# Exercise02

May 3, 2017

## 1 Exercise Sheet 1: *Programming and Visualisation*

Machine Intelligence 2 SS 2017, Obermayer/Augustin/Guo due: **2017-05-03** Group: Outlaws (Muhammed Cengizhan Özmen, Zhanwang Chen, Sedat Koca, Huajun Li, Khaled Mansour)

*Used Python version 2.7.11 (https://www.continuum.io/downloads, http://ipython.org/install.html)*

```
In [1]: import sys
        print(sys.version)
```

```
2.7.13 |Anaconda 4.3.1 (64-bit)| (default, Dec 20 2016, 23:09:15)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
```

```
In [2]: import IPython
        print(IPython.__version__)
```

```
5.1.0
```

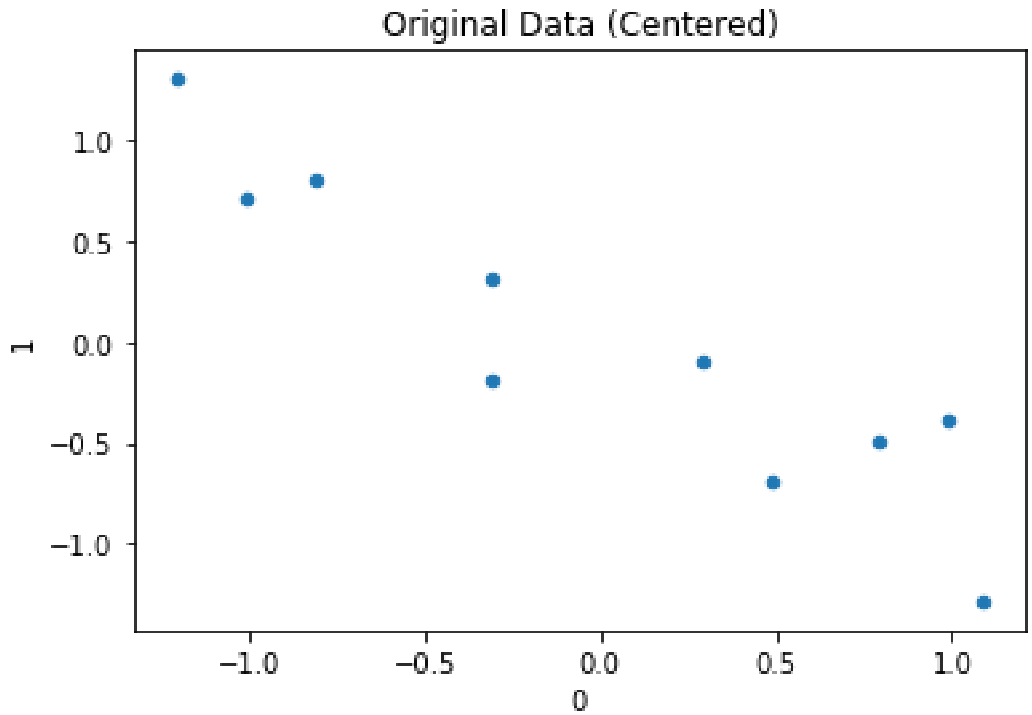
```
In [3]: # load frequently used libraries
        import os
        import numpy as np
        import math
        import matplotlib.pyplot as plt
        import pandas as pd
        import matplotlib.image as mpimg
        from skimage.util import view_as_windows
        from sklearn.decomposition import PCA
        from pandas.tools.plotting import scatter_matrix
        from mpl_toolkits.mplot3d import Axes3D
        from matplotlib.collections import LineCollection
        from matplotlib import cm
        %matplotlib inline
```

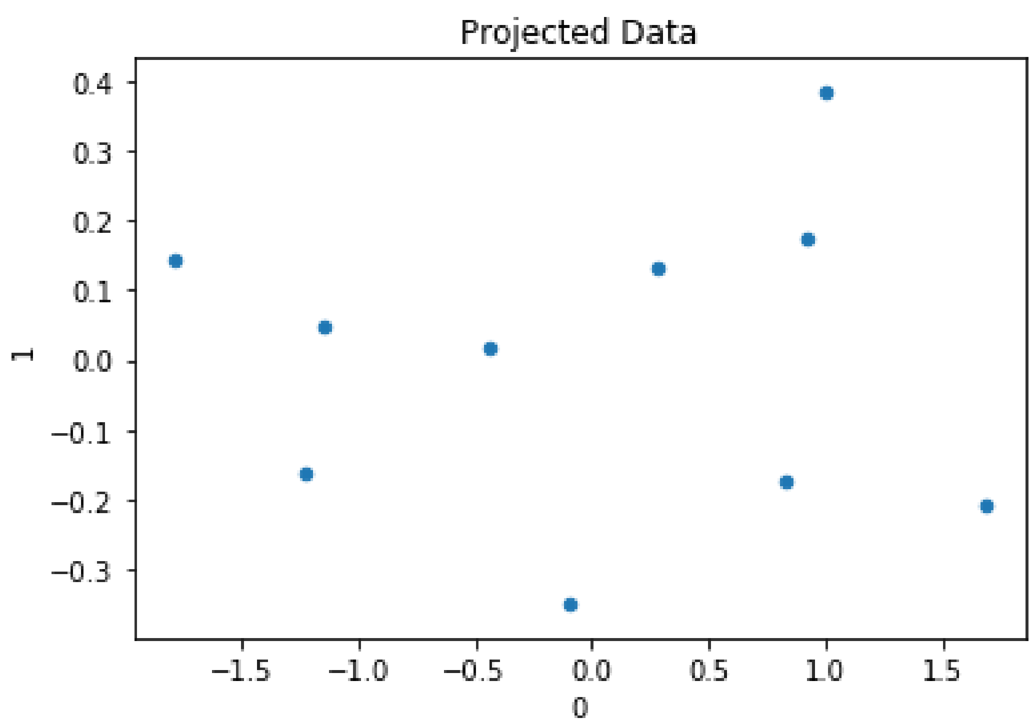
### 1.1 N.2.1 *PCA: 2-dimensional Toy Data*

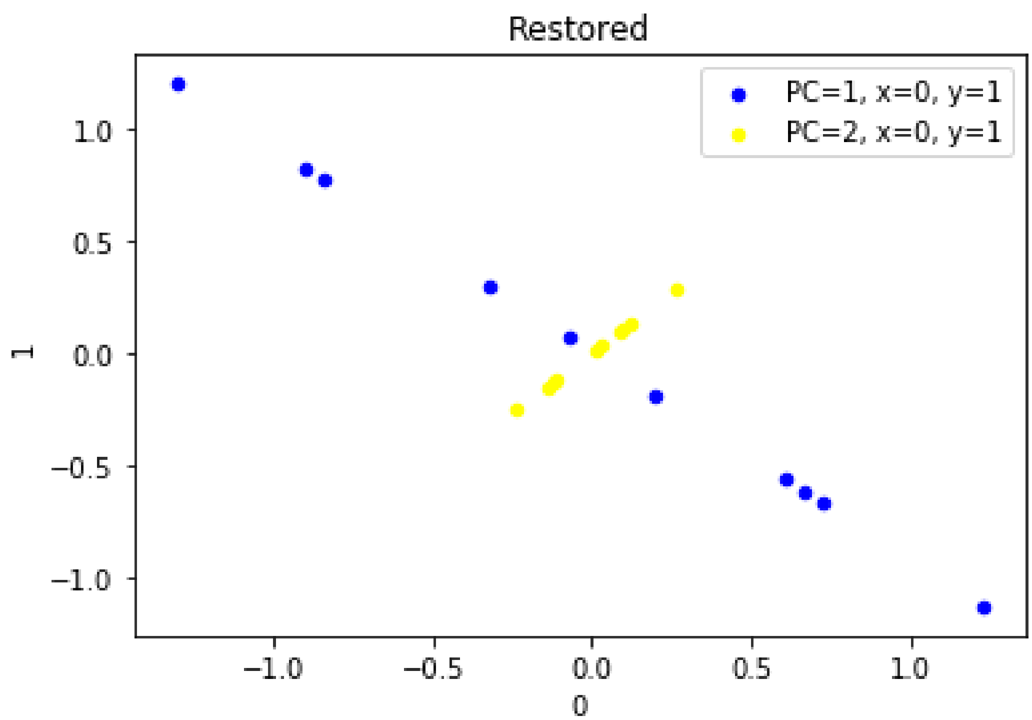
**PCA using covariance and eigen value decomposition**

In [4]:
```
# Read data from file:
pca2d = pd.read_csv('Data/pca-data-2d.dat', header=None, delim_whitespace=T

# Task 1.a:

# Center the data
pca2d_centered = pca2d - np.mean(pca2d)
# Plot as scatter
pca2d_centered.plot(kind='scatter',x=0,y=1, title='Original Data (Centered)

# Task 1.b
# Principal Component Analysis (PCA)

# covariance
pca2d_cov = pca2d_centered.cov()
#Eigen value and eigen vector
eig_val, eig_vec = np.linalg.eig(pca2d_cov)
#projection of data from pca2d_centered.dot with eigen vector
pca2d_proj = pca2d_centered.dot(eig_vec)
# Plot of projected data
pca2d_proj.plot(kind='scatter',x=0,y=1, title='Projected Data')
# Task 1.c
# Projection on principle components
# pca2d_centered.dot(eig_vec.T[0])
pca2d_1PC = np.matrix(pca2d_proj[0]).T
# pca2d_centered.dot(eig_vec.T[1])
pca2d_2PC = np.matrix(pca2d_proj[1]).T

rePCA_1PC = pd.DataFrame(pca2d_1PC.dot(np.matrix(eig_vec.T[0])))
rePCA_2PC = pd.DataFrame(pca2d_2PC.dot(np.matrix(eig_vec.T[1])))

ax = rePCA_1PC.plot(kind='scatter',x=0,y=1, color='blue',label='PC=1, x=0,
rePCA_2PC.plot(kind='scatter',x=0,y=1, color='yellow',label='PC=2, x=0, y=1
```

Out[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc5804ae650>
```


Original Data (Centered)
1.0
0.5
0.0
−0.5
−1.0
1
−1.0
−0.5
0.0
0.5
1.0
0

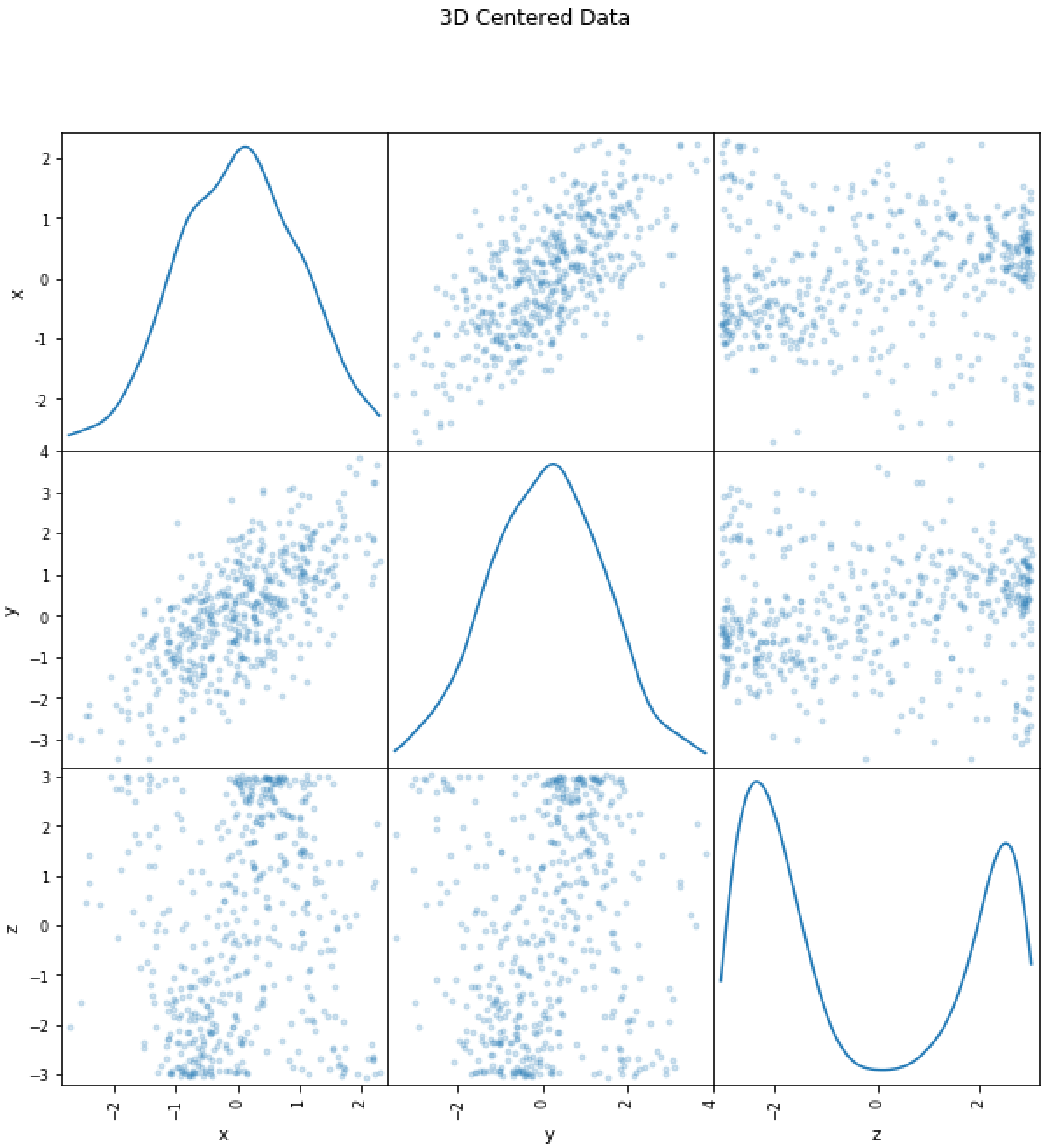

Projected Data
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
1
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
0



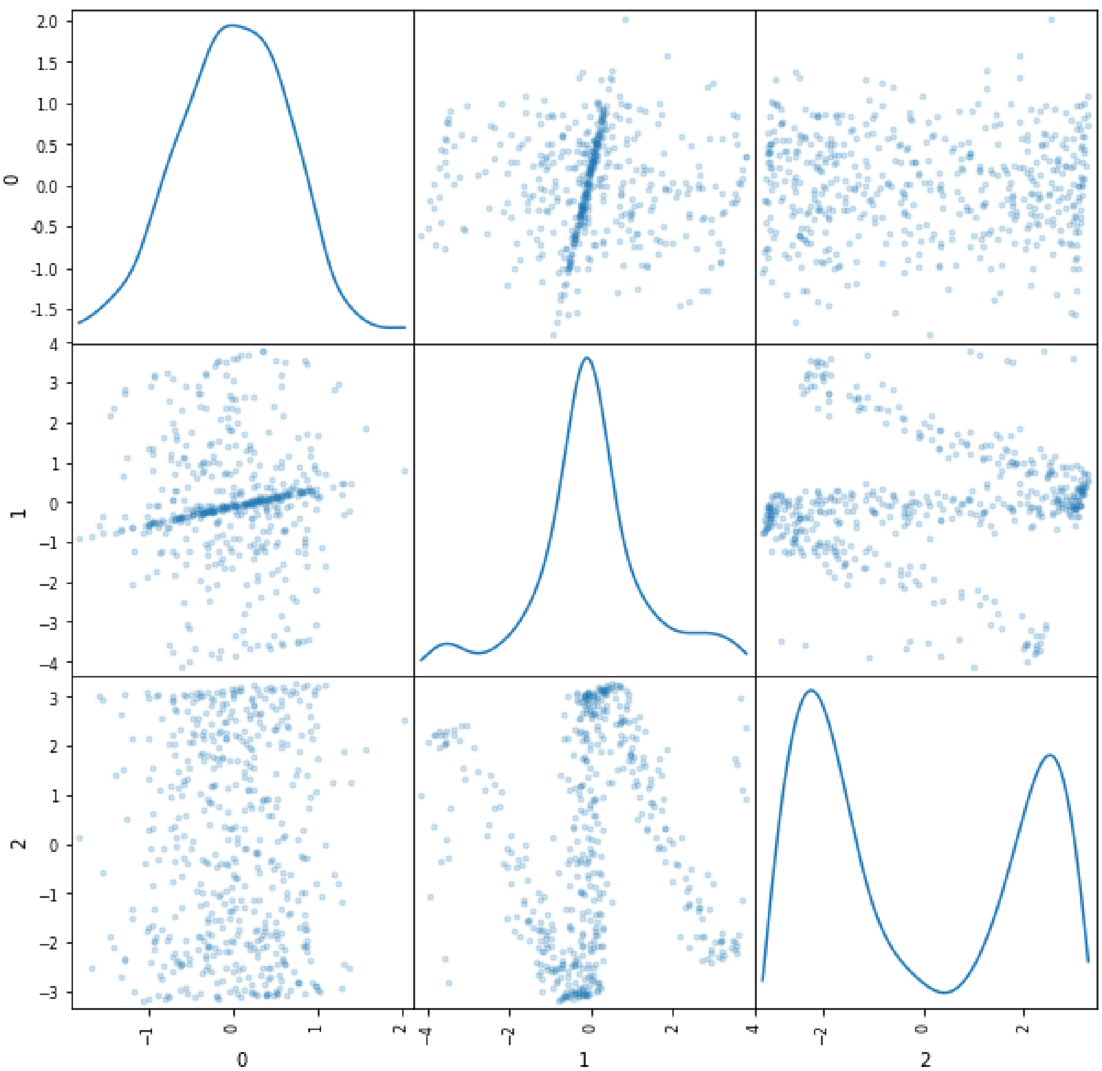

## 1.2 N.2.2 *PCA: 3-dimensional Toy Data*

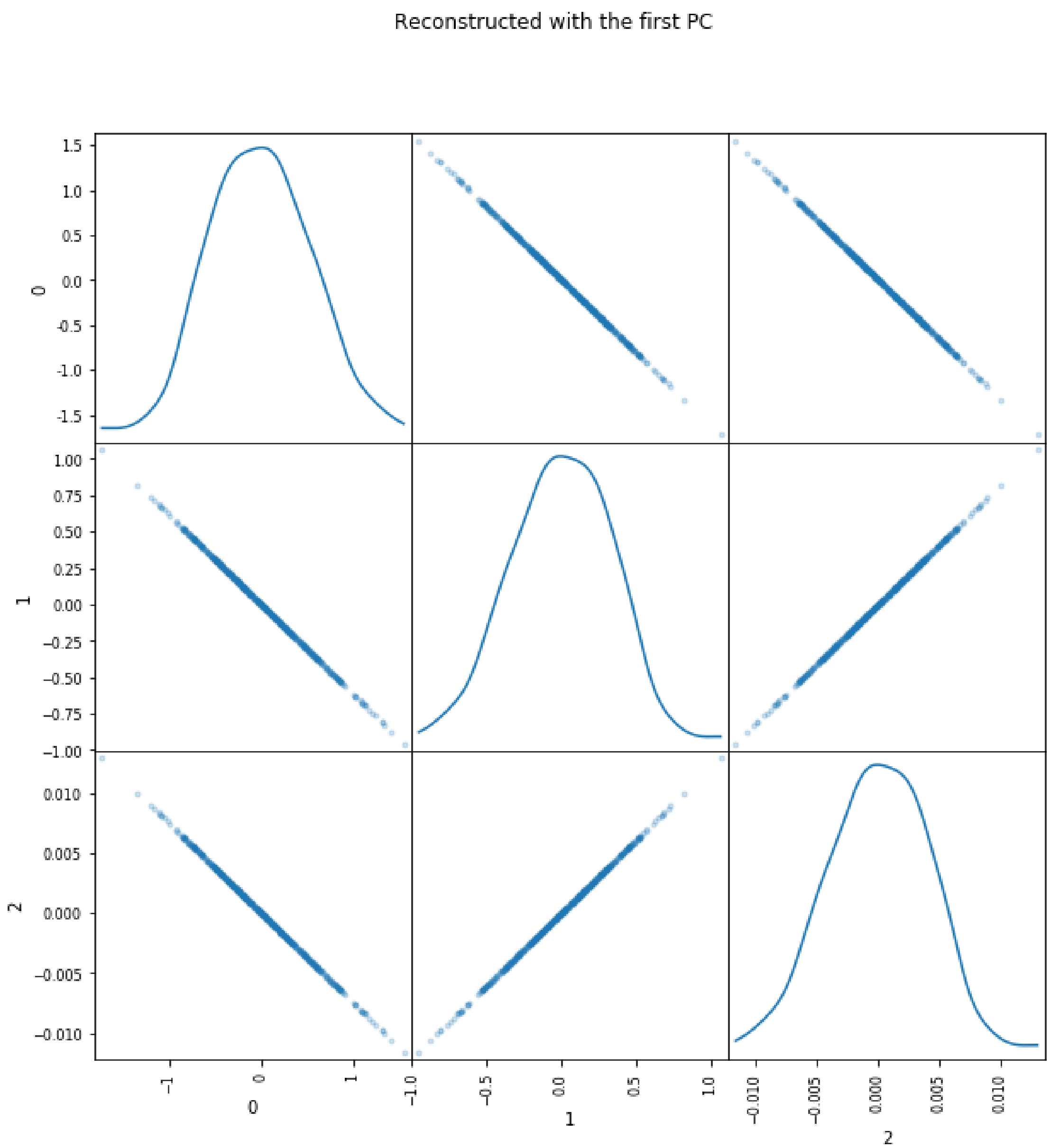
In [5]:
```
# Task 2.a:

# Read data from file:
pca3d = pd.read_csv('Data/pca-data-3d.txt')

# Center the data
pca3d_centered = pca3d - np.mean(pca3d)

# Plot the scatter matrix
scatter_matrix(pca3d, alpha=0.2, figsize=(10, 10), diagonal='kde')
plt.suptitle('3D Centered Data')
plt.show(block=False)

# Task 2.b:
# Principal Component Analysis (PCA)

# covariance
pca3d_cov = pca3d_centered.cov()
#Eigen value and eigen vector
eig_val, eig_vec = np.linalg.eig(pca3d_cov)
#projection of data from pca3d_centered.dot with eigen vector
```

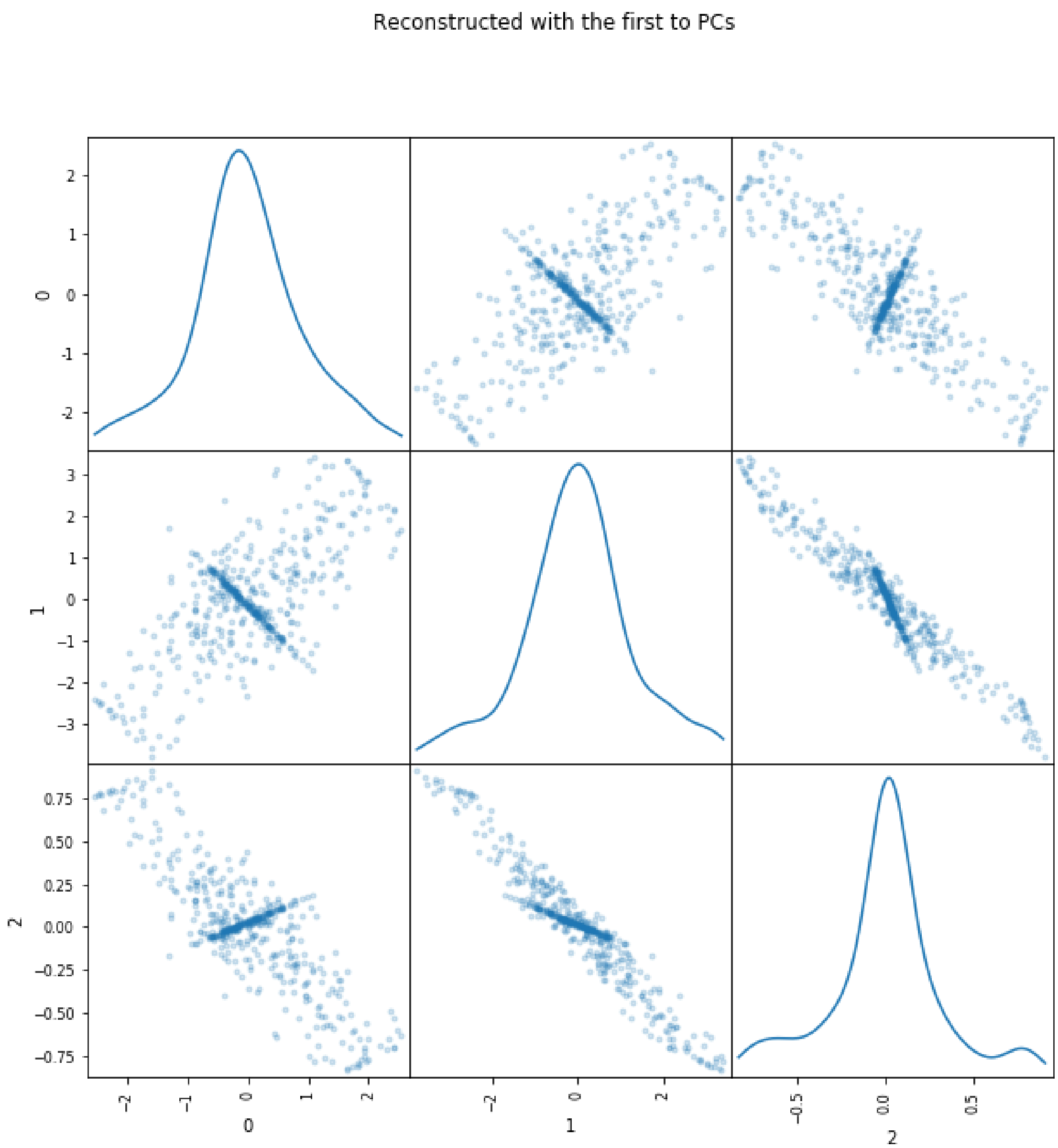
```
pca3d_proj = pca3d_centered.dot(eig_vec)
# Plot the scatter matrix
scatter = scatter_matrix(pca3d_proj, alpha=0.2, figsize=(10, 10), diagonal=
plt.suptitle('3D Projected Data')
plt.show(block=False)

# Task 2.c
# Projection on principle components

pca3d_1PC = np.matrix(pca3d_proj[0]).T
pca3d_2PC = np.matrix(pca3d_proj.T[0:2]).T
pca3d_3PC = np.matrix(pca3d_proj)

# Reconstruction with the first PC
re3dPCA_1PC = pd.DataFrame(pca3d_1PC.dot(np.matrix(eig_vec.T[0])))
# Reconstruction with the first two PCs
re3dPCA_2PC = pd.DataFrame(pca3d_2PC.dot(np.matrix(eig_vec.T[0:2])))
# Reconstruction with all the PCs
re3dPCA_3PC = pd.DataFrame(pca3d_3PC.dot(np.matrix(eig_vec.T)))

# Plotting the scatters
scatter = scatter_matrix(re3dPCA_1PC, alpha=0.2, figsize=(10, 10), diagonal
plt.suptitle('Reconstructed with the first PC')
plt.show(block=False)

scatter = scatter_matrix(re3dPCA_2PC, alpha=0.2, figsize=(10, 10), diagonal
plt.suptitle('Reconstructed with the first to PCs')
plt.show(block=False)

scatter = scatter_matrix(re3dPCA_3PC, alpha=0.2, figsize=(10, 10), diagonal
plt.suptitle('Reconstructed with all three PCs')
plt.show(block=False)

# Comment:
# Obviously when we use all the PCs to reconstruct
# the data, it is the most efficient way of reconstruction.
# We can check it by checking all of the scatter matrix plots
# of three situations stated in 2.2.c
```

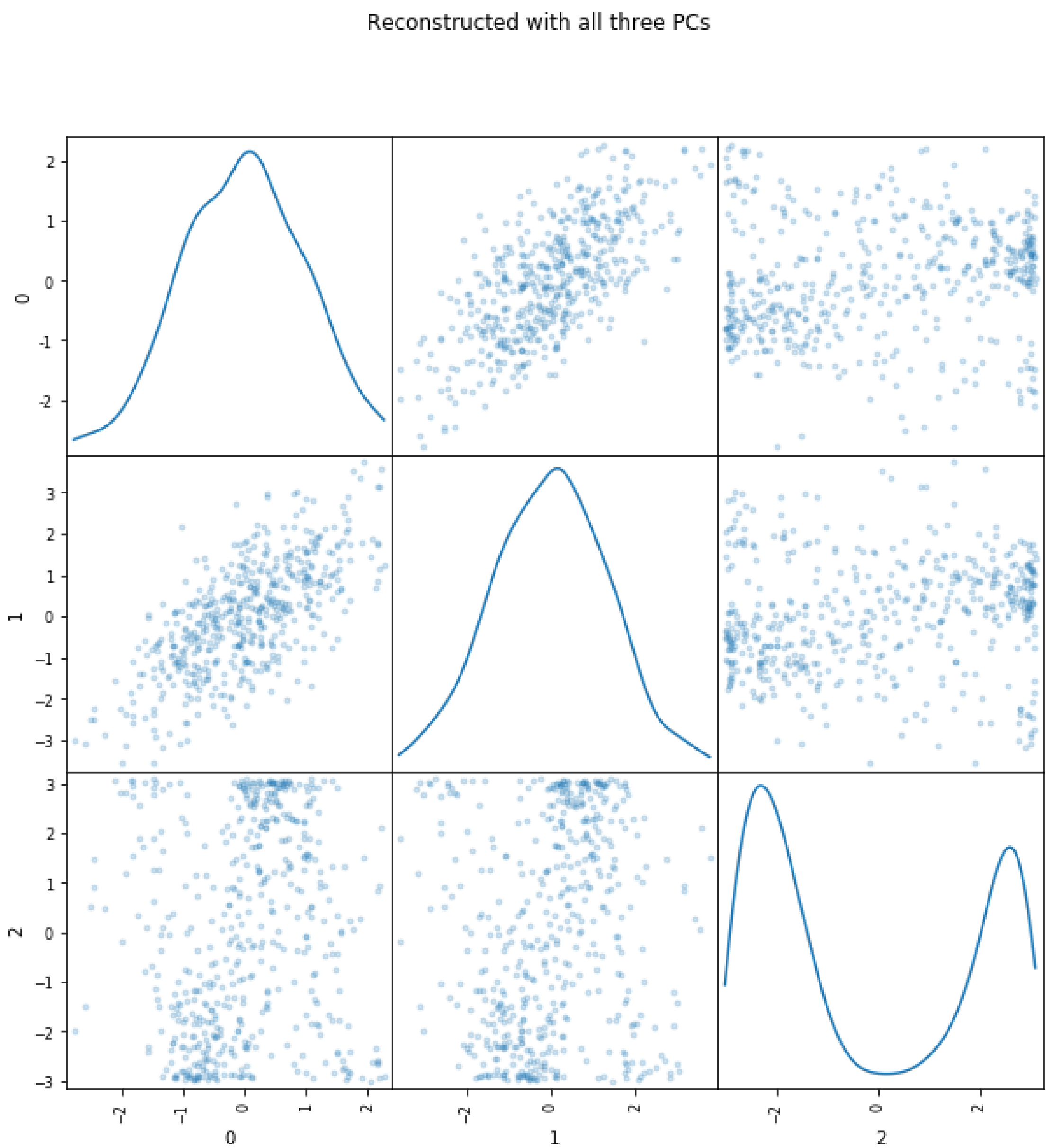

3D Centered Data
x
y
z

3D Projected Data




Reconstructed with the first PC
0
1
2


Reconstructed with the first to PCs
0
1
2

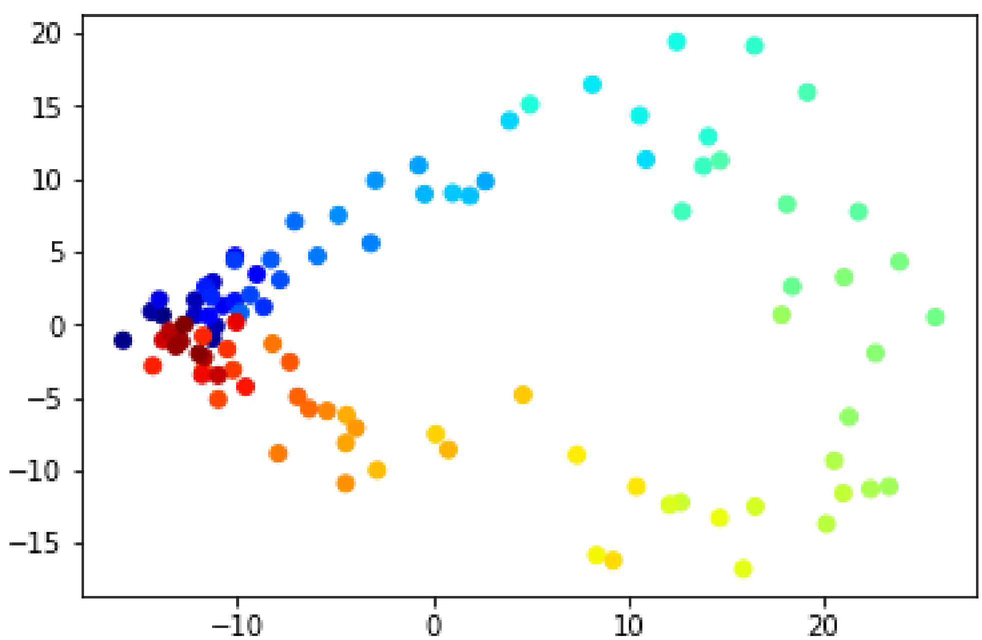


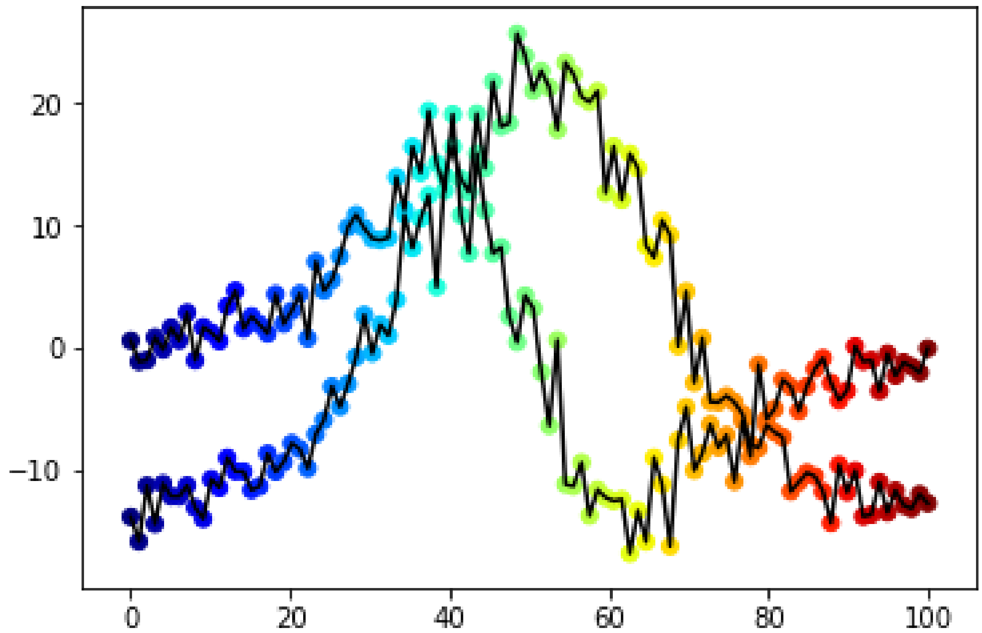

## 1.3 N.2.3 *Projections of a Dynamical System*

```
In [6]: # 2.3 a
        # load data
        expdat = pd.DataFrame.from_csv('Data/expDat.txt', parse_dates=False)
        expdatCentered = (expdat - expdat.mean(axis=0))
        # find 20 principal components
        pca3 = PCA(n_components=20)
        # extract principal components
        pca3.fit(expdat)
```

Out[6]: PCA(copy=True, iterated_power='auto', n_components=20, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)

In [7]:
```
# 2.3 b
# project data onto the first two PCs
pc12 = np.vstack((pca3.components_[0], pca3.components_[1]))
# project data onto pairs of primary components
projected12 = np.dot(expdatCentered, pc12.T)
# display color coded scatter plot of the data projected onto first two pri
time = np.linspace(0,100,100)
plt.scatter(projected12[:,:1],projected12[:,1:], c=time, cmap=cm.jet);
plt.show(block=False)

# project data onto first PC
pc1 = np.reshape(pca3.components_[0], (-1,pca3.components_.shape[0]))
projected1 = np.dot(expdatCentered, pc1.T)
# project data onto second PC
pc2 = np.reshape(pca3.components_[1], (-1,pca3.components_.shape[0]))
projected2 = np.dot(expdatCentered, pc2.T)

fig = plt.figure(1)
fig.clf()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(time, projected1, c=time, cmap=cm.jet)
ax.plot(time, projected1, c="black")
ax = fig.add_subplot(1, 1, 1)
ax.scatter(time, projected2, c=time, cmap=cm.jet)
ax.plot(time, projected2, c="black")
plt.show(block=False)
```



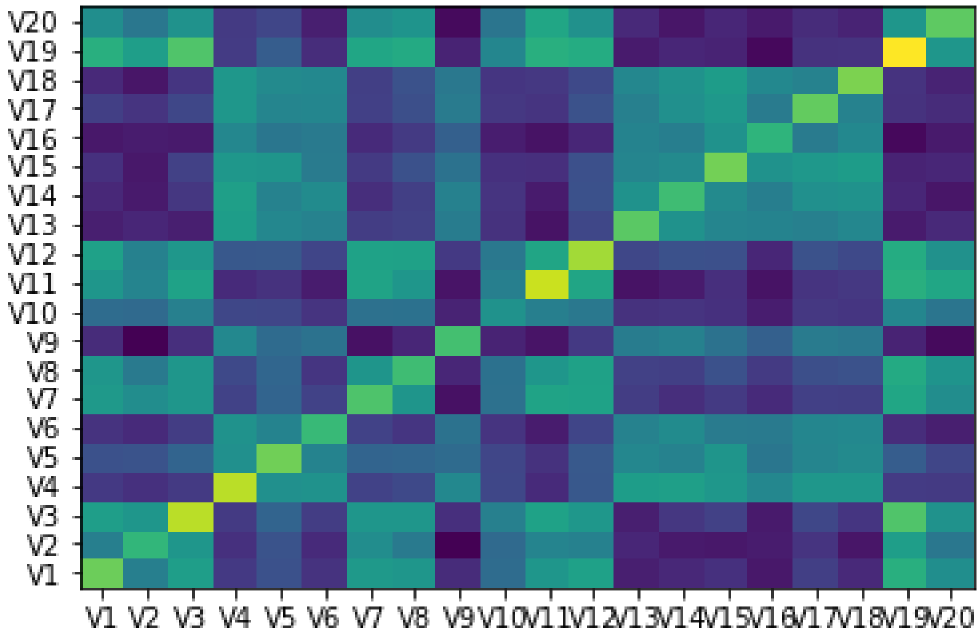


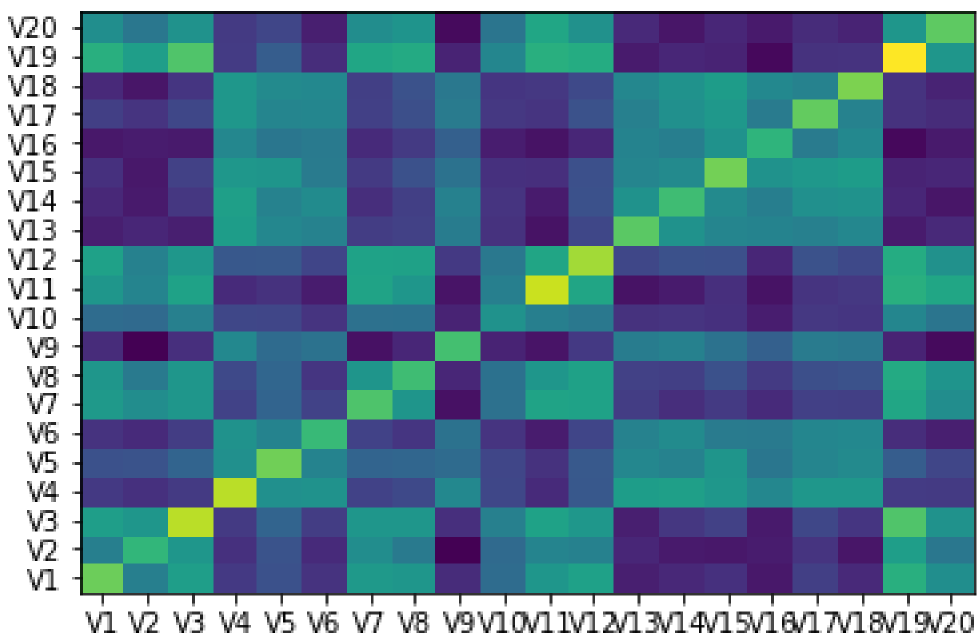

```
In [8]: # 2.3 c
        # shuffle data
        expdatshuffled = expdat.copy()
        for _ in range(expdatshuffled.shape[1]):
            expdatshuffled.apply(np.random.shuffle, axis=1)
        print expdat.head(1)
        print expdatshuffled.head(1)
```

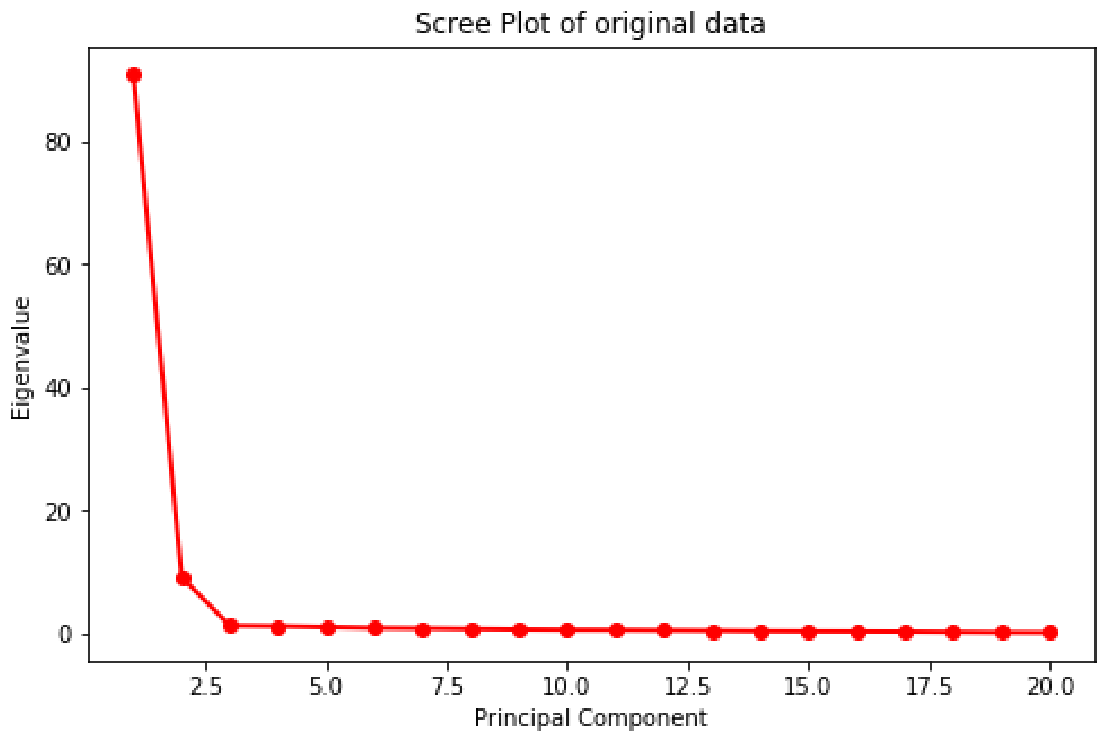
```
   V1  V2  V3  V4  V5  V6  V7  V8  V9  V10  V11  V12  V13  V14  V15  V16  V17  \
1   1   0   0   2   2   0   2   3   2    3    2    3    3    5    1    1    1

   V18  V19  V20
1    4    2    1
   V1  V2  V3  V4  V5  V6  V7  V8  V9  V10  V11  V12  V13  V14  V15  V16  V17  \
1   1   0   0   2   2   0   2   3   2    3    2    3    3    5    1    1    1

   V18  V19  V20
1    4    2    1
```

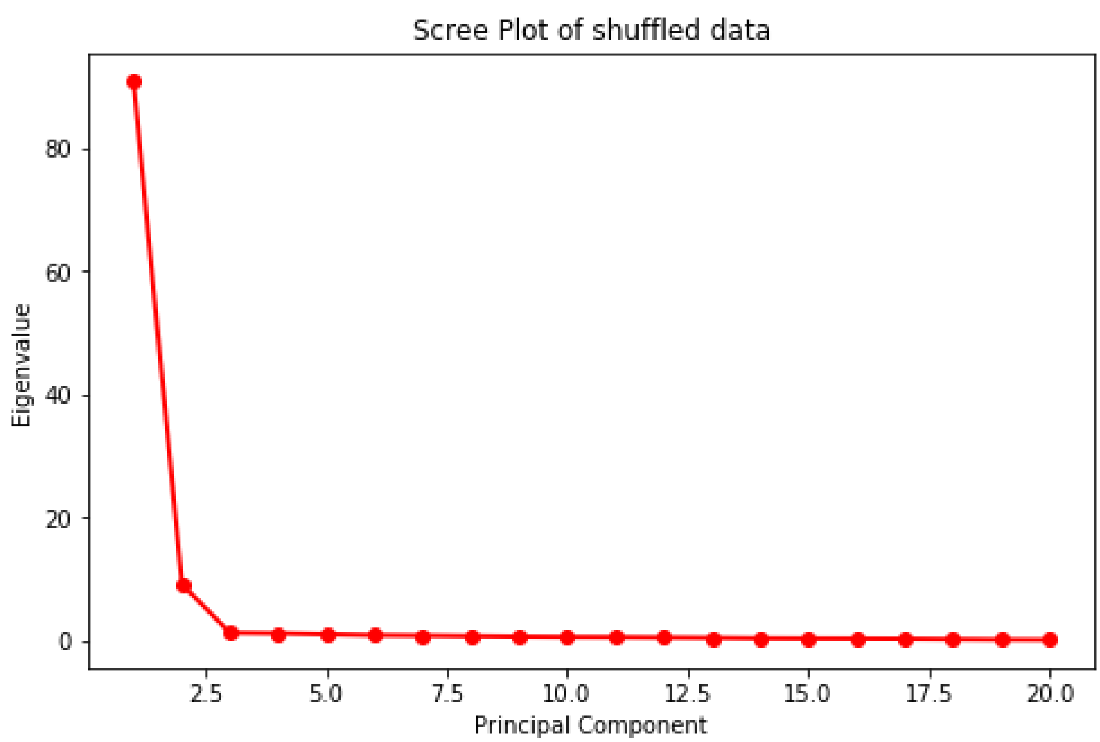
```
In [9]: # 2.3 d
        # compute covariances
        expdatcov = expdat.cov()
```

```
expdatshuffledcov = expdatshuffled.cov()

# display covariances
plt.pcolor(expdatcov)
plt.yticks(np.arange(0.5, len(expdatcov.index), 1), expdatcov.index)
plt.xticks(np.arange(0.5, len(expdatcov.columns), 1), expdatcov.columns)
plt.show(block=False)
plt.pcolor(expdatshuffledcov)
plt.yticks(np.arange(0.5, len(expdatshuffledcov.index), 1), expdatshuffledc
plt.xticks(np.arange(0.5, len(expdatshuffledcov.columns), 1), expdatshuffle
plt.show(block=False)

# display scree plots
# first, do singular value decomposition to obtain eigenvalues
U, S, V = np.linalg.svd(expdat)
# now compute the magintude of the eigenvalues
eigvals = S**2 / np.cumsum(S)[-1]
fig = plt.figure(figsize=(8,5))
# plot the eigenvalues
plt.plot(np.arange(len(expdat.columns)) + 1, eigvals , 'ro-', linewidth=2)
plt.title('Scree Plot of original data')
plt.xlabel('Principal Component')
plt.ylabel('Eigenvalue')
plt.show(block=False)

# same as above for shuffled data
U, S, V = np.linalg.svd(expdatshuffled)
eigvalsshuffled = S**2 / np.cumsum(S)[-1]
fig = plt.figure(figsize=(8,5))
plt.plot(np.arange(len(expdatshuffled.columns)) + 1, eigvalsshuffled, 'ro-'
plt.title('Scree Plot of shuffled data')
plt.xlabel('Principal Component')
plt.ylabel('Eigenvalue')
plt.show(block=False)
```


V20
V19
V18
V17
V16
V15
V14
V13
V12
V11
V10
V9
V8
V7
V6
V5
V4
V3
V2
V1
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20


V20
V19
V18
V17
V16
V15
V14
V13
V12
V11
V10
V9
V8
V7
V6
V5
V4
V3
V2
V1
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20


Scree Plot of original data
Eigenvalue
0
20
40
60
80
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Principal Component


Scree Plot of shuffled data
Eigenvalue
0
20
40
60
80
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Principal Component

## 1.4 N.2.4 *Image Data Compression and Reconstruction*

In [10]:
```
import numpy as np
from PIL import Image

# Task 4.a

# nature pictures

natimg = np.array([])

for item in range(10):
    image = Image.open('Data/imgpca/n' + str(item + 1) + '.jpg')
    imgdata = np.array(image.getdata())
    natimg = np.append(natimg,imgdata[0:(256*500)])

natimg = natimg.reshape(5000,256)

# building pictures

buildimg = np.array([])

for item in range(10):
    image = Image.open('Data/imgpca/n' + str(item + 1) + '.jpg')
    imgdata = np.array(image.getdata())
    buildimg = np.append(buildimg,imgdata[0:(256*500)])

buildimg = buildimg.reshape(5000,256)
```