zxw534 - 491 - A5

April 26, 2018

1 491 Assignment 5

1.1 EX1

In this exercise, I will write a function that can separate sounds from the mixed sound.

According to the ICA algorithm, the mixed sounds comes after some sounds mixed.

$$S = \begin{bmatrix} S_1 \\ S_2 \\ ... \\ S_i \\ S_n \end{bmatrix}$$

S is the matrix that includes sounds.

$X = AS$, A is the mixing matrix.

In this exercise, our purpose is separating S from X.

We have X now.

We can initialize A as an unit matrix.

Then we get the first S from $S = A^{-1}X$.

After that the function will iteratively refresh A by using the equation: $dA = -A(zs^T + I)$, where $z = (logP(s))'$

Because $P(s) \propto e^{\frac{-|s|^{2/(1+\beta)}}{2}}$

So $z = (\frac{-|s|^{2/(1+\beta)}}{2})'$

In order to calculate easily, we can set $\beta = 1$

Hence, $z = (\frac{-|s|}{2})'$

When $s \geq 0$, z = 0.5

When $s < 0$, z = -0.5

```
In [1]: import numpy as np
```

```
In [2]: def bss(X):
            n,m = X.shape
            iteration = 1000
            #eta is the learning rate
            eta = 0.05
            # A is the mixing matrix
            A = np.identity(n)
            I = np.identity(n)
```

```
    for i in range (iteration):
        s = np.linalg.inv(A).dot(X)
        a,b = s.shape
        z = np.zeros((a,b))
        for x in range (a):
            for y in range(b):
                if (s[x][y]>=0):
                    z[x][y] = -0.5
                else:
                    z[x][y] = 0.5
        dtA = A.dot(z.dot(s.transpose())+I)
        A = A - eta*dtA
    return A,s
```

1.2 EX2

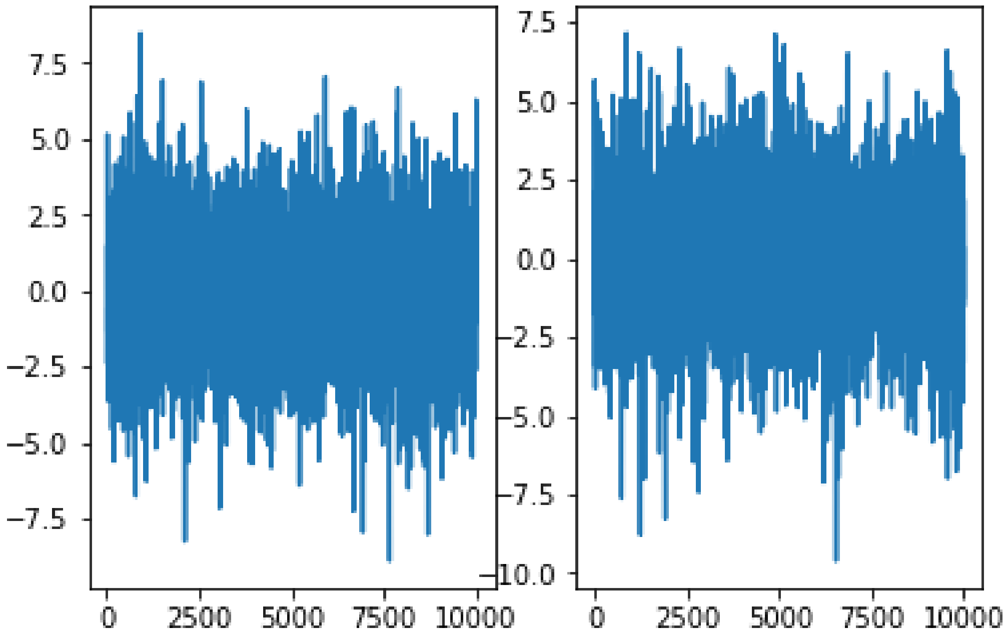
In this exercise, I will synthetic mixtures from a known mixing matrix then use the function in exercise 1 to separate the samples.

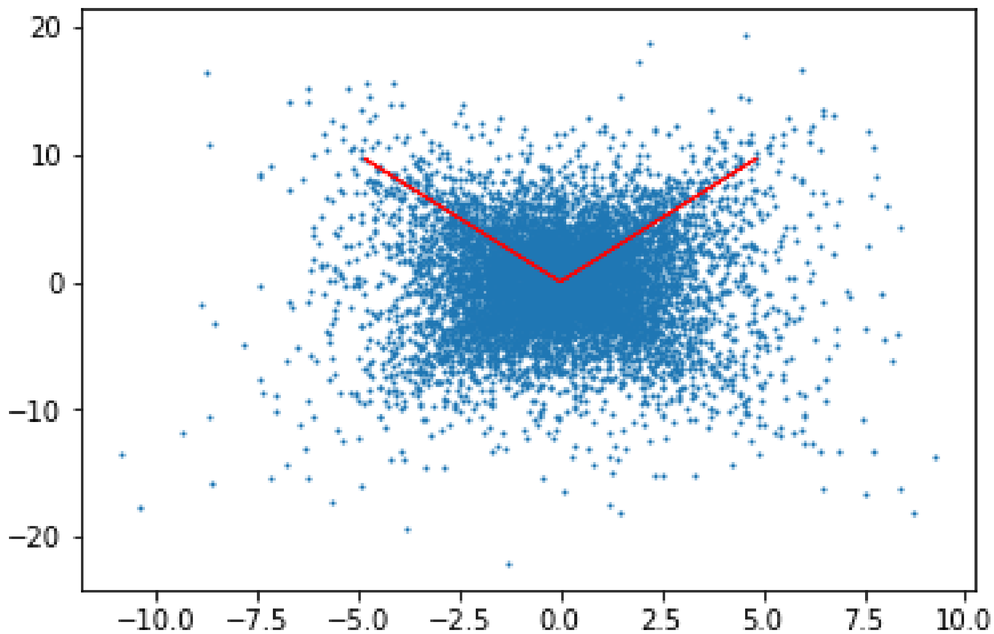
```
In [3]: import matplotlib.pyplot as plt
```

```
In [4]: def syntheticDataGenerate(verifyA, nsamples):
            n = verifyA.shape[0]
            laplace = np.random.laplace(0,1,(n,nsamples))
            plt.subplot(1,2,1)
            plt.plot(laplace[0])
            plt.subplot(1,2,2)
            plt.plot(laplace[1])
            synthData = verifyA.dot(laplace)
            return synthData
```

```
In [5]: def drawDataWithMixingMatrix(data, mat):
            # plot data points
            plt.scatter(data[0], data[1], s=1)
            # calculate axis length
            lenAxis = np.sqrt(np.sum(np.square(mat), axis=0))
            # calculate scale for illustration
            scale = np.min(np.max(np.abs(data), axis=1) / lenAxis.T)
            # draw axis as arrow
            plt.arrow(0, 0, scale * mat[0,0], scale * mat[1,0], shape='full', color='r')
            plt.arrow(0, 0, scale * mat[0,1], scale * mat[1,1], shape='full', color='r')
```

```
In [6]: nsamples = 10000
        verifyA = np.asarray([[-1, 1],[2, 2]])
        synthData = syntheticDataGenerate(verifyA, nsamples)
        plt.figure()
        drawDataWithMixingMatrix(synthData, verifyA)
```



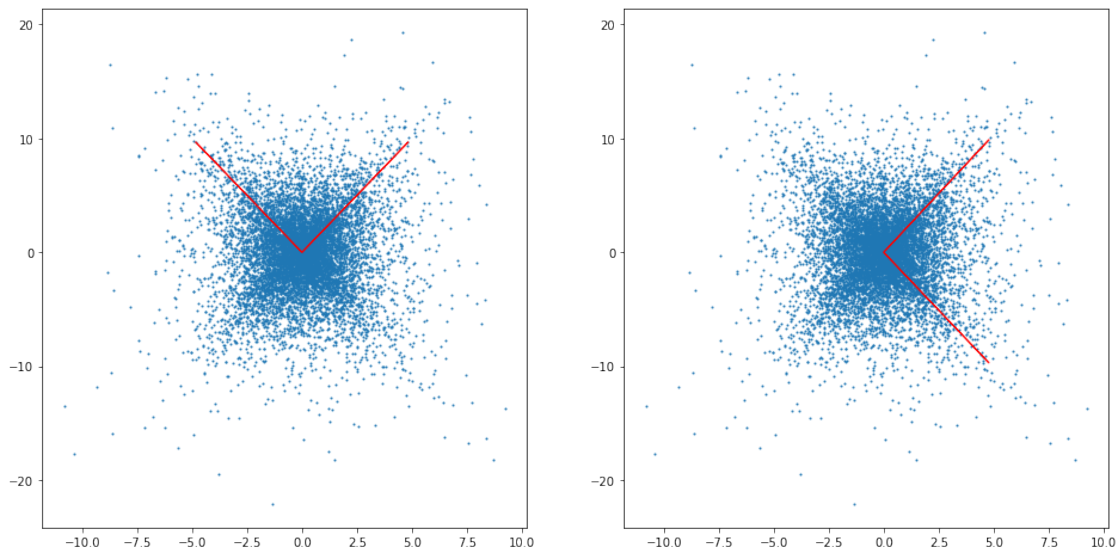


Verify my function in exercise 1.

```
In [7]: estimateA, recoverData = bss(synthData)
```

In [8]:
```
def compareMixingMatrix(data, matA, matB):
    plt.figure(figsize=(16, 8))
    # plot first mixing matrix
    plt.subplot(1,2,1)
    drawDataWithMixingMatrix(data, matA)
    # plot first mixing matrix
    plt.subplot(1,2,2)
    drawDataWithMixingMatrix(data, matB)
```

In [9]:
```
compareMixingMatrix(synthData,verifyA, estimateA)
```



According to the result, my function is correct.

1.3 EX3

Apply my function to the sound mixtures to recover the original sources.

In [10]:
```
from scipy.io import wavfile

srate, data1 = wavfile.read('data/bach.wav')
_, data2 = wavfile.read('data/speech.wav')
```

In [11]:
```
def newaudionorm(data):
    length = []
    for i in range (len(data)):
        length.append(data[i].shape[0])
    n = min(length)
    for j in range(len(data)):
        if data[j].shape[0]>n:
            data[j] = data[i][0:n]
```

```
        # check
        for k in range(len(data)):
            if data[k].shape[0]<n:
                print('the size of data is not uniform')

        data = np.asarray(data).astype('float')
        lbound, ubound = np.min(data), np.max(data)
        if lbound == ubound:
            offset = lbound
            scalar = 1
            data = np.zeros(size=data.shape)
        else:
            offset = (lbound + ubound) / 2
            scalar = 1 / (ubound - lbound)
            data = (data - offset) * scalar
        # return normalized data
        return data
```

```
In [12]: def simpleMixer(S):
             nchannel = S.shape[0]
             # generate a random matrix
             A = np.random.uniform(size = (nchannel,nchannel))
             # generate mixed audio data
             X = A.dot(S)

             return X, A
```

```
In [13]: gtruthS = newaudionorm([data1,data2])
```

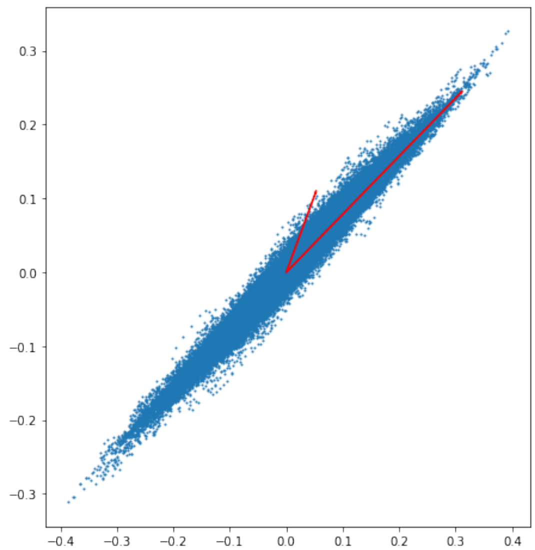
```
In [14]: X, gtruthA = simpleMixer(gtruthS)
```

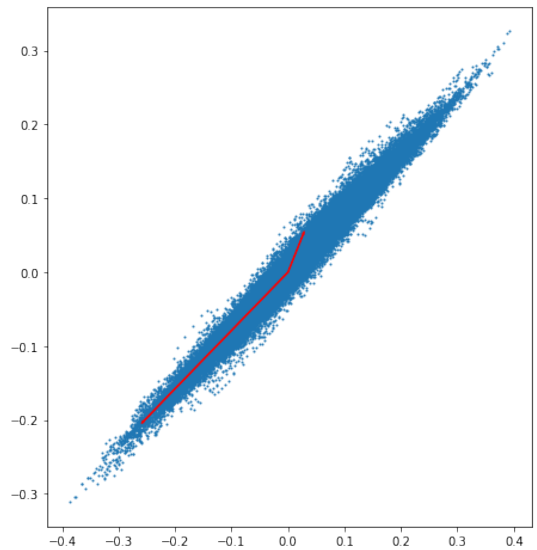
```
In [15]: wavfile.write('data/A5/EX3/mixedTrackA.wav', srate, X[0])
         wavfile.write('data/A5/EX3/mixedTrackB.wav', srate, X[1])
```

```
In [16]: A, S = bss(X)
```

```
In [17]: S = newaudionorm(S)
         wavfile.write('data/A5/EX3/separatedTrackA.wav', 22050, S[0])
         wavfile.write('data/A5/EX3/separatedTrackB.wav', 22050, S[1])
```

```
In [18]: compareMixingMatrix(X, gtruthA, A)
```





You can listen the files, then you will find that I have separated the two sounds. And I think more iterations will cause better result, but with more and more iterations, the value of A will change little.

1.4 Separate 3 sounds

```
In [53]: _, data3 = wavfile.read('data/bigbrain.wav')

In [54]: gtruthS1 = newaudionorm([data1,data2,data3])
         X1, gtruthA1 = simpleMixer(gtruthS1)

In [55]: wavfile.write('data/A5/three_sounds/mixedTrackA.wav', srate, X1[0])
         wavfile.write('data/A5/three_sounds/mixedTrackB.wav', srate, X1[1])
         wavfile.write('data/A5/three_sounds/mixedTrackC.wav', srate, X1[2])

In [56]: A1, S1 = bss(X1)

In [57]: S1 = newaudionorm(S1)
         wavfile.write('data/A5/three_sounds/separatedTrackA.wav', 22050, S1[0])
         wavfile.write('data/A5/three_sounds/separatedTrackB.wav', 22050, S1[1])
         wavfile.write('data/A5/three_sounds/separatedTrackC.wav', 22050, S1[2])
```

Use different functions to show three dimension.

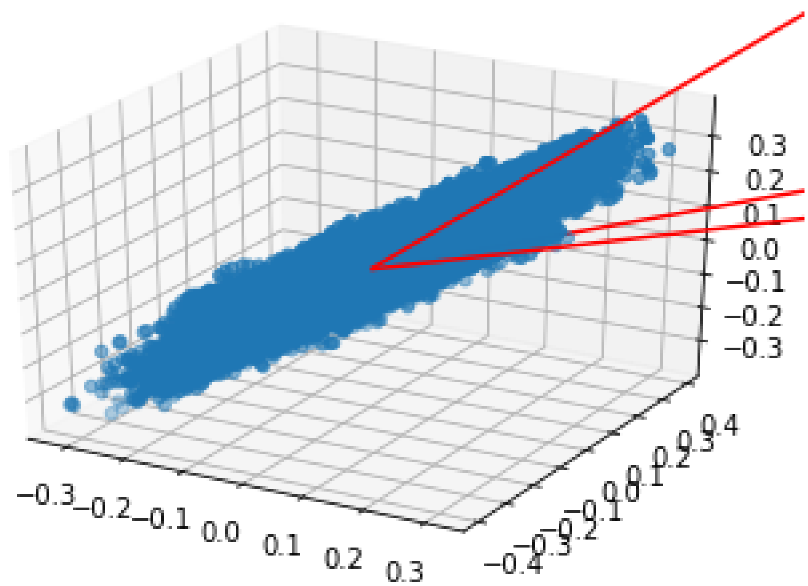
```
In [58]: from mpl_toolkits.mplot3d import Axes3D

In [59]: def drawDataWithMixingMatrix3D(data, mat):
             fig = plt.figure()
             ax = fig.gca(projection='3d')
             ax.scatter(data[0], data[1],data[2])
             lenAxis = np.sqrt(np.sum(np.square(mat), axis=0))
```

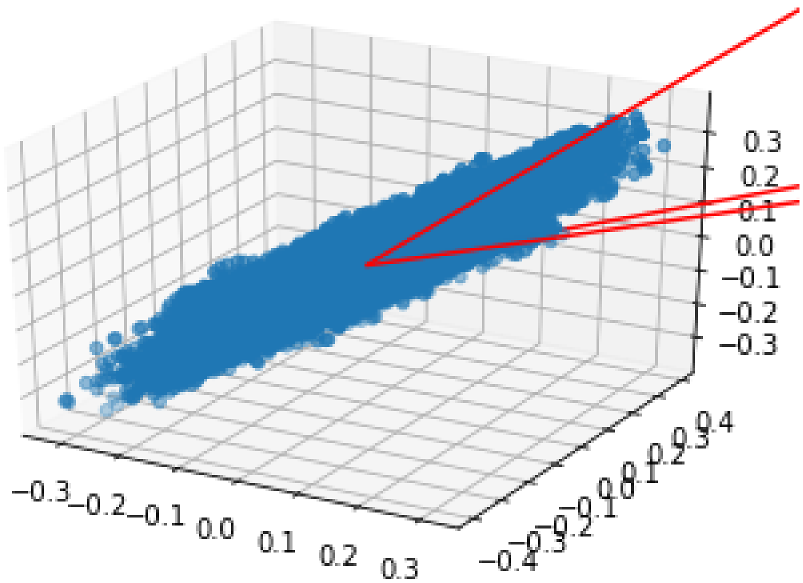
```
            scale = np.min(np.max(np.abs(data), axis=1) / lenAxis.T) *20
            ax.quiver3D(0, 0, 0, scale*mat[0,0], scale*mat[1,0], scale*mat[2,0], color = "r")
            ax.quiver3D(0, 0, 0, scale*mat[0,1], scale*mat[1,1], scale*mat[2,1], color = "r")
            ax.quiver3D(0, 0, 0, scale*mat[0,2], scale*mat[1,2], scale*mat[2,2], color = "r")
            plt.show()
```

```
In [60]: def compareMixingMatrix3D(data, matA, matB):
             drawDataWithMixingMatrix3D(data, matA)
             plt.figure()
             drawDataWithMixingMatrix3D(data, matB)
```

```
In [61]: compareMixingMatrix3D(X1, gtruthA1, A1)
```




```
<Figure size 432x288 with 0 Axes>
```



With same iterations and learning rate, the function can separate three sounds well.