Question 1: *Vary the sample size and examine the results.*

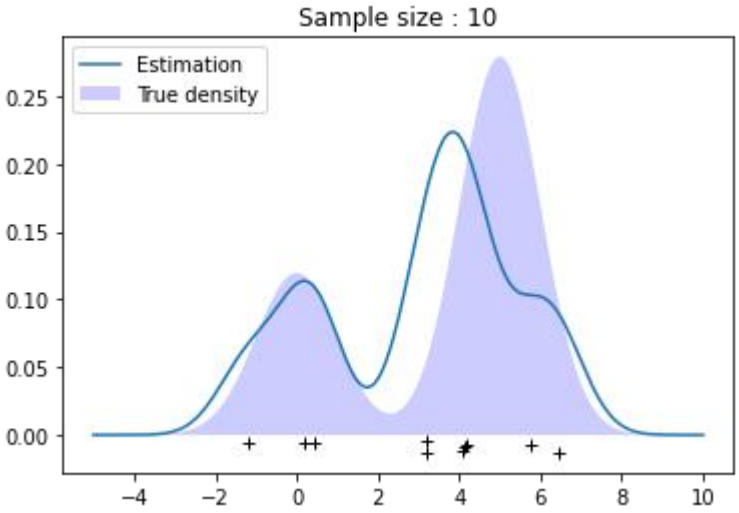
Answer:

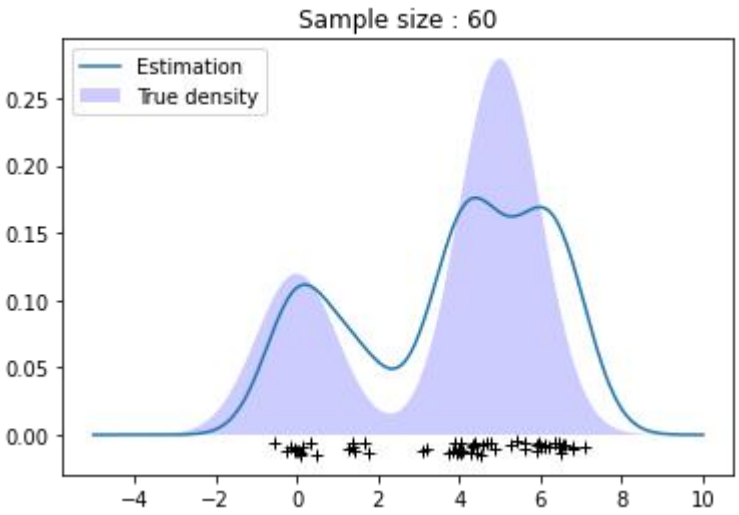
We define a function call generate_sample_withDiffererntSize(sample size) which can vary the sample size and plot the according images contains with true density and estimation.

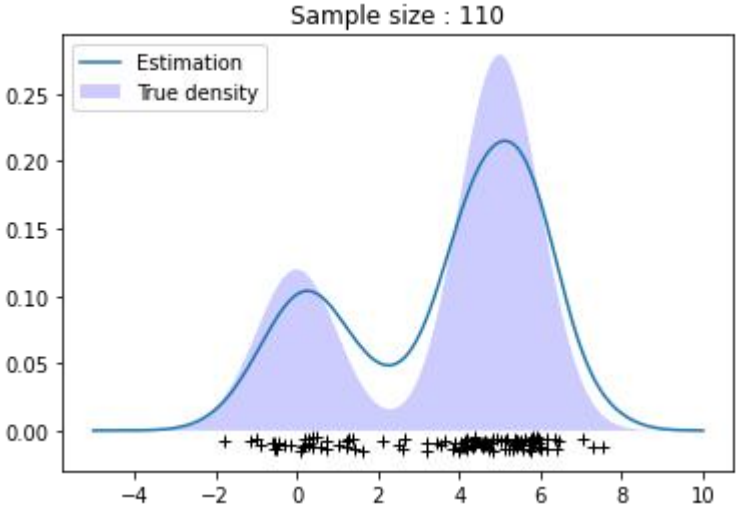
```
# define a function to generate different sample size
def generate_sample_withDifferentSize(sample_size):
  N = sample_size
  X = np.concatenate((np.random.normal(0, 1, int(0.3 * N)), np.random.normal(5, 1, int(0.7 * N))))[:, np.newaxis]
  X_plot = np.linspace(-5, 10, 1000)[:, np.newaxis]
  true_density=(0.3*norm(0,1).pdf(X_plot[:,0])+0.7*norm(5,1).pdf(X_plot[:,0]))
  # density estimation by Gaussian kernels
  kde = KernelDensity(kernel='gaussian', bandwidth=0.75).fit(X)
  density = np.exp(kde.score_samples(X_plot))
  # display: true density and estimate
  fig = plt.figure()
  ax = fig.add_subplot(111)
  ax.fill(X_plot[:,0], true_density, fc='b', alpha=0.2, label= 'True density')
  ax.plot(X_plot[:,0], density, '-', label="Estimation")
  ax.plot(X[:, 0], -0.005 - 0.01 * np.random.random(X.shape[0]), '+k')
  plt.title("Sample size : " + str(sample_size))
  ax.legend(loc='upper left')
  plt.show()

l=[i for i in range(30,330,20)]
for size in l :
  generate_sample_withDifferentSize(size)
```

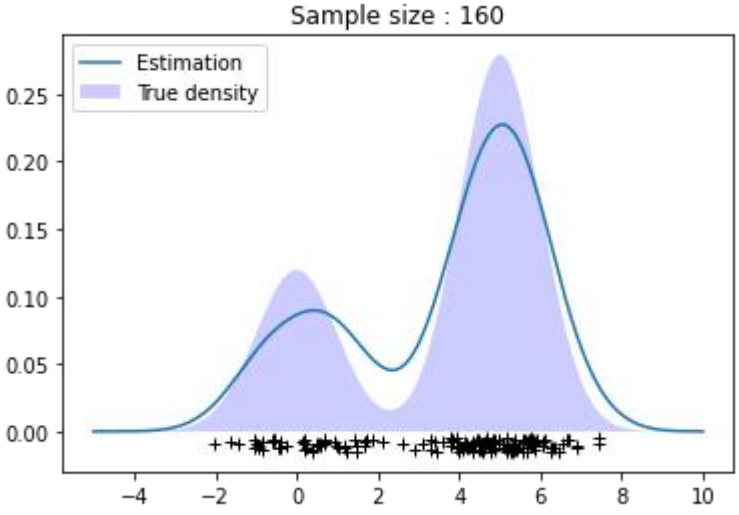
figure 1.1: Code for vary the sample size

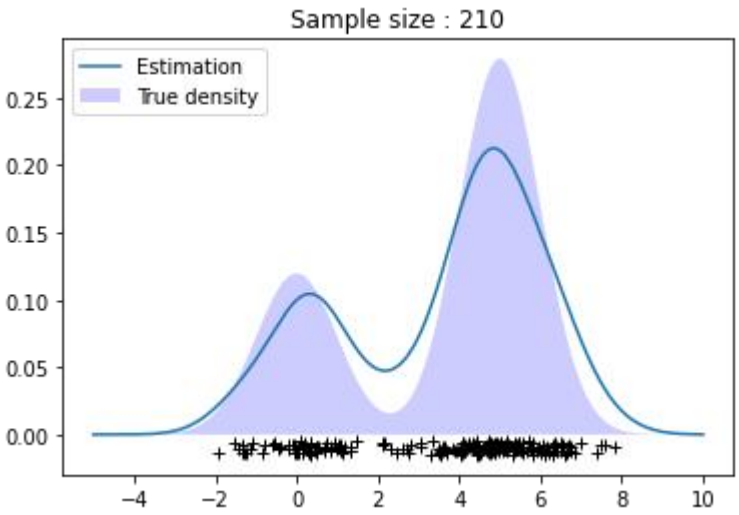
We plot the data size from 10 to 510 with 50 data interval. When the datas volumn is just 10 and 60, the estimator had hard time align the true density. After 110 data points, is becoming stable. We can conclude that when the data volumn is low, the estimation is sometime hard to estimate the precise density, as the data volumn increasing, the estimation of the density is tending to be more stable.

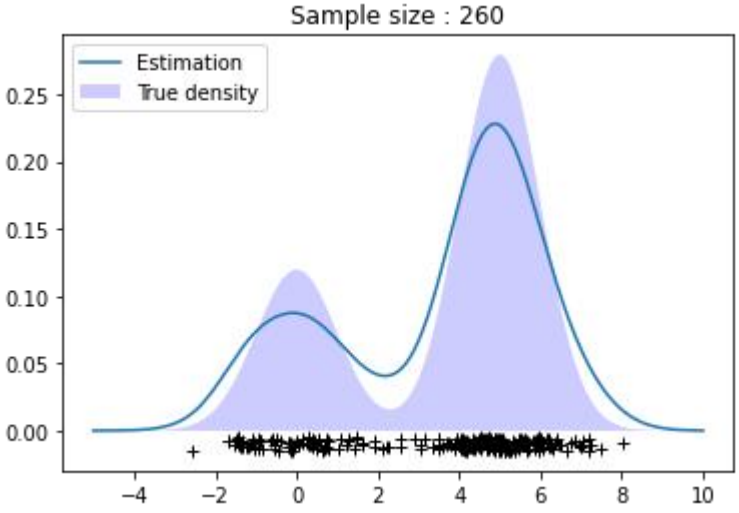


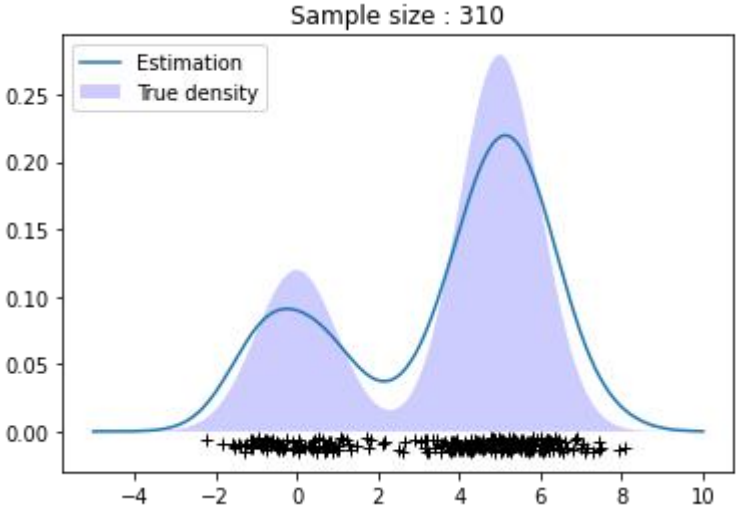

Sample size : 60
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10

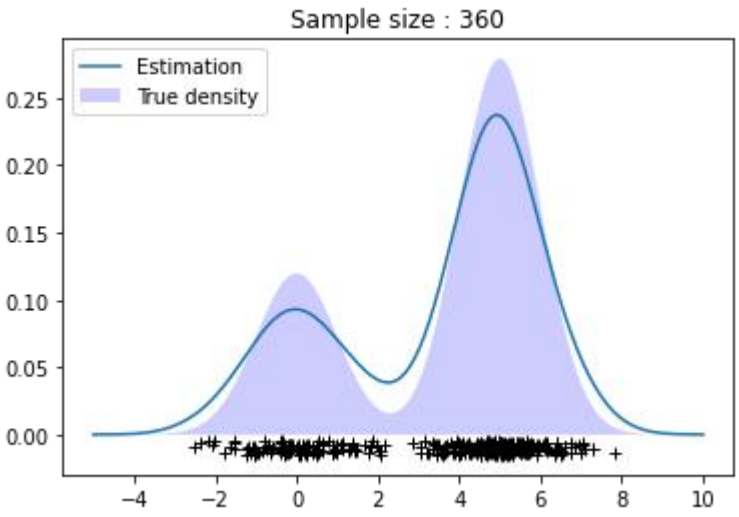

Sample size : 110
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10

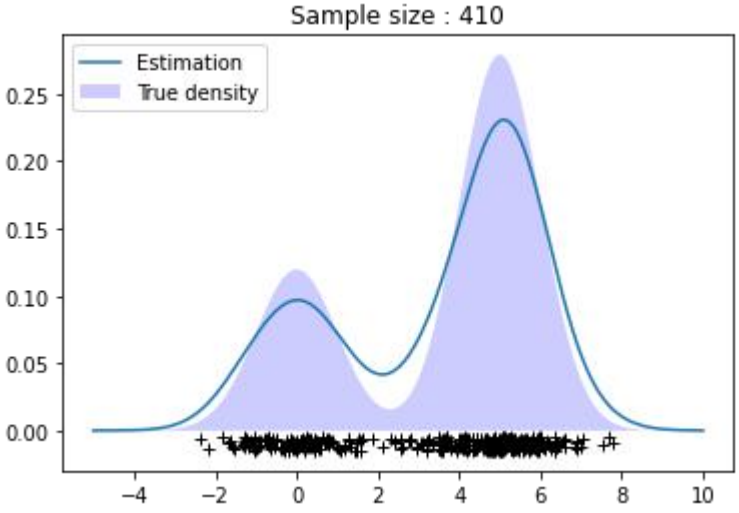

Sample size : 160
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10

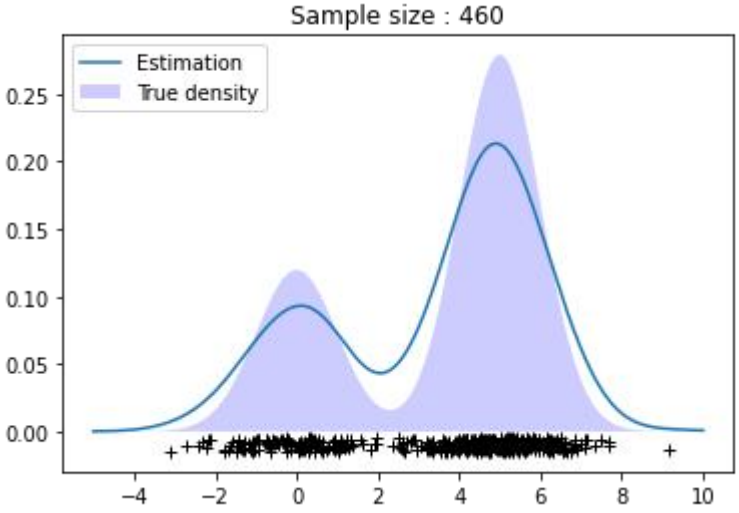

Sample size : 210
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10


Sample size : 260
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10


Sample size : 310
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10


Sample size : 360
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10


Sample size : 410
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10


Sample size : 460
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10

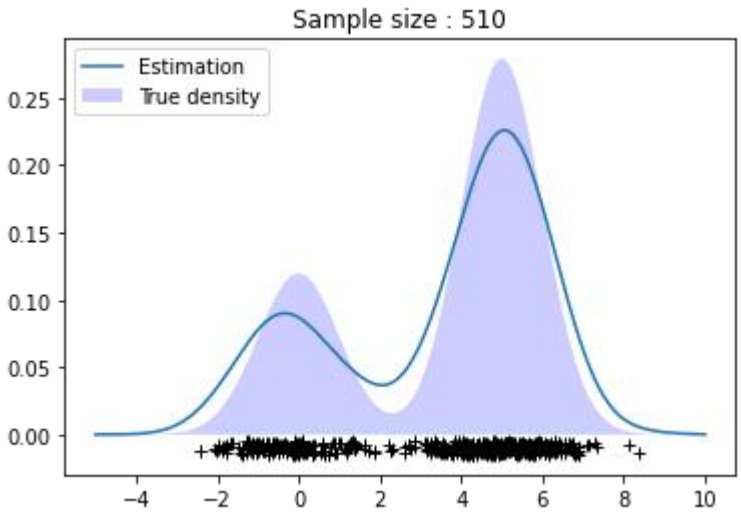


figure 1.2 Estimation and true density with different data volumn

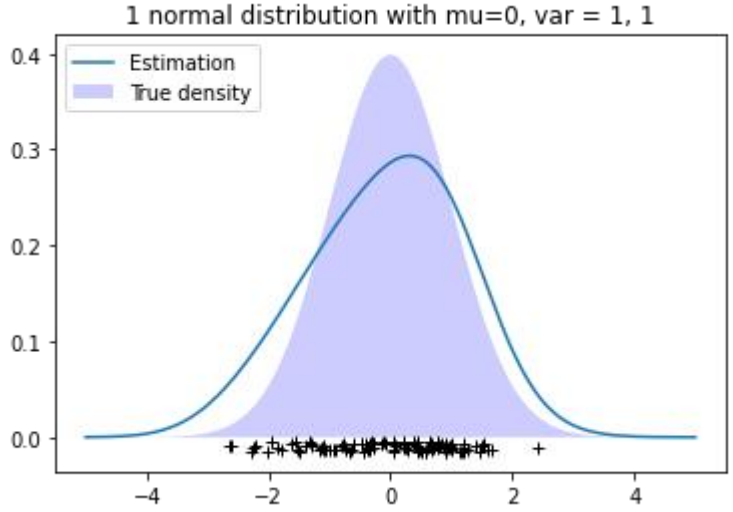
Question 2: *Vary the number of normal distributions when generating the sample and examine the results.*

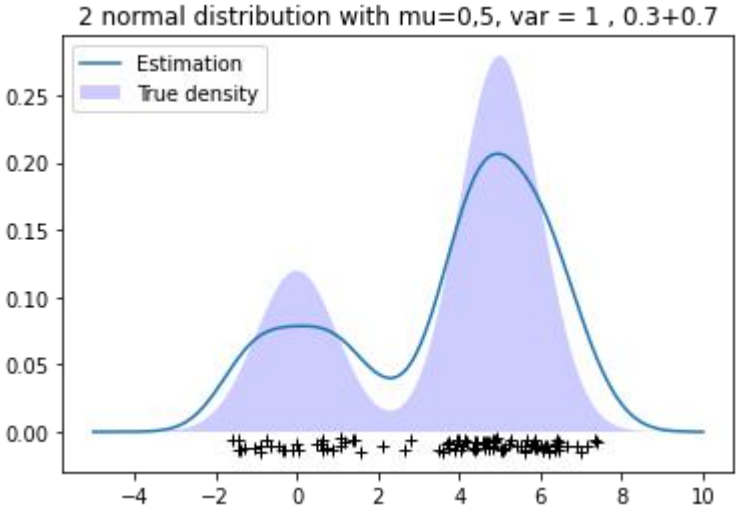
Answer:

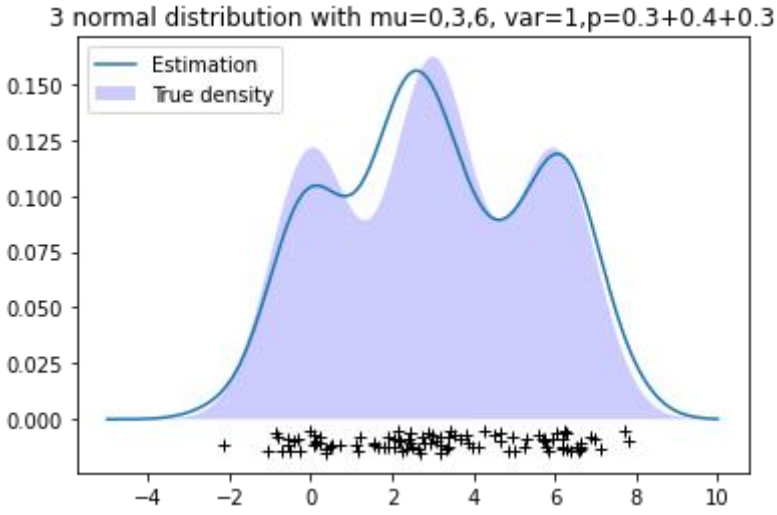
We generate different number of normal distributions and plot the images with estimation and true density. For the reason of easy comparation, the variance of normal distribution are all equal to 1.

When just have 1 normal distribution , the estimator is about 0.1 lower then the true density value when x = mu. When we have 2 normal distribution, the estimator value when x = mu is also a little bit lower. When we have 3 dense normal diribution , the estimator is good. When we have 4 dense normal distribution, the estimator doesn't work very well, the estimation of 4 sparse normal distribution still a little bit lower but align better than the dense one. The 5 normal distribution case is almost as same as four.

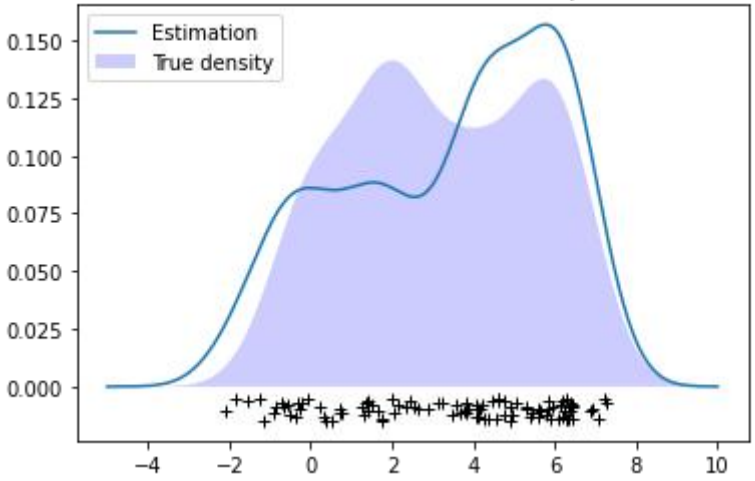
In conclusion, as the number of normal distributions increase, the estimator is hard to have precise estimation especiall when the normal distributions are very dense. And the estimators are tend to have lower value(large variance) around the mean.



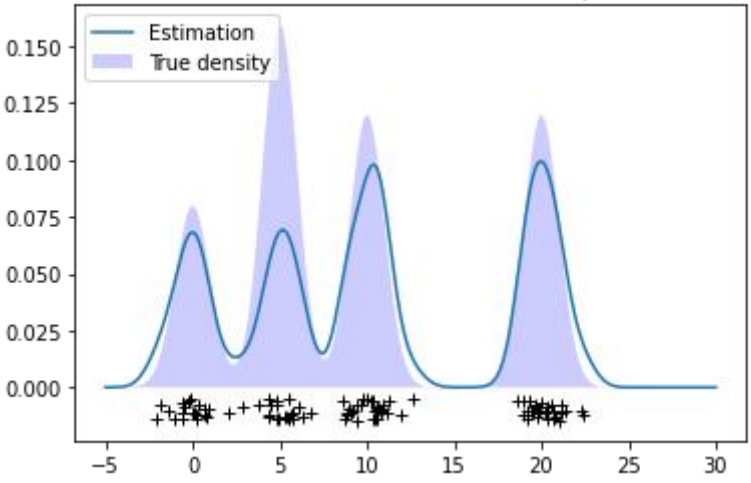




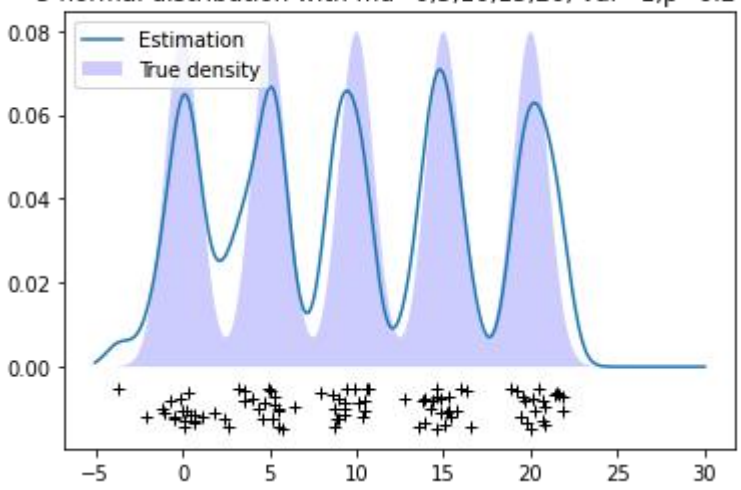
4 normal distribution with mu=0,2,4,6, var=1,p=0.2+0.3+0.2+0.3



4 normal distribution with mu=0,5,10,20, var=1,p=0.2+0.2+0.3+0.3



5 normal distribution with mu=0,5,10,15,20, var=1,p=0.2*5



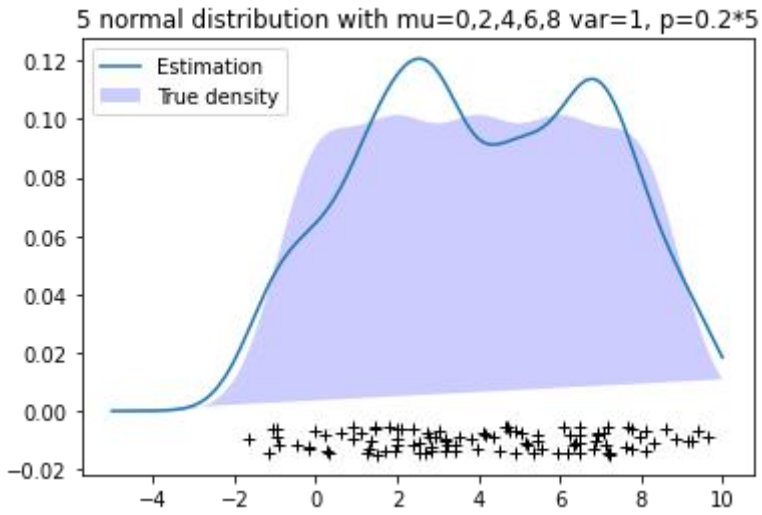


figure 2: Estimation and true density with different number of normal distribution

Question 3: *Vary the* `bandwidth` *value and examine the results.*

Answer:

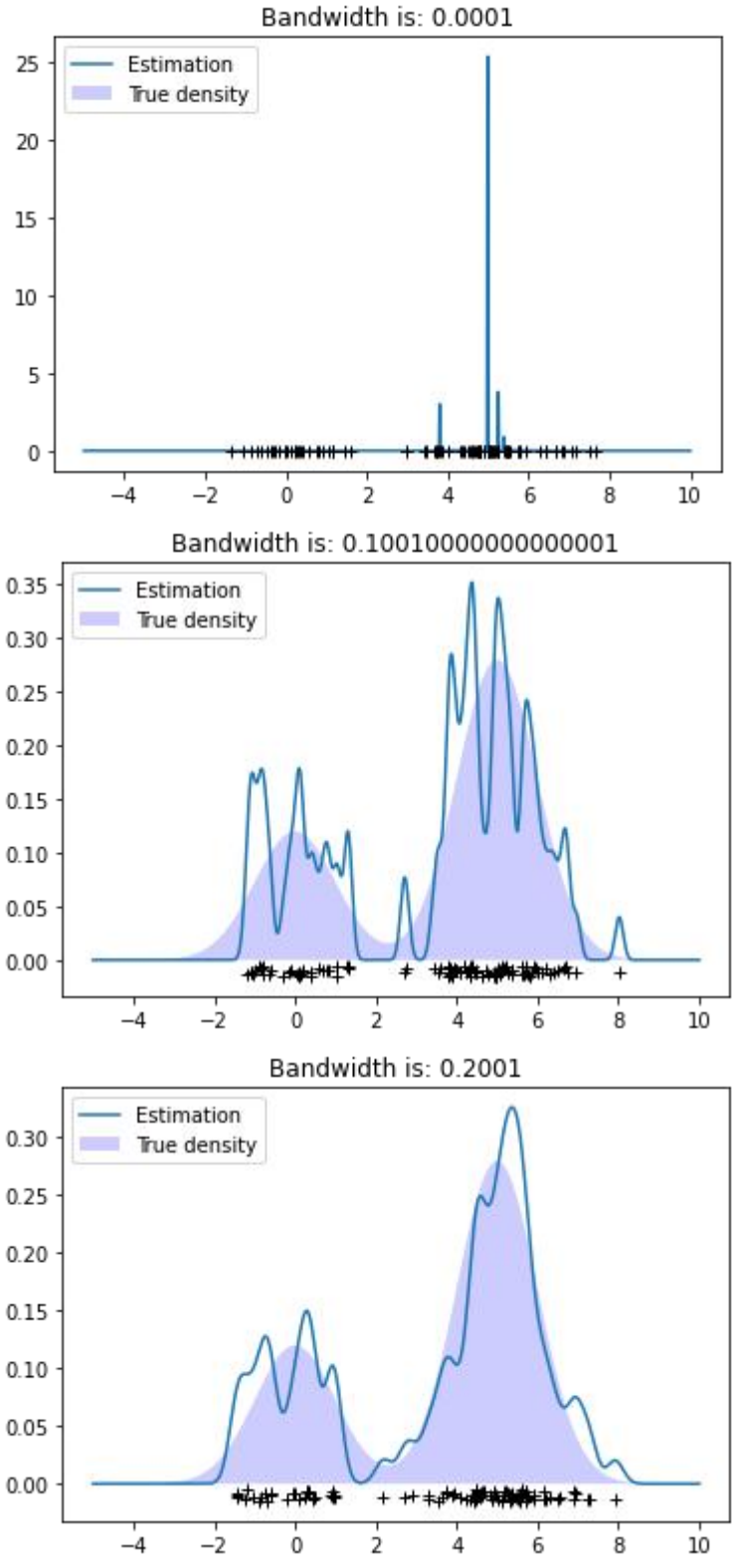
we define a function called generate_withDifferentBandwidth() that can pass a bandwidth parameter to generate and plot the estimation and true density.

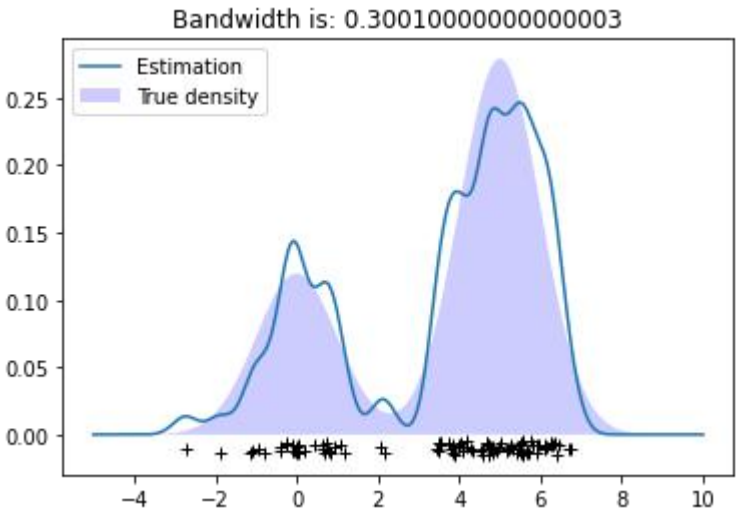
```python
# define a function to generate with different bandwidth
def generate_withDifferentBandwidth(bandwidth):
  N = 100
  X = np.concatenate((np.random.normal(0, 1, int(0.3 * N)), np.random.normal(5, 1, int(0.7 * N))))[:, np.newaxis]
  X_plot = np.linspace(-5, 10, 1000)[:, np.newaxis]
  true_density=(0.3*norm(0,1).pdf(X_plot[:,0])+0.7*norm(5,1).pdf(X_plot[:,0]))
  # density estimation by Gaussian kernels
  kde = KernelDensity(kernel='gaussian', bandwidth= bandwidth).fit(X)
  density = np.exp(kde.score_samples(X_plot))
  # display: true density and estimate
  fig = plt.figure()
  ax = fig.add_subplot(111)
  ax.fill(X_plot[:,0], true_density, fc='b', alpha=0.2, label= 'True density')
  ax.plot(X_plot[:,0], density, '-', label="Estimation")
  ax.plot(X[:, 0], -0.005 - 0.01 * np.random.random(X.shape[0]), '+k')
  plt.title(f"Bandwidth is: {bandwidth}")
  ax.legend(loc='upper left')
  plt.show()

l=[i for i in np.arange(0.0001,2.5,0.1)]
for bandwidth in l :
  generate_withDifferentBandwidth(bandwidth)
```

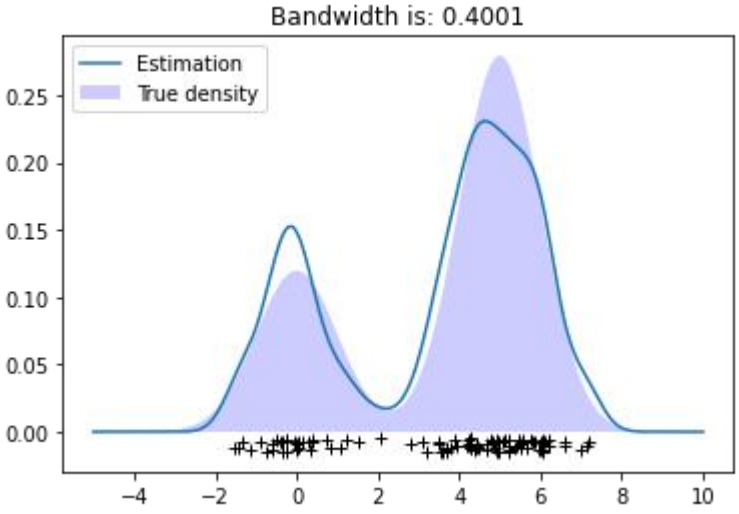
figure 3.1: Code for vary the bandwidth

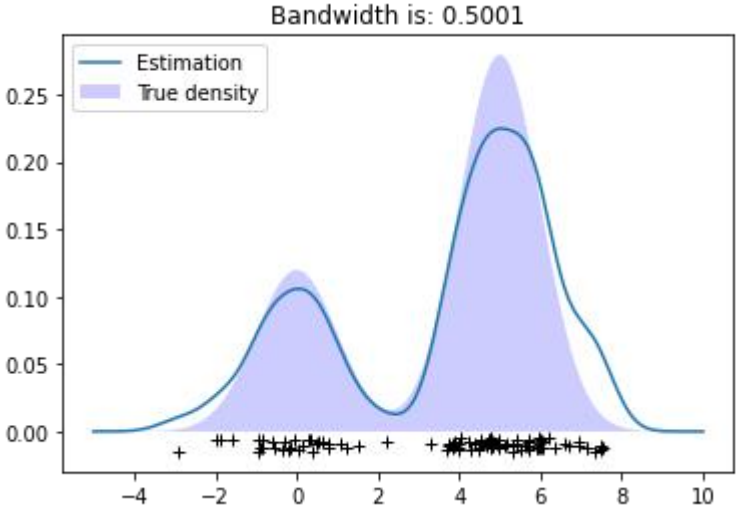
When the bandwidth is really low, like 0.0001, there are just few vertical lines in the plot. When the bandwidth increase to 0.1, there are lots of zig zag in the plot. When the bandwidth is increasing from 0.2 to 0.6, the estimation become smoother. After 0.6, the estimation become flat.

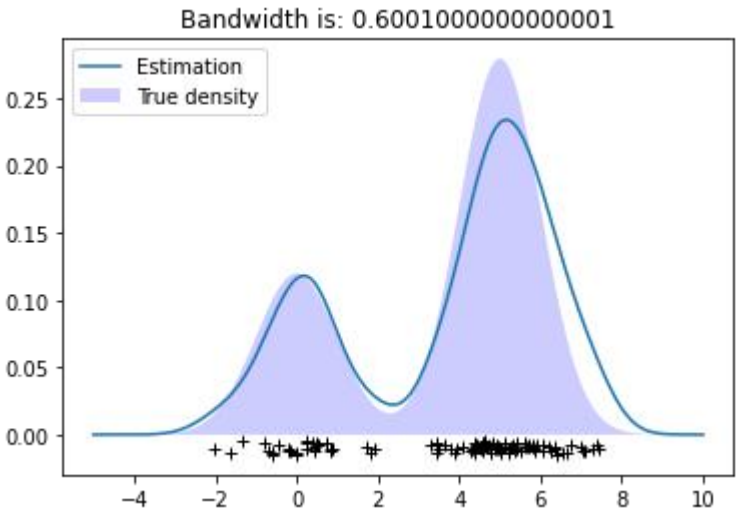
As API said “The bandwidth here acts as a smoothing parameter, controlling the tradeoff between bias and variance in the result. A large bandwidth leads to a very smooth (i.e. high-bias) density distribution. A small bandwidth leads to an unsmooth (i.e. high-variance) density distribution”.

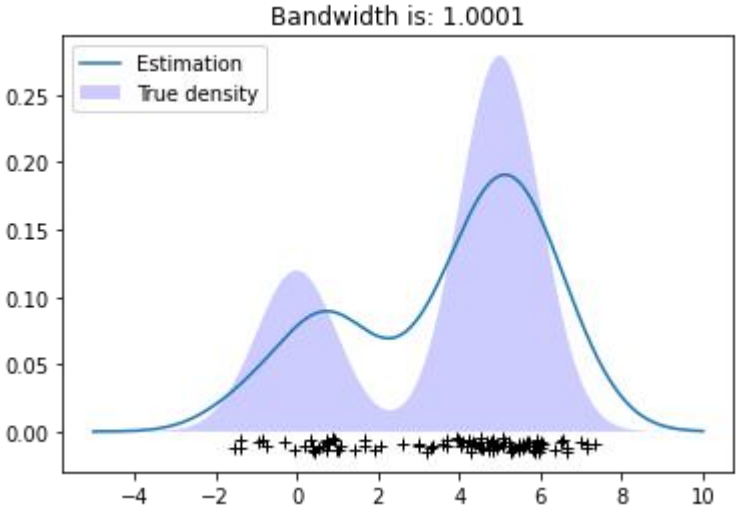

Bandwidth is: 0.0001
Estimation
True density
Bandwidth is: 0.10010000000000001
Estimation
True density
Bandwidth is: 0.2001
Estimation
True density

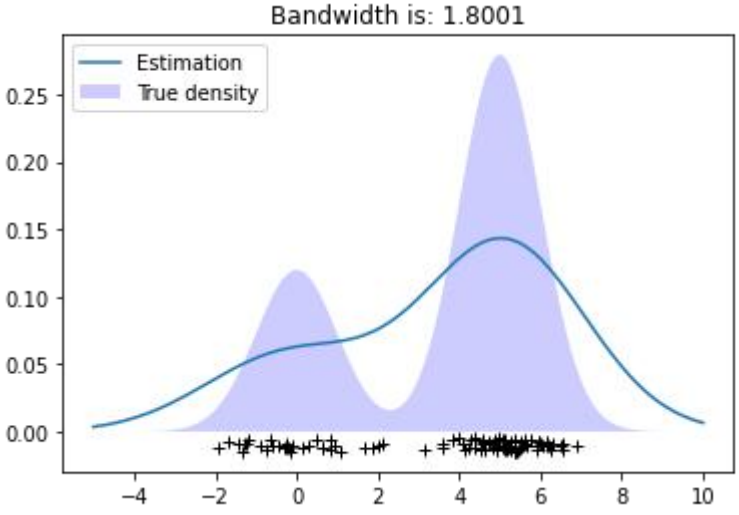

Bandwidth is: 0.30010000000000003
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10


Bandwidth is: 0.4001
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10


Bandwidth is: 0.5001
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10


Bandwidth is: 0.6001000000000001
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10


Bandwidth is: 1.0001
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10


Bandwidth is: 1.8001
Estimation
True density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6
8
10

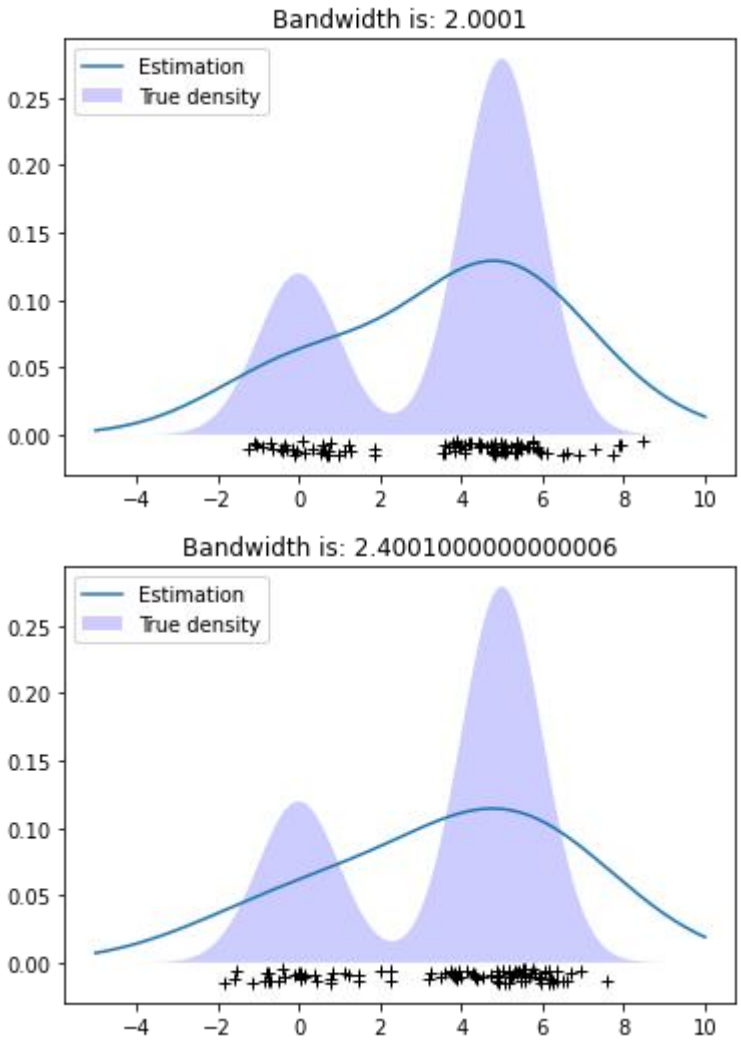


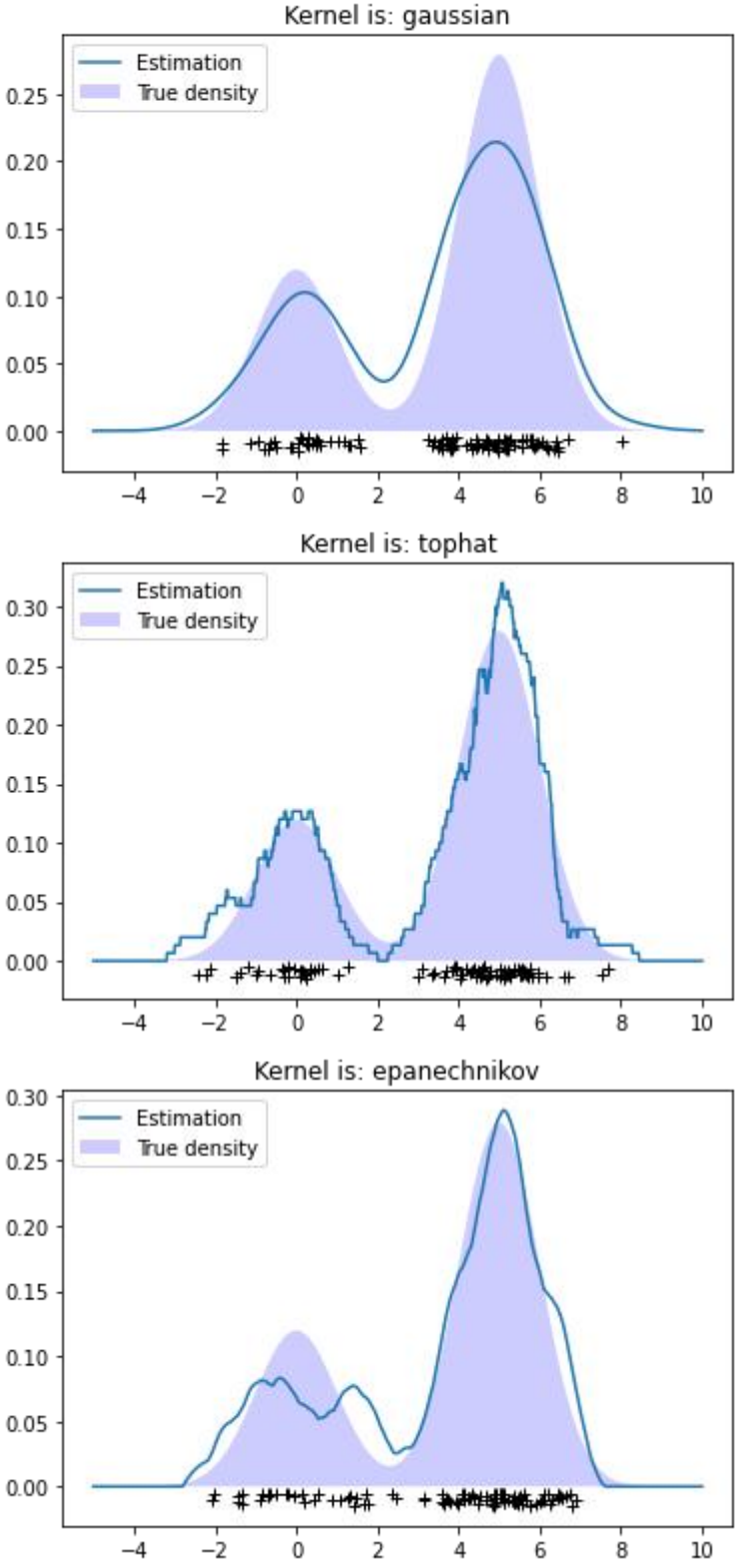
figure 3.2: Estimation and true denstiy with different bandwidth

Question 4: *Vary the kernel type and examine the results.*

Answer:

We change the kernel parameter in the list ['gaussian', 'tophat', 'epanechnikov', 'exponential', 'linear', 'cosine'].

As you can see, the gaussian is best align with the true density, simply because the data we generate are gaussian distribution. tophat has lots of zig zag, other is not smoother as gaussian.



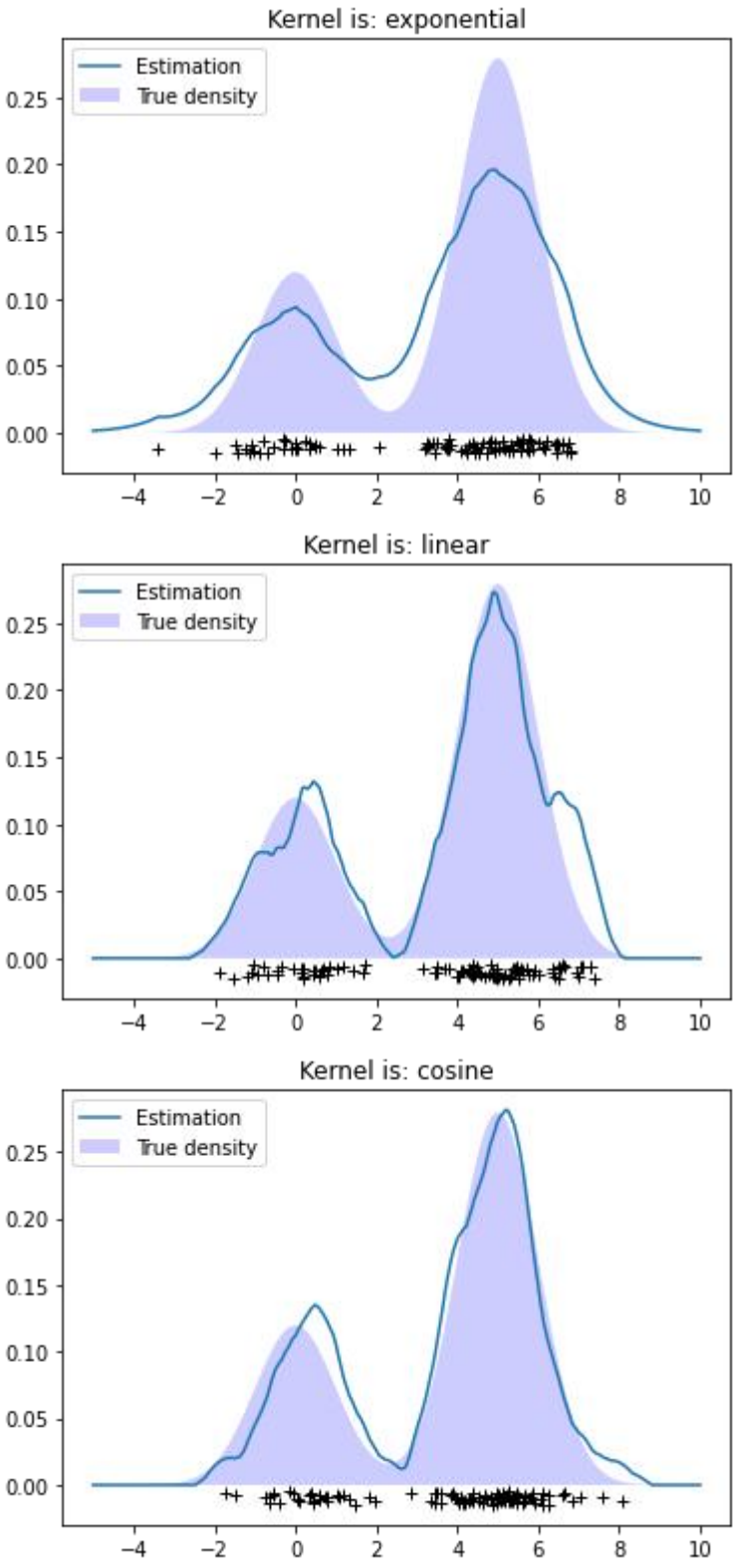


figure 4: Vary the kernel type

Question 5: *Vary the* `bandwidth` *value and examine the results.*

Answer:

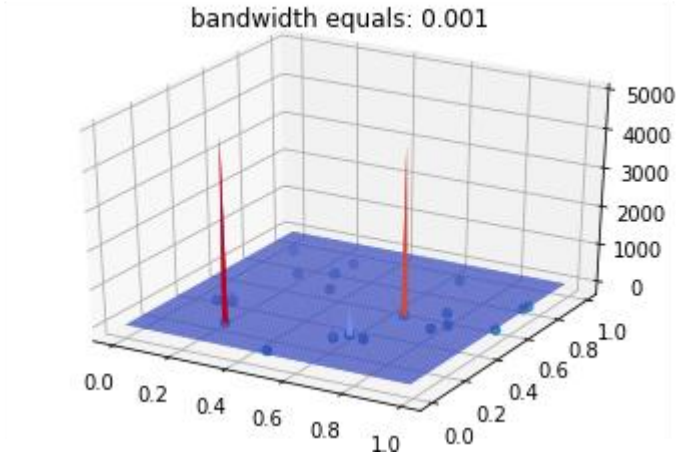
We define a function that can generate different estimator according to different bandwidth in 2D data.

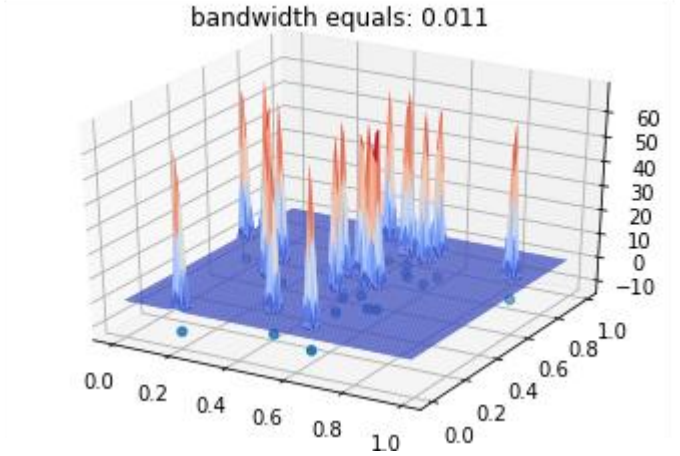
```
def generate1_withDifferentBandwidth(bandwidth):
  N = 20
  # random variables N rows 2 columns
  kd = np.random.rand(N, 2)
  # set grid for visualization
  grid_size = 100
  Gx = np.arange(0, 1, 1/grid_size)
  Gy = np.arange(0, 1, 1/grid_size)
  Gx, Gy = np.meshgrid(Gx, Gy)
  # set bandwidth for kernel
  bw = bandwidth
  # estimation, then density calculation on the grid
  kde3 = KernelDensity(kernel='gaussian', bandwidth=bw).fit(kd)
  Z= np.exp(kde3.score_samples(np.hstack(((Gx.reshape(grid_size*grid_size))[:,np.newaxis], (Gy.reshape(grid_size*grid_size)[:,np.newaxis])))))
  # display
  from mpl_toolkits.mplot3d import Axes3D
  from matplotlib import cm
  fig = plt.figure()
  ax = fig.gca(projection='3d')
  ax.plot_surface(Gx, Gy, Z.reshape(grid_size,grid_size), rstride=1,
  cstride=1, cmap=cm.coolwarm, linewidth=0,
  antialiased=True)
  ax.scatter(kd[:,0], kd[:,1], -10)
  plt.title(f"bandwidth equals: {bandwidth}")
  plt.show()


l=[i for i in np.arange(0.001,0.2,0.01)]
for bandwidth in l :
  generate1_withDifferentBandwidth(bandwidth)
```

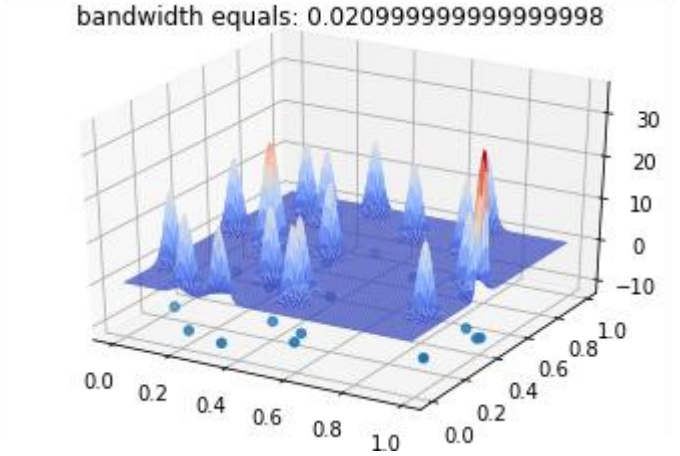
figure 5.1: Code for vary bandwidth in 2D data

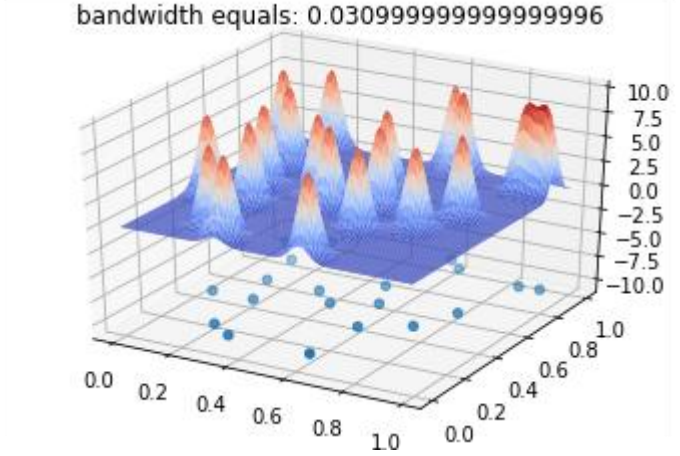
As you can see in the plots, when the bandwidth is extreamly small, the plots just have few vertical spikes. When the bandwidth is under 0.2, the spikes are still very sharp. From 0.2 to 0.6, the spikes become more and more smoother. Around 0.7 you can see the beautiful bell shape distributions.When bandwidth keep growing, the plots are tend to be flat.

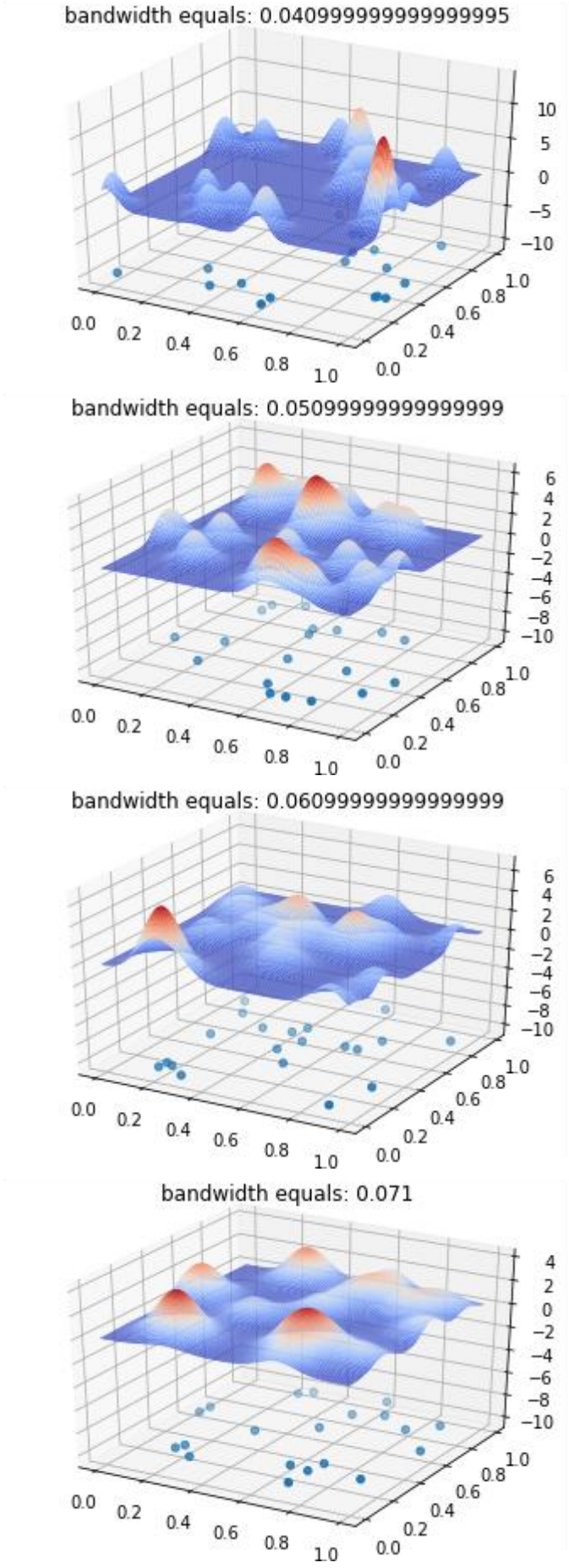
I believe we can have the same conclusion as problem 3. As API said “The bandwidth here acts as a smoothing parameter, controlling the tradeoff between bias and variance in the result. A large bandwidth leads to a very smooth (i.e. high-bias) density distribution. A small bandwidth leads to an unsmooth (i.e. high-variance) density distribution”.

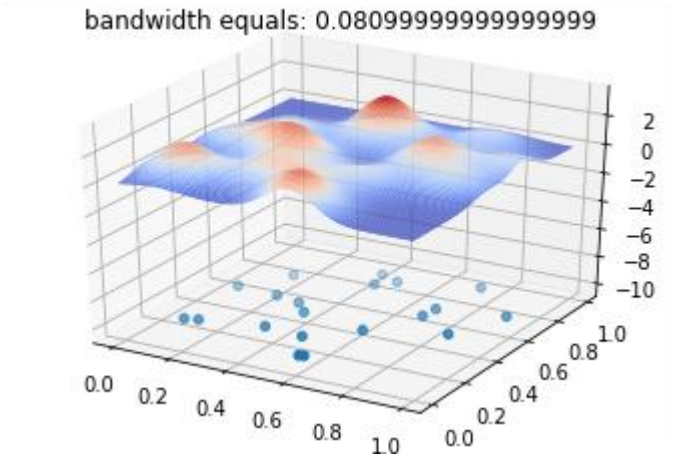

bandwidth equals: 0.001
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0

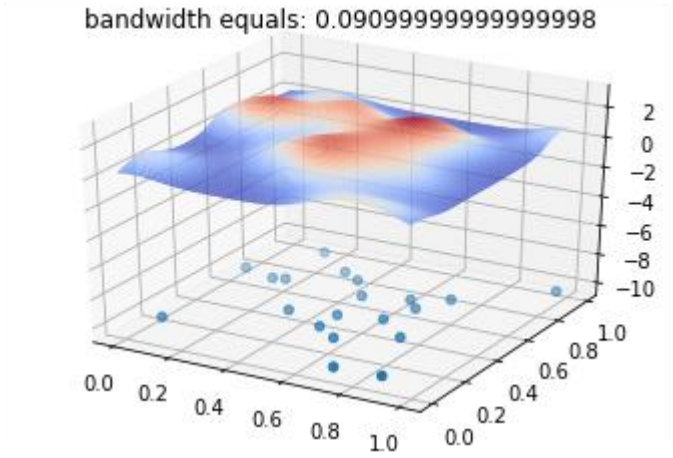

bandwidth equals: 0.011
60
50
40
30
20
10
0
−10
0.0
0.2
0.4
0.6
0.8
1.0

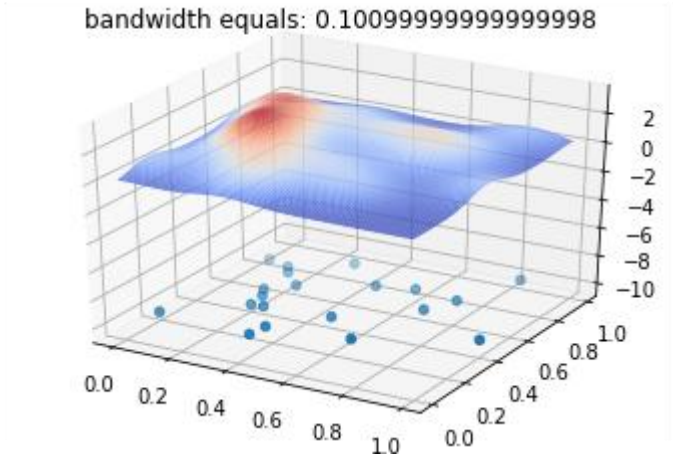

bandwidth equals: 0.020999999999999998
30
20
10
0
−10
0.0
0.2
0.4
0.6
0.8
1.0

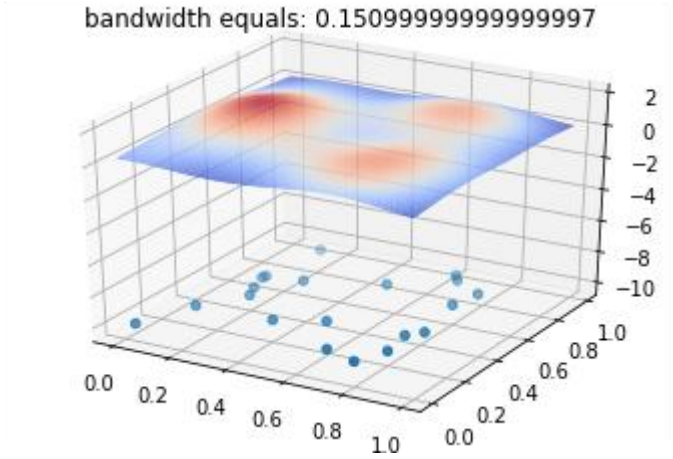

bandwidth equals: 0.030999999999999996
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0.0
0.2
0.4
0.6
0.8
1.0


bandwidth equals: 0.040999999999999995
10
5
0
−5
−10
0.0
0.2
0.4
0.6
0.8
1.0
bandwidth equals: 0.05099999999999999
6
4
2
0
−2
−4
−6
−8
−10
bandwidth equals: 0.06099999999999999
bandwidth equals: 0.071


bandwidth equals: 0.08099999999999999
2
0
−2
−4
−6
−8
−10
0.0
0.2
0.4
0.6
0.8
1.0


bandwidth equals: 0.09099999999999998
2
0
−2
−4
−6
−8
−10
0.0
0.2
0.4
0.6
0.8
1.0


bandwidth equals: 0.10099999999999998
2
0
−2
−4
−6
−8
−10
0.0
0.2
0.4
0.6
0.8
1.0


bandwidth equals: 0.15099999999999997
2
0
−2
−4
−6
−8
−10
0.0
0.2
0.4
0.6
0.8
1.0

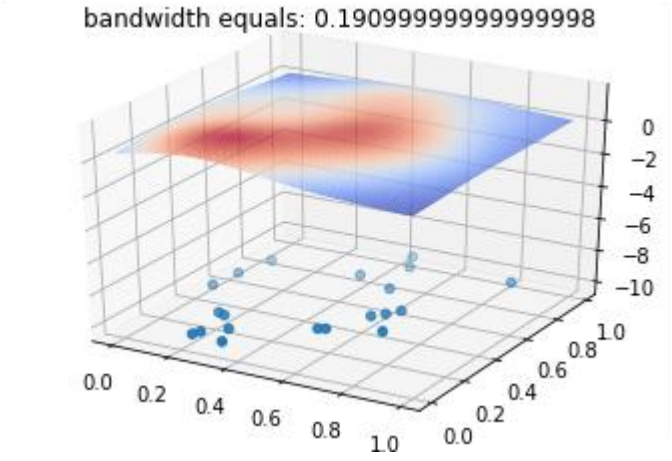


figure 5.2: Estimatoions with different bandwidth in 2D data

Question 6: *Generate the sample from a sum of 2 different normal distributions (with larger N) and examine the results.*

Answer:

We generate two normal distributions using np.random.randn(), one is +(1.0,1.0), two are different. And this time we use a larger N = 1000 rather than 100.

```
# generate sample
N = 1000
kd1 = np.random.randn(int(N/2), 2) + (1.0,1.0)
kd2 = np.random.randn(int(N/2), 2)
kd = np.concatenate((kd1, kd2))

# define the grid for the visualization
grid_size = 100
Gx = np.arange(-1, 3, 4/grid_size)
Gy = np.arange(-1, 3, 4/grid_size)
Gx, Gy = np.meshgrid(Gx, Gy)

# define the size of the bandwidth
bw = 0.2

# estimate and compute density in the grid
kde4 = KernelDensity(kernel='gaussian', bandwidth=bw).fit(kd)
Z= np.exp(kde4.score_samples(np.hstack(((Gx.reshape(grid_size*grid_size))[:,np.newaxis], (Gy.reshape(grid_size*grid_size)[:,np.newaxis])))))
# display
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(Gx, Gy, Z.reshape(grid_size,grid_size), rstride=1,
cstride=1, cmap=cm.coolwarm, linewidth=0,
antialiased=True)
ax.scatter(kd[:,0], kd[:,1], 0)
plt.show()
```

figure 6.1: Code for sum of 2D different normal distributions

Below is the result, what a beauty! Doesn't like rand, the normal distribution are not the spikes, they are beautiful bell shape. And two mixture guassian model is grand, like mountains.

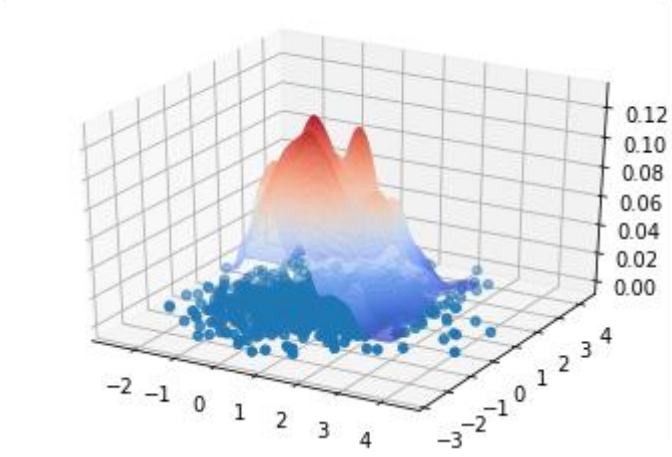


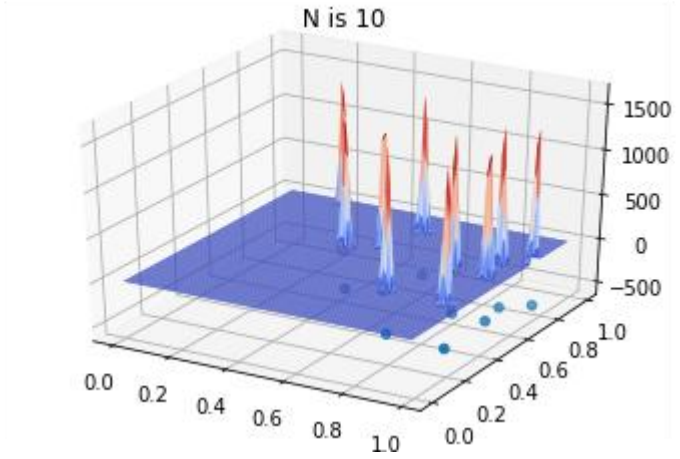
figure 6.2: Plot of sum of 2D different normal distributions

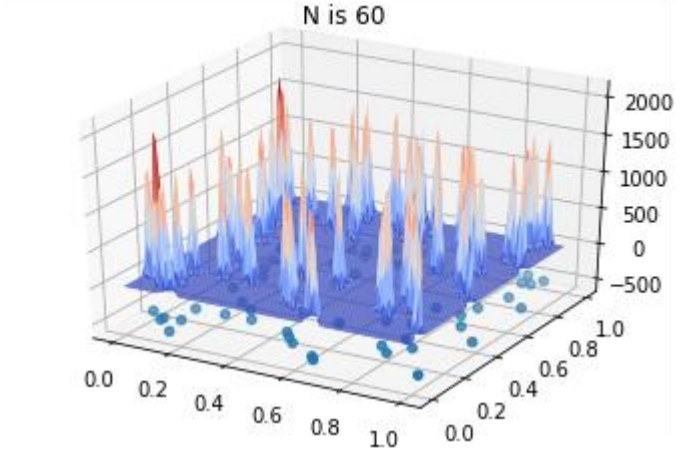
Question 7: *Vary the sample size and examine the results. What are the consequences of an increase in N?*

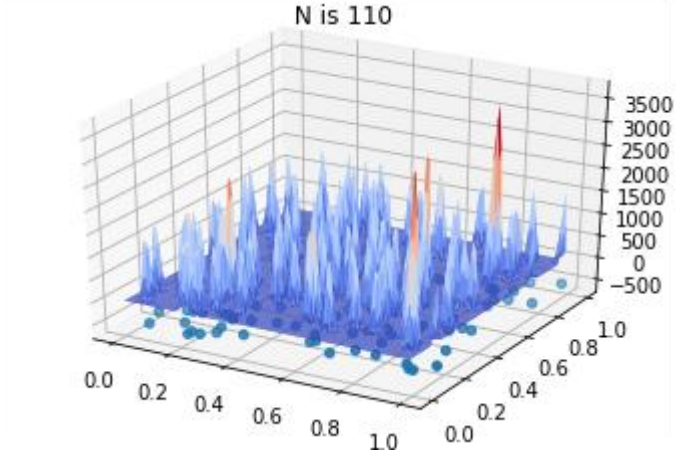
Answer:

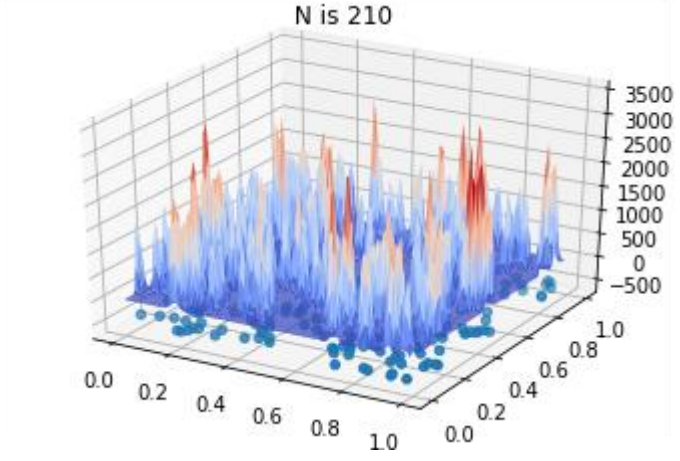
In this example, we use the calculation function of a two-dimensional normal distribution in (x,y), bivariate_normal (x, y, sigmax, sigmay, mux, muy, sigmaxy), where sigmax is the variance of X, sigmay the variance of Y, sigmaxy the covariance between X and Y, mux the mean of X and muy the mean of Y.

We increase the size of N from a low number to a big number. Below is the plots. As you can see, as the N increase, the spikes are increased. When there are enough spikes in the space, the distribution is tend to be like bivariate normal we definded before.


N is 10
1500
1000
500
0
−500
0.0
0.2
0.4
0.6
0.8
1.0


N is 60
2000
1500
1000
500
0
−500
0.0
0.2
0.4
0.6
0.8
1.0


N is 110
3500
3000
2500
2000
1500
1000
500
0
−500
0.0
0.2
0.4
0.6
0.8
1.0


N is 210
3500
3000
2500
2000
1500
1000
500
0
−500
0.0
0.2
0.4
0.6
0.8
1.0

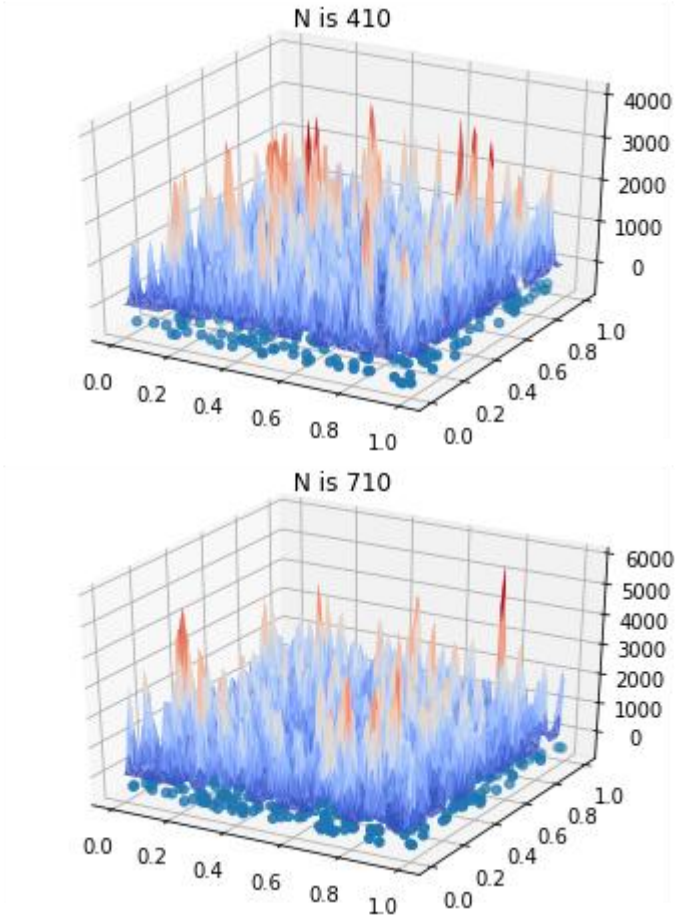


figure 7: Plots with different sample size

Question 8: *Apply principal component analysis to this data and estimate the density of the projected data on the first principal axis (or the first two principal axes). Vary the "window" width (*`bandwidth`*). Visualize the results.*

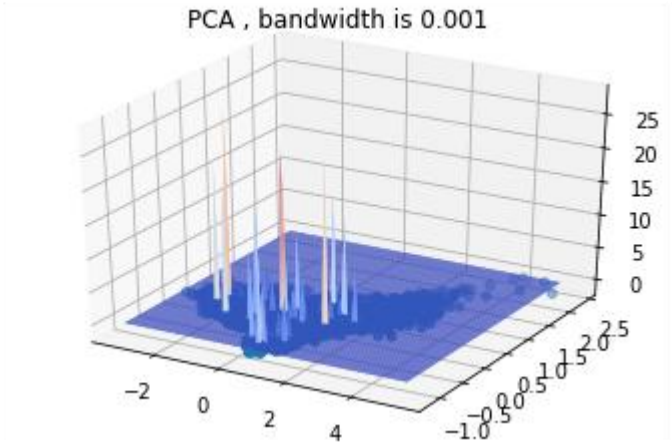
Answer:

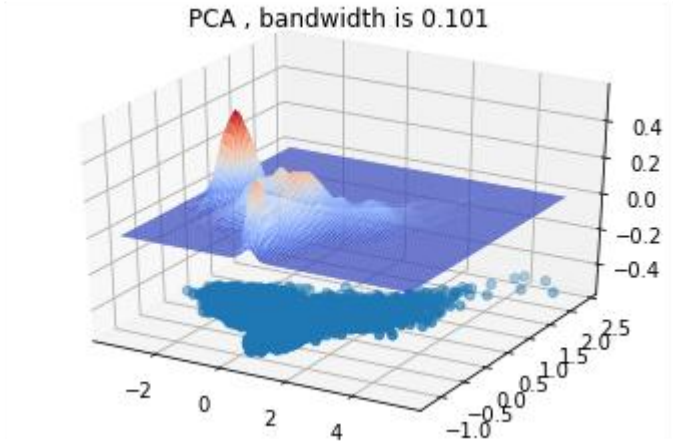
We define a function that we can pass a bandwidth parameter to change the bandwidth using PCA to reduce the data dimension.

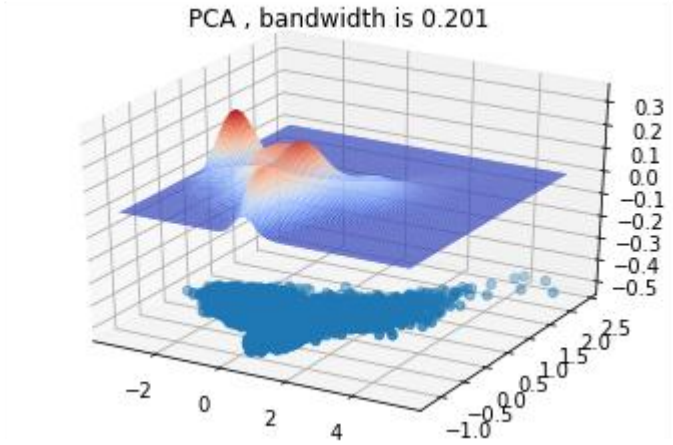
```
def text(bd):
  from sklearn.decomposition import PCA
  pca = PCA(n_components=2)
  pca.fit(textures[:,0:40])
  texturesp = pca.transform(textures[:,:40])
  # set grid for visualization
  grid_size = 100
  mx = min(texturesp[:,0])
  Mx = max(texturesp[:,0])
  my = min(texturesp[:,1])
  My = max(texturesp[:,1])
  xstep = (Mx - mx) / grid_size
  ystep = (My - my) / grid_size
  Gx = np.arange(mx, Mx, xstep)
  Gy = np.arange(my, My, ystep)
  Gx, Gy = np.meshgrid(Gx, Gy)
  ######## TO BE COMPLETED #########
  bandwidth = bd

  kde3 = KernelDensity(kernel='gaussian', bandwidth=bandwidth).fit(texturesp)
  Z= np.exp(kde3.score_samples(np.hstack(((Gx.reshape(grid_size*grid_size))[:,np.newaxis], (Gy.reshape(grid_size*grid_size)[:,np.newaxis])))))

  # display
  from mpl_toolkits.mplot3d import Axes3D
  from matplotlib import cm
  fig = plt.figure()
  ax = fig.gca(projection='3d')
  ax.plot_surface(Gx, Gy, Z.reshape(grid_size,grid_size), rstride=1,
  cstride=1, cmap=cm.coolwarm, linewidth=0,
  antialiased=True)
  ax.scatter(texturesp[:,0], texturesp[:,1], -0.5)
  plt.show()

l=[i for i in np.arange(0.001, 0.2, 0.1)]
for n in l :
  text(n)
```

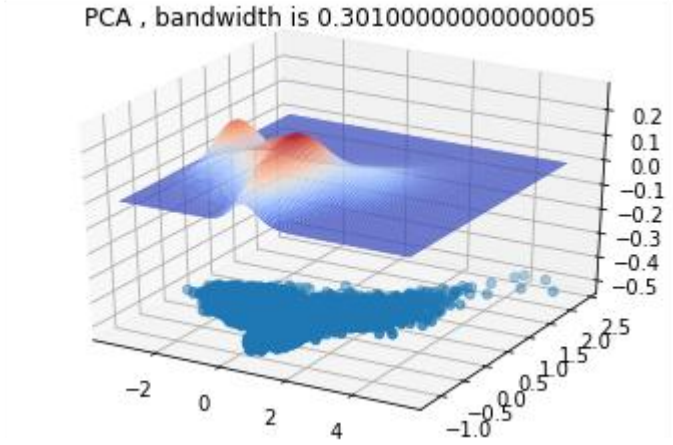
figure 8.1: Code for using PCA to transform multidimension data and estimate

As you can see in the plot, when bandwidth is extremly low, just few spikes in the space. When the bandwidth is around 0.1, the plot display its features. When the bandwidth is around 0.2, the plot is smooth and the plot is become flat after 0.2.




PCA , bandwidth is 0.101
0.4
0.2
0.0
−0.2
−0.4
−2
0
2
4
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0


PCA , bandwidth is 0.201
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−2
0
2
4
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0


PCA , bandwidth is 0.30100000000000005
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−2
0
2
4
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0

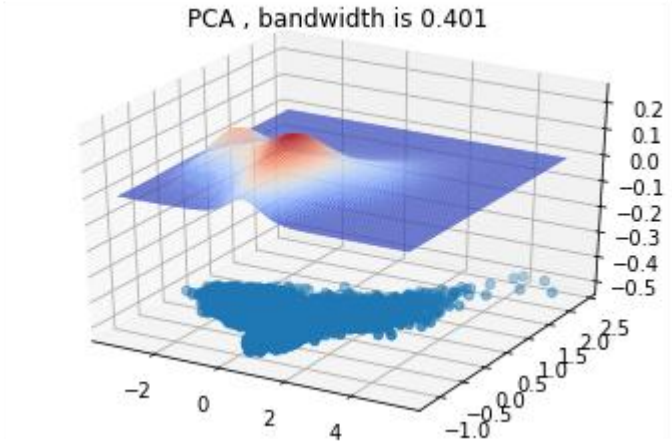


figure 8.2: Plots with different bandwidth using PCA

Question 9: *Apply discriminant analysis to this data and estimate the density of the projected data on the first discriminant axis (or the first two discriminant axes). Vary the "window" width (*`bandwidth`*). Visualize the results.*

Answer:

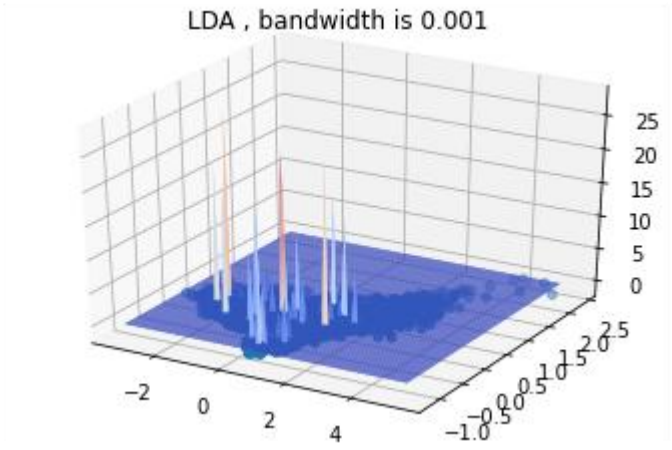
We define a function that we can pass a bandwidth parameter to change the bandwidth using LDAto reduce the data dimension.

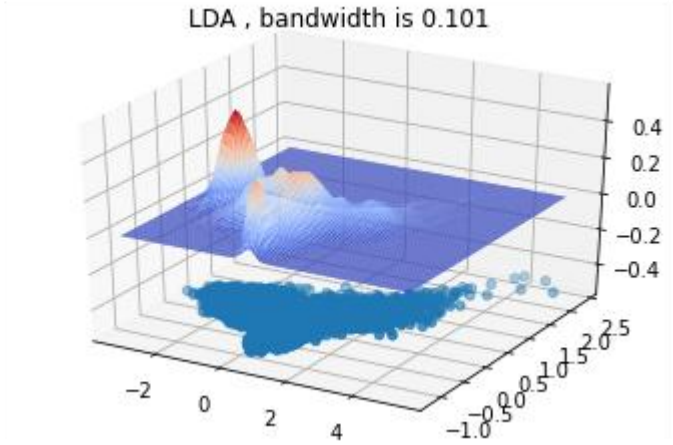
```
def text1(bd):
  from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
  lda = LDA(n_components=2).fit(textures[:,0:40],textures[:,40]).transform(textures[:,:40])
  # set grid for visualization
  grid_size = 100
  mx = min(texturesp[:,0])
  Mx = max(texturesp[:,0])
  my = min(texturesp[:,1])
  My = max(texturesp[:,1])
  xstep = (Mx - mx) / grid_size
  ystep = (My - my) / grid_size
  Gx = np.arange(mx, Mx, xstep)
  Gy = np.arange(my, My, ystep)
  Gx, Gy = np.meshgrid(Gx, Gy)
  ######## TO BE COMPLETED #########
  bandwidth = bd

  kde3 = KernelDensity(kernel='gaussian', bandwidth=bandwidth).fit(texturesp)
  Z= np.exp(kde3.score_samples(np.hstack(((Gx.reshape(grid_size*grid_size))[:,np.newaxis], (Gy.reshape(grid_size*grid_size)[:,np.newaxis])))))

  # display
  from mpl_toolkits.mplot3d import Axes3D
  from matplotlib import cm
  fig = plt.figure()
  ax = fig.gca(projection='3d')
  ax.plot_surface(Gx, Gy, Z.reshape(grid_size,grid_size), rstride=1,
  cstride=1, cmap=cm.coolwarm, linewidth=0,
  antialiased=True)
  ax.scatter(texturesp[:,0], texturesp[:,1], -0.5)
  plt.title(f'LDA , bandwidth is {bandwidth}')
  plt.show()

l=[i for i in np.arange(0.001,1,0.1)]
for n in l :
  text1(n)
```

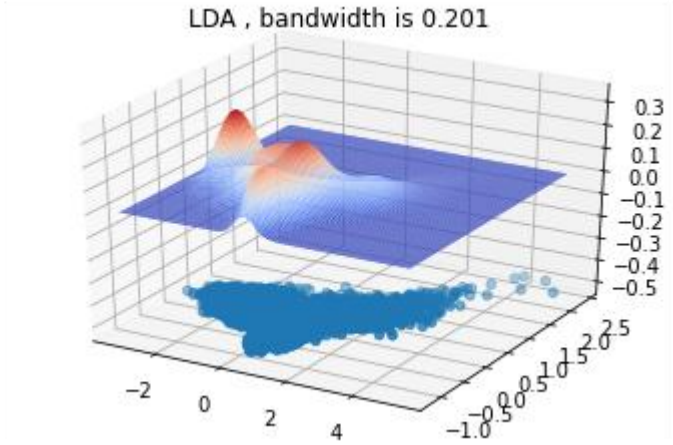
figure 9.1: Code for using LDAto transform multidimension data and estimate

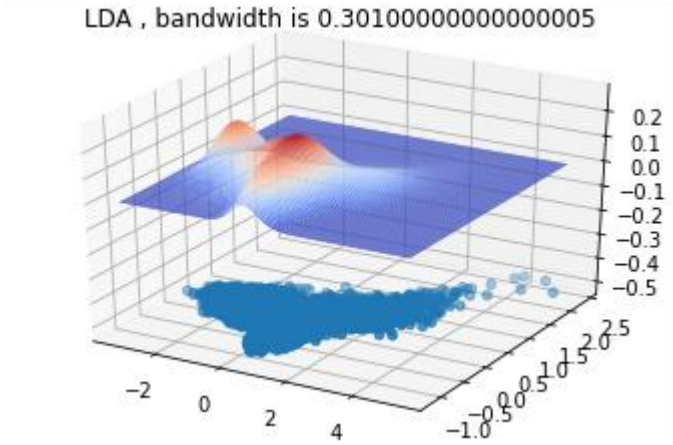
The LDA almost has the same result as PCA in analysising the texture.dat dataset.

As you can see in the plot, when bandwidth is extremly low, just few spikes in the space. When the bandwidth is around 0.1, the plot display its features. When the bandwidth is around 0.2, the plot is smooth and the plot is become flat after 0.2.




LDA , bandwidth is 0.101
0.4
0.2
0.0
−0.2
−0.4
−2
0
2
4
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0


LDA , bandwidth is 0.201
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−2
0
2
4
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0


LDA , bandwidth is 0.30100000000000005
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−2
0
2
4
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0

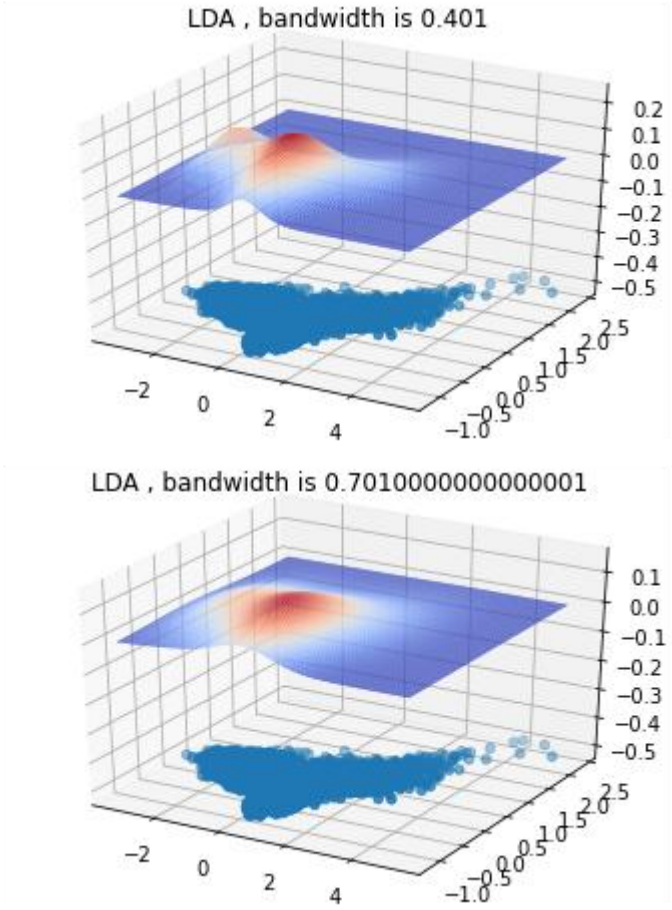


figure 9.2: Plots with different bandwidth using LDA