6.1

In [ ]:

```
# imports
import os
from scipy.io.wavfile import write
from IPython.core.display import HTML, display
from __future__ import division
from matplotlib import style
style.use("ggplot")


import scipy as sp
import numpy as np
import pylab as plt
#import pandas as pd
import warnings

warnings.filterwarnings('ignore')
sp.random.seed(1)
```

In [ ]:

```
# play functions in iPython notebook
try:
    from IPython.display import Audio
    def wavPlayer(data, rate):
        display(Audio(data, rate=rate))
except ImportError:
    pass
```

In [ ]:

```
# Intializations
```

In [ ]:

```
# laod soundfiles.
s1 = np.fromfile('sound1.dat', dtype=float, sep='\n')
s2 = sp.fromfile('sound2.dat', dtype=float, sep='\n')

#create random noise
std = np.std(s1)
size = s1.shape[0]
s3 = np.random.normal(0,std,size)
s4 = np.random.laplace(0,1,size)
#stack
s = sp.stack((s1, s2, s3))
slap = sp.stack((s1,s2,s4))
```

In [ ]:

```
# random nxn mixing matrix
A = sp.random.randint(1, high=11, size=9)
A = A.reshape((3,3))
A = A + sp.eye(3)

# compute true W
W_true = sp.linalg.inv(A)

print('A:')
print(A)
print('W_true:')
print(W_true)
print(sp.linalg.det(W_true))

x = sp.dot(A, s)
sp.io.wavfile.write('mixed1' + '.wav', 8192, x[0,:])
```

In [ ]:

```
#remove structure
idx = sp.random.permutation(18000)
x_shuffled = x[: , idx]
```

In [ ]:

```
#center to zero mean
x_mean = sp.mean(x_shuffled, axis=1).reshape((x.shape[0], 1))
print('Mean of audio signals:')
print(x_mean.shape)

# center shuffled data
x_shuffled_centered = x_shuffled - x_mean

# center unshuffled data
x_centered = x - x_mean
print (x_centered.shape)
```

In [ ]:

```
#rand vals. for unmixing
W = sp.random.randint(1, high=2, size=9)
W = W.reshape((3,3)) * 0.1
W = W + sp.eye(3) * 0.1
```

In [ ]:

```
# define natural gradient with learning rate eta
```

In [ ]:

```
def ICA_natural_gradient(W, x):
    gradient = sp.concatenate((sp.dot(W, x), sp.dot(W, x)), axis=1)
    gradient = sp.concatenate((gradient,sp.dot(W,x)), axis=1)
    gradient = sp.special.expit(gradient) * 2
    gradient = sp.ones((gradient.shape)) - gradient

    gradient = (gradient * x)
    normalization = sp.dot(W.T, W)

    gradient = sp.dot(gradient, normalization)
    return gradient
```

In [ ]:

```
def ICA(W, x, n=0.0085, l=0.9):
    vals = []
    t = 1
    rate = n
    for _ in range(3):
            for i in range(x.shape[1]):
                # adaptive learning rate !? --> bad results and not wanted
                rate = rate*l

                # select random datapoint
                #idx = sp.random.randint(0, high=18000, size=1)
                #datapoint = x[:, idx].reshape((2,1))

                # choose next datapoint
                datapoint = x[:, i].reshape((3,1))

                gradient = 0
                gradient = ICA_natural_gradient(W, datapoint)
                W = W + rate * gradient

                if (t % 1000) == 0:
                    val = sp.sum((rate * gradient) ** 2)
                    vals.append(val)

                t = t + 1
    return W, vals

def unmix_sources(W, x):
    return sp.dot(W, x)
```

In [ ]:

```
# do things for normal dist
# for unshuffled data
W_un, vals_regular = ICA(sp.copy(W), x_centered)
s_un= unmix_sources(W_un, x_centered)

# for shuffled data
W_shuffled, vals_shuffled = ICA(sp.copy(W), x_shuffled_centered)
s_shuffled = unmix_sources(W_shuffled, x_shuffled_centered)



print('W_natural (W learned from unshuffled data:')
print(W_un)

print('W_natural_shuffled (W learned from shuffled data:')
print(W_shuffled)
```

In [ ]:

```
def plot_and_play_data(data, title, label1, label2, label3):
    sp.io.wavfile.write(label1 + '.wav', 8192, data[0, :])
    sp.io.wavfile.write(label2 + '.wav', 8192, data[1, :])
    sp.io.wavfile.write(label3 + '.wav', 8192, data[2, :])
    print(label1 + ':')
    wavPlayer(data[0, :], 8192)
    print(label2 + ':')
    wavPlayer(data[1, :], 8192)
    print(label3 + ':')
    wavPlayer(data[2, :], 8192)

    # plot data
    x_axis = sp.arange(data.shape[1])

    plt.figure()
    plt.plot(x_axis, data[1,:], 'b-', label=label2, alpha=0.5)
    plt.plot(x_axis, data[0,:], 'r-', label=label1, alpha=0.5)
    plt.plot(x_axis, data[2,:], 'g-', label=label3, alpha=0.5)
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), ncol=1)
    plt.grid(True)
    plt.xlabel('time')
    plt.ylabel('sound')
    plt.title(title)
    plt.show()
```

In [ ]:

```
# (i) Plot & Play the original sources
#plot_and_play_data(s, 'Original sounds (sources)', 'sound 1', 'sound 2','sound 3
```

In [ ]:

```
# (ii) Plot & Play the mixed sources before the data permutation
#plot_and_play_data(x, 'Mixes sounds - before permutation', 'observation 1', 'obs
```

In [ ]:

```
# (iii) Plot & Play the mixed sources after the data permuta#tion
#plot_and_play_data(x_shuffled, 'Mixes sounds - after permutation', 'observation
```

In [ ]:

```
# (iv) Plot & Play the recovered signals using the unpermuted data
plot_and_play_data(s_un, 'recovered sounds - natural gradient', 'source 1 (natura
```

In [ ]:

```
# do things for laplacian dist
# for unshuffled data
W_un_lap, vals_lap = ICA(sp.copy(W), x_centered)
s_un_lap= unmix_sources(W_un_lap, x_centered)

# for shuffled data
W_shuff_lap, vals_shuffled = ICA(sp.copy(W), x_shuffled_centered)
s_shuff_lap = unmix_sources(W_shuff_lap, x_shuffled_centered)



print('W_natural (W learned from unshuffled data:')
print(W_un_lap)

print('W_natural_shuffled (W learned from shuffled data:')
print(W_shuff_lap)
```

In [ ]:

```
# (iv) Plot & Play the recovered signals using the unpermuted data
plot_and_play_data(s_un_lap, 'recovered sounds - natural gradient', 'source 1 (na
```

In [ ]:

```
# (iv) Plot & Play the recovered signals using the permuted data
plot_and_play_data(s_shuff_lap, 'recovered sounds - natural gradient', 'source 1
```

In [ ]:

```
# 6.3
```

In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
import scipy.io
import numpy as np
from numpy.linalg import eigh
```

In [ ]:

```
DataSets=scipy.io.loadmat('distrib.mat')
Normal=DataSets['normal']
Uniform=DataSets['uniform']
Laplacian=DataSets['laplacian']
```

In [ ]:

```
(a) Apply the following mixing matrix A to the original data s:
```

In [ ]:

```
A=np.reshape([4,3,2,1],(2,2))
MixN=np.dot(A,Normal)
MixU=np.dot(A,Uniform)
MixL=np.dot(A,Laplacian)
```

In [ ]:

```
(b) Center the mixed data to zero mean.
```

In [ ]:

```
Center_N=(MixN.T-MixN.mean(axis=1)).T
Center_U=(MixU.T-MixU.mean(axis=1)).T
Center_L=(MixL.T-MixL.mean(axis=1)).T
```

In [ ]:

```
# (c) Decorrelate the data by applying principal component analysis (PCA) and prc
# the principal components (PCs).
```

In [ ]:

```
def PCA(data):
    Cov=np.cov(data)
    E,V=eigh(Cov)
    key = np.argsort(E)[::-1]
    E, V = E[key], V[:, key]
    NewData=np.dot(data.T,V)
    return E,V,NewData.T

E_N,V_N,NewData_N=PCA(Center_N)
E_U,V_U,NewData_U=PCA(Center_U)
E_L,V_L,NewData_L=PCA(Center_L)
```

In [ ]:

```
#(d) Scale the data to unit variance in each PC direction (now the data is whiten
```

In [ ]:

```
Scale_N=(NewData_N.T/np.std(NewData_N,axis=1)).T
Scale_U=(NewData_U.T/np.std(NewData_U,axis=1)).T
Scale_L=(NewData_L.T/np.std(NewData_L,axis=1)).T
```

In [ ]:

```
# (e) Rotate the data by different angles θ
```

In [ ]:

```
angelList=np.linspace(0,2*np.pi,100)
def Rotataion(data,angelList):
    for angel in angelList:
        r=np.reshape([np.cos(angel),-np.sin(angel),np.sin(angel),np.cos(angel)],(
        x=np.dot(r,data)
```

In [ ]:

```
#(f) Find the minimum and maximum kurtosis value for the first dimension and rota
# accordingly.
```

In [ ]:

```

```

In [ ]:

```
# Plot the original dataset (sources) and the mixed dataset after the steps (a),
# and (f) as a scatter plot and display the respective marginal histograms. For s
# the kurtosis value as a function of angle for each dimension.
```

In [ ]:

```
def dplot(data1,data2,data3,title):
    plt.figure(figsize=(15,5))
    axScatter=plt.subplot(131)
    divider = make_axes_locatable(axScatter)
    axHistx = divider.append_axes("top", 1.5, pad=0.1, sharex=axScatter)
    axHistx.set_title('Normal')
    axHisty = divider.append_axes("right", 1.5, pad=0.1, sharey=axScatter)
    plt.setp(axHistx.get_xticklabels() + axHisty.get_yticklabels(),visible=False)
    axScatter.scatter(data1[0], data1[1])
    binwidth = 0.25
    xymax = np.max([np.max(np.fabs(data1[0])), np.max(np.fabs(data1[1]))])
    lim = (int(xymax/binwidth) + 1)*binwidth
    bins = np.arange(-lim, lim + binwidth, binwidth)
    axHistx.hist(data1[0],bins=bins)
    axHisty.hist(data1[1],bins=bins, orientation='horizontal')

    axScatter=plt.subplot(132)
    divider = make_axes_locatable(axScatter)
    axHistx = divider.append_axes("top", 1.5, pad=0.1, sharex=axScatter)
    axHisty = divider.append_axes("right", 1.5, pad=0.1, sharey=axScatter)
    axHistx.set_title('Uniform')
    plt.setp(axHistx.get_xticklabels() + axHisty.get_yticklabels(),visible=False)
    axScatter.scatter(data2[0], data2[1])
    axScatter.set_xlabel(title,fontsize=14, color='red')
    binwidth = 0.25
    xymax = np.max([np.max(np.fabs(data2[0])), np.max(np.fabs(data2[1]))])
    lim = (int(xymax/binwidth) + 1)*binwidth
    bins = np.arange(-lim, lim + binwidth, binwidth)
    axHistx.hist(data2[0],bins=bins)
    axHisty.hist(data2[1],bins=bins, orientation='horizontal')

    axScatter=plt.subplot(133)
    divider = make_axes_locatable(axScatter)
    axHistx = divider.append_axes("top", 1.5, pad=0.1, sharex=axScatter)
    axHisty = divider.append_axes("right", 1.5, pad=0.1, sharey=axScatter)
    axHistx.set_title('Laplacian')
    plt.setp(axHistx.get_xticklabels() + axHisty.get_yticklabels(),visible=False)
    axScatter.scatter(data3[0], data3[1])
    binwidth = 0.5
    xymax = np.max([np.max(np.fabs(data3[0])), np.max(np.fabs(data3[1]))])
    lim = (int(xymax/binwidth) + 1)*binwidth
    bins = np.arange(-lim, lim + binwidth, binwidth)
    axHistx.hist(data3[0],bins=bins)
    axHisty.hist(data3[1],bins=bins, orientation='horizontal')
    plt.show()
```

In [ ]:

```
dplot(Normal,Uniform,Laplacian,'Original Data')
```

In [ ]:

```
dplot(MixN,MixU,MixL,'Mixed Data after a step')
```

In [ ]:

```
dplot(Center_N,Center_U,Center_L,'Center Data after b step')
```

In [ ]:

```
dplot(NewData_N,NewData_U,NewData_L,'Data after PCA after c step')
```

In [ ]:

```
dplot(Scale_N,Scale_U,Scale_L,'Scale Data after d step')
```