Problem Set Three

Zhixian(Jason) Yu

03/20/2018

1 Theory

Problem 1/3

Because $\phi^{(i)}$, $\psi^{(i)}$, $u^{(i)}$ and $v^{(i)}$ are all unit vectors, we have $\|u^{(i)}\| = 1$ and $\|v^{(i)}\| = 1$. In addition, ${\phi^{(i)}}^T\phi^{(i)} = 1$ and ${\psi^{(i)}}^T\psi^{(i)} = 1$. Therfore, the cosine of the angle between $u^{(i)}$ and $v^{(i)}$ is:

$$\begin{aligned}
cos\theta &= \frac{{u^{(i)}}^T v^{(i)}}{\|u^{(i)}\|\|v^{(i)}\|} \\
&= \left(Q_X\phi^{(i)}\right)^T Q_Y\psi^{(i)} \\
&= {\phi^{(i)}}^T {Q_X}^T Q_Y\psi^{(i)} \\
&= {\phi^{(i)}}^T \phi^{(i)}\lambda_i {\psi^{(i)}}^T \psi^{(i)} \\
&= \lambda_i
\end{aligned}$$

Problem 2

We have:

$$u = Q_X\phi, v = Q_Y\psi$$

and:

$$u = Xa, v = Yb$$

Therefore:

$$Xa = Q_X\phi, Yb = Q_Y\psi \tag{1}$$

Since $X = Q_X R_X, Y = Q_Y R_Y$, so equation 1 can be rewirtten as:

$$Q_X R_X a = Q_X\phi, Q_Y R_Y b = Q_Y\psi$$

Thus:

$$Q_X(R_X a - \phi) = 0, Q_Y(R_Y b - \psi) = 0$$

Because both Q_X and Q_Y are orthonormal basis, we get $R_X a = \phi$ and $R_Y b = \psi$. Therefore:

$$a = R_X^\dagger\phi, b = R_Y^\dagger\psi$$

Problem 4

Suppose we have solutions $\psi^{(i)}$ and $\psi^{j)}$. The following two equations hold:

$$\alpha_i X^T X\psi^{(i)} = \beta_i Q^T Q\psi^{(i)} \tag{2}$$

and

$$\alpha_j X^T X\psi^{(j)} = \beta_j Q^T Q\psi^{(j)} \tag{3}$$

We can take the transpose on both sides of the equation 2 and we get:

$$\alpha_i {\psi^{(i)}}^T X^T X = \beta_i {\psi^{(i)}}^T Q^T Q \tag{4}$$

We then times $\psi^{(j)}$ on both the right sides of the equation 5 and we get:

$$\alpha_i {\psi^{(i)}}^T X^T X\psi^{(j)} = \beta_i {\psi^{(i)}}^T Q^T Q\psi^{(j)} \tag{5}$$

From equation 3, we know that:

$$Q^TQ\psi^{(j)} = \frac{\alpha_j}{\beta_j}X^TX\psi^{(j)} \tag{6}$$

Therefore equation 5 can be rewritten as:

$$\alpha_i{\psi^{(i)}}^TX^TX\psi^{(j)} = \beta_i{\psi^{(i)}}^T(\frac{\alpha_j}{\beta_j}X^TX\psi^{(j)}) \tag{7}$$

Thus:

$$(\alpha_i\beta_j - \beta_i\alpha_j){\psi^{(i)}}^TX^TX\psi^{(j)} = 0 \tag{8}$$

The requirement for GSVD demands that $\alpha_i + \beta_i = 1$, therefore:

$$\begin{aligned}\alpha_i\beta_j - \beta_i\alpha_j &= (1-\beta_i)\beta_j - (1-\beta_j)\beta_i \\ &= \beta_j - \beta_i\end{aligned}$$

So equation 8 becomse:

$$(\beta_j - \beta_i){\psi^{(i)}}^TX^TX\psi^{(j)} = 0$$

When $i \neq j$, we have $\beta_j \neq \beta_i$, therefore ${\psi^{(i)}}^TX^TX\psi^{(j)} = 0$.
When $i = j$, because $\frac{X^TX\psi^{(i)}}{\beta_j} = ({\psi^{(i)}}^{-1})^T$, ${\psi^{(i)}}^TX^TX\psi^{(i)} = \beta_i$.
Thus ${\psi^{(i)}}^TX^TX\psi^{(j)} = \beta_i\delta_{ij}$.

Similarly, if we perform similar derivations on equation 3 (taking the tranpose and timing $\psi^{(i)}$), we can get the following equation:

$$(\beta_j - \beta_i){\psi^{(i)}}^TQ^TQ\psi^{(j)} = 0$$

And we can conclude that ${\psi^{(i)}}^TQ^TQ\psi^{(j)} = \alpha_i\delta_{ij}$.

Problem 5

Here I use X_s, N_s, S_s to denote the shifted version of X, N, S, respectively. We know that $dX = X - X_s$, so:

$$\begin{aligned}dX^TdX &= (X - X_s)^T(X - X_s) \\ &= (S - S_s + N - N_s)^T(S - S_s + N - N_s) \\ &= S^TS - S^TS_s + S^TN - S^TN_s - S_s^TS + S_s^TS_s - S_s^TN + S_s^TN_s + \\ &\quad N^TS - N^TS_s + N^TN - N^TN_s - N_s^TS + N_s^TS_s - N_s^TN + N_s^TN_s\end{aligned}$$

If N is pure noise, we can assume that N is independent of the signal S. In addition, shifted noise N_s, which is another random noise matrix, is also independent of the signal S. Similarly, shifted signal S_s is independent of N and N_s. Therefore $N^TS = 0, N_s^TS = 0, N^TS_s = 0, N_s^TS_s = 0$. So we get:

$$dX^TdX = S^TS - S^TS_s - S_s^TS + S_s^TS_s + N^TN - N^TN_s - N_s^TN + N_s^TN_s$$

Another assumption we can make is that any vector from N_s and another vector from N are orthogonal because two random noise vectors from a high-dimensional space are highly likely to be independent. Therefore ${N_s}^TN = 0$. and we get:

$$dX^TdX = S^TS - S^TS_s - S_s^TS + S_s^TS_s + N^TN + N_s^TN_s$$

Thirdly, if S is a smooth signal, shifting S will not dramatically change the values of each element of S, therefore $S^T S \approx S^T S_s$, so:

$$dX^T dX = N^T N + N_s^T N_s$$

Finally, because N_s is just another random noise matrix drawn from the same space of N, we can approximate $N^T N$ using $N_s^T N_s$, therefore $dX^T dX = 2N^T N$, and:

$$N^T N = \frac{1}{2} dX^T dX$$

2 Programming

The following code sets up the environment and import packages.

```
import scipy.io as spio
import numpy as np
import matplotlib.pyplot as plt
```

Problem 1

The following code defines a function to execute CCA algorithm.

```
def CCA(X, Y, mean_substract=True):
    '''
    :param X, Y : input vectors to analyze CCA,
    '''
    if mean_substract:
        X = X - np.mean(X, axis=0)[None, :]
        Y = Y - np.mean(Y, axis=0)[None, :]
    Qx, Rx = np.linalg.qr(X)
    Qy, Ry = np.linalg.qr(Y)
    phi, s, psi = np.linalg.svd(Qx.T@Qy)
    return phi, s, psi, Qx, Rx, Qy, Ry
```

First we need to load the data and separate the data into X and Y, as shown below.

```
mat = spio.loadmat('data/MardiaExamData.mat', squeeze_me=True)
data = mat['EXAMS']
X = data[:, :2]
Y = data[:, 2:]
```

a) We can call the CCA function to compute a and b.

```
phi, s, psi, Qx, Rx, Qy, Ry = CCA(X, Y, True)
a = np.linalg.pinv(Rx)@phi
b = np.linalg.pinv(Ry)@psi
```

The result is $a_1 = \begin{bmatrix}0.00276961 & 0.00551701\end{bmatrix}^T$, $a_2 = \begin{bmatrix}0.00682024 & -0.00808835\end{bmatrix}^T$, $b_1 = \begin{bmatrix}0.01052326 & -0.00104472 & 0.00050926\end{bmatrix}^T$, $a_2 = \begin{bmatrix}0.00788944 & -0.010537 & 0.00149612\end{bmatrix}^T$.

b) The following code is used to computer α_i and β_i.

```
alpha = X@a
beta = Y@b
plt.plot(alpha[:, 0], 'r')
plt.plot(beta[:, 0], 'g')
plt.show()
```

The result is shown in figure 1. We can see that there is a strong correlation between α and β.

c) $u^{(1)}, u^{(2)}, v^{(1)}, v^{(2)}$ are computed as following. From the SVD on $Q_X^T Q_Y$ we can get different sets of singular vectors associated with different singular values, therefore we can get different u, v through different sets of singular vectors.

```
u = Qx@phi
v = Qy@psi
```

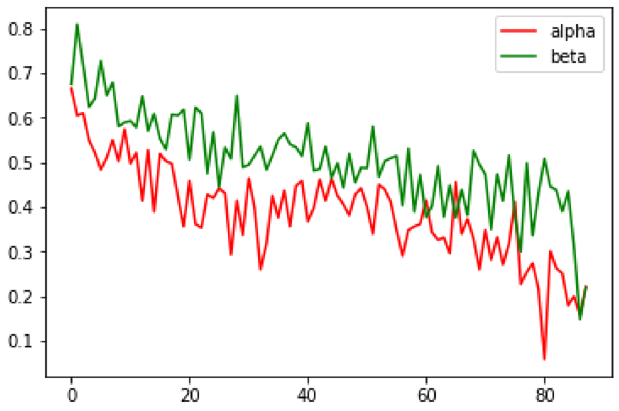

Figure 1: α_i and β_i are plotted.

c) The largest singular value we get from SVD on $Q_X^T Q_Y$ is 0.66305211. The cosine of the angle between $u^{(1)}$ and $v^{(1)}$ is calculated using the following code, and the result is 0.65207818. They are very close to each other.

```
u[:, 0].T@v[:,0] / np.linalg.norm(u[:, 0])/ np.linalg.norm(v[:, 0])
```

Problem 2

Data is loaded first using the following code. Data is also mean-centered.

```
mat = spio.loadmat('data/MNFdata.mat', squeeze_me=True)
X = mat['X']
X = X - np.mean(X, axis=0)
Xs = np.vstack((X[1:, :], X[0, :])) # shifted X
dX = X - Xs
```

a)-i) The following code finds the maximum noise fraction basis *phi* by solving the eigenvector problem.

```
Y = np.linalg.inv(dX.T@dX/2)@X.T@X
w, psi = np.linalg.eig(Y)
phi = X@psi
```

The following code finds the maximum noise fraction basis *phi* by solving the generalized eigenvector problem.

```
import scipy
C = X.T@X
D = dX.T@dX/2
w, psi = scipy.linalg.eig(C, D)
phi = X@psi
```

The following code finds the maximum noise fraction basis *phi* by solving the GSVD problem. Because python does not have a gsvd() function, I called the gsvd() from matlab in python.

```
import matlab.engine
eng = matlab.engine.start_matlab()
u,v,x,c,s = eng.gsvd(matlab.double((dX/np.sqrt(2)).tolist()), matlab.double
   (X.tolist()), 0,nargout=5)
psi = np.linalg.inv(np.array(x).T)
phi = X@psi
```

b) The basis vectors from different methods are shown in figure 2, 3 and 4. All three methods seem to be able to separate noise and signal, and generate similar basis vectors. Some of the basis vectors have different direction, thus the figure seems flipped along x axis.

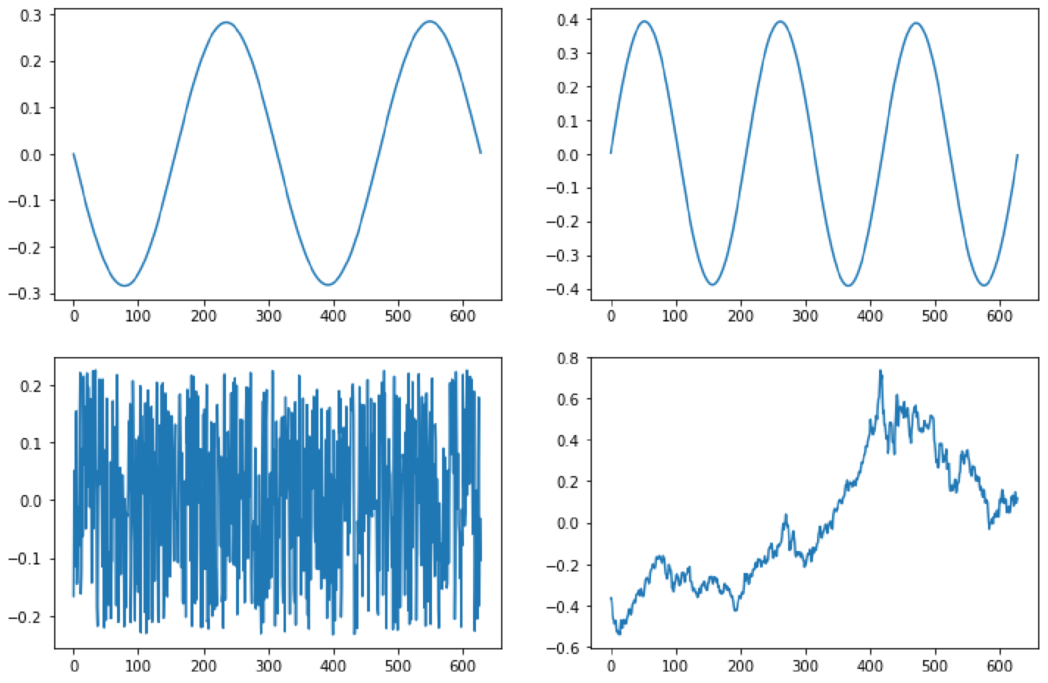
.

Figure 2: Basis vectors from eigenvector method.

c) The following code is used to compute the SVD of the data matrix and plot the basis vectors.

```
u, s, vh = np.linalg.svd(X, full_matrices=False)

plt.figure(figsize=(12,8))
for i in range(2):
    for j in range(2):
        plt.subplot(2,2, i*2 + j + 1)
        plt.plot(u[:, i*2+j])
plt.show()
```

Figure 5 shows the basis vectors from SVD. SVD does not separate the noise with signal. This makes sense because SVD finds the best axis that maximizes the variance.

d) The following code is used to project data matrix onto the first 2 MNF basis vectors and the first 2 SVD basis vectors.

```
B_mnf = phi[:, :2]
B_svd = u[:, :2]
proj_mnf, proj_svd = B_mnf@B_mnf.T@X, B_svd@B_svd.T@X

plt.figure(figsize=(12,8))
for i in range(2):
    for j in range(2):
```

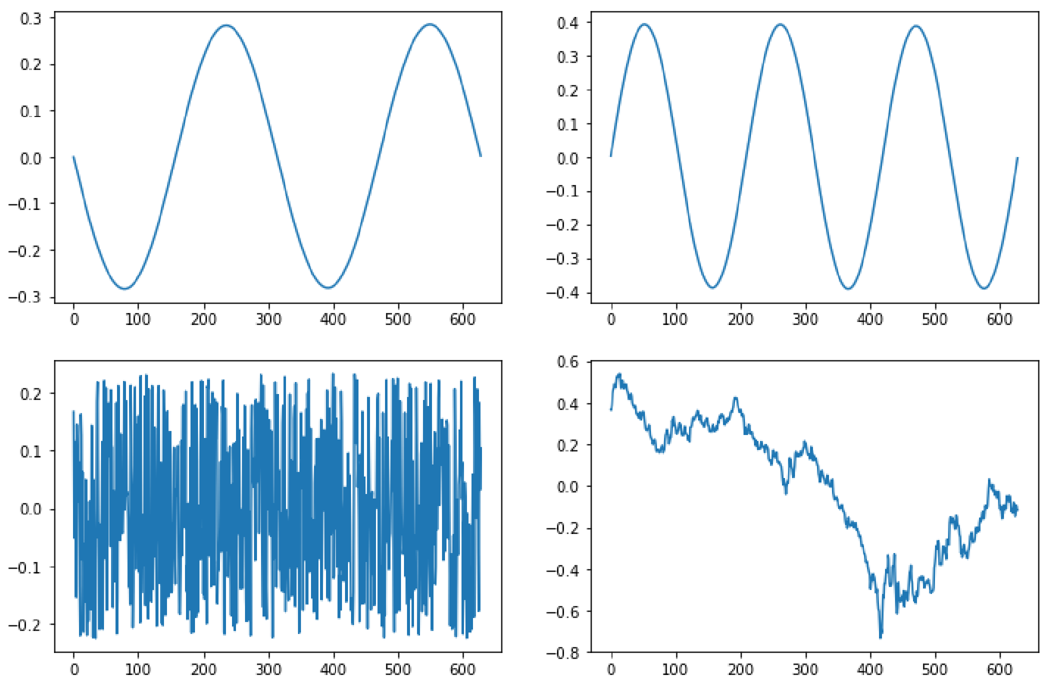
.

Figure 3: Basis vectors from generalized eigenvector method.

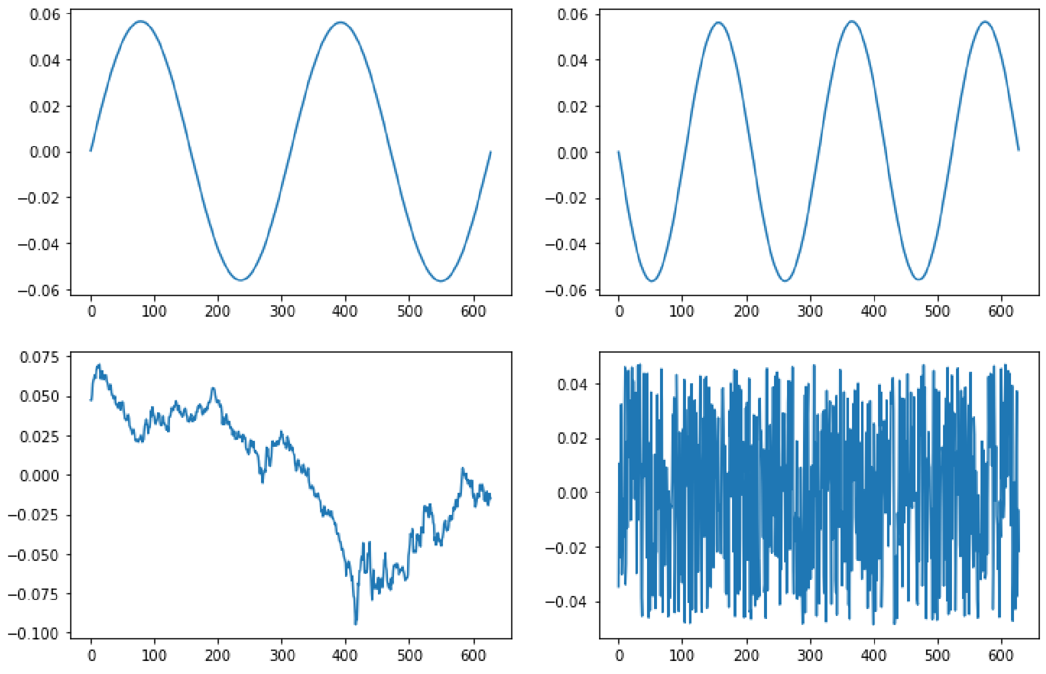
.

Figure 4: Basis vectors from GSVD method.

```
        plt.subplot(2,2, i*2 + j + 1)
        plt.plot(proj_mnf[:, i*2+j])
```

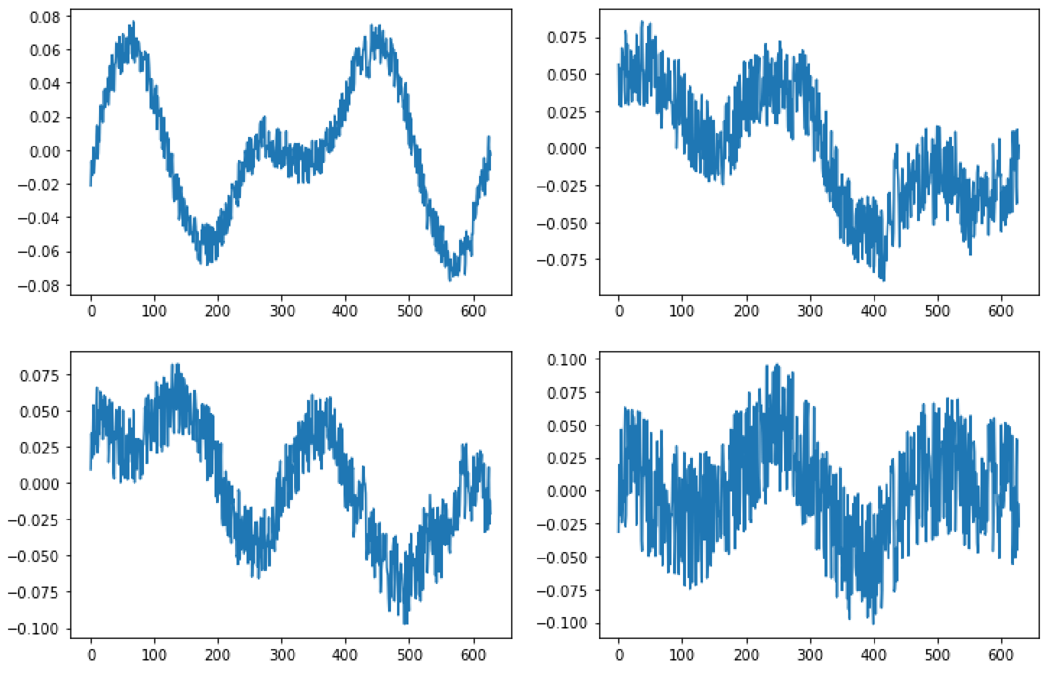

Figure 5: Basis vectors from SVD on data matrix.

```
plt.show()

plt.figure(figsize=(12,8))
for i in range(2):
    for j in range(2):
        plt.subplot(2,2, i*2 + j + 1)
        plt.plot(proj_svd[:, i*2+j])
plt.show()
```

Figures 6 and 7 show the projections. The projections on the MNF basis vectors are smooth because they are linear combination of two smooth curves. On the contrary, projections on the SVD basis vectors are noisy because the SVD basis vectors do not separate actual noise from signal.

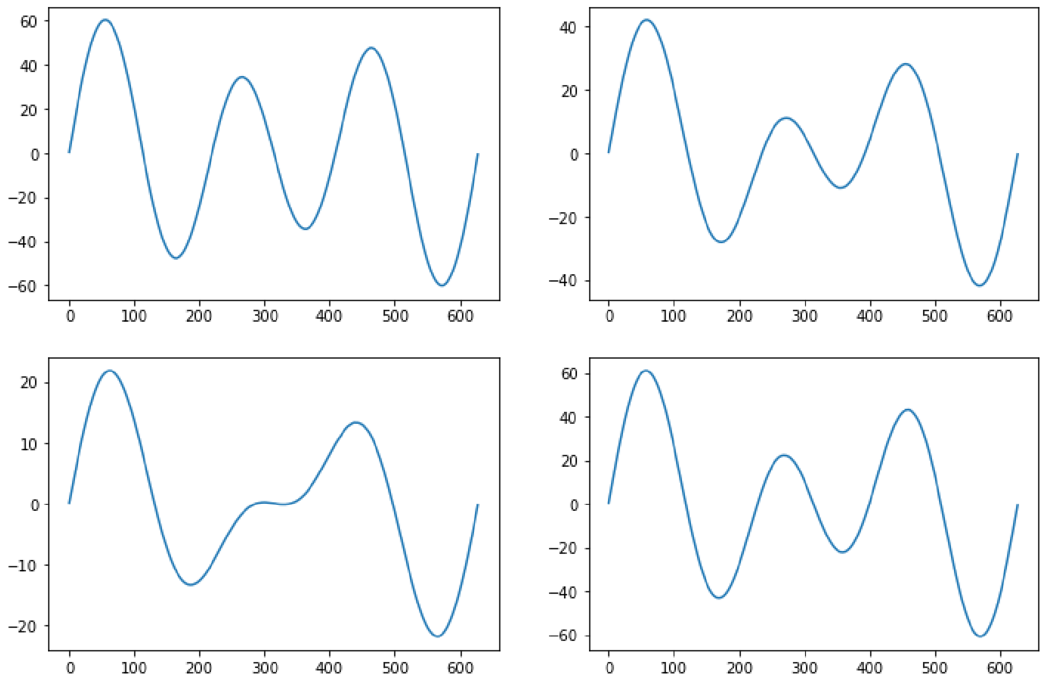

Figure 6: Projection of X onto first 2 MNF basis vectors.

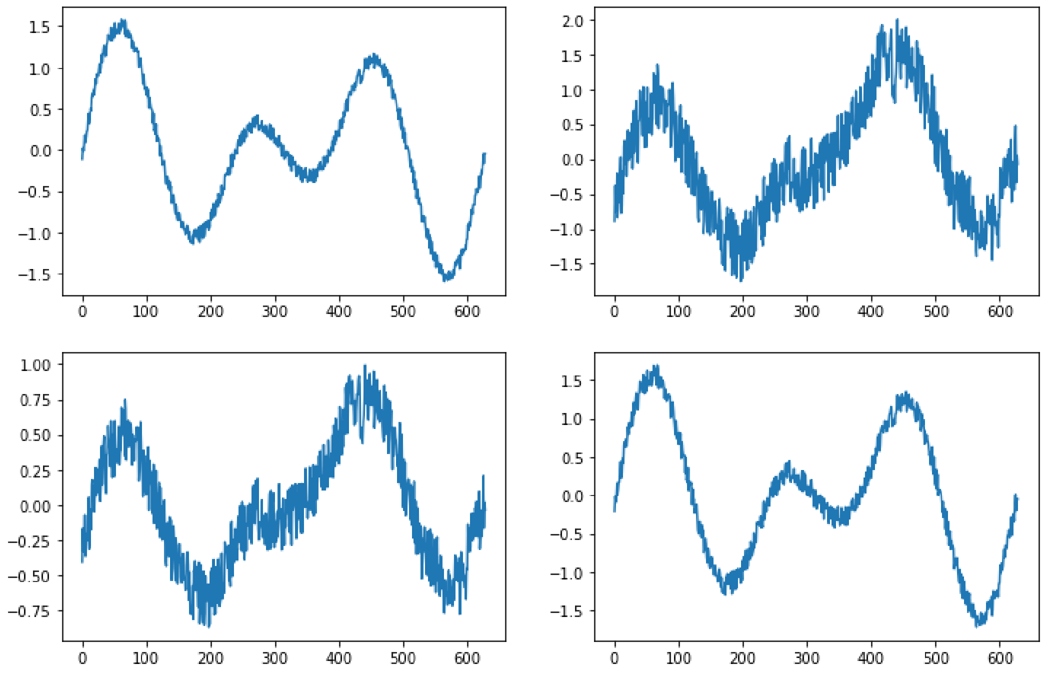

Figure 7: Projection of X onto first 2 SVD basis vectors.