NEW

June 9, 2016

In [135]:
```
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
```

1 Initialization

In [159]:
```
#(a) Load the sound files. Each of the N = 2 sources is sampled at at 8192 Hz and contains p
s = np.array([np.loadtxt("sounds/sound1.dat"), np.loadtxt("sounds/sound2.dat")])
N = s.shape[0]
```

In [163]:
```
#(b) Create a random (& invertible) N ×N mixing matrix A and mix the sources: x(α) = As(α)
A = np.zeros((N, N))
while np.linalg.det(A) == 0:
    A = np.random.uniform(0.0, 1.0, (N, N))
print(A)
x = np.dot(A, s)
#print(x.shape)
#print(x)
```

```
[[ 0.86351992  0.4879736 ]
 [ 0.05267788  0.6253859 ]]
```

In [164]:
```
#(c) Remove the temporal structure by permuting the columns of the N × p matrix X randomly.
xs = np.random.permutation(x)
#print(xs)
```

In [167]:
```
#(d) Calculate the correlations between the sources and the mixtures: ρ(si,xj) = cov(si,xj) /
#p = np.cov(s, xs) / (np.var(s) * np.var(xs))
#p = [((np.cov(s[i], x[j]) / (np.std(s[i]) * np.std(x[j])))[0,1],i,j) for i in range(N) for j
#print(p)
def correlations(N, s, x):
    p = np.zeros((N, N))
    for i in range(N):
        for j in range(N):
            p[i, j] = (np.cov(s[i], x[j]) / (np.std(s[i]) * np.std(x[j])))[0,1]
    return p
print(correlations(N, s, x))
```

```
[[ 0.87074079  0.08516133]
 [ 0.49295234  0.99652948]]
```

In [168]:
```
#(e) Center the data to zero mean.
#print( np.mean(x, axis=1).shape )
x_mean = np.mean(xs, axis=1).reshape(-1,1)
xsc = xs - x_mean
#print(xsc.shape)
```

```
In [169]: #(f) Initialize the unmixing matrix W with random values.
          #s = W.x, s: Nxp, x: Nxp -> W: NxN
          W = np.random.uniform(0, 1, (N, N))
```

2 Optimization

```
In [170]: def fhat(y):
              return 1 / (1 + np.exp(-y))
          def fhatpp_fhatp(y):
              return 1 - 2*fhat(y)

In [171]: #(a) Compute the update matrix ΔW using the \regular" gradient.
          def dW(eta, W, x):
              #x0 x1
              #x0 x1
              xv = np.vstack( (x.T, x.T) )
              #x0 x0
              #x1 x1
              xh = np.hstack( (x.reshape(-1,1), x.reshape(-1,1)) )
              #W-1 + f(W . xh) * xv
              return eta * (np.linalg.pinv(W).T + np.multiply( fhatpp_fhatp(np.dot(W, xh)), xv))

In [172]: #(b) Compute the update matrix ΔW using the natural gradient as described in the lecture note
          def dWn(eta, W, x):
              return np.dot(dW(eta, W, x), np.dot(W.T, W))

In [173]: #(c) Choose a suitable learning rate η and apply both versions to the data to unmix the sourc
          eta = .2
          epsilon = .001
          Wa = W.copy()
          Wb = W.copy()
          AConverged = False
          BConverged = False
          print("Initial W:\n"+str(W))
          for t in range(1,18000):
              xa = xsc[:,np.random.randint(xsc.shape[1])]

              regular = dW(eta/float(t), Wa, xa)
              natural = dWn(eta/float(t), Wb, xa)
              if not AConverged:
                  if regular.any() > epsilon:
                      Wa = Wa + regular
                  else:
                      AConverged = True
                      print("(a) converged at: "+str(t))
              else:
                  pass

              if not BConverged:
                  if natural.any() > epsilon:
                      Wb = Wb + natural
                  else:
                      BConverged = True
                      print("(b) converged at: "+str(t))
```

```
            else:
                pass
        print("Wa:\n"+str(Wa))
        print("Wb:\n"+str(Wb))
```

```
Initial W:
[[ 0.24800548  0.57166945]
 [ 0.95237266  0.16830033]]
Wa:
[[-0.03450289  1.5594804 ]
 [ 1.92166763 -0.35650371]]
Wb:
[[-0.73894737  1.79073814]
 [ 2.68314697 -0.32222722]]
```

3 Results

In [174]:
```
#(a) Plot & Play
#(i) the original sounds (e.g. use scipy.io.wavfile to save playable files),
#(ii) the mixed sources (before and after the data permutation),
#and the recovered signals (estimated sources) ^s = Wx using the unpermuted data.

#retrieve shats
shata = np.dot(Wa, x)
shata_decentered = shata + x_is_mean
shatb = np.dot(Wb, x)
shatb_decentered = shatb + x_is_mean

#(i)
fig = plt.figure(figsize=(20,30))
limits = [-8,8]

ax1 = plt.subplot(821)
ax1.plot(range(s.shape[1]), s[0])
plt.title('Source 1')
plt.ylabel('Original Sources')
ax1.set_ylim(limits)

ax2 = plt.subplot(822)
ax2.plot(range(s.shape[1]), s[1])
plt.title('Source 2')
ax2.set_ylim(limits)


#(ii)
ax1 = plt.subplot(823)
ax1.plot(range(x.shape[1]), x[0])
ax1.set_ylim(limits)
plt.ylabel('Mixed Sources')

ax2 = plt.subplot(824)
ax2.plot(range(x.shape[1]), x[1])
ax2.set_ylim(limits)
```

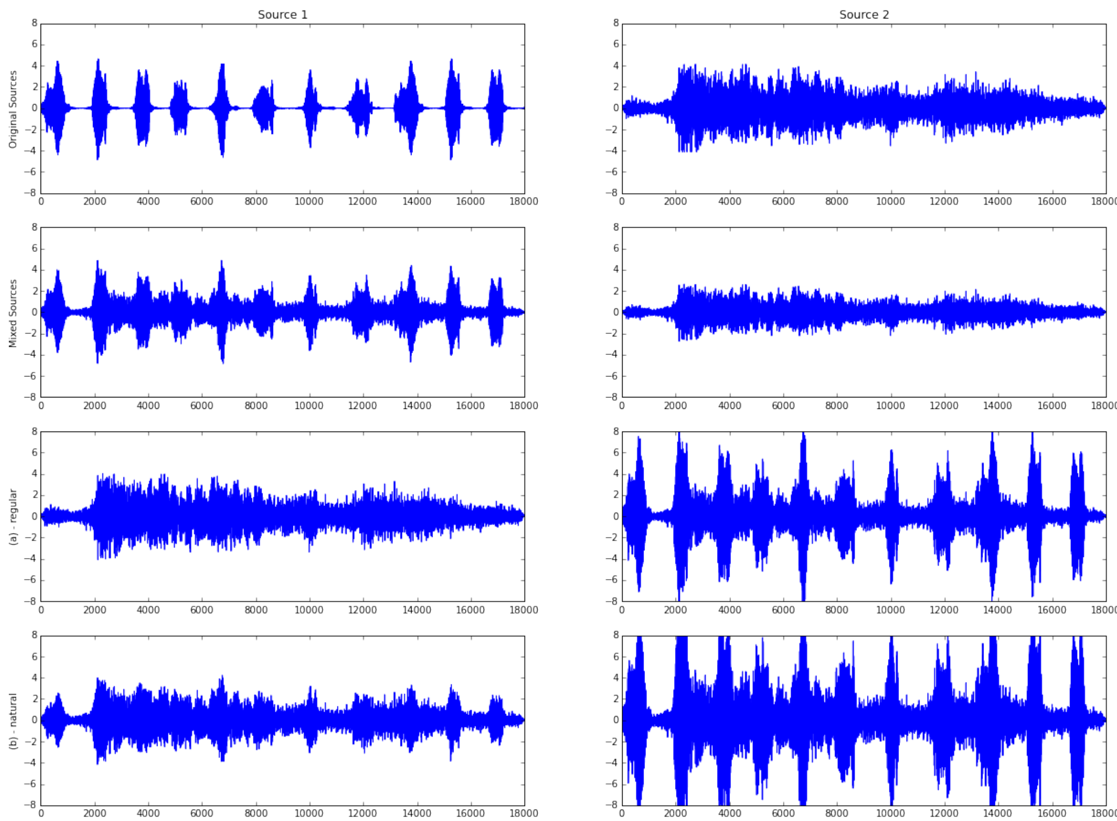
```
#(iii)
ax3m = plt.subplot(825)
ax3m.plot(range(shata_decentered.shape[1]), shata_decentered[0])
ax3m.set_ylim(limits)
plt.ylabel('(a) - regular')

ax4m = plt.subplot(826)
ax4m.plot(range(shata_decentered.shape[1]), shata_decentered[1])
ax4m.set_ylim(limits)

ax5 = plt.subplot(827)
ax5.plot(range(shatb_decentered.shape[1]), shatb_decentered[0])
ax5.set_ylim(limits)
plt.ylabel('(b) - natural')

ax6 = plt.subplot(828)
ax6.plot(range(shatb_decentered.shape[1]), shatb_decentered[1])
ax6.set_ylim(limits)


plt.show()
```




```
In [175]: #(b) Calculate the correlations (as above) between the true sources and the estimations.
          mixings = (correlations(N, s, x))
          natural_mean = (correlations(N, shatb_decentered, x))
```

```python
    regular_mean = (correlations(N, shata_decentered, x))

    print("\nMixes")
    print(mixings)
    print("\nRegular")
    print(regular_mean)
    print("\nNatural")
    print(natural_mean)
```

```
Mixes
[[ 0.87074079  0.08516133]
 [ 0.49295234  0.99652948]]

Regular
[[ 0.53966194  0.99962198]
 [ 0.99471409  0.47597849]]

Natural
[[-0.10610345  0.76117059]
 [ 0.99791481  0.50901291]]
```

In [174]:
```python
#(c) For every 1000th update, plot ||ΔW||2F :=  Ni=1,j=1(Δwi,j)2 to compare the convergence
```

In [175]:
```python
#(d) Plot the density of the mixed, unmixed, and true signals & interpret your results.
```