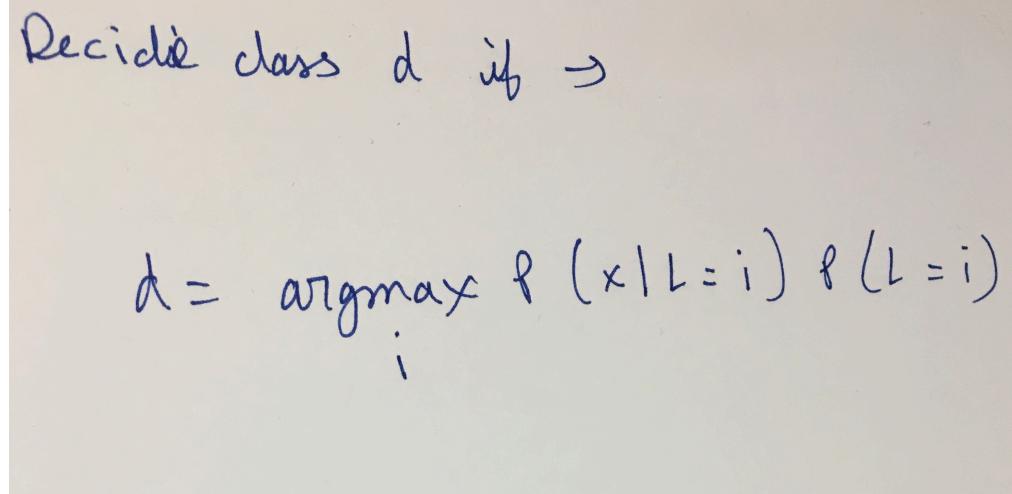


Github repo for this exam- https://github.com/rohinarora/EECE5644-Machine_Learning/tree/master/Exam1

Additionally all the code and figures are present in this file, and also attached on blackboard as additional reference Please refer whatever you find convenient

Answer 1 Github https://github.com/rohinarora/EECE5644-Machine_Learning/blob/master/Exam1/Q1.ipynb

- 3 Gaussian classes are generated by applying linear transformations to a gaussian with zero mean and I (identity) covariance.
- Minimizing $P(\text{error})$ implies MAP estimate.
- For MAP, we need to select the class that maximizes un-normalized posterior-
$$\text{decide class } D = \underset{i}{\operatorname{argmax}} P(x|L=i)P(L=i)$$



Decide class d if \rightarrow

$$d = \underset{i}{\operatorname{argmax}} f(x|L=i) f(L=i)$$

- Number of samples generated from each class-

Class 1- 1492

Class 2- 3550

Class 3- 4958

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
In [2]: def multivariate_normal(x, d, mean, covariance):
    """pdf of the multivariate normal distribution."""
    x_m = x - mean
    return (1. / (np.sqrt((2 * np.pi)**d * np.linalg.det(covariance))) *
           np.exp(-(np.linalg.solve(covariance, x_m).T.dot(x_m)) / 2))
def highlight_max(data, color='yellow'):
    """
    highlight the maximum in a Series or DataFrame
    """

    attr = 'background-color: {}'.format(color)
    if data.ndim == 1: # Series from .apply(axis=0) or axis=1
        is_max = data == data.max()
        return [attr if v else '' for v in is_max]
    else: # from .apply(axis=None)
        is_max = data == data.max().max()
        return pd.DataFrame(np.where(is_max, attr, ''),
                           index=data.index, columns=data.columns)
```

```
In [187]: def generate_GMM_samples(prior,number_of_samples,sig1,sig2,sig3,u1,u2,u3):
    ...
    Args:
        prior of class 1= prior[0]
        prior of class 2 = prior[1]
        prior of class 3 = 1-prior[0]-prior[1]

        number_of_samples

        class 1- u_1, sig_1
        class 2- u_2, sig_2
        class 2- u_3, sig_3

    x is samples from zero-mean identity-covariance Gaussian sample generators

    generating class 1- A1*x+b1
    generating class 2- A2*x+b2
    generating class 2- A3*x+b3

    Minimizing P(error) implies MAP estimate.

    For MAP, we need to find un-normalized posterior- P(x|L=i)P(L=i). Let us call it Check(i) for class i.

    Check(L=1)=N(u_1, sig_1)*Prior(L=1)
    Check(L=2)=N(u_2, sig_2)*Prior(L=2)
    Check(L=3)=N(u_3, sig_3)*Prior(L=3)

    Decide class i where Check(L=i) is greatest out of i=1 to 3

    Count the total misclassifications for class 1, class 2 and class 3

    P(error)= Total errors/Total samples
                = Total errors/10000

    ...

from matplotlib.pyplot import figure
txt="Plot of data sampled from 3 gaussians along with their MAP estimated class labels."
fig = plt.figure(figsize=(20,20));
fig.text(.35,0.09,txt,fontsize=15);
samples_class1=[ ]
samples_class2=[ ]
samples_class3=[ ]
sig_1=np.matrix(sig1)
sig_2=np.matrix(sig2)
sig_3=np.matrix(sig3)
u_1=np.matrix(u1).transpose()
u_2=np.matrix(u2).transpose()
u_3=np.matrix(u3).transpose()
prior=prior
A1=np.linalg.cholesky(sig_1)
b1=u_1
```

```

A2=np.linalg.cholesky(sig_2)
b2=u_2

A3=np.linalg.cholesky(sig_3)
b3=u_3

zero_mean=[0,0]
cov=[[1,0],[0,1]]
for i in range(number_of_samples):
    uniform_sample=np.random.uniform()

    sample_from_zero_mean_identity_covariance=np.random.multivariate
_normal(zero_mean,cov,[1]).transpose()

    if uniform_sample<prior[0]:
        '''sample from class class 1'''
        sample=A1.dot(sample_from_zero_mean_identity_covariance)+b1
        samples_class1.append(sample)
    elif uniform_sample>(prior[1]+prior[0]):
        '''sample from class class 3'''
        sample=A3.dot(sample_from_zero_mean_identity_covariance)+b3
        samples_class3.append(sample)
    else:
        '''sample from class class 2'''
        sample=A2.dot(sample_from_zero_mean_identity_covariance)+b2
        samples_class2.append(sample)

samples_class1_final=np.hstack(samples_class1)
samples_class2_final=np.hstack(samples_class2)
samples_class3_final=np.hstack(samples_class3)
a=np.squeeze(np.asarray(samples_class1_final.transpose()[:,1]))
b=np.squeeze(np.asarray(samples_class1_final.transpose()[:,0]))

c=np.squeeze(np.asarray(samples_class2_final.transpose()[:,1]))
d=np.squeeze(np.asarray(samples_class2_final.transpose()[:,0]))

e=np.squeeze(np.asarray(samples_class3_final.transpose()[:,1]))
f=np.squeeze(np.asarray(samples_class3_final.transpose()[:,0]))


# A list containing entries of class prediction for labels coming fr
om class i
classify_1=[]
classify_2=[]
classify_3=[]

for a in samples_class1_final.transpose():
    pred_class_1=multivariate_normal(a.transpose(), 2, u_1, sig_1)
    pred_class_2=multivariate_normal(a.transpose(), 2, u_2, sig_2)
    pred_class_3=multivariate_normal(a.transpose(), 2, u_3, sig_3)
    prediction=0
    if (pred_class_1> pred_class_2) and (pred_class_1> pred_class_3
):
        prediction=1
    elif(pred_class_2> pred_class_3):
        prediction=2

```

```

else:
    prediction=3
    classify_1.append(prediction)

for a in samples_class2_final.transpose():
    pred_class_1=multivariate_normal(a.transpose(), 2, u_1, sig_1)
    pred_class_2=multivariate_normal(a.transpose(), 2, u_2, sig_2)
    pred_class_3=multivariate_normal(a.transpose(), 2, u_3, sig_3)
    prediction=0
    if (pred_class_1> pred_class_2) and (pred_class_1> pred_class_3):
        prediction=1
    elif(pred_class_2> pred_class_3):
        prediction=2
    else:
        prediction=3
    classify_2.append(prediction)

for a in samples_class3_final.transpose():
    pred_class_1=(multivariate_normal(a.transpose(), 2, u_1, sig_1))
*prior[0]
    pred_class_2=(multivariate_normal(a.transpose(), 2, u_2, sig_2))
*prior[1]
    pred_class_3=(multivariate_normal(a.transpose(), 2, u_3, sig_3))
*(1-prior[0]-prior[1])
    prediction=0
    if (pred_class_1> pred_class_2) and (pred_class_1> pred_class_3):
        prediction=1
    elif(pred_class_2> pred_class_3):
        prediction=2
    else:
        prediction=3
    classify_3.append(prediction)
df_main=pd.DataFrame(data=[[np.sum(np.array(classify_1)==1),np.sum(n
p.array(classify_2)==1),np.sum(np.array(classify_3)==1)],\
                           [np.sum(np.array(classify_1)==2),np.sum(np.array(clas
sify_2)==2),np.sum(np.array(classify_3)==2)],\
                           [np.sum(np.array(classify_1)==3),np.sum(np.array(clas
sify_2)==3),np.sum(np.array(classify_3)==3)]],columns=[1,2,3])
df_main.index=[1,2,3]

a=np.squeeze(np.asarray(samples_class1_final.transpose()[:,1]))
b=np.squeeze(np.asarray(samples_class1_final.transpose()[:,0]))

c=np.squeeze(np.asarray(samples_class2_final.transpose()[:,1]))
d=np.squeeze(np.asarray(samples_class2_final.transpose()[:,0]))

e=np.squeeze(np.asarray(samples_class3_final.transpose()[:,1]))
f=np.squeeze(np.asarray(samples_class3_final.transpose()[:,0]))
#plt.scatter(b,a,color='r',marker='*',label='class 1',s=50)
#plt.scatter(d,c,color='g',marker='*',label='class 2',s=50)
#plt.scatter(f,e,color='b',marker='*',label='class 3',s=50)

df=pd.DataFrame(data= { 'x' : np.squeeze(np.asarray(samples_class3_fin

```

```

al.T[:,0]).tolist(),\
        'y': np.squeeze(np.asarray(samples_class3_final.T[:,1])).tolist(),\
            'label': np.squeeze(np.asarray(classify_3)))}
    plt.scatter((df[df['label']==3]['x']).values,(df[df['label']==3]['y']).values,label="True label class 3, Decision class 3",marker='^')
    plt.scatter((df[df['label']==2]['x']).values,(df[df['label']==2]['y']).values,label="True label class 3, Decision class 2",marker='*',s=80)
    plt.scatter((df[df['label']==1]['x']).values,(df[df['label']==1]['y']).values,label="True label class 3, Decision class 1",marker='*',s=80)

    df=pd.DataFrame(data= {'x': np.squeeze(np.asarray(samples_class2_final.T[:,0])).tolist(),\
        'y': np.squeeze(np.asarray(samples_class2_final.T[:,1])).tolist(),\
            'label': np.squeeze(np.asarray(classify_2)))}
    plt.scatter((df[df['label']==3]['x']).values,(df[df['label']==3]['y']).values,label="True label class 2, Decision class 3",marker='*',s=80)
    plt.scatter((df[df['label']==2]['x']).values,(df[df['label']==2]['y']).values,label="True label class 2, Decision class 2",marker='^')
    plt.scatter((df[df['label']==1]['x']).values,(df[df['label']==1]['y']).values,label="True label class 2, Decision class 1",marker='*',s=80)

    df=pd.DataFrame(data= {'x': np.squeeze(np.asarray(samples_class1_final.T[:,0])).tolist(),\
        'y': np.squeeze(np.asarray(samples_class1_final.T[:,1])).tolist(),\
            'label': np.squeeze(np.asarray(classify_1)))}
    plt.scatter((df[df['label']==3]['x']).values,(df[df['label']==3]['y']).values,label="True label class 1, Decision class 3",marker='*',s=80)
    plt.scatter((df[df['label']==2]['x']).values,(df[df['label']==2]['y']).values,label="True label class 1, Decision class 2",marker='*',s=80)
    plt.scatter((df[df['label']==1]['x']).values,(df[df['label']==1]['y']).values,label="True label class 1, Decision class 1",marker='^')

plt.legend();
plt.title('Trivariate gaussian samples',fontsize=15)
plt.xlabel('x1',fontsize=20)
plt.ylabel('x2',fontsize=20);
plt.legend()
plt.show();
print ("Samples from class 1 - ", samples_class1_final.shape[1])
print ("Samples from class 2 - ", samples_class2_final.shape[1])
print ("Samples from class 3 - ", samples_class3_final.shape[1])
from matplotlib.pyplot import figure
txt="Plot of data sampled 1 gaussian along with their MAP estimates."
fig = plt.figure(figsize=(20,20));
fig.text(.35,0.09,txt,fontsize=15);
df=pd.DataFrame(data= {'x': np.squeeze(np.asarray(samples_class1_final.T[:,0])).tolist(),\
        'y': np.squeeze(np.asarray(samples_class1_final.T[:,1])).tolist(),\
            'label': np.squeeze(np.asarray(classify_1)))}

```

```

        'label': np.squeeze(np.asarray(classify_1)))})
    plt.scatter((df[df['label']==3]['x']).values,(df[df['label']==3]['y']
]).values,label="True label class 1, Decision class 3",marker='*',s=80)
    plt.scatter((df[df['label']==2]['x']).values,(df[df['label']==2]['y'
]).values,label="True label class 1, Decision class 2",marker='*',s=80)
    plt.scatter((df[df['label']==1]['x']).values,(df[df['label']==1]['y'
]).values,label="True label class 1, Decision class 1",marker='^')

plt.legend();
plt.title('Gaussian samples from class 1, along with MAP decision la
bels',fontsize=15)
plt.xlabel('x1',fontsize=20)
plt.ylabel('x2',fontsize=20);
plt.legend()
plt.show();

print ('\n\n\n')

from matplotlib.pyplot import figure
txt="Plot of data sampled 1 gaussian along with their MAP estimates.
"
fig = plt.figure(figsize=(20,20));
fig.text(.35,0.09,txt,fontsize=15);
df=pd.DataFrame(data= {'x': np.squeeze(np.asarray(samples_class2_fin
al.T[:,0])).tolist(),\
                      'y': np.squeeze(np.asarray(samples_class2_final.T[:,1])).tolist(),
                     'label': np.squeeze(np.asarray(classify_2)))})
    plt.scatter((df[df['label']==3]['x']).values,(df[df['label']==3]['y'
]).values,label="True label class 2, Decision class 3",marker='*',s=80)
    plt.scatter((df[df['label']==2]['x']).values,(df[df['label']==2]['y'
]).values,label="True label class 2, Decision class 2",marker='^')
    plt.scatter((df[df['label']==1]['x']).values,(df[df['label']==1]['y'
]).values,label="True label class 2, Decision class 1",marker='*',s=80)

plt.legend();
plt.title('Gaussian samples from class 2, along with MAP decision la
bels',fontsize=15)
plt.xlabel('x1',fontsize=20)
plt.ylabel('x2',fontsize=20);
plt.legend()
plt.show();

print ('\n\n\n')
from matplotlib.pyplot import figure
txt="Plot of data sampled 1 gaussian along with their MAP estimates.
"
fig = plt.figure(figsize=(20,20));
fig.text(.35,0.09,txt,fontsize=15);
df=pd.DataFrame(data= {'x': np.squeeze(np.asarray(samples_class3_fin
al.T[:,0])).tolist(),\
                      'y': np.squeeze(np.asarray(samples_class3_final.T[:,1])).tolist(),
                     'label': np.squeeze(np.asarray(classify_3)))})
    plt.scatter((df[df['label']==3]['x']).values,(df[df['label']==3]['y'
])

```

```

]).values,label="True label class 3, Decision class 3",marker='^')
    plt.scatter((df[df['label']==2]['x']).values,(df[df['label']==2]['y']
]).values,label="True label class 3, Decision class 2",marker='*',s=80)
    plt.scatter((df[df['label']==1]['x']).values,(df[df['label']==1]['y'
]).values,label="True label class 3, Decision class 1",marker='*',s=80)

plt.legend();
plt.title('Gaussian samples from class 3, along with MAP decision la
bels',fontsize=15)
plt.xlabel('x1',fontsize=20)
plt.ylabel('x2',fontsize=20);
plt.legend()
plt.show();
return df_main
'''

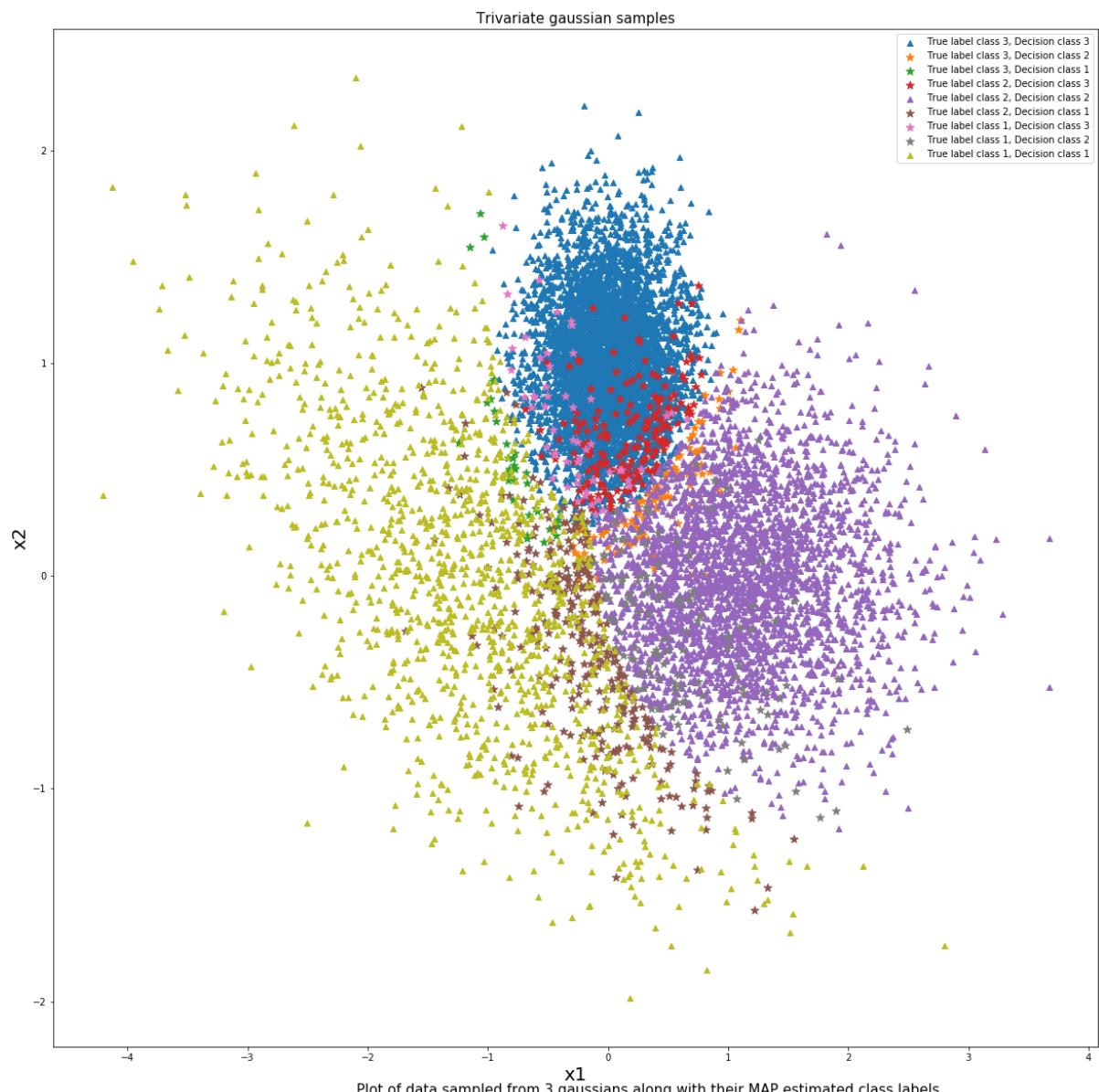
if len(mis_classify)==0:
    pass
else:
    class1_mis_classify=np.vstack(mis_classify).transpose()
    a=np.squeeze(np.asarray(class1_mis_classify.transpose()[:,1]))
    b=np.squeeze(np.asarray(class1_mis_classify.transpose()[:,0]))
    plt.scatter(a,b,color='b',marker='x',label='class 1 labels miscl
assified',s=100)

mis_classify=[]
for a in samples_class2_final.transpose():
    b= multivariate_normal(a.transpose(), 2, u_1, sig_1)*prior > multivariate_
normal(a.transpose(), 2, u_2, sig_2)*(1-prior)
    if b==True:
        mis_classify.append(a)
    num_class2_mis_classify= (len(mis_classify))

if len(mis_classify)==0:
    pass
else:
    class2_mis_classify=np.vstack(mis_classify).transpose()
    a=np.squeeze(np.asarray(class2_mis_classify.transpose()[:,1]))
    b=np.squeeze(np.asarray(class2_mis_classify.transpose()[:,0]))
    plt.scatter(a,b,color='r',marker='H',label='class 2 labels miscl
assified',s=100);
    errors= num_class1_mis_classify+num_class2_mis_classify
    print ("P(error) = ",errors/400)
''' ;

```

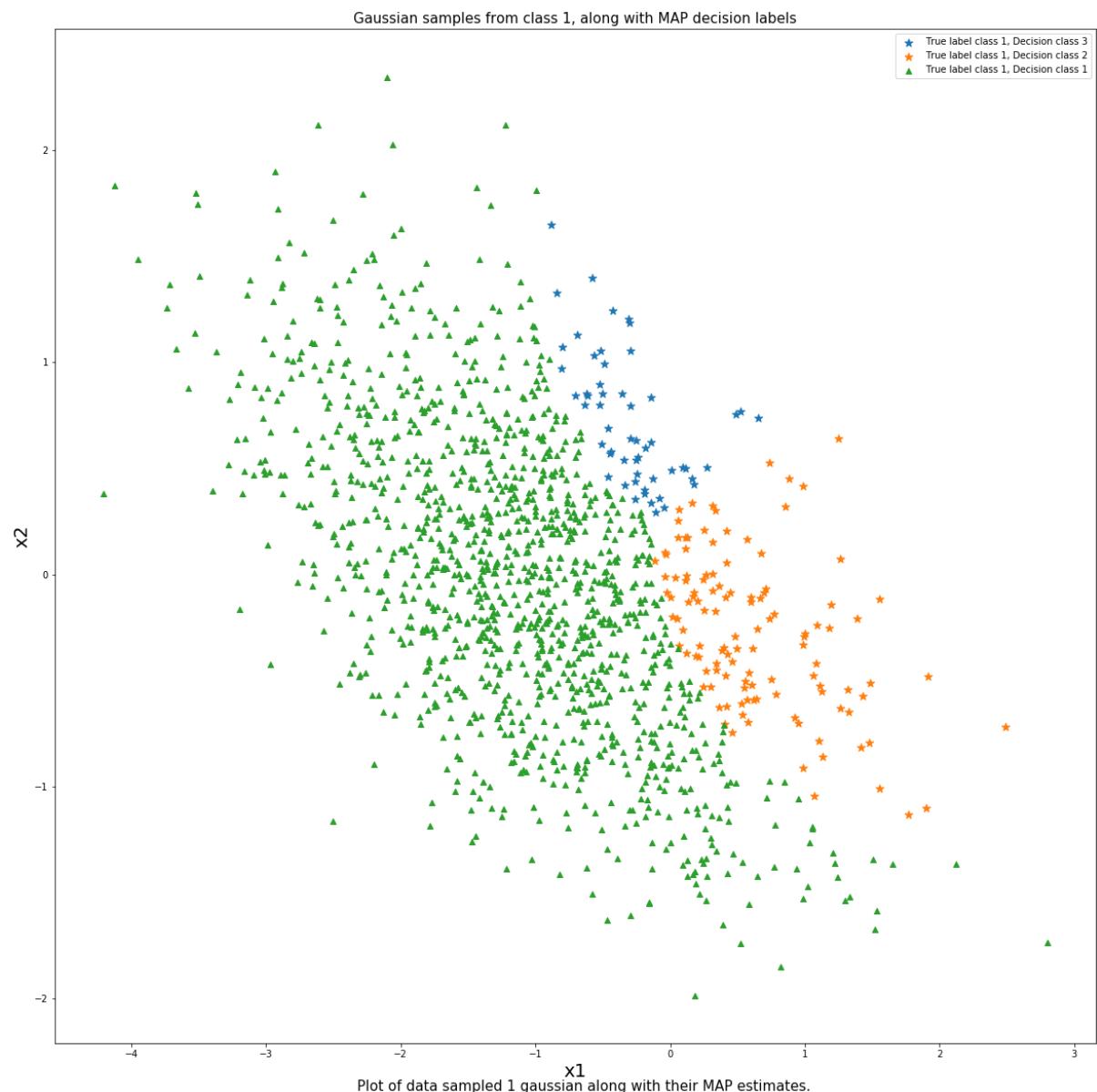
```
In [188]: df=generate_GMM_samples(prior=[0.15,0.35],number_of_samples=10000,sig1=[[1,-0.4],[-0.4,0.5]],sig2=[[0.5,0],[0,0.2]],sig3=[[0.1,0],[0,0.1]],u1=[-1,0],u2=[1,0],u3=[0,1]);
```



Samples from class 1 - 1445

Samples from class 2 - 3504

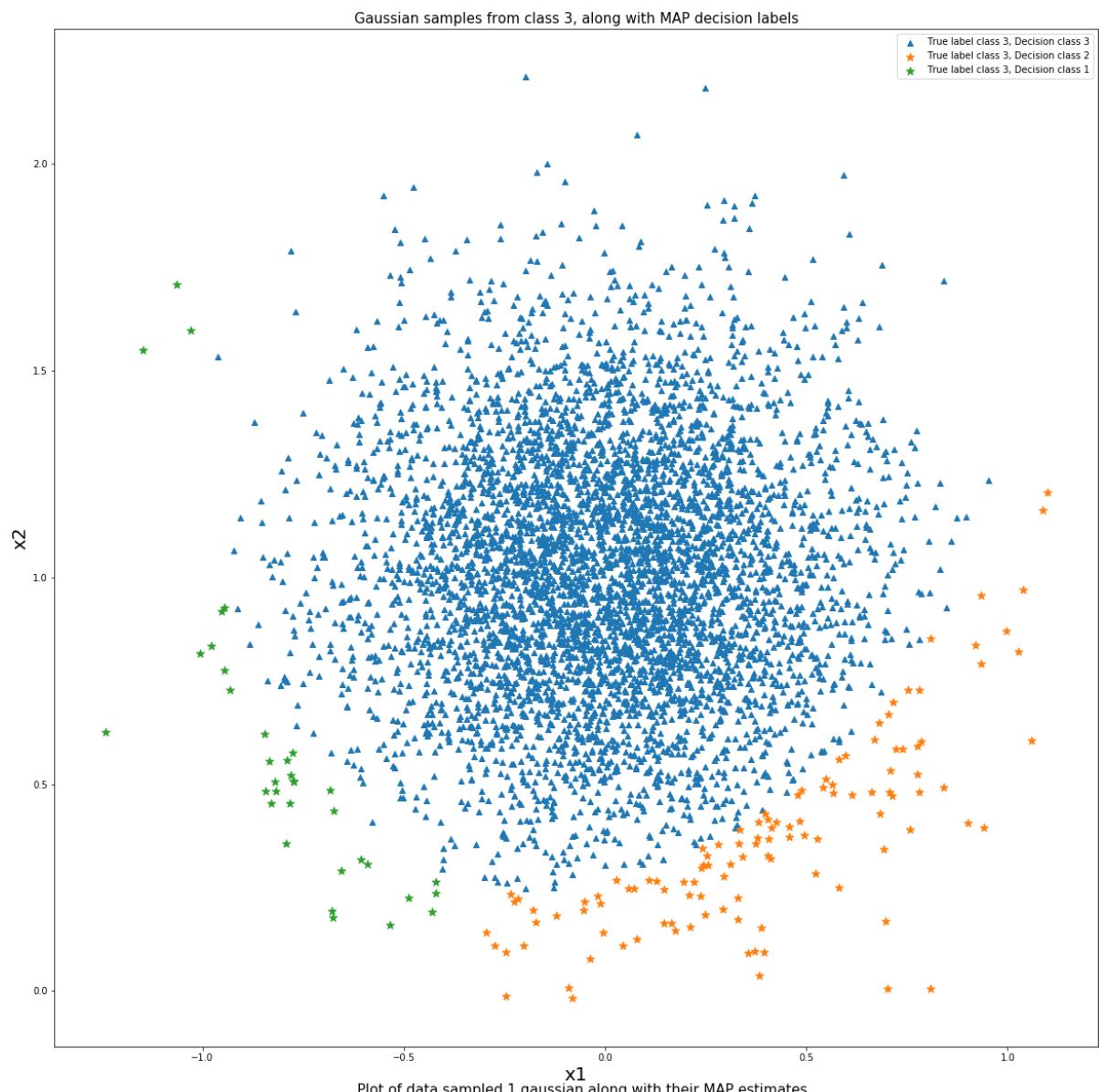
Samples from class 3 - 5051



\n



\n



```
In [189]: print ("Columns have the True class; \nand Rows have decided class. \n Confusion matrix below")
df.style.apply(highlight_max)
```

Columns have the True class;
and Rows have decided class.
Confusion matrix below

Out[189]:

	1	2	3
1	1272	300	35
2	118	3016	116
3	55	188	4900

```
In [190]: print("Total samples misclassified : ",df.mask(np.eye(3, dtype = bool)).sum().sum())
print("P(error) : ",(df.mask(np.eye(3, dtype = bool)).sum().sum())/100,
      "%")
```

```
Total samples misclassified : 812.0
P(error)%: 8.12 % or P(error)= 0.0812
```

With just 8% P(error), MAP classifier correctly classifies majority of the points. Samples where class conditional of true class label is high, are correctly classified. As the points are fairly distributed, majority of the points are that way.

For eg. consider class i.

$\text{Posterior}(i) = P(x|\text{class }=i)^* P(i)$.

This is compared against Posterior of all classes j,k,a, $j \neq i$

The class conditional ($P(x|\text{class }=i)$) is gaussian. Exponential decaying function. As separation of classes increases, "class conditional" is the main dominating term. As the exponentials evaluated at large distances decay way faster than difference in class priors of class i and j. Class priors are fixed, and roughly on the same order for all classes. Since the points are fairly scattered, MAP classifier gives just 8% P(error)

```
In [ ]:
```

Answer 2

Github- https://github.com/rohinarora/EECE5644-Machine_Learning/blob/master/Exam1/Q2.ipynb

Answer 2

Let $\vec{\theta} = \begin{bmatrix} x \\ y \end{bmatrix}$

prior of $\vec{\theta} \rightarrow$

$$P \begin{bmatrix} x \\ y \end{bmatrix} = (2\pi\sigma_x^2\sigma_y^2)^{-1} e^{-\frac{1}{2} \begin{bmatrix} x \\ y \end{bmatrix}^T \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \end{bmatrix}}$$

To find $\vec{\theta}_{MAP} = \underset{\theta}{\operatorname{argmax}} P(\vec{\theta} | \vec{R})$

where $\vec{R} = (r_1, r_2, \dots, r_K) \rightarrow K$ measurements

$$r_i = d_{ri} + n_i$$

$$d_{ri} = \sqrt{(x_{ri} - x_i)^2 + (y_{ri} - y_i)^2}$$

where (x_i, y_i) is the sensor position (known to us)

$$\theta_{MAP} = \operatorname{argmax}_{\theta} P(\vec{\theta} | \vec{R})$$

$$= \operatorname{argmax}_{\theta} \frac{P(\vec{R} | \vec{\theta}) P(\vec{\theta})}{P(\vec{R})} \quad \text{Bayes Rule}$$

$$= \operatorname{argmax}_{\theta} P(\vec{R} | \vec{\theta}) P(\vec{\theta})$$

we can take log of probability as log is increasing funct.

$$\theta_{MAP} = \operatorname{argmax}_{\theta} \left[\log P(\vec{R} | \vec{\theta}) + \log P(\vec{\theta}) \right]$$

\downarrow
 $v_1, v_2, v_3, \dots, v_n$

$$= \operatorname{argmax}_{\theta} \left[\cancel{\log \prod_{i=1}^k} P(v_i | \vec{\theta}) + \log P(\vec{\theta}) \right]$$

\downarrow
observations are independent

$$= \operatorname{argmax}_{\theta} \left[\sum_{i=1}^k \log P(v_i | \vec{\theta}) + \log P(\vec{\theta}) \right]$$

$$= \underset{\theta}{\operatorname{argmax}} \left[\sum_{i=1}^K \log p(r_i | \theta) + \log \left((2\pi\sigma_x\sigma_y)^{-1} e^{-\frac{1}{2} \begin{bmatrix} x \\ y \end{bmatrix}^T \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}} \right) \right]$$

$$r_i | \theta = \sqrt{(x_i - x)^2 + (y_i - y)^2} + \text{constant}$$

$$r_i \sim N(0, \sigma_i^2)$$

$$\therefore M_{r_i | \theta} = \sqrt{(x_i - x)^2 + (y_i - y)^2} = d_{ix,y} = d(x, y)$$

$$\sigma_{r_i | \theta} = \sigma_i$$

$$\therefore r_i | \theta \sim N(d_{ix,y}, \sigma_i^2)$$

$\therefore r_i | \theta$ is a gaussian with mean $d_{ix,y}$ and variance σ_i^2 .

$$\Rightarrow \theta_{MAP} = \underset{\theta}{\operatorname{argmax}} \left[\sum_{i=1}^K \log \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(r_i - d_{ix,y})^2}{2\sigma_i^2}} + \log (2\pi\sigma_x\sigma_y)^{-1} - \frac{1}{2} \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \end{bmatrix} \right]$$

independent of (x, y)

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmax}} \left[\sum_{i=1}^K \left[\log \frac{1}{\sqrt{2\pi\sigma_i^2}} - \frac{(r_i - d_i(x,y))^2}{2\sigma_i^2} \right] \right]$$

independent of θ

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmax}} \left[\sum_{i=1}^K \left[- \frac{(r_i - ((x_i - x)^2 + (y_i - y)^2))}{2\sigma_i^2} \right] \right]$$

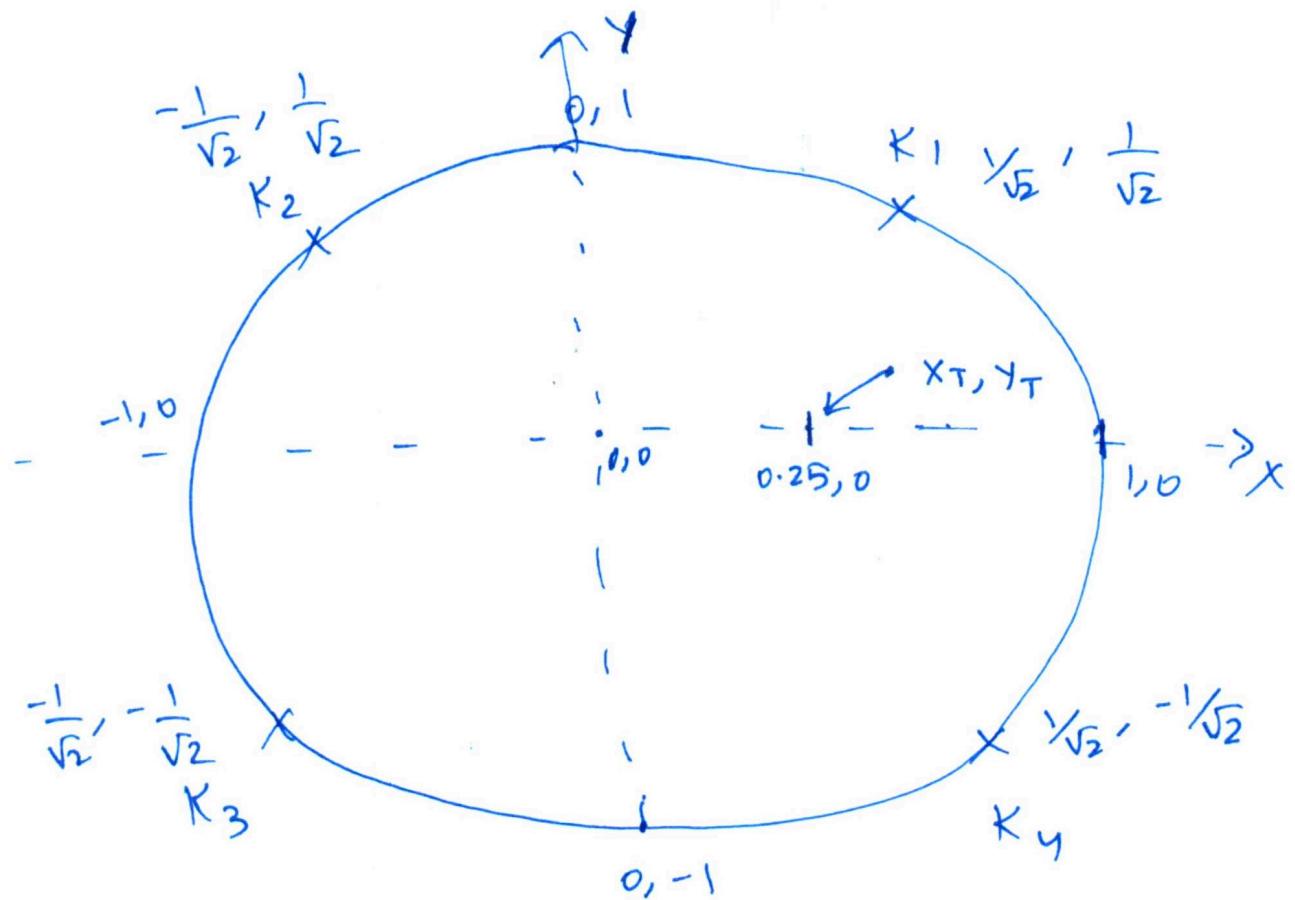
$$- \frac{1}{2} \left[\begin{matrix} x & y \end{matrix} \right] \left[\begin{matrix} \frac{1}{\sigma_x^2} & 0 \\ 0 & \frac{1}{\sigma_y^2} \end{matrix} \right] \left[\begin{matrix} x \\ y \end{matrix} \right]$$

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmin}} \left[\sum_{i=1}^K \frac{(r_i - \sqrt{(x_i - x)^2 + (y_i - y)^2})^2}{\sigma_i^2} + \left(\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2} \right) \right]$$

where $\theta = [x \ y]^T$

solving this optimization will give θ_{MAP} or $\begin{bmatrix} x_{MAP} \\ y_{MAP} \end{bmatrix}$

Visualization of the data-



The code is well commented explaining the steps

sigma_x=0.1

sigma_y=0.1

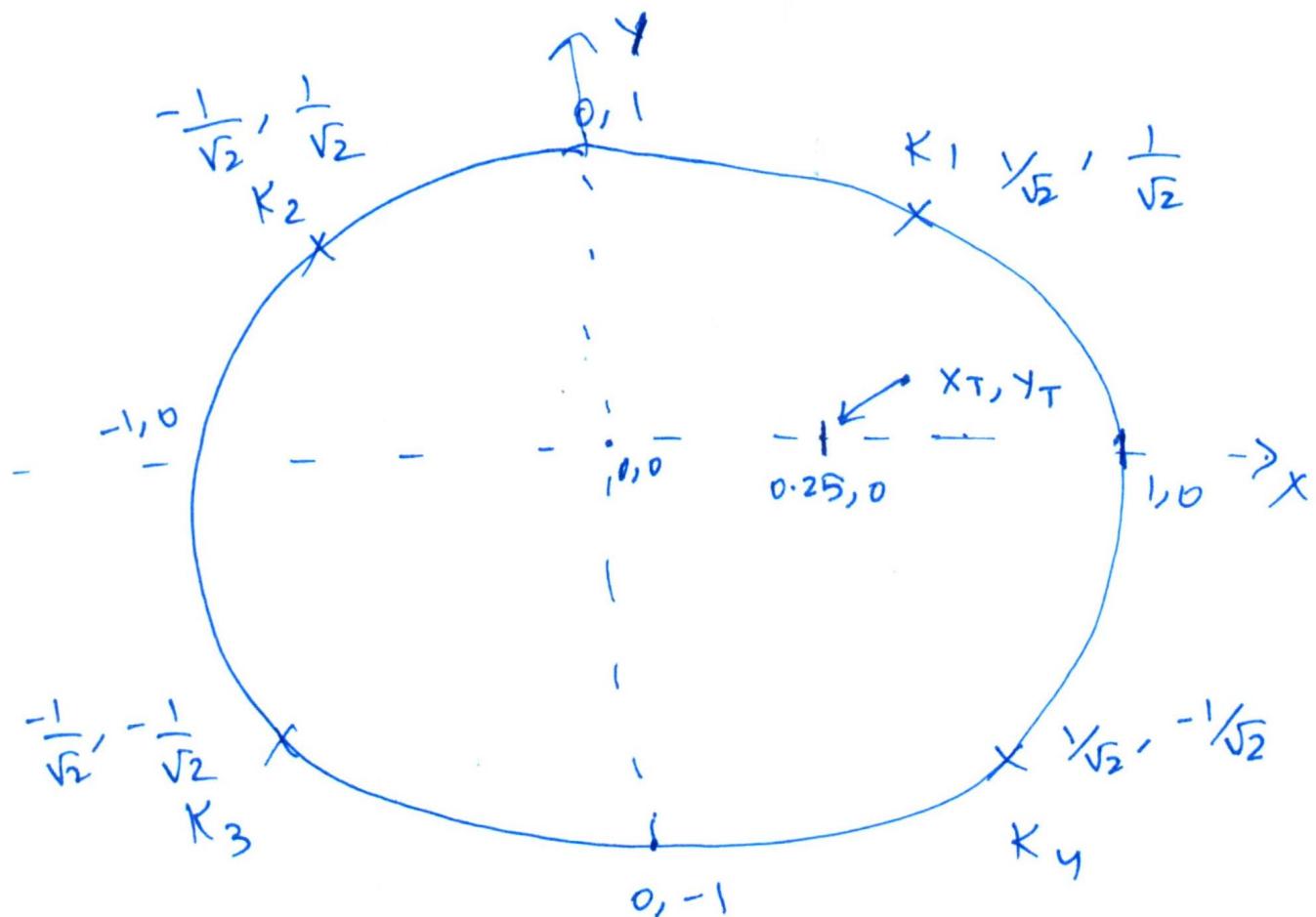
sigma_i=.01

In [2]:

```
''
imports
''

import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import sympy as sym
```

Viz of the data-



```
In [3]: '''
Position the sensor. k1-k4 represent the 4 sensors
'''

k1=(1/np.sqrt(2),1/np.sqrt(2))
k2=(-1/np.sqrt(2),1/np.sqrt(2))
k3=(-1/np.sqrt(2),-1/np.sqrt(2))
k4=(1/np.sqrt(2),-1/np.sqrt(2))

'''
True location of the object
'''

xy_true=(0.2,0.3)

sensors=[k1,k2,k3,k4] # make a list of sensors

# set the sigma_x, sigma_y, sigma_i
sigma_x=0.1
sigma_y=0.1
sigma_i=.01 #0.3 for submission
```

```
In [4]: def dist_points(p1,p2):
    '''
    L2 distance between points p1 and p2
    Assumes p1 and p2 are both tuples (x,y)
    '''

    return ((p1[0]-p2[0])**2+(p1[1]-p2[1])**2)**(0.5)
```

```
In [5]: def measure_value(k):
    '''
    measure the distance between sensor k and true position "xy_true". Add gaussian noise with std sigma_i to it
    '''

    measurement=dist_points(k,xy_true)+np.random.normal(scale=sigma_i)
    if measurement>0:
        return measurement
    else:
        measure_value(k)
```

In [6]:

```
''
Calculate the distance of sensor to true position "xy_true", by calling
the function measure_value
Store the resulting values in a dictionary
'''
```

```
sensor_measurements=dict()
sensor_measurements[k1]=measure_value(k1)
sensor_measurements[k2]=measure_value(k2)
sensor_measurements[k3]=measure_value(k3)
sensor_measurements[k4]=measure_value(k4)
print ("sensor data is :\n", sensor_measurements)
print ('\n\n')
for j,i in enumerate(sensor_measurements):
    print ("Distance of sensor k"+str(j+1)+" from true positon",sensor_m
easurements[i])
```

sensor data is :

```
{(0.7071067811865475, 0.7071067811865475): 0.6451761942076968, (-0.707
1067811865475, 0.7071067811865475): 1.0024351553984014, (-0.70710678118
65475, -0.7071067811865475): 1.3675174669665544, (0.7071067811865475, -
0.7071067811865475): 1.1276801471635933}
```

Distance of sensor k1 from true positon 0.6451761942076968

Distance of sensor k2 from true positon 1.0024351553984014

Distance of sensor k3 from true positon 1.3675174669665544

Distance of sensor k4 from true positon 1.1276801471635933

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmin}} \left[\sum_{i=1}^k \frac{(x_i - \sqrt{(x_i - x)^2 + (y_i - y)^2})^2}{\sigma_i^2} + \left(\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2} \right) \right]$$

where $\theta = [x \ y]^T$

solving this optimization will give θ_{MAP} or $\begin{bmatrix} x_{MAP} \\ y_{MAP} \end{bmatrix}$

```
In [7]: # Pre define the contour_level. This sets the contour levels for all the plots.
contour_level=[ ]
for i in range(0,300,10):
    contour_level.append(i)

# Create a meshgrid of X-Y
x = np.linspace(-2, 2, 1000)
y = np.linspace(-2, 2, 1000)
X, Y = np.meshgrid(x, y)

# Below functions give the MAP objective values for different cases. Refer above image for the formula.
def f0(x, y):
    '''
    MAP objective if no sensor were present.
    '''
    prior=(x**2)/(sigma_x**2)+(y**2)/(sigma_y**2)
    return prior

def f1(x, y):
    '''
    MAP objective if 1 sensor was present.
    '''

    first_senor=(np.square(sensor_measurements[k1]-np.sqrt((k1[0]-x)**2+(k1[1]-y)**2)))*(1/sigma_i**2)
    prior=(x**2)/(sigma_x**2)+(y**2)/(sigma_y**2)
    return first_senor+prior

def f2(x, y):
    '''
    two sensor
    '''

    prior=(x**2)/(sigma_x**2)+(y**2)/(sigma_y**2)
    first_senor=(np.square(sensor_measurements[k1]-np.sqrt((k1[0]-x)**2+(k1[1]-y)**2)))*(1/sigma_i**2)
    second_senor=(np.square(sensor_measurements[k3]-np.sqrt((k3[0]-x)**2+(k3[1]-y)**2)))*(1/sigma_i**2)
    return first_senor+second_senor+prior

def f3(x, y):
    '''
    three sensor
    '''

    prior=(x**2)/(sigma_x**2)+(y**2)/(sigma_y**2)
    first_senor=(np.square(sensor_measurements[k1]-np.sqrt((k1[0]-x)**2+(k1[1]-y)**2)))*(1/sigma_i**2)
    second_senor=(np.square(sensor_measurements[k2]-np.sqrt((k2[0]-x)**2+(k2[1]-y)**2)))*(1/sigma_i**2)
    third_senor=(np.square(sensor_measurements[k3]-np.sqrt((k3[0]-x)**2+(k3[1]-y)**2)))*(1/sigma_i**2)
    return first_senor+second_senor+third_senor
```

```

(k3[1]-y)**2)))*(1/sigma_i**2)
    return first_senor+second_senor+third_senor+prior

def f4(x, y):
    """
    four sensor
    """

    prior=(x**2)/(sigma_x**2)+(y**2)/(sigma_y**2)
    first_senor=(np.square(sensor_measurements[k1]-np.sqrt((k1[0]-x)**2+(k1[1]-y)**2)))*(1/sigma_i**2)
    second_senor=(np.square(sensor_measurements[k2]-np.sqrt((k2[0]-x)**2+(k2[1]-y)**2)))*(1/sigma_i**2)
    third_senor=(np.square(sensor_measurements[k3]-np.sqrt((k3[0]-x)**2+(k3[1]-y)**2)))*(1/sigma_i**2)
    forth_senor=(np.square(sensor_measurements[k4]-np.sqrt((k4[0]-x)**2+(k4[1]-y)**2)))*(1/sigma_i**2)
    return first_senor+second_senor+third_senor+forth_senor+prior
    """


```

Plotting function.

*Takes in argument objective function and number of sensors
Objective function can be one of functions defined above- f0,f1,f2,f3,f4
num_sensors can be 0-4*

```

def _plot_(f,num_sensors):
    Z = f(X, Y)    # compute Z for objective function f
    from matplotlib.pyplot import figure
    fig=figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')

    # plot contour
    plt.contourf(X, Y, Z, 20, cmap='RdGy',levels=contour_level);

    # set true labels and sensor positions
    plt.plot([xy_true[0]], [xy_true[1]], marker='+', markersize=20, color="blue",label="True xy",mew=2)
    if num_sensors==1:
        plt.plot([k1[0]], [k1[1]], marker='o', markersize=10, color="black",label="Sensor 1")
    elif num_sensors==2:
        plt.plot([k1[0]], [k1[1]], marker='o', markersize=10, color="black",label="Sensor 1")
        plt.plot([k3[0]], [k3[1]], marker='o', markersize=10, color="black",label="Sensor 2")
    elif num_sensors==3:
        plt.plot([k1[0]], [k1[1]], marker='o', markersize=10, color="black",label="Sensor 1")
        plt.plot([k2[0]], [k2[1]], marker='o', markersize=10, color="black",label="Sensor 2")
        plt.plot([k3[0]], [k3[1]], marker='o', markersize=10, color="black",label="Sensor 3")


```

```
    elif num_sensors==4:  
        plt.plot([k1[0]], [k1[1]], marker='o', markersize=10, color="black",label="Sensor 1")  
        plt.plot([k2[0]], [k2[1]], marker='o', markersize=10, color="black",label="Sensor 2")  
        plt.plot([k3[0]], [k3[1]], marker='o', markersize=10, color="black",label="Sensor 3")  
        plt.plot([k4[0]], [k4[1]], marker='o', markersize=10, color="black",label="Sensor 4")  
  
        plt.xlim(-2,2)  
        plt.ylim(-2,2)  
        plt.colorbar();  
        plt.legend();  
    return fig
```

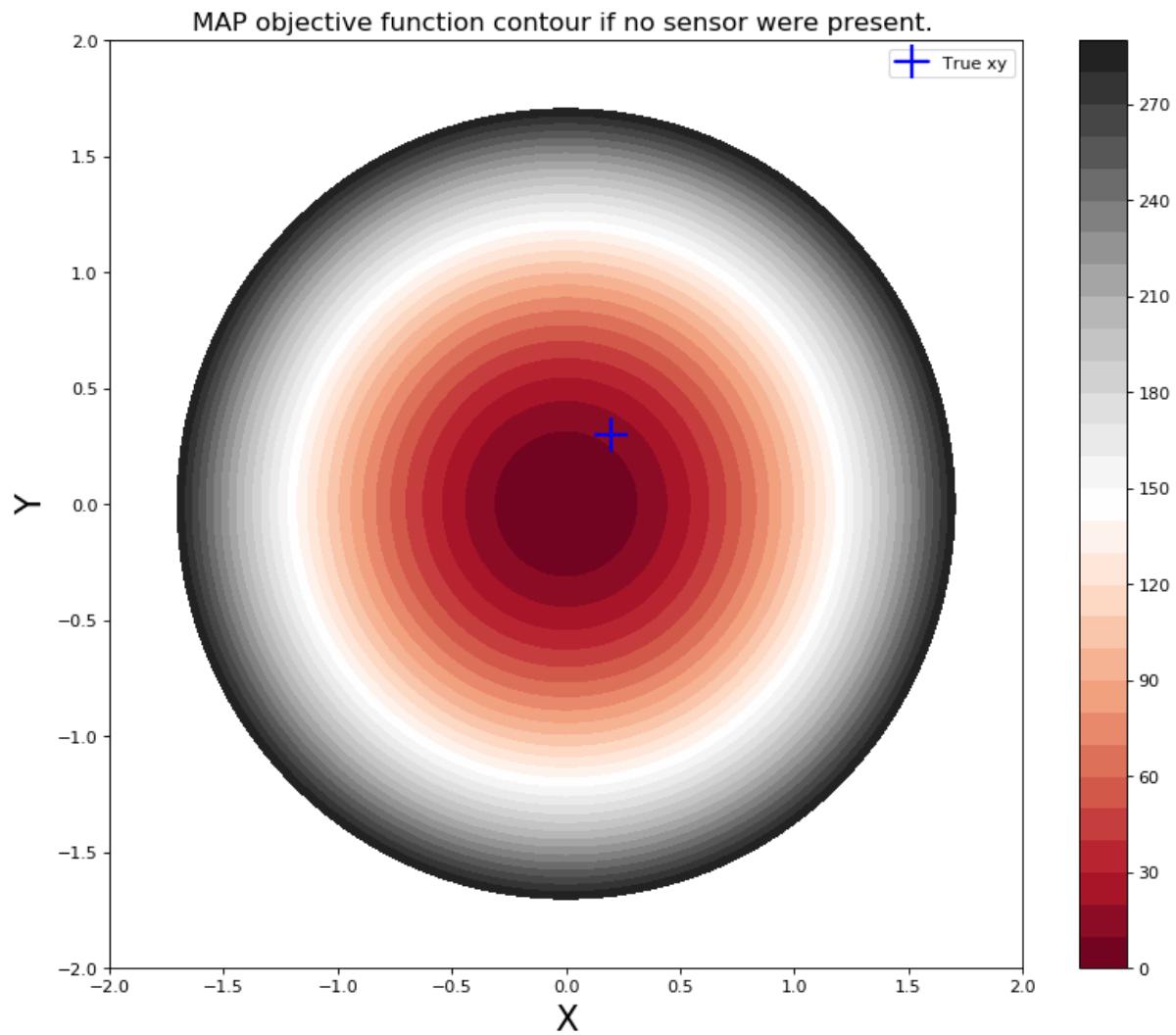
```
In [8]: fig=_plot_(f0,0)
# MAP estimate matches with prior value.
plt.title('MAP objective function contour if no sensor were present. ',fontsize=15)
plt.xlabel('X',fontsize=20)
plt.ylabel('Y',fontsize=20);
fig.text(.15,0.025,'The objective is centered at origin 0,0 -> which is
the prior value',fontsize=15);

fig=_plot_(f1,1)
plt.title('MAP objective function contour if 1 sensor was present. ',fontsize=15)
plt.xlabel('X',fontsize=20)
plt.ylabel('Y',fontsize=20);
fig.text(.15,0.025,'The objective center moves closer to the true va
lue',fontsize=15);

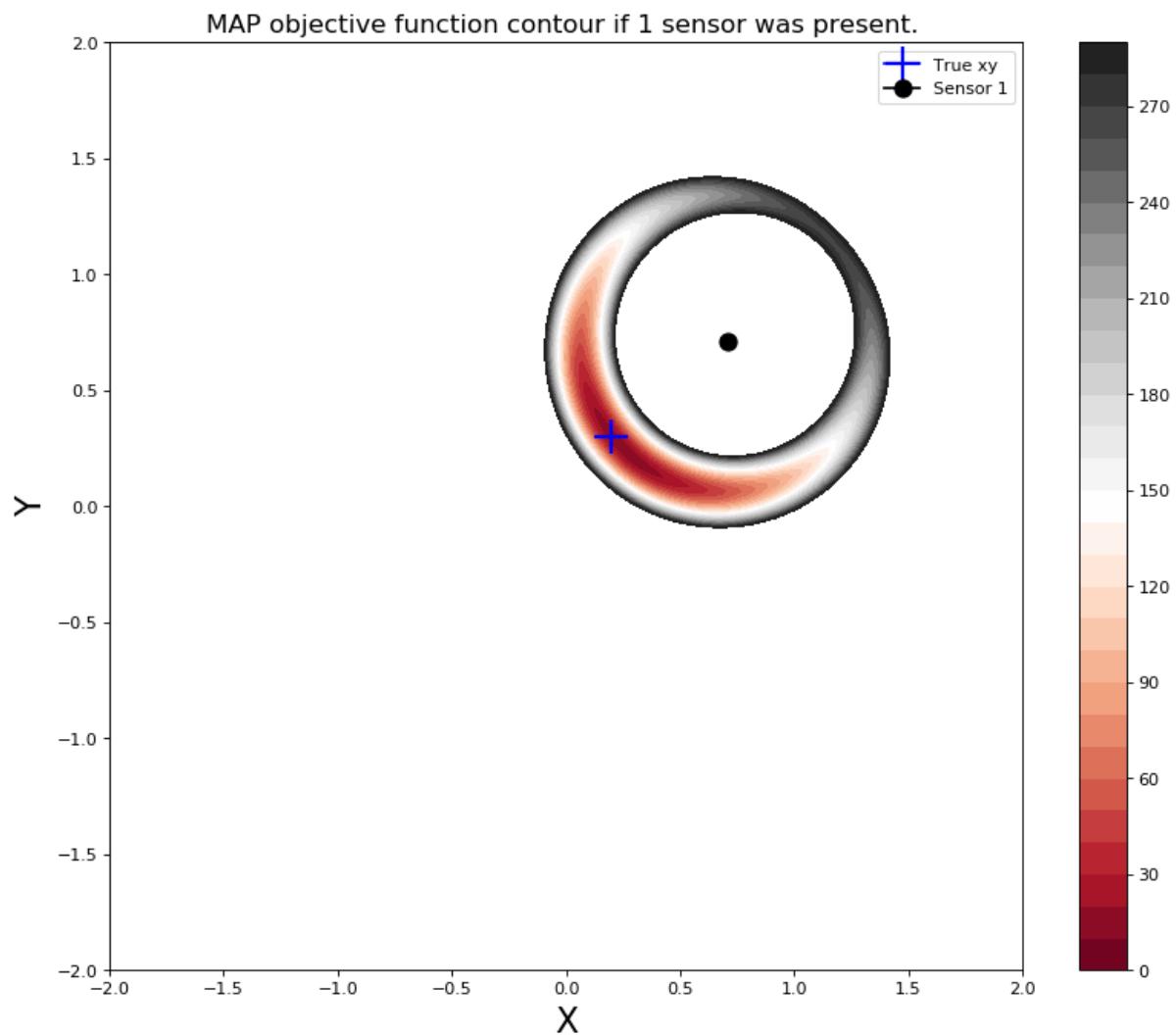
fig=_plot_(f2,2)
plt.title('MAP objective function contour if 2 sensors were present. ',fontsize=15)
plt.xlabel('X',fontsize=20)
plt.ylabel('Y',fontsize=20);
fig.text(.15,0.025,'The objective center moves further closer to the true
value',fontsize=15);

fig=_plot_(f3,3)
plt.title('MAP objective function contour if 3 sensors were present. ',fontsize=15)
plt.xlabel('X',fontsize=20)
plt.ylabel('Y',fontsize=20);
fig.text(.15,0.025,'The objective center moves very close to the true va
lue',fontsize=15);

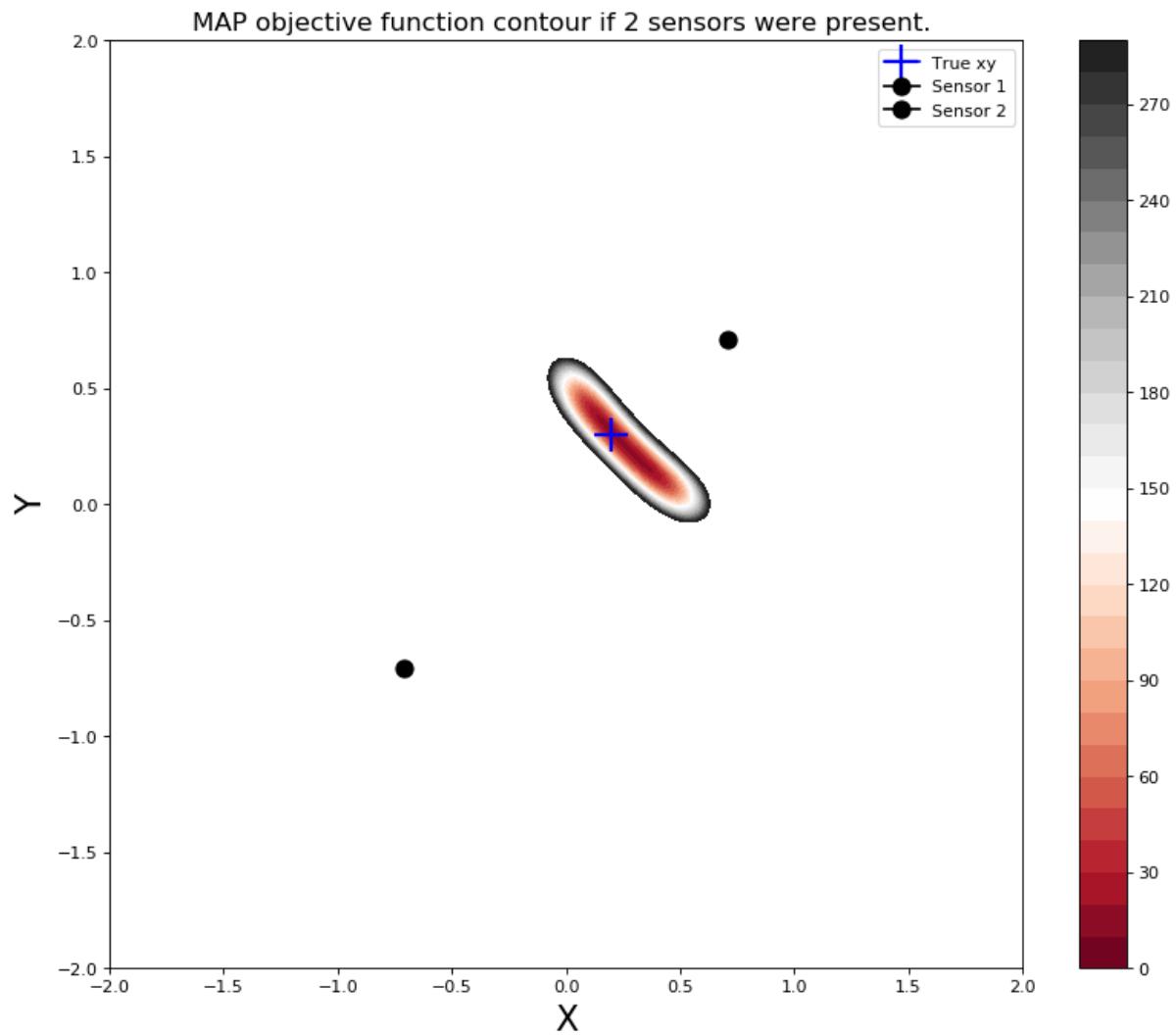
fig=_plot_(f4,4)
plt.title('MAP objective function contour if 4 sensors were present. ',fontsize=15)
plt.xlabel('X',fontsize=20)
plt.ylabel('Y',fontsize=20);
fig.text(.15,0.025,'The objective center almost coincidences to the true
value',fontsize=15);
```



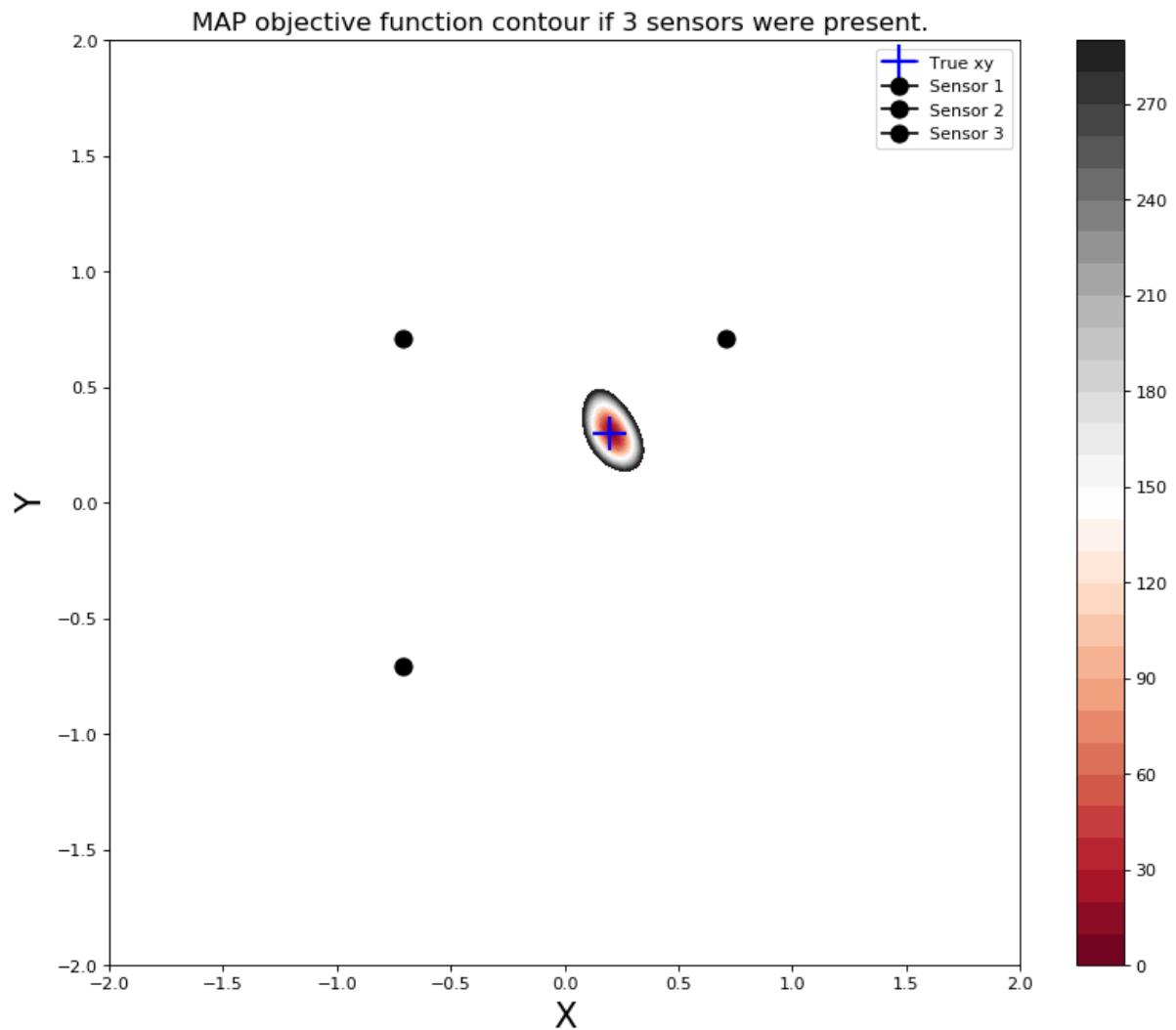
The objective is centered at origin 0,0 -> which is the prior value



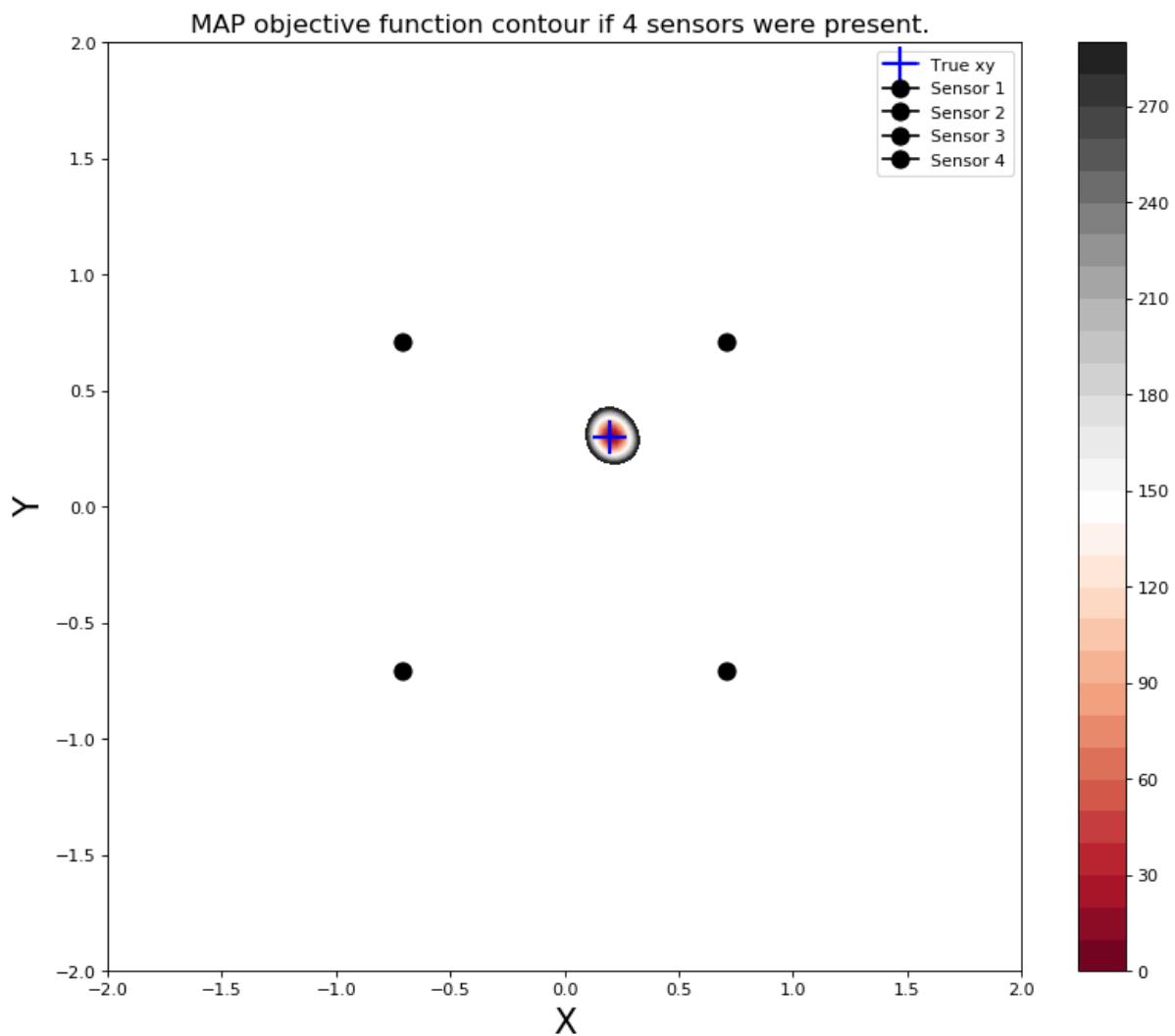
The objective center moves closer to the true value



The objective center moves further closer to the true value



The objective center moves very close to the true value

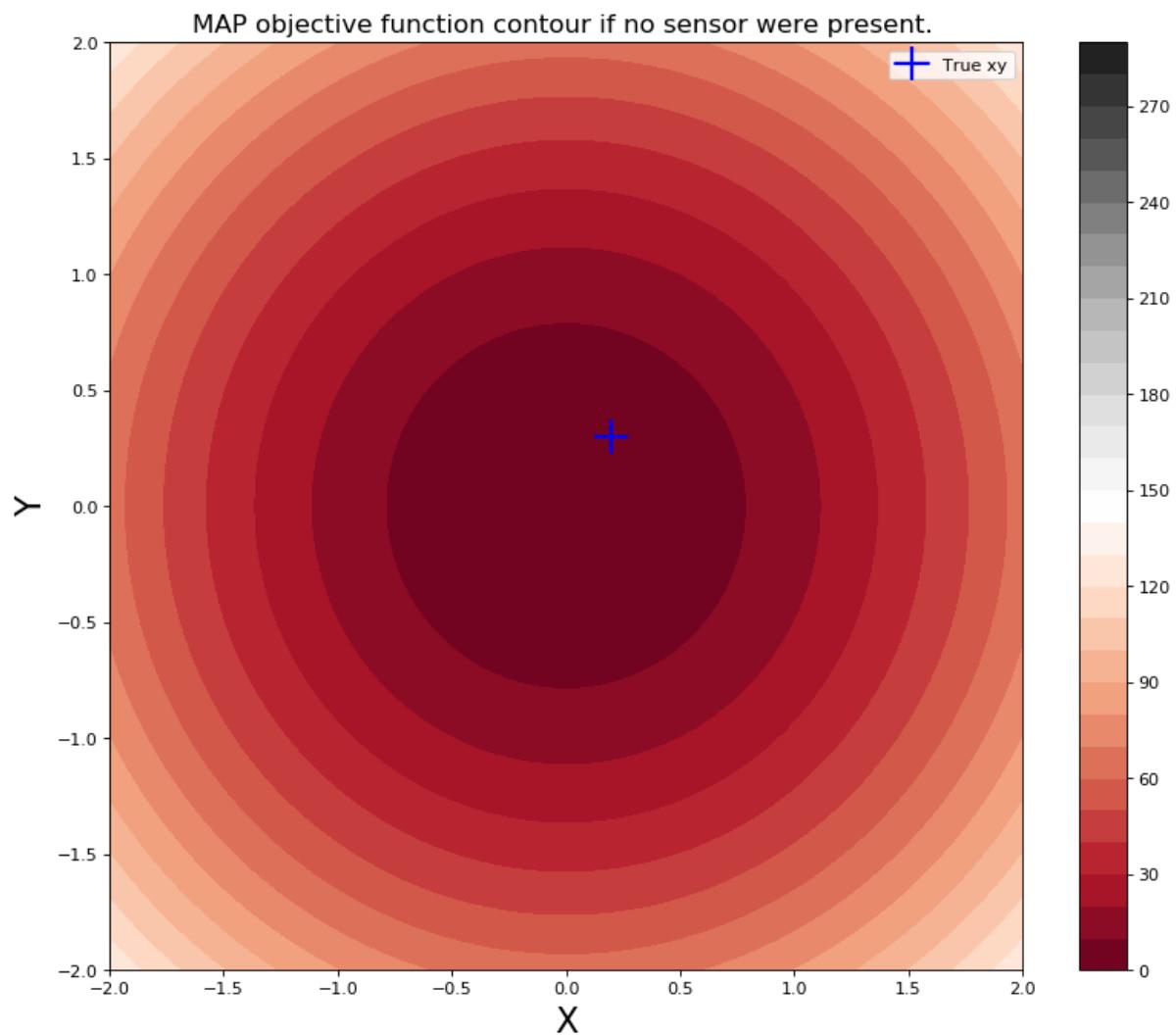


The objective center almost coincidences to the true value

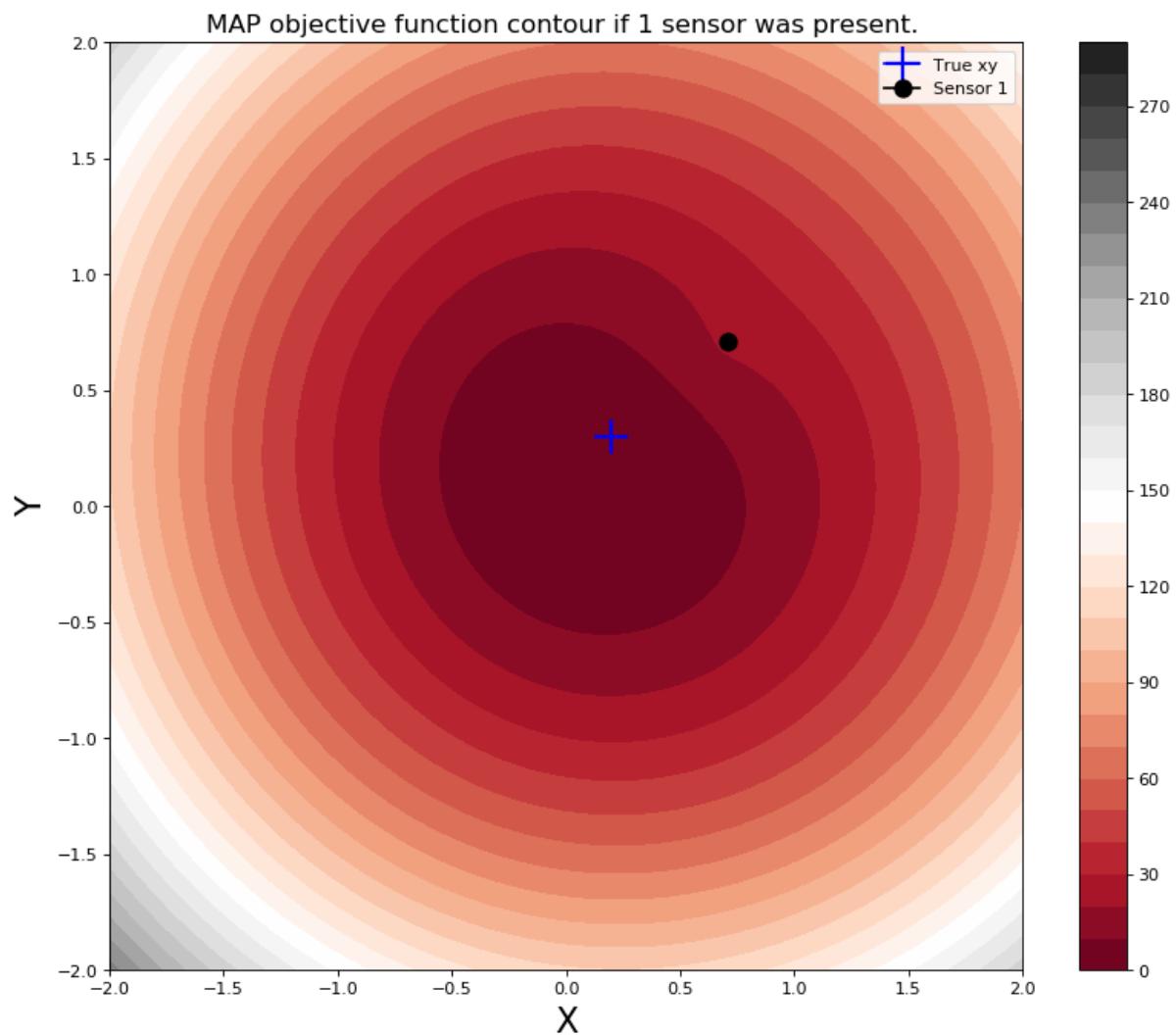
In []:

Changing the sigmas-

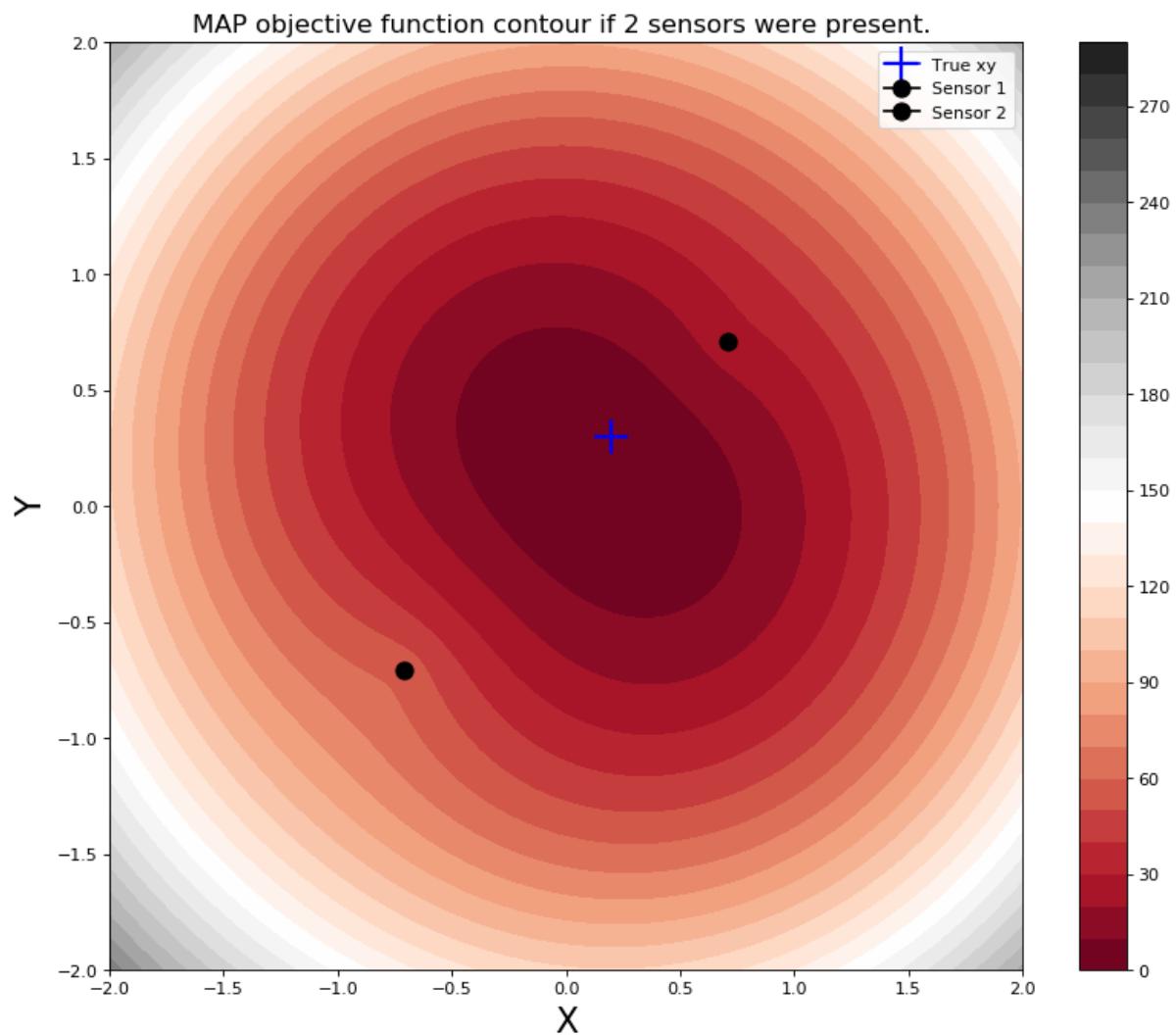
**sigma_x=0.25
sigma_y=0.25
sigma_i=0.3**



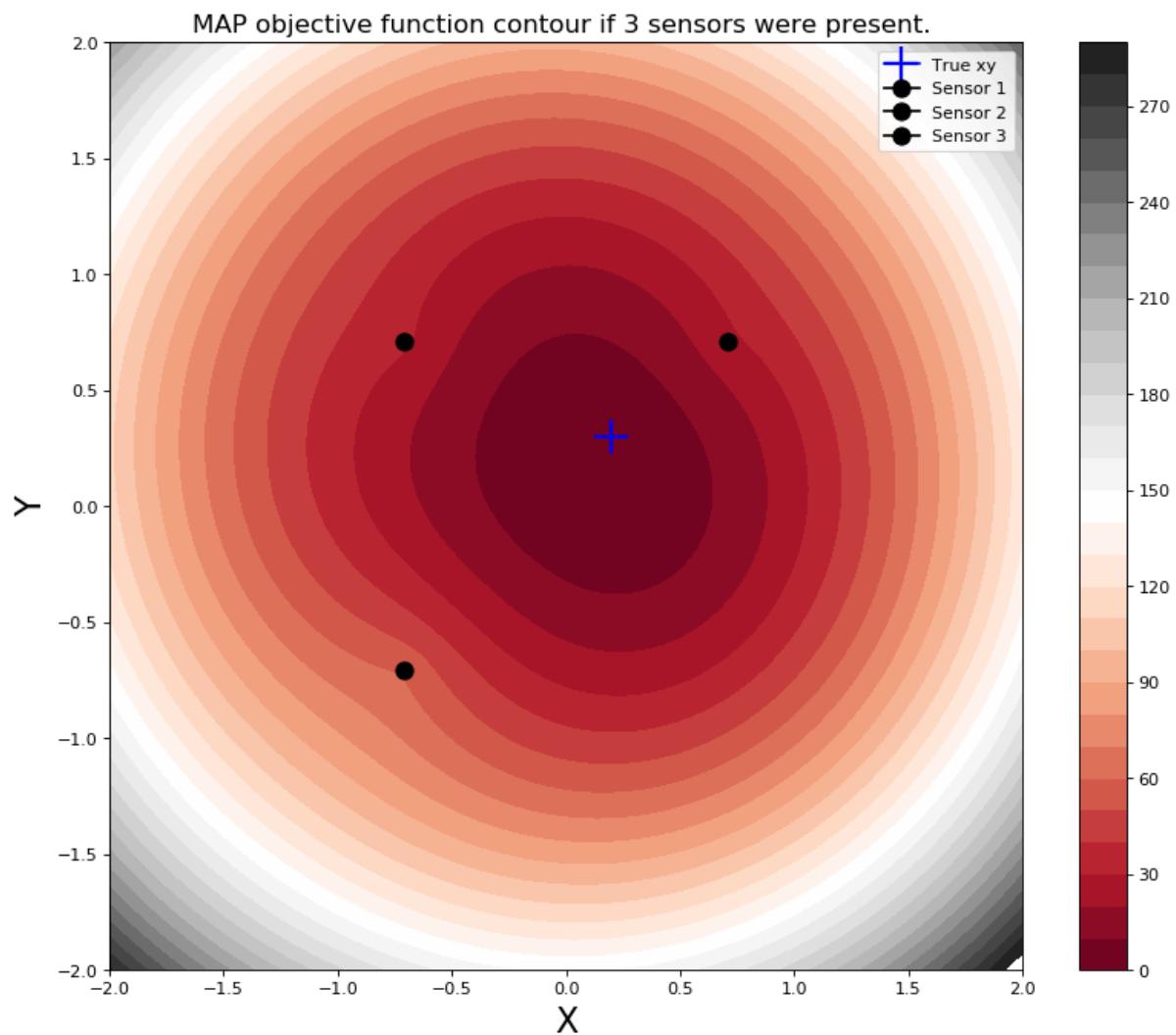
The objective is centered at origin 0,0 -> which is the prior value



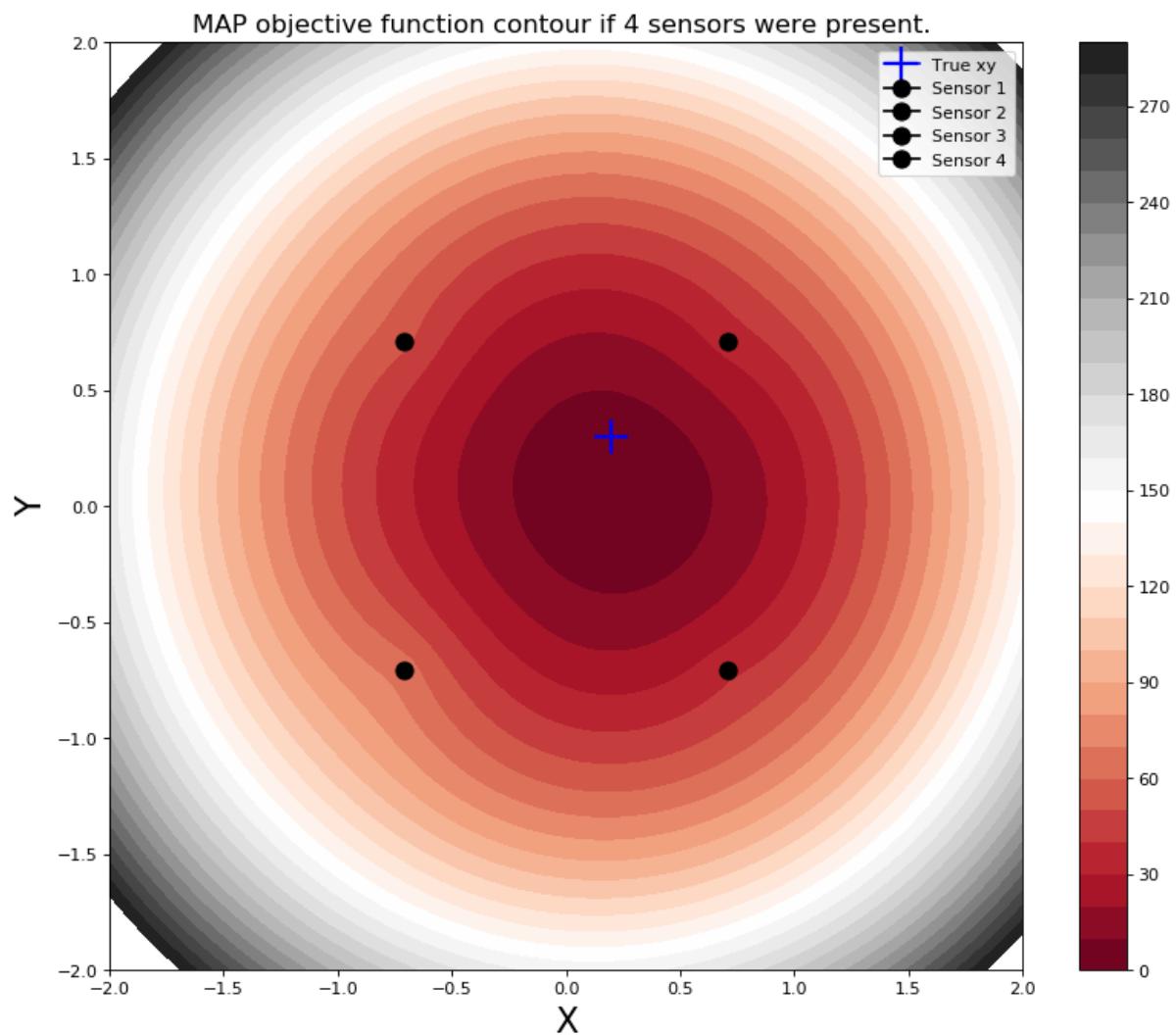
The objective center moves closer to the true value



The objective center moves further closer to the the true value



The objective center moves very close to the true value



The objective center almost coincidences to the true value

In []:

- Behavior of the MAP estimate based on centre of contour-

The MAP estimate moves closer to the true position of object as we get data from more sensors. In the contour plots, black regions are peaks and red regions are the valleys. valleys represent minima. In 1st plot, we don't even see black contours. The objective is extremely uncertain about the true location of object. As number of sensors increase, we see red region getting concentrated in a region, and black circles coming in the view. This represents deeper valleys, and more confidence about the true location. Further, the red circles are very close to true location in last image (4 sensors). MAP objective is also exactly correct about the true location in this case.

Answer 3

given $\rightarrow w \in \mathbb{R}^4$

$w \sim N(0, (\gamma^2 \Sigma))$ $\Sigma \rightarrow$ covariance matrix

$\vec{D} = (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$

$$y_i = ax_i^3 + bx_i^2 + cx_i + d + v$$

$$v \sim N(0, \sigma^2)$$

To find $\rightarrow \underset{w}{\operatorname{argmax}} P(\vec{w} | \vec{D})$

$$\vec{w}_{MAP} = \underset{w}{\operatorname{argmax}} \frac{P(\vec{D} | \vec{w}) P(\vec{w})}{P(\vec{D})}$$

$$= \underset{w}{\operatorname{argmax}} \left[\ln P(\vec{D} | \vec{w}) + \ln P(\vec{w}) \right]$$

D_1, D_2, \dots, D_N
↓ independent samples

$$= \underset{w}{\operatorname{argmax}} \left[\ln \prod_{i=1}^N P(D_i | \vec{w}) + \ln P(\vec{w}) \right]$$

$$= \underset{w}{\operatorname{argmax}} \left[\sum_{i=1}^N \ln (P(D_i | \vec{w})) + \ln P(\vec{w}) \right]$$

$$\ln p(y_i | \vec{w}) = (y_i, \vec{x}_i) | \vec{w}$$

$$P(y_i | \vec{x}_i, w) = ax_i^3 + bx_i^2 + cx_i + d + \mathcal{N}$$

↳ gaussian with mean $w^T \vec{x}$

and standard deviation σ^2

$$\text{where } w^T \vec{x} = ax_i^3 + bx_i^2 + cx_i + d$$

$$\therefore P(y_i | \vec{x}_i, w) = N(w^T \vec{x}_i, \sigma^2)$$

$$\ln P(y_i | \vec{x}_i, w) = \ln \left[\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - w^T \vec{x}_i)^2}{2\sigma^2}} \right]$$

↳ independent of \vec{w}

$$W_{MAP} = \underset{w}{\operatorname{argmax}} \left[\sum_{i=1}^N -\frac{(y_i - w^T \vec{x}_i)^2}{2\sigma^2} + \frac{1}{2\lambda} \|w\|^2 \right]$$

$$\ln \left(\frac{1}{((2\pi)^n |\Sigma|)^{1/2}} \exp \left(-\frac{1}{2} (\vec{w})^T \Sigma^{-1} \vec{w} \right) \right)$$

↳ independent of \vec{w}

$$W_{MAP} = \underset{w}{\operatorname{argmax}} \sum_{i=1}^N -\frac{(y_i - w^T x_i)^2}{2\sigma^2} - \frac{1}{2} (w^T \Sigma^{-1} w)$$

$$W_{MAP} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^N \frac{(y_i - w^T x_i)^2}{2\sigma^2} + \frac{1}{2} (w^T \Sigma^{-1} w)$$

where $\Sigma = \gamma^2 I$

$\therefore W_{MAP} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^N \frac{(y_i - w^T x_i)^2}{2\gamma^2} + \frac{1}{2} (w^T \gamma^2 I w)$

$$w_{MAP} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^n \frac{(y_i - w^T x_i)^2}{2\sigma^2} + \frac{1}{2} (w^T \Sigma^{-1} w)$$

$$\text{where } \Sigma = \gamma^2 I$$

$$w_{MLE} = \underset{w}{\operatorname{argmin}} \left(\frac{1}{\sigma^2} \right) \left(\frac{1}{2} \right) (xw - y)^T (xw - y) + \frac{1}{2} (w^T \Sigma^{-1} w)$$

$$= \underset{w}{\operatorname{argmin}} \left(\frac{1}{2\sigma^2} \right) (xw - y)^T (xw - y) + \frac{1}{2\gamma^2} w^T w$$

Take derivative and set to 0 →

$$\nabla_w f = \frac{1}{\sigma^2} (x^T x w - x^T y) + \frac{1}{\gamma^2} w = 0$$

~~$$\left[\frac{x^T x}{\sigma^2} + \frac{1}{\gamma^2} \right] w = \frac{x^T y}{\sigma^2}$$~~

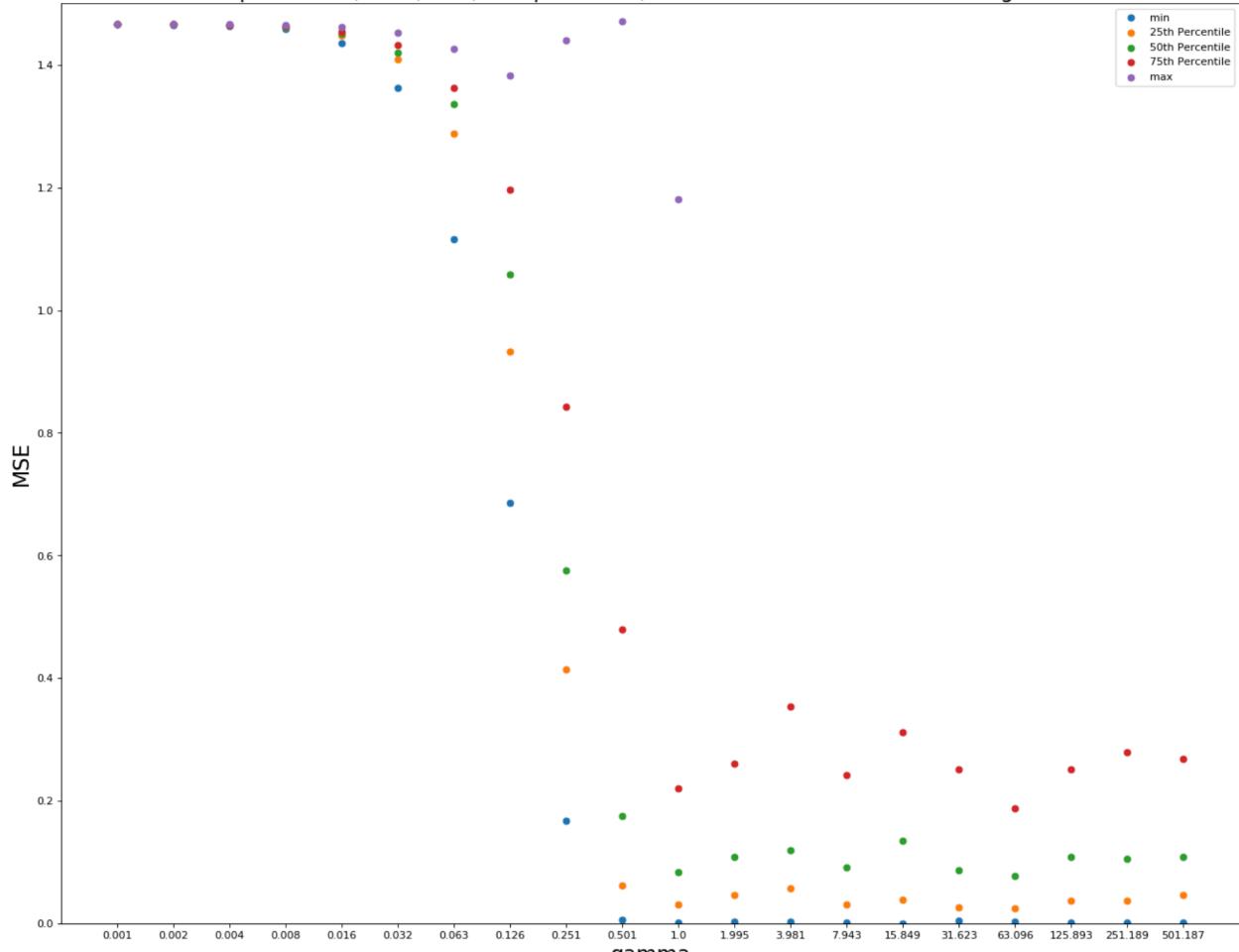
~~$$w = \cancel{\left(\frac{\sigma^2 x^T x + \gamma^2}{\sigma^2} \right) x^T y}$$~~

$$\left[X^T X + \frac{\sigma^2}{\gamma^2} I \right] w = X^T y$$



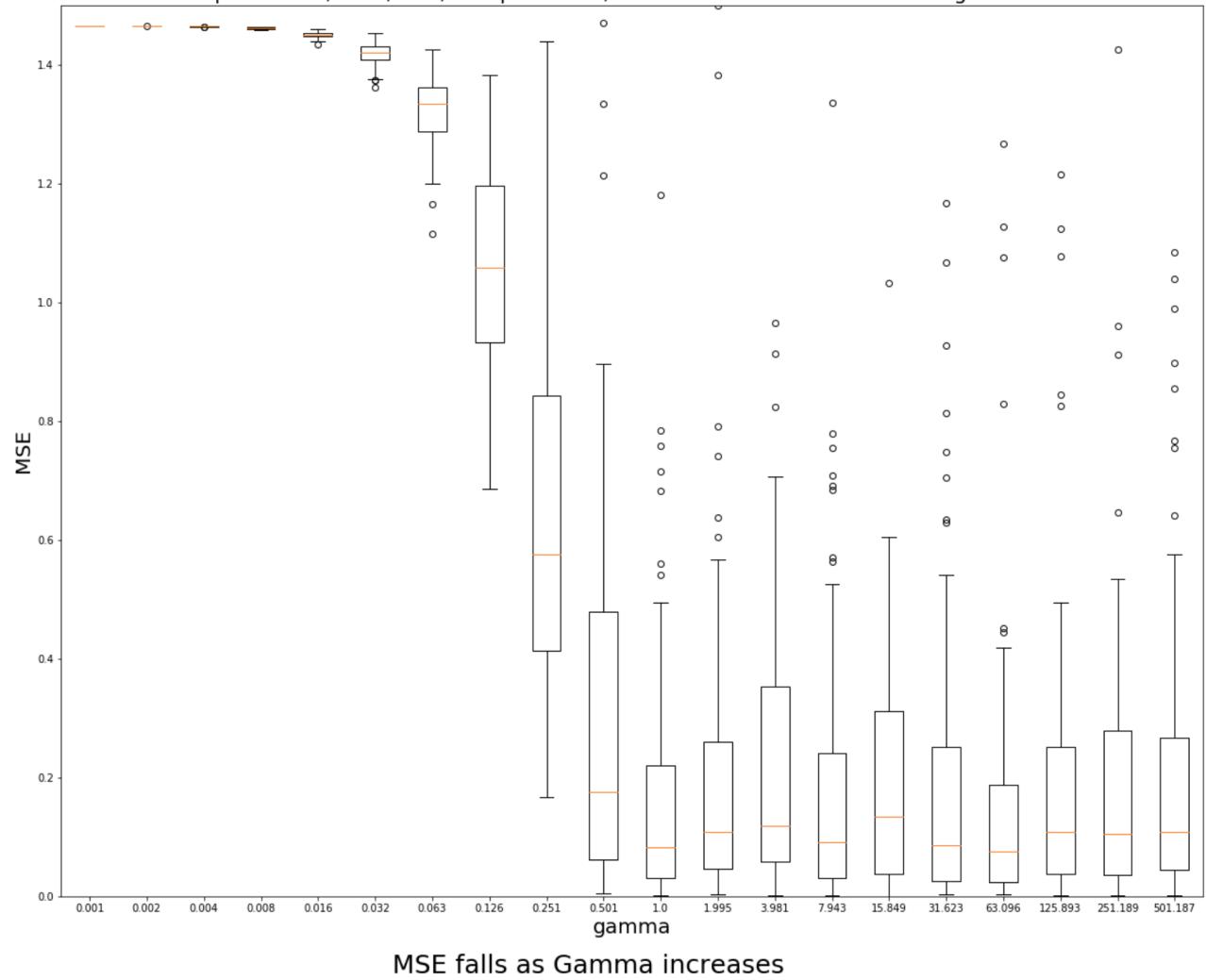
$$w = \left(X^T X + \frac{\sigma^2}{\gamma^2} I \right)^{-1} (X^T y)$$

Scatter plot of min, 25th,50th,75th percentile, and max values of MSE for each gamma value



MSE falls as Gamma increases

Box plot of min, 25th,50th,75th percentile, and max values of MSE for each gamma value



In [182]:

```
'''  
ML estimate  
'''  
  
C=np.matmul((X_stack.T),X_stack)  
A=np.linalg.inv(C)  
B=np.matmul(X_stack.T,Y)  
w_ml=np.matmul(A,B)  
print ("W_ML is \n",w_ml)  
  
print ("\n\nWith Gamma : ",gamma," , W_MAP is \n",w_map)  
print ("\nAs gamma increases, w_MAP indeed approaches to w_ml")  
  
W_ML is  
[[ 1.00279534]  
[-0.19552492]  
[-0.65031524]  
[ 0.13020576]]  
  
With Gamma : 100000.0 , W_MAP is  
[[ 1.00279534]  
[-0.19552492]  
[-0.65031524]  
[ 0.13020576]]  
  
As gamma increases, w_MAP indeed approaches to w_ml
```

Appendix

Consider least square expression \rightarrow

$$\underset{w}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^N (\gamma_i - x^T w)^2$$

independent of w

in vectorized form \rightarrow

$$\underset{w}{\operatorname{argmin}} f(w) = \frac{1}{2} \| (x^w - \gamma)^T (x^w - \gamma) \|$$

x^w : $1000, 4$
 γ : $1000, 1$
 w : $4, 1$
 $x^w - \gamma$: $1000, 1$

$$= \frac{1}{2} \| (x^w - \gamma)^T (x^w - \gamma) \|$$

\rightarrow assume w is feature vector of size 4 ; and 1000 samples

$$\underset{w}{\operatorname{argmin}} \frac{1}{2} \| (x^w - \gamma)^T (x^w - \gamma) \|$$

$$= \underset{w}{\operatorname{argmin}} \frac{1}{2} (w^T x^T x^w - w^T x^T \gamma - \gamma^T x^w + \gamma^T \gamma)$$

$$= \frac{1}{2} \text{tr} \left(w^T x^T x w - w^T x^T y - y^T x w + y^T y \right)$$

↓
trace

$$= \frac{1}{2} \left(\text{tr} w^T x^T x w - 2 \text{tr} y^T x w \right)$$

$$\nabla_w f = \frac{1}{2} \left(x^T x \overset{w}{\cancel{w}} + x^T x \overset{w}{\cancel{w}} - 2 x^T y \right)$$

$$= x^T x w - x^T \cancel{y}$$

↓

formula used for derivative

where $X \in \mathbb{R}^{1000 \times 4}$

$X^T \in \mathbb{R}^{n \times 1000}$

$\gamma \in \mathbb{R}^{1000 \times 1}$

$w \in \mathbb{R}^{n \times 1}$

assuming 1000 samples.

For 1 sample $\rightarrow \vec{x}_i = [x_i^3, x_i^2, x_i, x_i^0]$

$\vec{w} = [a, b, c, d]$

```
In [1]: import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
In [2]: '''
true roots
'''
x=sym.symbols('x')
r1=-0.8
r2=0.2
r3=0.8
y=(x-r1)*(x-r2)*(x-r3)
z=sym.expand(y.simplify())
z
```

Out[2]: $x^3 - 0.2x^2 - 0.64x + 0.128$

```
In [3]: '''
true values of W parameter
'''

a=(z.coeff(x**3))
b=(z.coeff(x**2))
c=(z.coeff(x))
d=z.coeff(x,n=0)
w=np.array([a,b,c,d])
w=np.array((w.reshape(-1,1)),dtype=float)
w
```

Out[3]: array([[1.],
 [-0.2],
 [-0.64],
 [0.128]])

In [225]:

```

'''  

X_stack.shape=(samples,4)  

'''  

num_samples=10
result={}
sigma=.1 # (Standard deviation of noise) (.01 seems decent) # fixed
gamma_low=-3
gamma_high=3
num_points=20
for gamma in np.logspace(gamma_low,gamma_high,num=num_points, endpoint=False):
else:
    result[gamma]=[]
    for i in range(100):
        Y=[ ]
        X=[ ]
        for i in range(num_samples):
            x=np.random.uniform(-1,1)
            y=w[0]*x**3+w[1]*x**2+w[2]*x**1+w[3]*x**0+np.random.normal(scale=sigma)
            Y.append(y)
            X.append(x)
        Y=np.array(np.array(Y).reshape(-1,1),dtype=float)
        X=np.array(X).reshape(-1,1)
        X_stack=np.hstack([X**3,X**2,X**1,X**0])
        C=np.matmul((X_stack.T),X_stack)+((sigma**2)/(gamma**2))*np.identity(4)
        A=np.linalg.inv(C)
        B=np.matmul(X_stack.T,Y)
        w_map=np.matmul(A,B)
        result[gamma].append(np.sum(np.square(w_map-w)))
label=[]
result_plot=[]
for gamma in np.logspace(gamma_low,gamma_high,num=num_points, endpoint=False):
else:
    label.append(np.round(gamma,3))
    result_plot.append(result[gamma])
fig1, ax1 = plt.subplots(figsize=(20,16))
from matplotlib.pyplot import figure
plt.ylim(0,1.5)
fig1.text(.35,0.06,'MSE falls as Gamma increases',fontsize=25);
plt.title('Box plot of min, 25th,50th,75th percentile, and max values of MSE for each gamma value',fontsize=20)
plt.ylabel('MSE',fontsize=20)
plt.xlabel('gamma',fontsize=20)
ax1.boxplot(result_plot,labels=label);

plt.show()
print ("\n\n\n")
from matplotlib.pyplot import figure
fig=figure(num=None, figsize=(20, 16), dpi=80, facecolor='w', edgecolor='k')
X=[]
for gamma in np.logspace(gamma_low,gamma_high,num=num_points, endpoint=False):
else:
    X.append(str(np.round(gamma,3)))

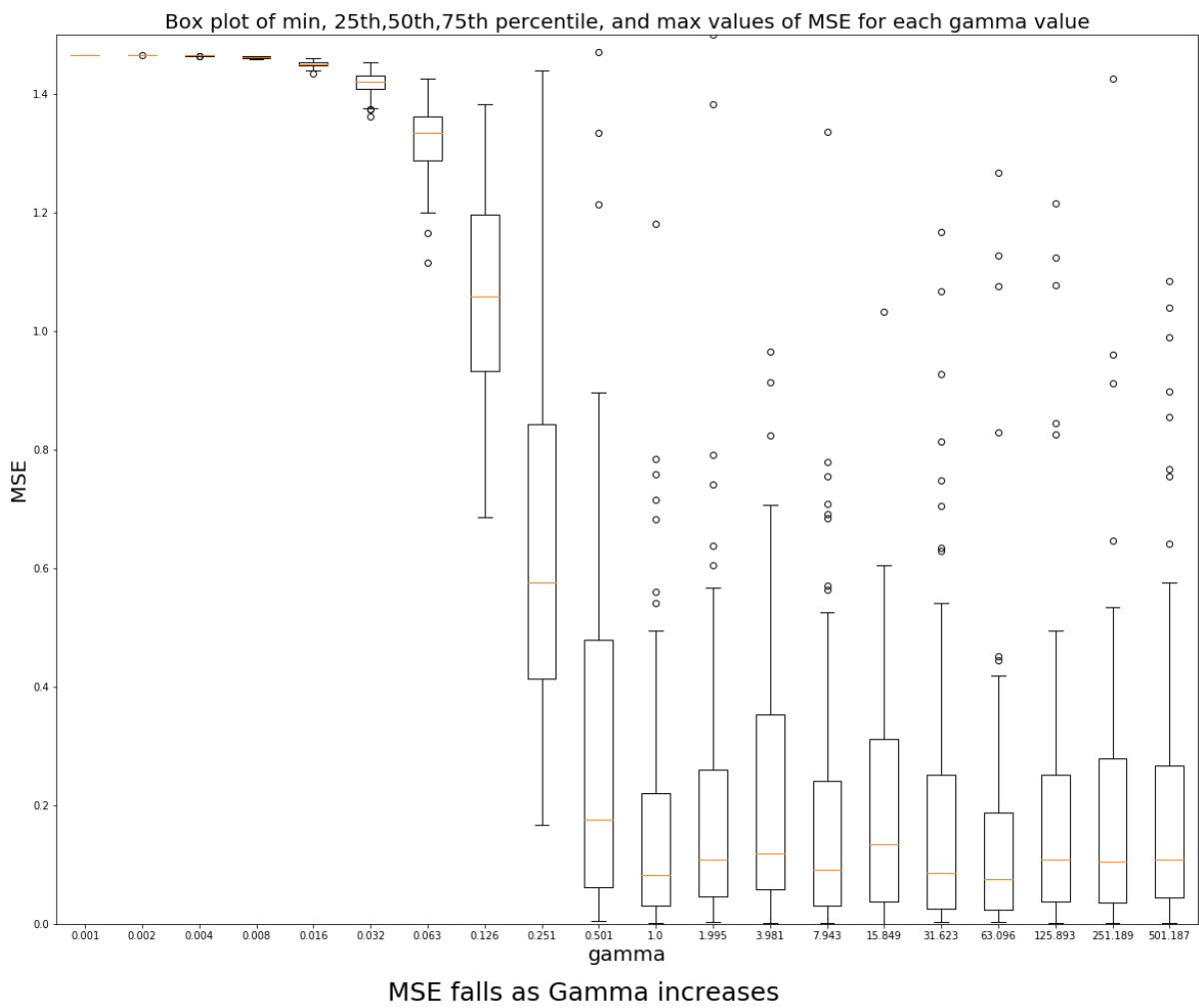
```

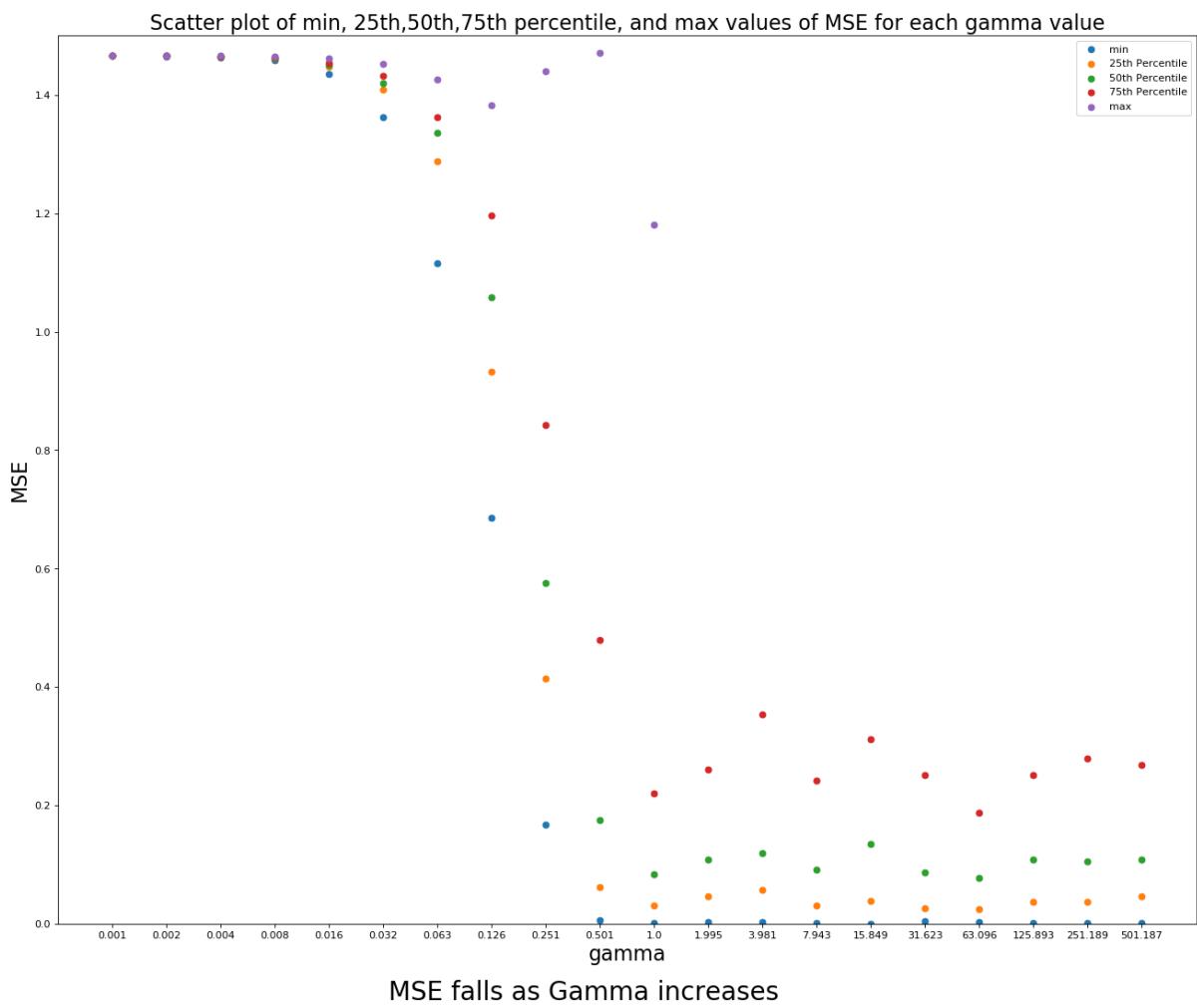
```
result_min=[ ]
for gamma in np.logspace(gamma_low,gamma_high,num=num_points, endpoint=False):
    result_min.append((np.array(result[gamma])).min())

result_max=[ ]
for gamma in np.logspace(gamma_low,gamma_high,num=num_points, endpoint=False):
    result_max.append((np.array(result[gamma])).max())

result_25=[ ]
for gamma in np.logspace(gamma_low,gamma_high,num=num_points, endpoint=False):
    result_25.append(np.percentile(np.array(result[gamma]),25))
result_50=[ ]
for gamma in np.logspace(gamma_low,gamma_high,num=num_points, endpoint=False):
    result_50.append(np.percentile(np.array(result[gamma]),50))

result_75=[ ]
for gamma in np.logspace(gamma_low,gamma_high,num=num_points, endpoint=False):
    result_75.append(np.percentile(np.array(result[gamma]),75))
plt.ylim(0,1.5)
plt.title('Scatter plot of min, 25th,50th,75th percentile, and max value s of MSE for each gamma value', fontsize=20)
plt.scatter(X,result_min,label='min')
plt.scatter(X,result_25,label='25th Percentile')
plt.scatter(X,result_50,label='50th Percentile')
plt.scatter(X,result_75,label='75th Percentile')
plt.scatter(X,result_max,label='max')
plt.ylabel('MSE', fontsize=20)
plt.xlabel('gamma', fontsize=20)
fig.text(.35,0.06,'MSE falls as Gamma increases', fontsize=25);
plt.legend();
```





In [182]:

```
'''  
ML estimate  
'''  
  
C=np.matmul((X_stack.T),X_stack)  
A=np.linalg.inv(C)  
B=np.matmul(X_stack.T,Y)  
w_ml=np.matmul(A,B)  
print ("W_ML is \n",w_ml)  
  
print ("\n\nWith Gamma : ",gamma,", W_MAP is \n",w_map)  
print ("\nAs gamma increases, w_MAP indeed approaches to w_ml")
```

```
W_ML is  
[[ 1.00279534]  
[-0.19552492]  
[-0.65031524]  
[ 0.13020576]]
```

```
With Gamma : 100000.0 , W_MAP is  
[[ 1.00279534]  
[-0.19552492]  
[-0.65031524]  
[ 0.13020576]]
```

As gamma increases, w_MAP indeed approaches to w_ml