Syed Ali

COSC 750 Assignment 3

Problem 1 a), c)

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import random
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import multivariate_normal
from numpy.linalg import det, inv
from numpy.random import seed, randint, random
from sklearn import datasets
```

Our 2 dimensional will be over variables x and y

In [2]:

```
N = 1000
X = np.linspace(-4, 4, N)
Y = np.linspace(-4, 4, N)
X, Y = np.meshgrid(X, Y)
```

Mean Vector and Covariance Matrix

In [3]:

```
mu = np.array([1 , 2])
sigma = np.array([[4, 4], [4 , 9]])
```

Pack X and Y into a single 3-dimensional array

In [4]:

```
pos = np.empty(X.shape + (2,))
pos[:, :, 0] = X
pos[:, :, 1] = Y

def multivariate_gaussian(pos, mu , sigma):
    """Return the multivariate Gaussian distribution on array pos. pos is an array cons
    x_1, x_2,x_3,....,x_k into its _last_dimension."""

    n = mu.shape[0]
    sigma_det = np.linalg.det(sigma)
    sigma_inv = np.linalg.inv(sigma)
    N = np.sqrt((2*np.pi)**n * sigma_det)
# This einsum call calculates (x-mu)T.sigma-1.(x-mu) in a vectorized way across all the
    fac = np.einsum('...k,kl,...l->...', pos-mu, sigma_inv,pos-mu)
    return np.exp(-fac / 2) / N
# The distribution on the variables X,Y packed into pos.
```

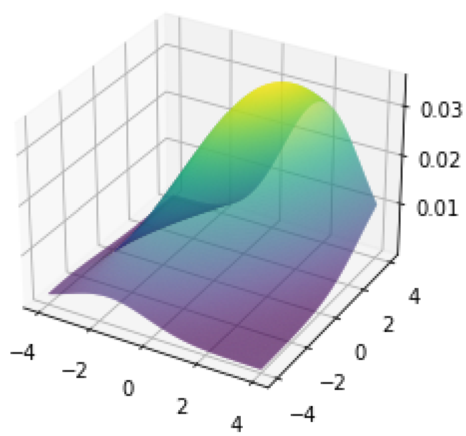
```
Z = multivariate_gaussian(pos, mu, sigma)
print(Z.shape)
```

```
(1000, 1000)
```

Create a surface plot and projected filled contour plot under it

In [5]:
```
fig = plt.figure()
ax = fig.gca(projection = '3d')
ax.plot_surface(X, Y, Z, rstride = 3,
                cstride = 3, linewidth = 1,
                antialiased = True, cmap = cm.viridis)
```

Out[5]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1d3b8434730>



Adjust the limits, ticks, and view angle

In [6]:
```
ax.set_zlim(-0.15, 0.2)
ax.set_zticks(np.linspace(0, 0.2, 5))
ax.view_init(27, -21)

plt.show()
```

Problem 1 b), c) with estimated mean and covariance

In [7]:
```
N = 1000
x = np.linspace(-4, 4, N)
y = np.linspace(-4, 4, N)
X, Y = np.meshgrid(x, y)
pos = np.dstack((X, Y))
mu = np.array([1, 1])
sigma = np.array([[.15, .25],[.25, .5]])
rv = multivariate_normal(mu, sigma, N)
Z = rv.pdf(pos)
print(Z)
```

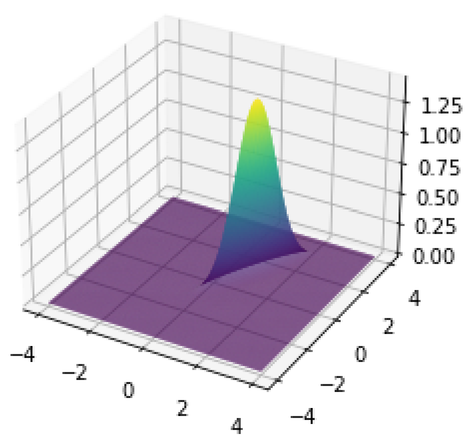
```
[[1.02139291e-065 2.27205687e-065 5.04117224e-065 ... 1.19085890e-272
```

```
 2.05308971e-273 3.53054317e-274]
[7.41160711e-066 1.65080493e-065 3.66745764e-065 ... 3.10398223e-272
 5.35826090e-273 9.22602157e-274]
[5.37400087e-066 1.19850020e-065 2.66602655e-065 ... 8.08432825e-272
 1.39735122e-272 2.40909097e-273]
...
[0.00000000e+000 0.00000000e+000 0.00000000e+000 ... 8.93714317e-024
 5.53459319e-024 3.41868174e-024]
[0.00000000e+000 0.00000000e+000 0.00000000e+000 ... 1.08156846e-023
 6.70653217e-024 4.14789726e-024]
[0.00000000e+000 0.00000000e+000 0.00000000e+000 ... 1.30790170e-023
 8.12037552e-024 5.02878539e-024]]
```

Create a surface plot and projected filled contour plot under it

In [8]:
```python
fig = plt.figure()
ax = fig.gca(projection = '3d')
ax.plot_surface(X, Y, Z, rstride = 3,
                cstride = 3, linewidth = 1,
                antialiased = True, cmap = cm.viridis)
```

Out[8]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1d3ba35fd90>
```



Adjust the limits, ticks, and view angle

In [9]:
```python
ax.set_zlim(-0.15, 0.2)
ax.set_zticks(np.linspace(0, 0.2, 5))
ax.view_init(27, -21)

plt.show()
```

Problem 1 d) Given Parameters

In [10]:
```python
mu = np.array([5 , -5,  6])
sigma = np.array([[5, 2, -1],
                  [2 , 5, 0],
                  [-1, 0, 4]])
```

Let us evaluate the integral using numerical and monte-carlo method

In [11]:
```
from scipy.stats.mvn import mvnun

cum = mvnun(np.array([-np.inf, -np.inf, -np.inf]), np.array([np.inf, np.inf, np.inf]),
print(cum[0])

def multpdf(X, mu, sigma, invsig):
    d = X.shape[0]
    den = (2*np.pi)**(d/2.0)*np.sqrt(det(sigma))
    p = np.exp(-0.5*np.dot((X-mu).T, np.dot(invsig, (X-mu))))/den
    return(p)

count = 0
nrandV = 100000
invsig = sigma
for i in range(nrandV):
    x1 = randint(-10, 10)
    x2 = randint(-10, 10)
    x3 = randint(-10, 10)
    p1 = random() * 0.0635
    X = np.array([x1, x2, x3])
    p = multpdf(X.T, mu.T, sigma, invsig)
    if(p1 < p):
        count += 1

vol = (count/nrandV) * 0.0635 * 20 * 20 * 20
print(vol)
```

```
1.0
0.01524
```

Problem 1 d) Estimated Parameters

In [12]:
```
mu = np.array([0 , 0,  0])
sigma = np.array([[1, 0, 0],
                  [0 , 1, 0],
                  [0, 0, 1]])
```

Let us evaluate the integral using numerical and monte-carlo method

In [13]:
```
from scipy.stats.mvn import mvnun

cum = mvnun(np.array([-np.inf, -np.inf, -np.inf]), np.array([np.inf, np.inf, np.inf]),
print(cum[0])

def multpdf(X, mu, sigma, invsig):
    d = X.shape[0]
    den = (2*np.pi)**(d/2.0)*np.sqrt(det(sigma))
    p = np.exp(-0.5*np.dot((X-mu).T, np.dot(invsig, (X-mu))))/den
    return(p)

count = 0
```

```
nrandV = 100000
invsig = sigma
for i in range(nrandV):
    x1 = randint(-10, 10)
    x2 = randint(-10, 10)
    x3 = randint(-10, 10)
    p1 = random() * 0.0635
    X = np.array([x1, x2, x3])
    p = multpdf(X.T, mu.T, sigma, invsig)
    if(p1 < p):
        count += 1

vol = (count/nrandV) * 0.0635 * 20 * 20 * 20
print(vol)
```

```
1.0
0.889
```

Problem 2 Iris data classification using Baysian decision

Import some data to play with

In [14]:

```
iris = datasets.load_iris()
X = iris.data
y = iris.target
print(X.shape)
print(y.shape)
```

```
(150, 4)
(150,)
```

Compute the mean value mue = E(X) for all three classes 0, 1, 2 as 2D-array, each column has mean vector class

In [15]:

```
mu = np.zeros(shape=(3,4), dtype = 'float')
mu1 = np.zeros(4, dtype ='float')
mu2 = np.zeros(4, dtype ='float')
mu3 = np.zeros(4, dtype ='float')
XX1 = X[0:25,]
XX2 = X[25:50,]
XX3 = X[50:75,]
for i in range(4):
    mu1[i] = np.mean(XX1[:,i])
    mu2[i] = np.mean(XX2[:,i])
    mu3[i] = np.mean(XX3[:,i])
mu[0,] = mu1
mu[1,] = mu2
mu[2,] = mu3

print(mu)
```

```
[[5.028 3.48  1.46  0.248]
 [4.984 3.376 1.464 0.244]
 [6.012 2.776 4.312 1.344]]
```

In [16]:

```
Bsig = np.zeros(shape=(3,4,4), dtype = 'float')
Bsig1 = np.cov(XX1.T)
Bsig2 = np.cov(XX2.T)
Bsig3 = np.cov(XX3.T)
Bsig[0,] = Bsig1
Bsig[1,] = Bsig2
Bsig[2,] = Bsig3
print(Bsig.shape)
print(Bsig[0])
print('\n', Bsig[1])
print('\n', Bsig[2])
iBsig = np.zeros(shape=(3,4,4), dtype = 'float')
for i in range(3):
    iBsig[i] = np.linalg.inv(Bsig[i])
print(iBsig[0])
print('\n', iBsig[1])
print('\n', iBsig[2])
```

```
(3, 4, 4)
[[0.16043333 0.11808333 0.02408333 0.01943333]
 [0.11808333 0.13583333 0.00625    0.02225   ]
 [0.02408333 0.00625    0.03916667 0.00658333]
 [0.01943333 0.02225    0.00658333 0.01093333]]

 [[ 0.09223333  0.0821      0.0094      0.00156667]
 [ 0.0821      0.1519      0.01785    -0.00348333]
 [ 0.0094      0.01785     0.0224      0.00581667]
 [ 0.00156667 -0.00348333  0.00581667  0.01173333]]

 [[0.30026667 0.10946667 0.18651667 0.05195    ]
 [0.10946667 0.1244     0.08863333 0.04651667]
 [0.18651667 0.08863333 0.19693333 0.06403333]
 [0.05195    0.04651667 0.06403333 0.04256667]]
[[ 20.91511858 -19.02488906 -11.21928438   8.29697556]
 [-19.02488906  28.52858243  12.48404679 -31.7588954 ]
 [-11.21928438  12.48404679  34.88789326 -26.47139439]
 [  8.29697556 -31.7588954  -26.47139439 157.28666562]]

 [[ 21.46802823 -12.09197012   2.6437278   -7.76687132]
 [-12.09197012  14.43115583  -9.13295941  10.42636306]
 [  2.6437278   -9.13295941  59.23189642 -32.42788463]
 [ -7.76687132  10.42636306 -32.42788463 105.43540841]]

 [[  9.52562884  -4.4293071   -9.43896043   7.41397689]
 [ -4.4293071   16.32356718   1.74370506 -15.05569328]
 [ -9.43896043   1.74370506  19.78327275 -20.14595735]
 [  7.41397689 -15.05569328 -20.14595735  61.20275728]]
```

In [17]:

```
dBsig = np.zeros(3, dtype = 'float')
rdBsig = np.zeros(3, dtype = 'float')
for i in range (3):
    dBsig[i] = np.linalg.det(Bsig[i])
    rdBsig[i] = np.sqrt(dBsig[i])
print(dBsig)
print(rdBsig)
```

```
[1.63966346e-06 1.39977535e-06 3.15184961e-05]
```

```
[0.00128049 0.00118312 0.00561413]
```

compute probability density of a given vector for p(x,k), where x is input vector and k is class number 0:2

In [18]:
```
def pdensity(x, m, s, rd):
    den = (2*np.pi)**(x.shape[0]/2) * rd
    return(np.exp(-0.5*np.dot((x-m), np.dot(s, x-m)))/den)
```

apriors for all three classes are equal = 1/3 and denominirs are equal for all the three classes the Posterior density for each class multiplied by a constant term in the decision can be from probability density

Let us apply for each vector and find max aposterior value for each vector from the sample

In [19]:
```
px = np.zeros(3,dtype = 'float')
mcl = 0
for i in range(25):
    x = X[i,]
    for j in range(3):
        px[j] = pdensity(x, mu[j,], Bsig[j,], rdBsig[j]) #p(x) to be divided by sum (p(
    mx = np.argmax(px)
    if(mx != 0):
        mcl += 1
for i in range (25,50):
    x = X[i,]
    for j in range(3):
        px[j] = pdensity(x, mu[j,], Bsig[j,], rdBsig[j]) #p(x) to be divided by sum (p(
    mx = np.argmax(px)
    if(mx != 0):
        mcl += 1
for i in range(50, 75):
    x = X[i,]
    for j in range(3):
        px[j] = pdensity(x, mu[j,], Bsig[j,], rdBsig[j]) #p(x) to be divided by sum (p(
    mx = np.argmax(px)
    if(mx != 0):
        mcl += 1
print('Misclassification = ', mcl, 'Accuracy = ', (75-mcl)*100.0/75)
```

```
Misclassification =  75 Accuracy =  0.0
```

Split Dataset into (drivetrain,

trainlabel,datatest,testlabel) 30X3 samples for training set and 20 X 3 for testing

In [20]:

```
def splitdata(X,y):
    Xtr = np.zeros(shape=(90,4), dtype = 'float')
    ytr = np.zeros(90, dtype = 'uint16')
    Xts = np.zeros(shape=(60,4), dtype = 'float')
    yts = np.zeros(60, dtype = 'uint16')

    Xtr[0:30,] = X[0:30,]
    Xtr[30:60,] = X[50:80,]
    Xtr[60:90,] = X[100:130,]

    Xts[0:20,] = X[30:50,]
    Xts[20:40,] = X[80:100,]
    Xts[40:60,] = X[130:150,]

    ytr[0:30] = y[0:30]
    ytr[30:60] = y[50:80]
    ytr[60:90] = y[100:130]

    yts[0:20,] = y[30:50]
    yts[20:40,] = y[80:100]
    yts[40:60,] = y[130:150]


    return Xtr, ytr, Xts, yts
```

Compute the mean value mue = E(X) for all three classes 0, 1, 2 as 2D-array, each column has mean vector class

In [21]:

```
mu = np.zeros(shape=(3,4), dtype = 'float')
mu1 = np.zeros(4, dtype ='float')
mu2 = np.zeros(4, dtype ='float')
mu3 = np.zeros(4, dtype ='float')

Xtr, ytr, Xts, yts = splitdata(X,y)


for i in range(4):
    mu1[i] = np.mean(Xtr[0:30,i])
    mu2[i] = np.mean(Xtr[30:60,i])
    mu3[i] = np.mean(Xtr[60:90,i])
mu[0,] = mu1
mu[1,] = mu2
mu[2,] = mu3

print(mu)
```

```
[[5.02666667 3.45       1.47333333 0.24666667]
 [6.07       2.79       4.33333333 1.35333333]
 [6.58333333 2.93333333 5.60333333 2.00666667]]
```

In [22]:

```
Bsig = np.zeros(shape=(3,4,4), dtype = 'float')
Bsig1 = np.cov(Xtr[0:30,].T)
```

```
Bsig2 = np.cov(Xtr[30:60,].T)
Bsig3 = np.cov(Xtr[60:90,].T)
Bsig[0,] = Bsig1
Bsig[1,] = Bsig2
Bsig[2,] = Bsig3
print(Bsig.shape)
print(Bsig[0])
print('\n', Bsig[1])
print('\n', Bsig[2])
```

```
(3, 4, 4)
[[0.13857471 0.10103448 0.01797701 0.01595402]
 [0.10103448 0.12258621 0.00172414 0.01931034]
 [0.01797701 0.00172414 0.03443678 0.00577011]
 [0.01595402 0.01931034 0.00577011 0.01016092]]

 [[0.29803448 0.10210345 0.19310345 0.05717241]
 [0.10210345 0.10782759 0.08517241 0.04434483]
 [0.19310345 0.08517241 0.21126437 0.07298851]
 [0.05717241 0.04434483 0.07298851 0.04464368]]

 [[0.47454023 0.1091954  0.39247126 0.04356322]
 [0.1091954  0.11195402 0.08781609 0.04701149]
 [0.39247126 0.08781609 0.39274713 0.06204598]
 [0.04356322 0.04701149 0.06204598 0.06547126]]
```

In [23]:
```
dBsig = np.zeros(3, dtype = 'float')
rdBsig = np.zeros(3, dtype = 'float')
for i in range (3):
    dBsig[i] = np.linalg.det(Bsig[i])
    rdBsig[i] = np.sqrt(dBsig[i])
print(dBsig)
print(rdBsig)
```

```
[1.21752890e-06 2.56554057e-05 9.80238892e-05]
[0.00110342 0.00506512 0.0099007 ]
```

In [24]:
```
def pdensity(x, m, s, rd):
    den = (2*np.pi)**(x.shape[0]/2) * rd
    return(np.exp(-0.5*np.dot((x-m), np.dot(s, x-m)))/den)
```

apriors for all three classes are equal = 1/3 and denominirs are equal for all the three classes the Posterior density for each class multiplied by a constant term in the decision can be from probability density

Let us apply for each vector and find max aposterior value for each vector from the sample

In [25]:
```
px = np.zeros(3, dtype='float')
mcl = 0
for i in range(60):
```

```python
    x = Xts[i,]
    for j in range(3):
        px[j] = pdensity(x, mu[j,], Bsig[j,], rdBsig[j]) #p(x) to be divided by sum (p(
    mx = np.argmax(px)
    if(mx != yts[i]):
        mcl += 1
print('Misclassification = ', mcl, 'Accuracy = ', (75-mcl)*100.0/75)
```

```
Misclassification =  40 Accuracy =  46.666666666666664
```

In [ ]:

```python

```