Exercise 1.1

```
np.cov(X_std.T)
```

```
array([[ 1.00671141, -0.11010327,  0.87760486,  0.82344326],
       [-0.11010327,  1.00671141, -0.42333835, -0.358937  ],
       [ 0.87760486, -0.42333835,  1.00671141,  0.96921855],
       [ 0.82344326, -0.358937  ,  0.96921855,  1.00671141]])
```

Exercise 1.2

```
sklearn_pca.explained_variance_ratio_
```

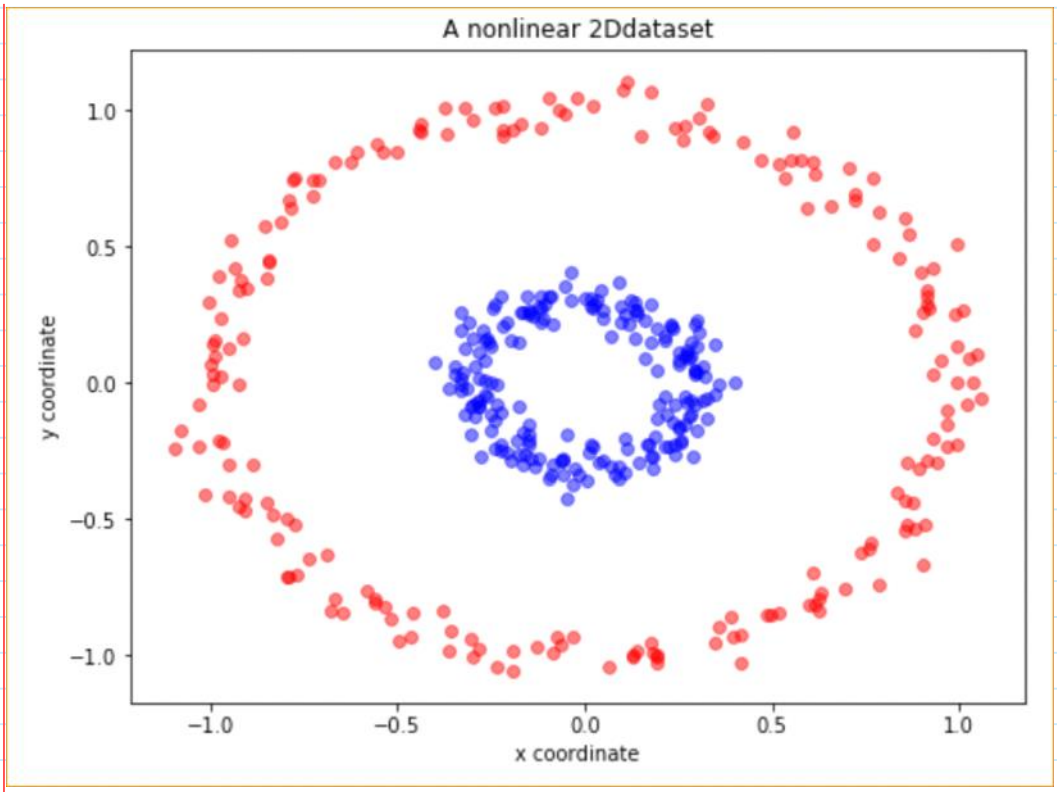
```
array([0.72770452, 0.23030523])
```

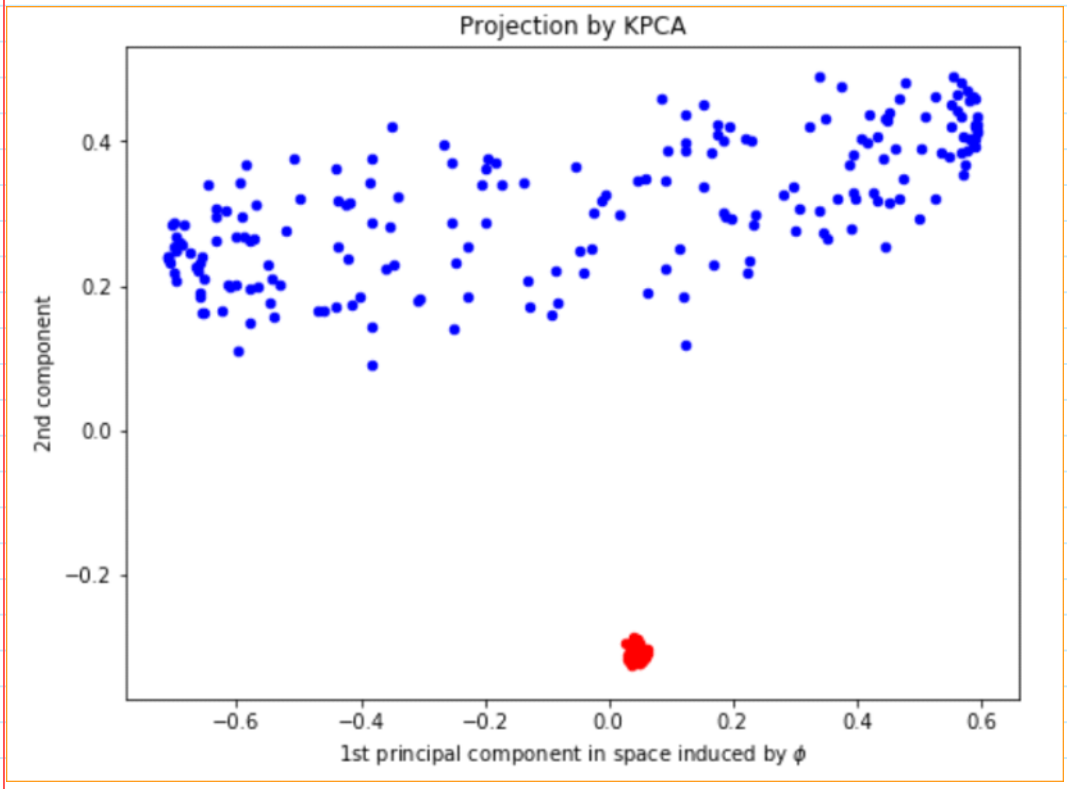
Exercise 1.3

The reason for why the version of principal component 2 obtained with sklearn is flipped is because of the signs.
If the manual PCA gives a positive number, the sklearn will give a negative number, hence a flipped version.

Exercise 1.4

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA, KernelPCA
from sklearn.datasets import make_circles

np.random.seed(0)

X, y = make_circles(n_samples=400, factor=.3, noise=.05)
# TODO
#1.
plt.figure(figsize=(8,6))

plt.scatter(X[y==0, 0], X[y==0, 1], color='red', alpha=0.5)
plt.scatter(X[y==1, 0], X[y==1, 1], color='blue', alpha=0.5)

plt.title('A nonlinear 2Ddataset')
plt.ylabel('y coordinate')
plt.xlabel('x coordinate')

plt.show()

#2.
scikit_kpca = KernelPCA(n_components=2, kernel='rbf', gamma=10)
X_skernpca = scikit_kpca.fit_transform(X)

X_kpca = scikit_kpca.fit_transform(X)

#3.
plt.figure(figsize=(8,6))
plt.scatter(X_kpca[y==0, 0], X_kpca[y==0, 1], color="red", s=20)
plt.scatter(X_kpca[y==1, 0], X_kpca[y==1, 1], color="blue", s=20)
plt.title("Projection by KPCA")
plt.xlabel("1st principal component in space induced by $\phi$")
plt.ylabel("2nd component")
```





Exercise 1.5

```
mf.get_rating(1, 3)
```

```
1.0105660878287315
```

Exercise 1.6

```
import pandas as pd

#1.
ratings_url="https://raw.githubusercontent.com/alexvlis/movie-recommendation-system/master/data/ratings.csv"
ratings=pd.read_csv(ratings_url)

moives_url="https://raw.githubusercontent.com/alexvlis/movie-recommendation-system/master/data/movies.csv"
movies=pd.read_csv(moives_url)
```

```
#2. Dropping column 'timestamp'
ratings = ratings.drop(['timestamp'], axis=1)
#print(ratings)

#3. Converting into a rating matrix
ratings = pd.pivot_table(ratings, values='rating', index='userId', columns='movieId', fill_value=0)
#print(ratings)
```

```
#4. Converting to numpy array and using matrix factorization
ratings = np.array(ratings)
mf = MF(ratings, K=100, alpha=0.04, beta=0.01, iterations=100)
mf.train()
```

Notice that mf.train() will give different result.
Sometimes, the indexes won't match in user_671 and rating_671
=> no results

```
#5. Extracting row 671th

R_hat = mf.full_matrix()
user_671 = ratings[670]
rating_671 = R_hat[670]

indexes = list()
for i in range(0, len(ratings)):
    if user_671[i] == 0:
        if rating_671[i] >= 4.9:
            indexes.append(i)

print(indexes)
```

```
[427, 527]
```

```
#6. Corresponding moviesId from indexes found in previous step
for i in indexes:
    print(movies.iloc[i])
    print()
```

```
movieId                                480
title                 Jurassic Park (1993)
genres     Action|Adventure|Sci-Fi|Thriller
Name: 427, dtype: object

movieId                                                595
title                          Beauty and the Beast (1991)
genres     Animation|Children|Fantasy|Musical|Romance|IMAX
Name: 527, dtype: object
```