Homework 10: ECE 8843

Danial Zuberi

December 1, 2018

Problem 1

This last week, we started with Optimization using Gradient descent. This ultimately becomes an iterative method to solve an optimization program, $\text{minimize}_{x \varepsilon \mathbb{R}^N} f(x)$. This turned into an iterative method, where each becomes the previous x minus some stepsize times the gradient of f at the previous x. This is especially useful when f is convex, since we can just ride the gradient to the solution. Step size selection is somewhat difficult, but a good idea is the backtracking line search, which involves scaling an initial step size until it is good enough. Another complication is adding constraints. How we handle this is by using the closest point operator after each step to project the solution into the space we are working in.

From here we moved to PCA and Gaussian Mixture Models. PCA is how we learn what subspace vector comes from given samples. This also gives us a way to create structured matrix factorizations, which has applications in facial recognition.

Finally, we moved on to GMM. Mixture models mean that the pdf for a random variable is a weighted sum of different component pdfs. Gaussian mixture models just mean that these pdfs are gaussian. The EM algorithm defines a way to solve for the component pdfs given a data set.

Problem 2

Part A

$$\text{maximize}_{S \varepsilon S^{6++}} \log\det S - \text{trace}(S\Sigma) \tag{1}$$

subject to S[1,4:6], S[2,4:6], S[3,5:6], S[4:6,1], S[4:6,2], S[5:6,3] all zero

Part B

Results under code in comment format:

```
import numpy as np
import scipy.io as scio
```

```
def inv(A):
	return np.linalg.inv(A)

def get_sigma():
	mat = scio.loadmat('hw10p2data.mat')
	X = mat.get('X')
	rows = np.shape(X)[0]
	col = np.shape(X)[1]
	sig = np.zeros((rows,rows))
	for n in range(0,col):
		sig = np.add( sig , np.outer(X[:,n],X[:,n]) )
	sig /= col
	return sig

sigma = get_sigma()

def Ps(A): #set the appropriate values to zero
	A[0, 3:6] = 0
	A[1, 3:6] = 0
	A[2, 4:6] = 0
	A[3:6, 0] = 0
	A[3:6, 1] = 0
	A[4:6, 2] = 0
	return A

def grad(S): #Find the gradient for a given S
	return sigma - inv(S)


def in_S(S): #check if matrix satisfies our condition,
#give 1e-12 tolerance for inversion errors
	if np.any(S[0,3:6] > 1e-10):
		return 0
	elif np.any(S[1,3:6] > 1e-12):
		return 0
	elif np.any(S[2,4:6] > 1e-12):
		return 0
	elif np.any(S[3:6,0] > 1e-12):
		return 0
	elif np.any(S[3:6,1] > 1e-12):
		return 0
	elif np.any(S[4:6,2] > 1e-12):
		return 0
	else:
		return 1
```

```
def alpha_search(S,g): #search for a high alpha that keeps the
    ↪ result in our set,
#reduce by 90% per iteration
       alpha = 1.0
       while(in_S(S - alpha*g) == 0):
              alpha *= 0.9
       return alpha

S= np.identity(6)
for k in range(0,int(1e4)):
       g = Ps(grad(S))
       a = alpha_search(S, g)
       S = S - a*g
       S = Ps(S)
R = inv(S)
np.set_printoptions(suppress=True)
print('Result:')
print(R)
print('Sample Cov:')
print(sigma)
np.savetxt('p2_R.txt',inv(S),delimiter=' ',newline='\n')
np.savetxt('p2_invR.txt',S,delimiter=' ',newline='\n')

#Result:
#[[ 0.16251014 -0.01555197  0.00967764 -0.00177352 -0.00010501  0.0001178 ]
# [-0.01555197  0.13665699  0.00023067 -0.00004227 -0.0000025   0.00000281]
# [ 0.00967764  0.00023067  0.12290445 -0.02252338 -0.00133363  0.00149608]
# [-0.00177352 -0.00004227 -0.02252338  0.14042444  0.00831469 -0.00932746]
# [-0.00010501 -0.0000025  -0.00133363  0.00831469  0.13024438  0.01727435]
# [ 0.0001178   0.00000281  0.00149608 -0.00932746  0.01727435  0.12111833]]
#Sample Cov:
#[[ 0.16251014 -0.01555197  0.00967764  0.00416033  0.00268576  0.00845445]
# [-0.01555197  0.13665699  0.00023067 -0.0084813  -0.00617924  0.00694164]
# [ 0.00967764  0.00023067  0.12290445 -0.02252338 -0.00195947  0.00093073]
# [ 0.00416033 -0.0084813  -0.02252338  0.14042444  0.00831469 -0.00932746]
# [ 0.00268576 -0.00617924 -0.00195947  0.00831469  0.13024438  0.01727435]
```

The sample covariance and output look quite similar. The biggest differences are in the entries of the conditionally independent entries.

Problem 3

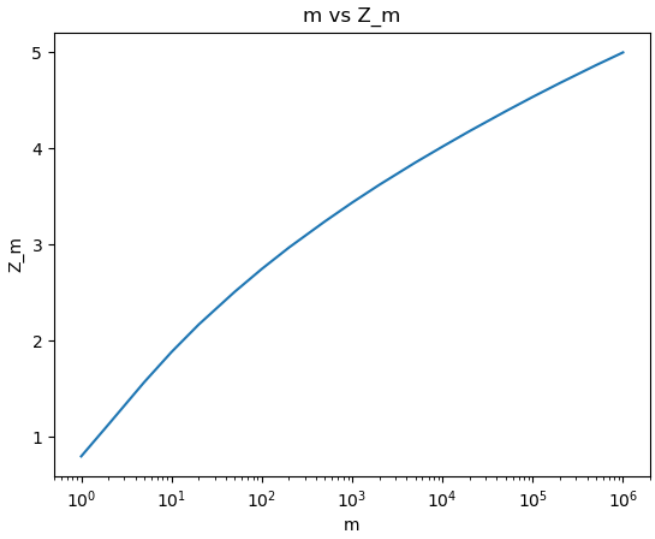
Code to generate the Z_m (separate code was used to plot this output)

```
import numpy as np
import scipy.io as scio

def find_max(M): #monte carlo simulation
	N = int(1e5)
	avg = 0.0;
	for i in range(1, N):
		x = np.random.randn(M,1)
		x = np.absolute(x)
		max_val = np.amax(x)
		avg += max_val
	avg = avg / float(N)
	return avg


ms = [1];
m = 1
i = 1
while m < 1e6: #generate the list of m to test
	type = i%3;
	if type == 1:
		m *=2
	if type == 2:
		m*=2.5
	if type == 0:
		m*=2
	i += 1
	ms.append(m)
ms = map(int, ms)
Zm = []
for m in ms:
	Zm.append(find_max(m))
print(Zm)
mZm = np.vstack([ms,Zm])
np.savetxt('Zm.csv',mZm,delimiter=',')
```

Plot:



Problem 4

Part A

```
import numpy as np
import scipy.stats as stat
import matplotlib.pyplot as plt

def gen_y(p):
	x = np.random.uniform(0,1)
	if x < p:
		return 0
	return 1

def gen_x(y):
	x = np.random.randn();
	if y == 0:
		x = -1 + 2*x
	else:
		x = 1 + 2*x
	return(x)

def R_h(theta,p):
	R = np.zeros(np.shape(theta))
```

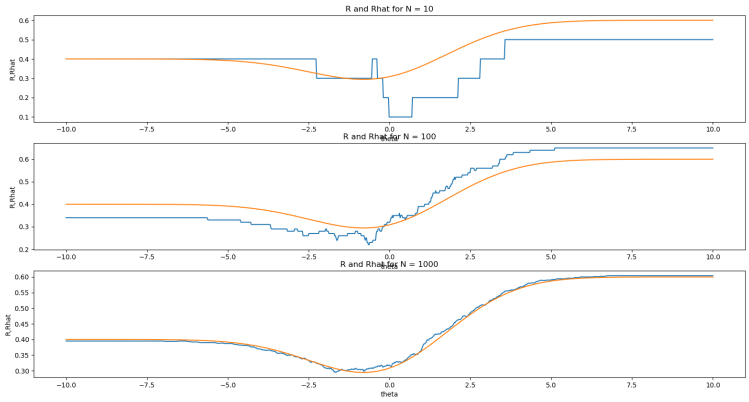
```
        for idx,t in enumerate(theta):
                R[idx] = (1 - p)*stat.norm.cdf((t - 1) / 2) + p*(1-
                    ↪ stat.norm.cdf((t + 1) / 2))
        return R

def find_nearest(array,value):
        idx = (np.abs(array - value)).argmin()
        return idx

N = 1e3
Ns = [10,100,1000]
R_minus_Rhat_08 = np.zeros(np.shape(Ns))
R_minus_Rhat_max = np.zeros(np.shape(Ns))
for idx,N in enumerate(Ns):
        plt.subplot(3,1,idx+1)
        theta = np.linspace(-10,10,int(1e3))
        loss = np.zeros(np.shape(theta))
        for n in range(1,int(N)):
                y = gen_y(0.4)
                x = gen_x(y)
                h_theta = np.zeros(np.shape(theta))
                l = np.zeros(np.shape(theta))
                for index,t in enumerate(theta):
                        if x < t:
                                h_theta[index] = 0
                        else:
                                h_theta[index] = 1
                for index,h in enumerate(h_theta):
                        if h == y:
                                l[index] = 0
                        else:
                                l[index] = 1
                loss = np.add(loss,l)
        loss /= N
        Rhat = loss
        R = R_h(theta,0.4)
        plt.plot(theta,Rhat)
        plt.plot(theta,R)
        plt.title("R and Rhat for N = " + str(int(N)))
        plt.xlabel("theta")
        plt.ylabel("R,Rhat")
        i = find_nearest(theta,0.8)
        R_minus_Rhat = np.abs(np.subtract(R,Rhat))
        R_minus_Rhat_08[idx] = R_minus_Rhat[i]
        R_minus_Rhat_max[idx] = np.amax(R_minus_Rhat)
print(R_minus_Rhat_08)
```

```
print(R_minus_Rhat_max)
print(np.amin(R))
plt.show()
```

Plot:



Part B

Code for this part and the following two parts:

```
import numpy as np
import scipy.stats as stat
#import matplotlib.pyplot as plt

def gen_y(p):
        x = np.random.uniform(0,1)
        if x < p:
                return 0
        return 1

def gen_x(y):
        x = np.random.randn();
        if y == 0:
                x = -1 + 2*x
        else:
                x = 1 + 2*x
        return(x)

def R_h(theta,p):
        R = np.zeros(np.shape(theta))
        for idx,t in enumerate(theta):
```

```
                R[idx] = (1 - p)*stat.norm.cdf((t - 1) / 2) + p*(1-
                    ↪ stat.norm.cdf((t + 1) / 2))
        return R

def find_nearest(array,value):
        idx = (np.abs(array - value)).argmin()
        return idx

N = 1e3
Ns = [10,100,1000]
R_minus_Rhat_08 = np.zeros(np.shape(Ns))
R_minus_Rhat_max = np.zeros(np.shape(Ns))
true_risk = np.zeros(np.shape(Ns))
T = int(1e4)
for trial in range(0,T):
        for idx,N in enumerate(Ns):
                #plt.subplot(3,1,idx+1)
                theta = np.linspace(-10,10,int(1e3))
                loss = np.zeros(np.shape(theta))
                for n in range(1,int(N)):
                        y = gen_y(0.4)
                        x = gen_x(y)
                        h_theta = np.zeros(np.shape(theta))
                        l = np.zeros(np.shape(theta))
                        for index,t in enumerate(theta):
                                if x < t:
                                        h_theta[index] = 0
                                else:
                                        h_theta[index] = 1
                        for index,h in enumerate(h_theta):
                                if h == y:
                                        l[index] = 0
                                else:
                                        l[index] = 1
                        loss = np.add(loss,l)
                loss /= N
                Rhat = loss
                R = R_h(theta,0.4)
                #plt.plot(theta,Rhat)
                #plt.plot(theta,R)
                #plt.title("R and Rhat for N = " + str(int(N)))
                #plt.xlabel("theta")
                #plt.ylabel("R,Rhat")
                i = find_nearest(theta,0.8)
                R_minus_Rhat = np.abs(np.subtract(R,Rhat))
                R_minus_Rhat_08[idx] += R_minus_Rhat[i]
```

```
                R_minus_Rhat_max[idx] += np.amax(R_minus_Rhat)
                minimizer_idx = Rhat.argmin()
                true_risk[idx] += R[minimizer_idx]
print(R_minus_Rhat_08/T)
print(R_minus_Rhat_max/T)
print(true_risk/T)
results = np.vstack([R_minus_Rhat_08/T,R_minus_Rhat_max/T,
    ↪ true_risk/T])
np.savetxt('p4_results.csv',results,delimiter=',')
```

N	10	100	1000
$\mathrm{E}[\lvert \mathrm{R}(\mathrm{h}_{0.8}) - \hat{\mathrm{R}}_{\mathrm{N}}(\mathrm{h}_{0.8})\rvert]$	.121	.037	.012

Part C

N	10	100	1000
$\mathrm{E}[\max_{\mathrm{h}_\vartheta \varepsilon \mathrm{H}} \lvert \mathrm{R}(\mathrm{h}_\vartheta) - \hat{\mathrm{R}}_{\mathrm{N}}(\mathrm{h}_\vartheta)\rvert]$	.268	.092	.029

Part D

N	10	100	1000
$\mathrm{E}[\mathrm{R}(\hat{\mathrm{h}}_{\mathrm{N}})]$	.340	.307	.297

Problem 5

```
import numpy as np
import scipy.io as scio
import matplotlib.pyplot as plt
from scipy.stats import norm

def Norm(x,mu,R):
       x.shape = [2,1]
       prob = 1 / (2*np.pi*np.linalg.det(R)) * np.exp(- 0.5 * np.
           ↪ matmul(np.transpose(x - mu),np.matmul(np.linalg.inv
           ↪ (R),(x-mu)) ))
       return prob

def contour(X,Y,mu,R):
       Zmap = np.zeros(np.shape(X))
       for i in range(0,np.shape(Zmap)[0]):
              for j in range(0,np.shape(Zmap)[1]):
                     this_point = np.array([X[i,j],Y[i,j]])
                     this_point.shape = [2,1]
                     Zmap[i,j] = Norm(this_point,mu,R)
```

```
        return Zmap
mat = scio.loadmat('hw10p5data.mat')
X = mat.get('X')
x_raw = X[0,:]
y_raw = X[1,:]
xn = []
for i in range(0,np.shape(X)[1]):
        xn.append(X[:,i])
        xn[i].shape = [2,1]
plt.scatter(x_raw,y_raw)
min_x = np.amin(x_raw)
max_x = np.amax(x_raw)
min_y = np.amin(y_raw)
max_y = np.amax(y_raw)
x = np.linspace(min_x*1.1,max_x*1.1,100)
y = np.linspace(min_y*1.1,max_y*1.1,100)
X,Y = np.meshgrid(x,y)
mus = [np.array([-0.9,0.1]),np.array([-0.5,0.07]),np.array
    ↪ ([0.02,0]),np.array([0.6,-0.03]),np.array([0.92,-0.06])]
for i in range(0,len(mus)):
        mus[i].shape = [2,1]
I = np.identity(2)
sigmas = [0.05*I,0.2*I,0.3*I,0.1*I,0.1*I]
betas = [0.1,0.1,0.1,0.1,0.1]
K = 200
Q = len(mus)
N = len(xn)

for k in range(0,K):
        gamma = np.zeros([N,Q])
        for n in range(0,N):
                for q in range(0,Q):
                        denom = 0
                        for q_ in range(0,Q):
                                denom += betas[q_] * Norm(xn[n],mus[
                                    ↪ q_],sigmas[q_])
                        gamma[n,q] = betas[q]*Norm(xn[n],mus[q],
                            ↪ sigmas[q]) / denom
        sum_gam = np.sum(gamma,0)
        betas = sum_gam / N
        temp_mus = []
        for q in range(0,Q):
                mu_q = 0
                for n in range(0,N):
                        mu_q += gamma[n,q]*xn[n]
                temp_mus.append(mu_q / sum_gam[q])
```

```
        mus = temp_mus
        temp_sigs = []
        for q in range(0,Q):
                numer = 0
                for n in range(0,N):
                        numer += gamma[n,q]*np.matmul((xn[n]- mus[q
                            ↪ ]),np.transpose(xn[n] - mus[q]))
                temp_sigs.append(numer / sum_gam[q])
        sigmas = temp_sigs

for q in range(0,Q):
        Zmap = contour(X,Y,mus[q],sigmas[q])
        plt.contour(X,Y,Zmap)
plt.title('EM Algoritm for ' + str(K) + ' iterations')
plt.xlabel('x[1]')
plt.ylabel('x[2]')
plt.show()
```