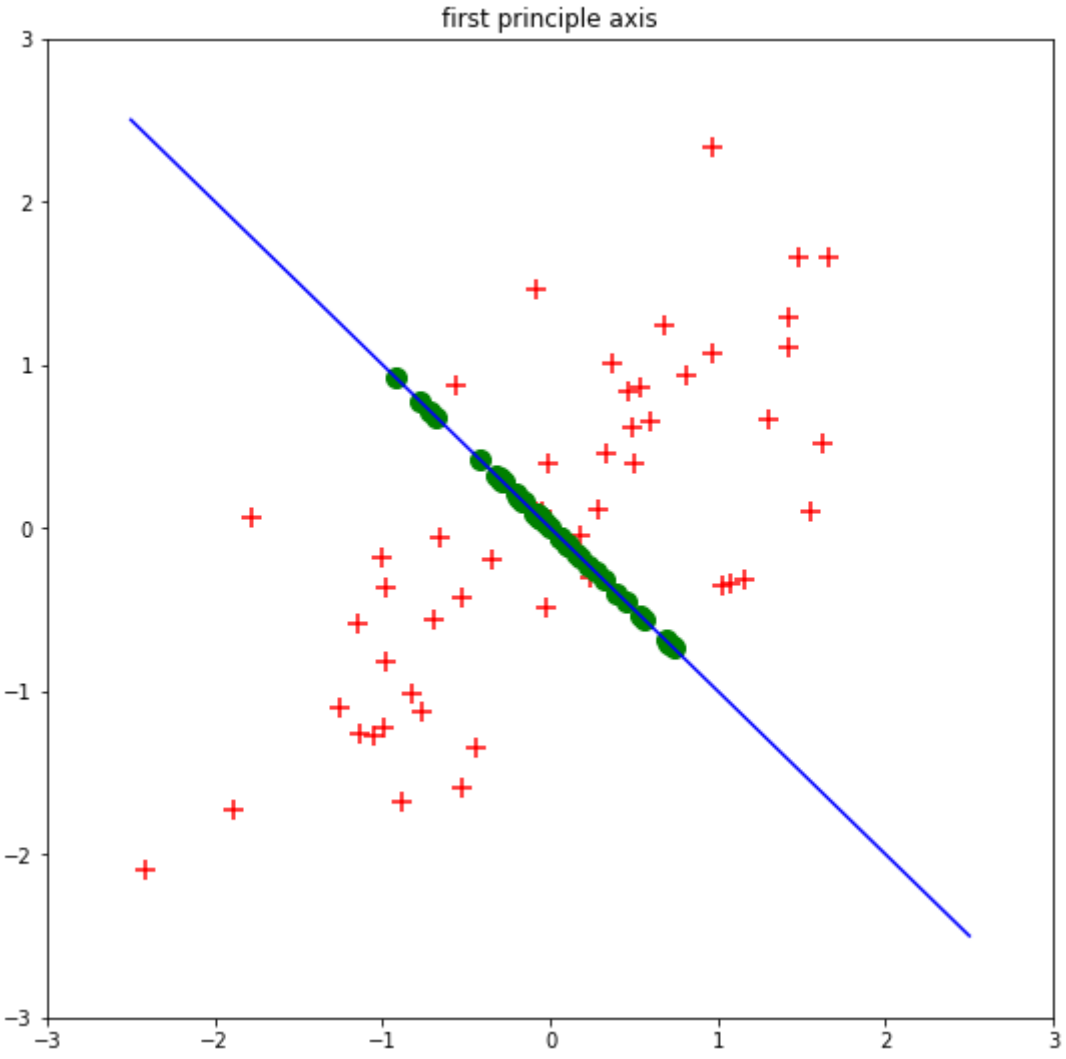
```
proj_1st = compute_projection(data_n, vec1)

# print(proj_1st)
# print(proj_1st[:,0])
# print(data_n)

x_proj_1st = np.zeros((len(xn),2))
y_proj_1st = np.zeros((len(yn),2))

x_proj_1st[:,0] = xn
x_proj_1st[:,1] = proj_1st[0]
y_proj_1st[:,0] = yn
y_proj_1st[:,1] = proj_1st[1]

print(x_proj_1st[0])
print(x_1st)

plt.figure(7,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_1st, y_1st, c='b')
plt.scatter(proj_1st[0], proj_1st[1], s=100, c='g')
for i in range(len(xn)):
    plt.plot(x_proj_1st[i], y_proj_1st[i], linestyle=':', c='b')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('distance to the first projection axis')
plt.show()
```

```
[-0.5233151 -1.058055 ]
[-2.5        -2.23684211 -1.97368421 -1.71052632 -1.44736842 -1.18421053
 -0.92105263 -0.65789474 -0.39473684 -0.13157895  0.13157895  0.39473684
  0.65789474  0.92105263  1.18421053  1.44736842  1.71052632  1.97368421
  2.23684211  2.5        ]
```


distance to the first projection axis
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

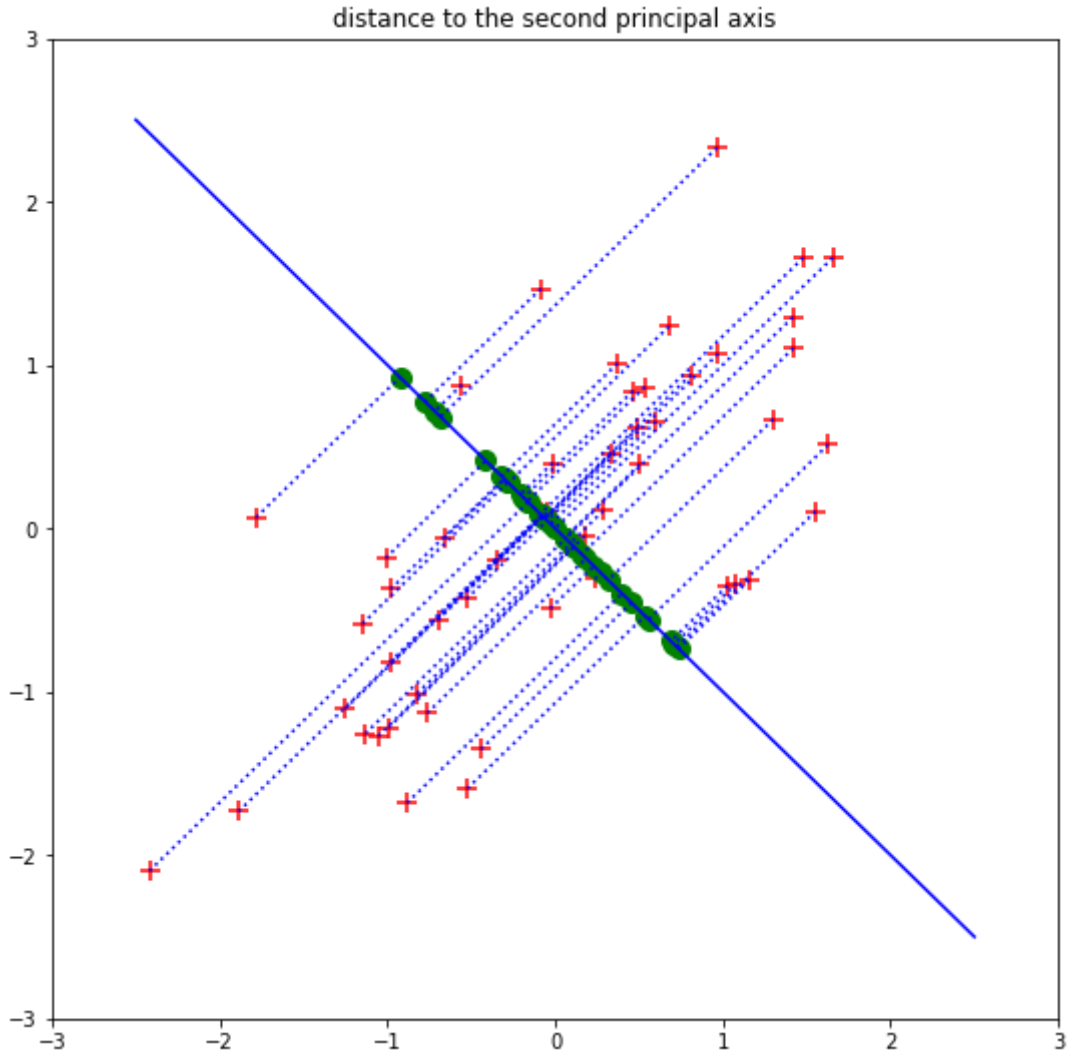
In [206]:

```
proj_2nd = compute_projection(data_n, vec2)

plt.figure(7,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_2nd, y_2nd, c='b')
plt.scatter(proj_2nd[0], proj_2nd[1], s=100, c='g')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('first principle axis')
plt.show()
```



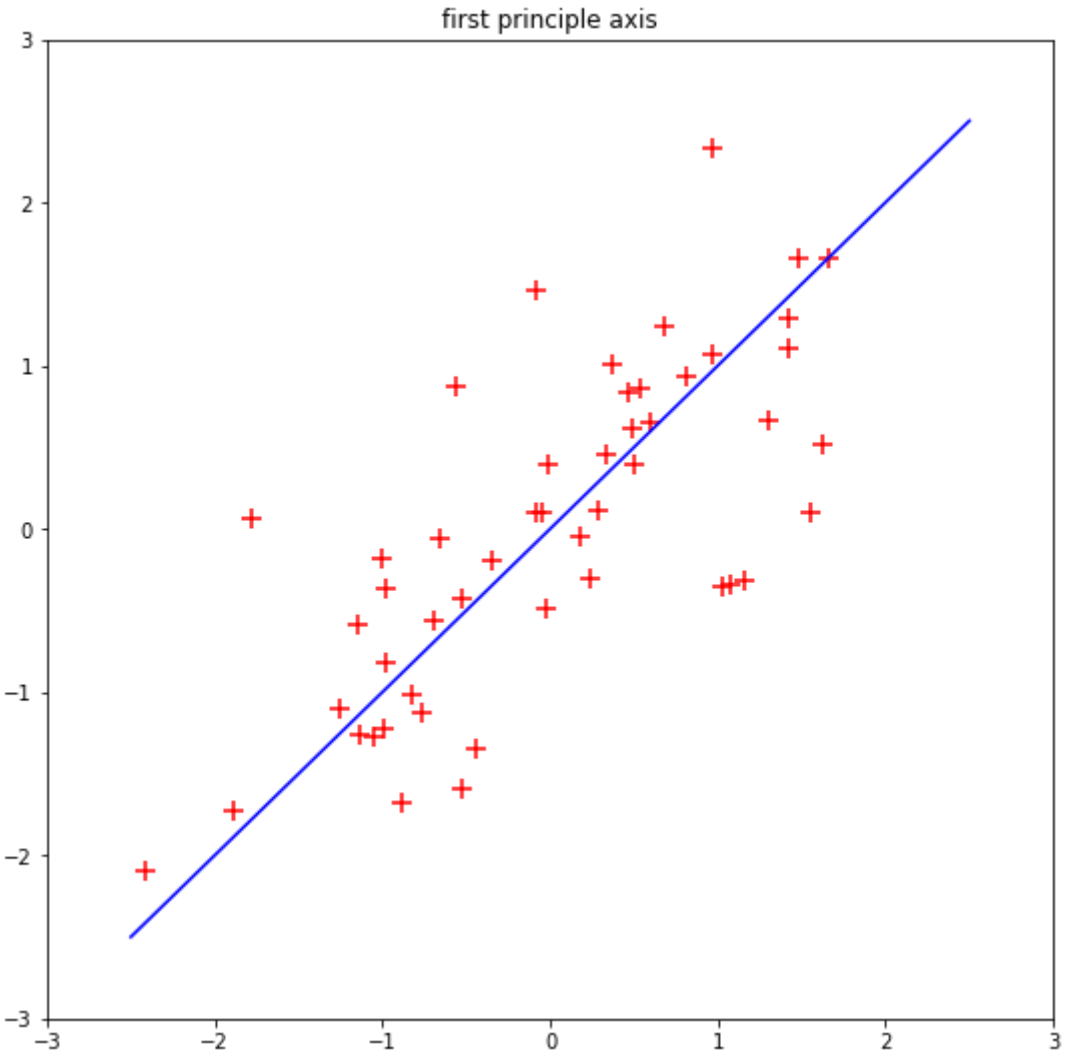
In [203]:

```
x_proj_2nd = np.zeros((len(xn),2))
y_proj_2nd = np.zeros((len(yn),2))

x_proj_2nd[:,0] = xn
x_proj_2nd[:,1] = proj_2nd[0]
y_proj_2nd[:,0] = yn
y_proj_2nd[:,1] = proj_2nd[1]

plt.figure(7,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_2nd, y_2nd, c='b')
plt.scatter(proj_2nd[0], proj_2nd[1], s=100, c='g')
for i in range(len(xn)):
    plt.plot(x_proj_2nd[i], y_proj_2nd[i], linestyle=':', c='b')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('distance to the second principal axis')
plt.show()
```



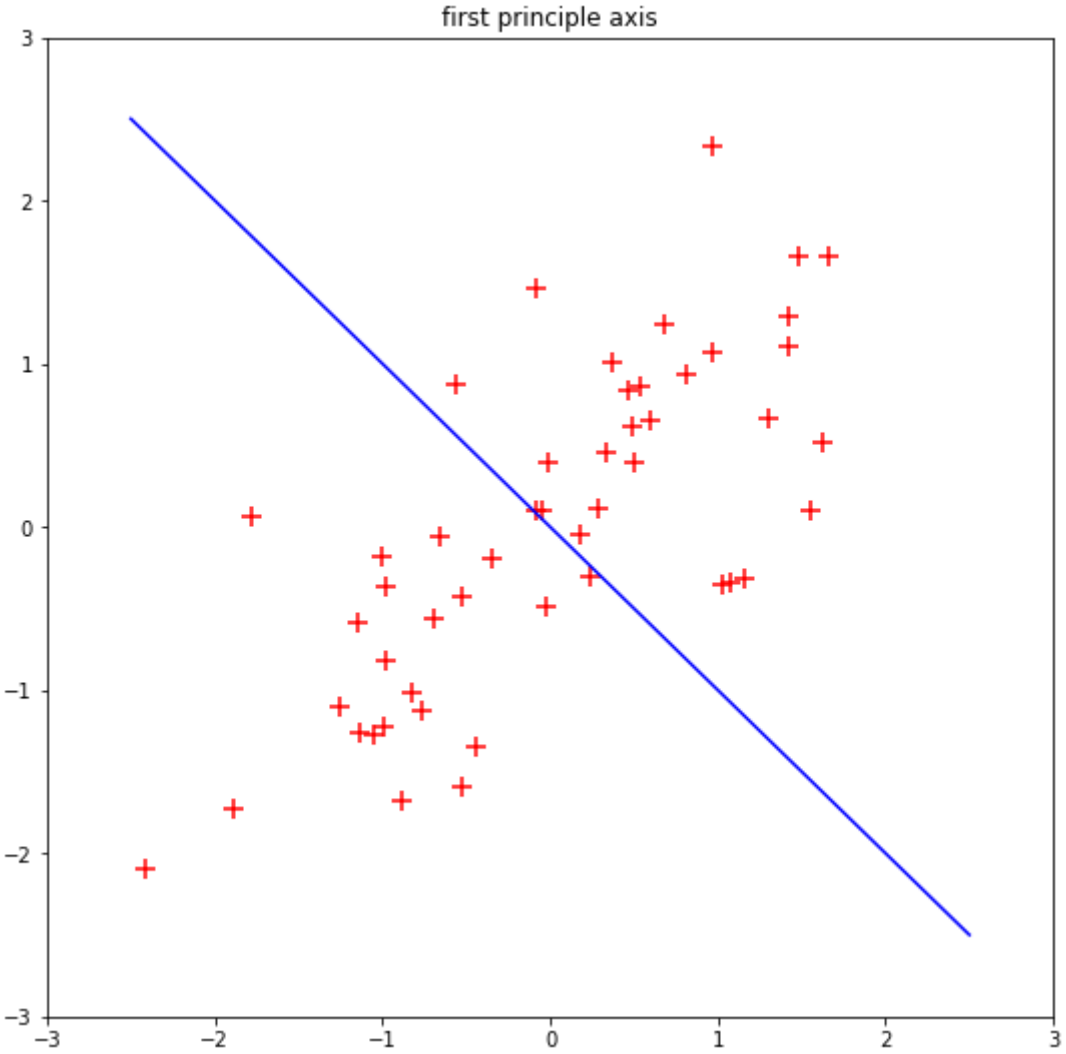
In [78]:

```
axis_1st = vec1[1] / vec1[0]
x_1st = np.linspace(-2.5, 2.5, 20)
y_1st = axis_1st * x_1st

plt.figure(4,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_1st, y_1st, c='b')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('first principle axis')
plt.show()
```



In [80]:

```
axis_2nd = vec2[1] / vec2[0]
x_2nd = np.linspace(-2.5, 2.5, 20)
y_2nd = axis_2nd * x_2nd

plt.figure(5,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_2nd, y_2nd, c='b')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('first principle axis')
plt.show()
```



Define funcitons

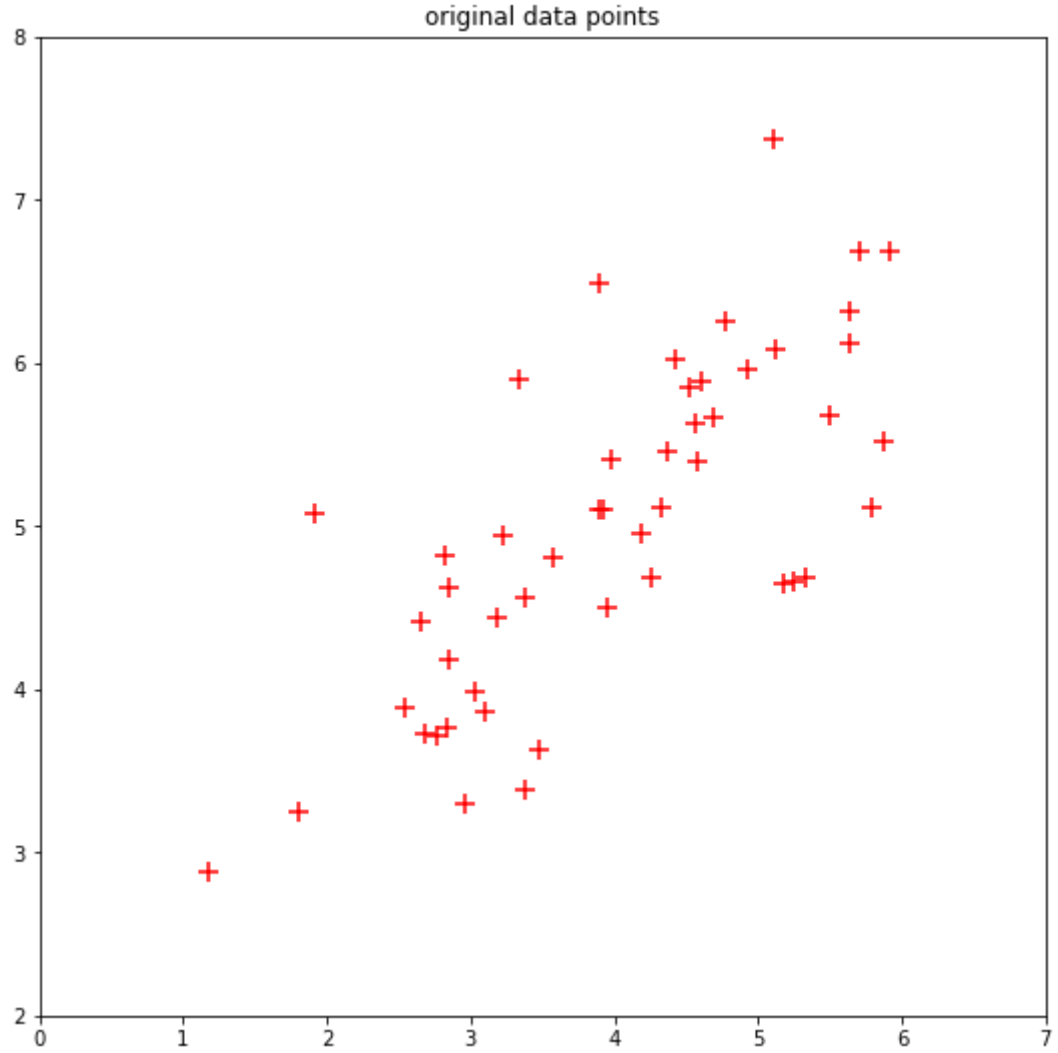
In [154]:

```
def normalize_data(x, y):

    xn = np.zeros(len(x))
    yn = np.zeros(len(y))

    xn = (x - np.mean(x)) / np.std(x) # normalize x. the mean of xn is zero and the standard deviati
    yn = (y - np.mean(y)) / np.std(y)# normalize y. the mean of yn is zero and the standard deviatio

    return xn, yn


def compute_covariance(x, y):

    data_n = np.zeros((len(x), 2))
    data_n[:,0] = x
    data_n[:,1] = y

    covar = np.cov(data_n.T) # compute the covariance matrix #

    return covar

# def compute_covariance2(x, y):

#     data_n = np.zeros((len(x), 2))
#     data_n[:,0] = x
#     data_n[:,1] = y

#     covar = 1 / len(x) * data_n.T @ data_n # compute the covariance matrix #

#     return covar


def compute_principal_direction(covariance):

    eig = np.linalg.eig
    direction = eig(covariance)[1] # compute the principal directions from the co-variance matrix #

    return direction


def compute_projection(point, axis):

#     a = (point@axis).T
#     b = a[:,np.newaxis]
#     projection = b * axis / (axis@axis) # compute the projection of point on the axis #
    axis_new = axis[:,np.newaxis]
    projection = axis_new @ axis_new.T @ point.T

    return projection


def compute_distance(point1, point2):

    distance = sum([(el_a - el_b)**2 for el_a, el_b in list(zip(point1, point2))]) # compute the Eu

    return distance
```

1. Plot the original data points [1pt]

In [207]:

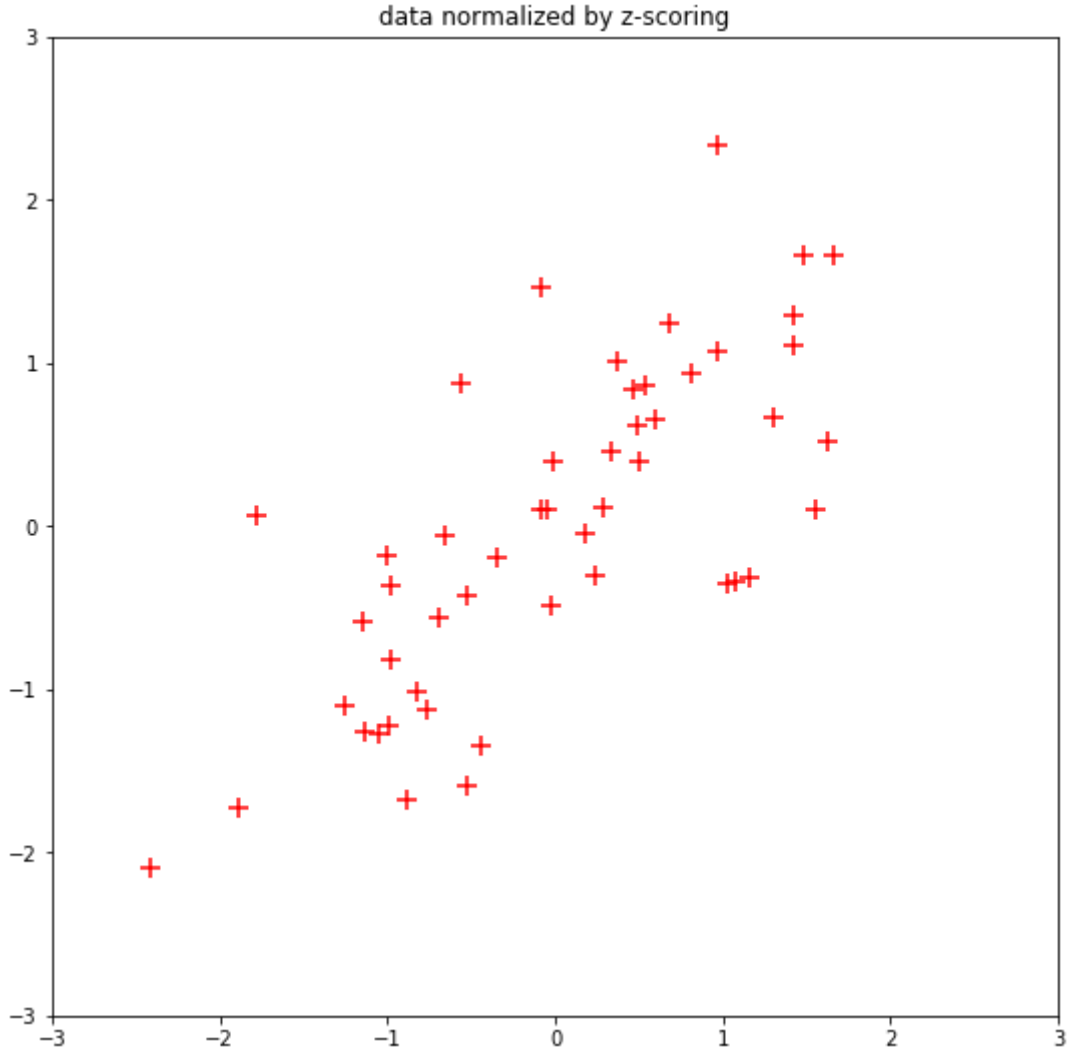
```
plt.figure(1,figsize=(9,9))
plt.scatter(x, y, s=100, c='r', marker='+', label='data')
plt.xlim(0, 7)
plt.ylim(2, 8)
plt.title('original data points')
plt.show()
```



2. Plot the normalized data points [1pt]

In [20]:

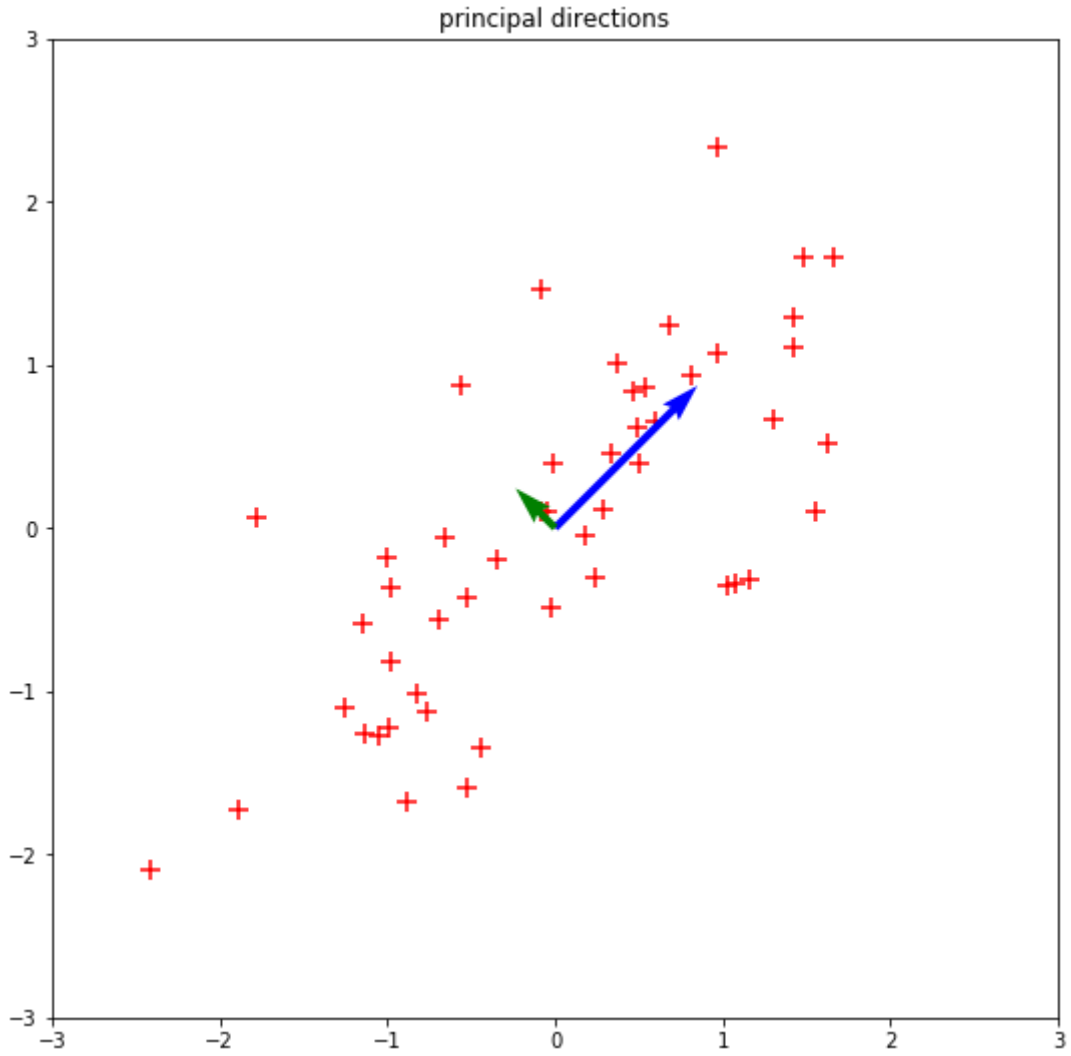
```
plt.figure(2,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('data normalized by z-scoring')
plt.show()
```



3. Plot the principal axes [2pt]

In [72]:

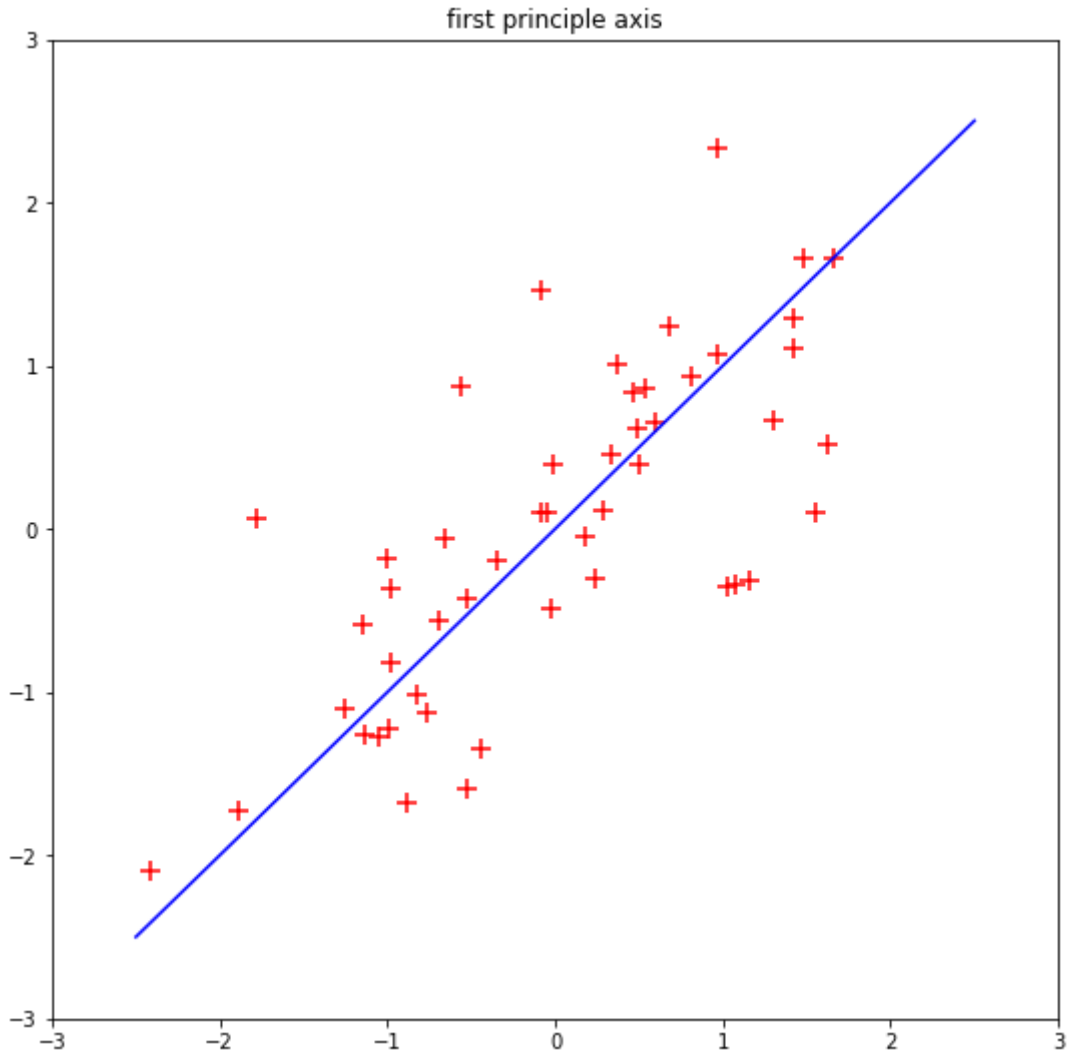
```
plt.figure(3,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.quiver(0, 0, vec1[0], vec1[1], color='b', scale=5, pivot='tail')
plt.quiver(0, 0, vec2[0], vec2[1], color='g', pivot='tail')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('principal directions')
plt.show()
```



4. Plot the first principal axis [3pt]

In [79]:

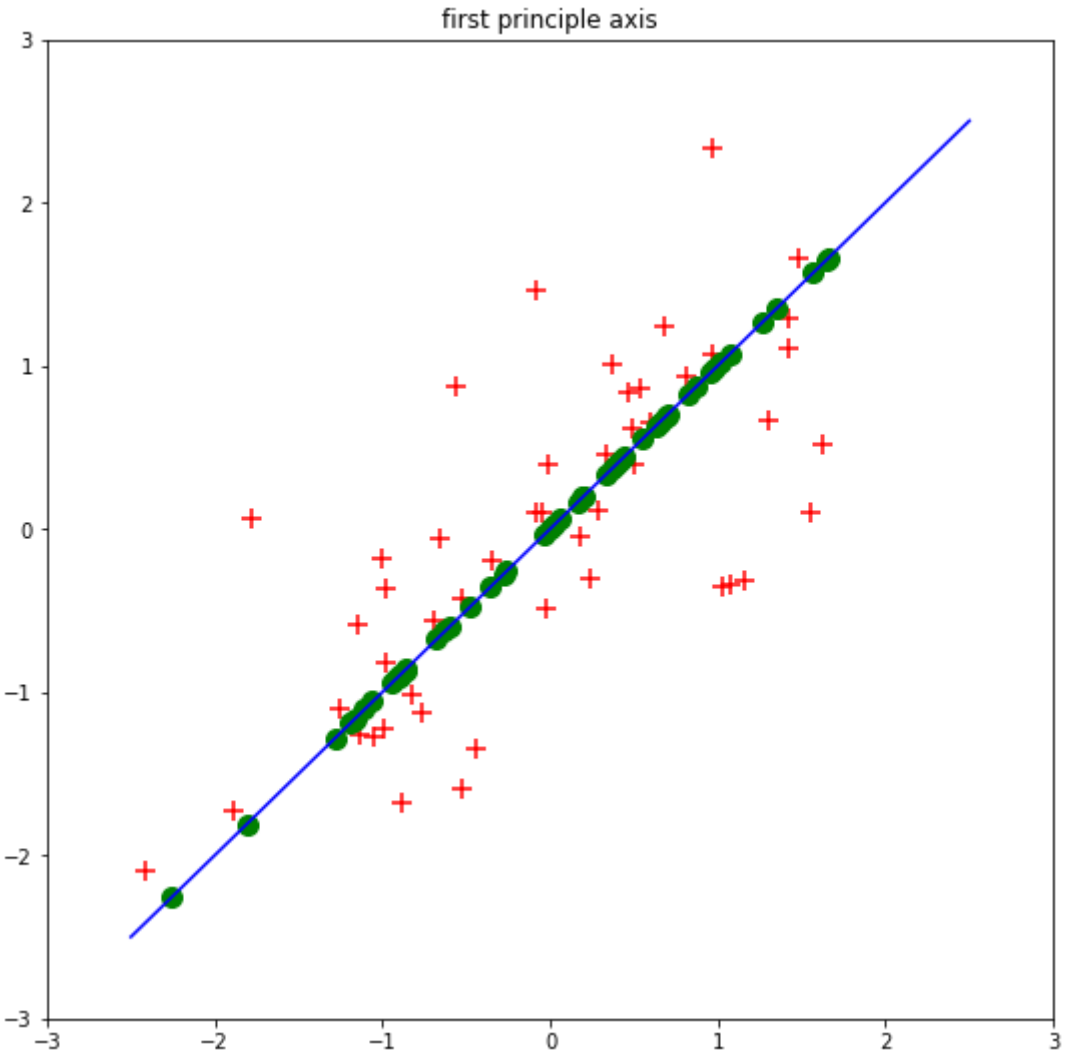
```
plt.figure(4,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_1st, y_1st, c='b')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('first principle axis')
plt.show()
```



5. Plot the project of the normalized data points onto the first principal axis [4pt]

In [151]:

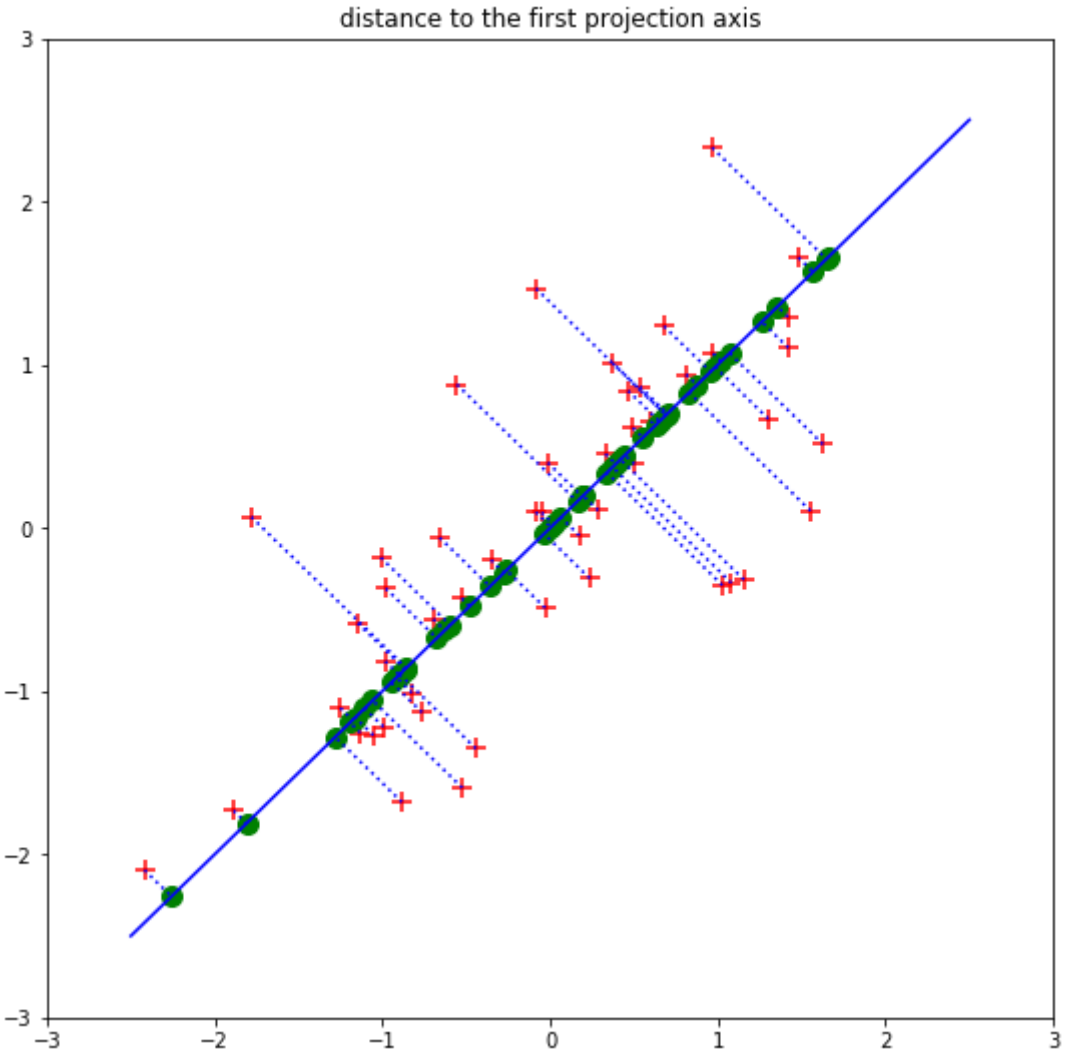
```
plt.figure(6,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_1st, y_1st, c='b')
plt.scatter(proj_1st[0], proj_1st[1], s=100, c='g')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('first principle axis')
plt.show()
```



6. Plot the lines between the normalized data points and their projection points on the first principal axis [3pt]

In [202]:

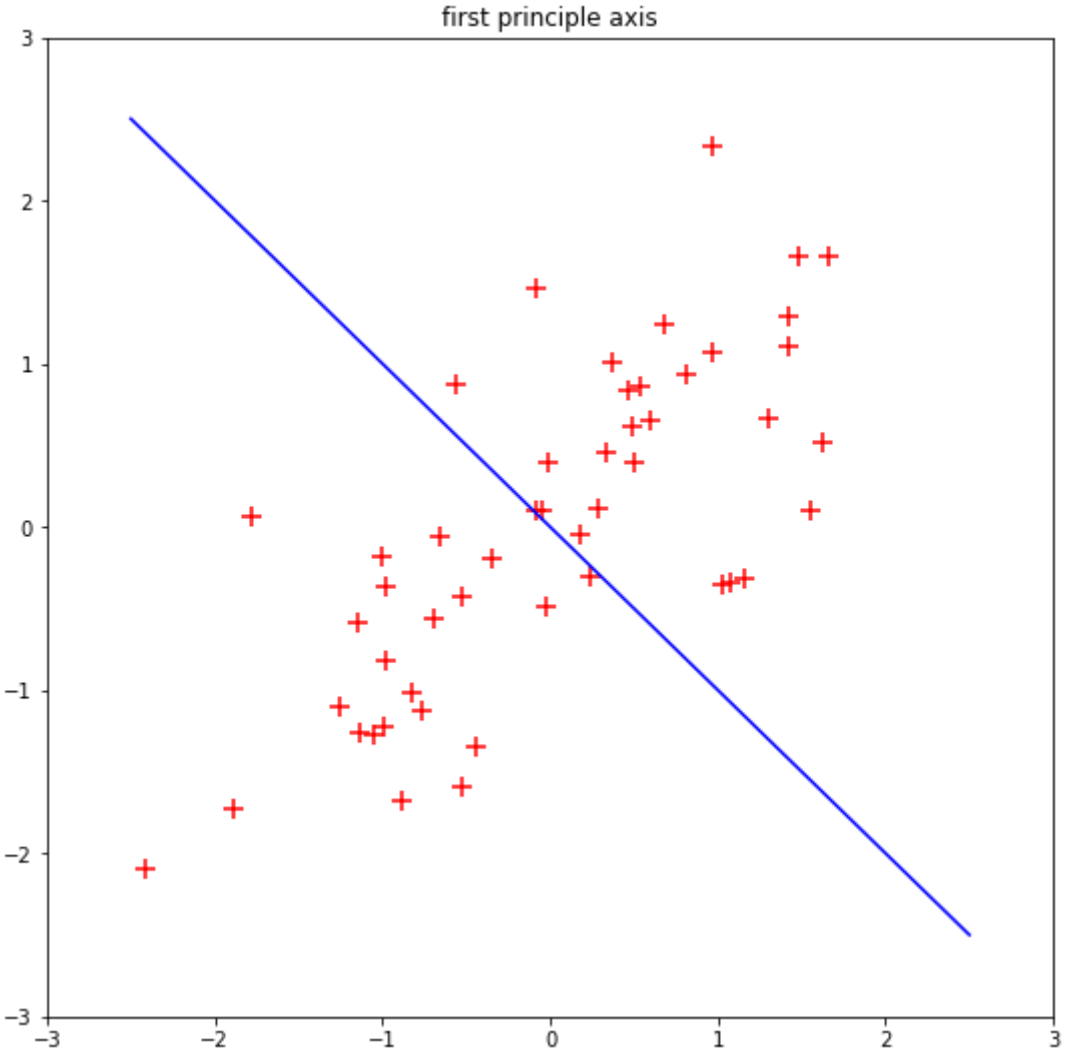
```
plt.figure(7,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_1st, y_1st, c='b')
plt.scatter(proj_1st[0], proj_1st[1], s=100, c='g')
for i in range(len(xn)):
    plt.plot(x_proj_1st[i], y_proj_1st[i], linestyle=':', c='b')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('distance to the first projection axis')
plt.show()
```



7. Plot the second principal axis [3pt]

In [81]:

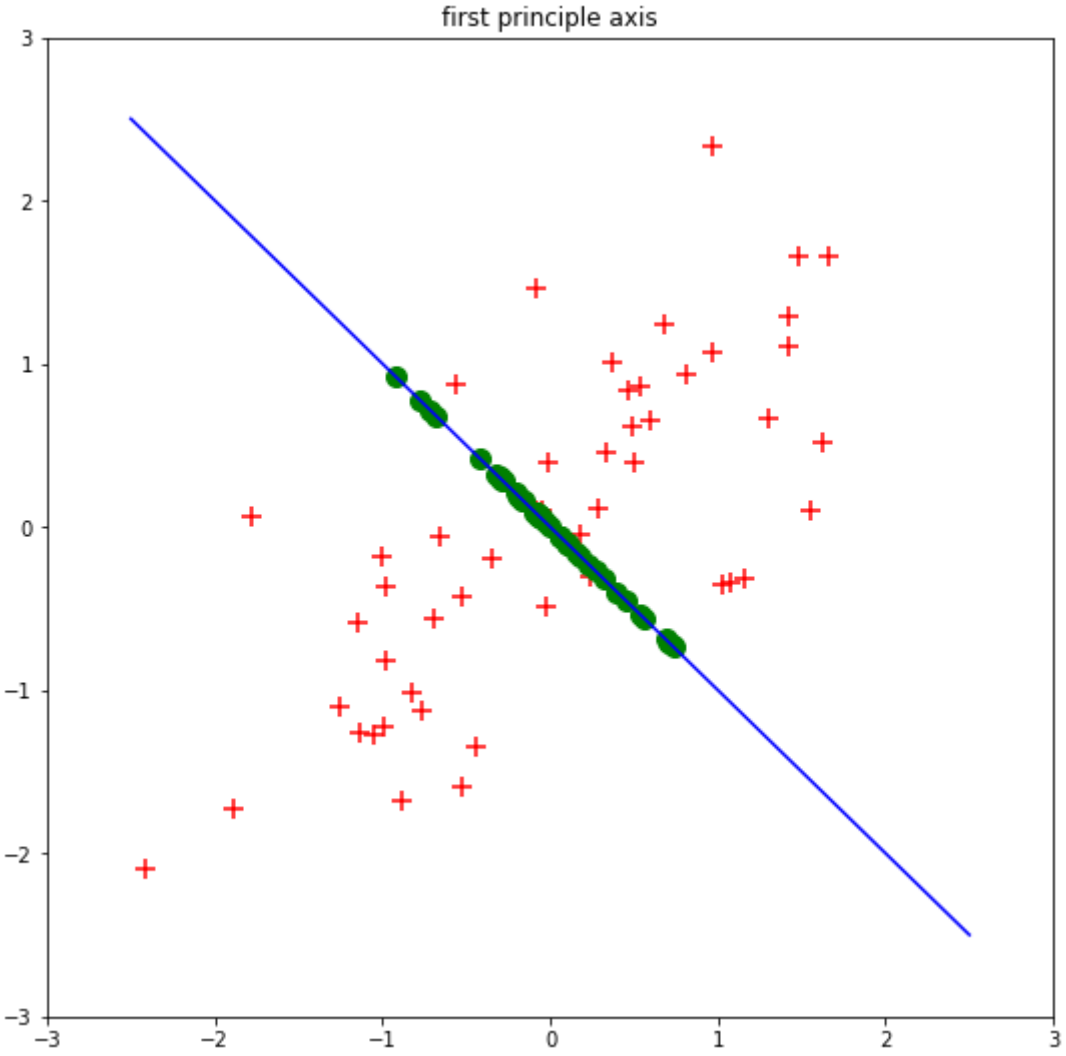
```
plt.figure(5,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_2nd, y_2nd, c='b')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('first principle axis')
plt.show()
```



8. Plot the project of the normalized data points onto the second principal axis [4pt]

In [153]:

```
plt.figure(7,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_2nd, y_2nd, c='b')
plt.scatter(proj_2nd[0], proj_2nd[1], s=100, c='g')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('first principle axis')
plt.show()
```



9. Plot the lines between the normalized data points and their projection points on the second principal axis [3pt]

In [204]:

```
plt.figure(7,figsize=(9,9))
plt.scatter(xn, yn, s=100, c='r', marker='+', label='data')
plt.plot(x_2nd, y_2nd, c='b')
plt.scatter(proj_2nd[0], proj_2nd[1], s=100, c='g')
for i in range(len(xn)):
    plt.plot(x_proj_2nd[i], y_proj_2nd[i], linestyle=':', c='b')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('distance to the second principal axis')
plt.show()
```