Question 1 (30%) ¶

Design a classifier that achieves minimum probability of error for a three-class problem where the class priors are respectively P(L = 1) = 0.15,P(L = 2) = 0.35,P(L = 3) = 0.5 and the classconditional data distributions are all Gaussians for two-dimensional data vectors: N ([−1 0],[1 −0.4 −0.4 0.5]),N ([1 0],[0.5 0 0 0.2]),N ([0 1],[0.1 0 0 0.1]). Generate 10000 samples according to this data distribution, keep track of the true class labels for each sample. Apply your optimal classifier designed as described above to this dataset and obtain decision labels for each sample. Report the following: • actual number of samples that were generated from each class; • the confusion matrix for your classifier consisting of number of samples decided as class r ∈ {1,2,3} when their true labels were class c ∈ {1,2,3}, using r, c as row/column indices; • the total number of samples misclassified by your classifier; • an estimate of the probability of error your classifier will achieves, based on these samples; • a visualization of the data as a 2-dimensional scatter plot, with true labels and decision labels indicated using two separate visulization cues, such as marker shape and marker color; • a clear but brief description of the results presented as described above.

In [1]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt2
```

Initializing the Data Generator function

In [2]:
```
def sample_data_generator(mean1, cov1, mean2, cov2, mean3, cov3, p1, p2, p3):
    np.random.seed(seed=2002)
    uniform_set = np.random.uniform(0, 1, 10000)
    prior_label = [1 if a < p1 else (2 if a < p1+p2 else 3) for a in uniform_s
et]
    print('1.] The number of samples generated for each class are as follows:'
)
    print('Label 1: {}'.format(prior_label.count(1)))
    print('Label 2: {}'.format(prior_label.count(2)))
    print('Label 3: {}'.format(prior_label.count(3)))
    data_1 = np.random.multivariate_normal(mean1, cov1, prior_label.count(1))
    data_2 = np.random.multivariate_normal(mean2, cov2, prior_label.count(2))
    data_3 = np.random.multivariate_normal(mean3, cov3, prior_label.count(3))
    return np.array(data_1), np.array(data_2), np.array(data_3)
```

In [3]:
```
# Generating data for Class Label 1:
mean1 = np.array([-1, 0])
cov1 = np.array([[1, -0.4], [-0.4, 0.5]])
p1 = 0.15
```

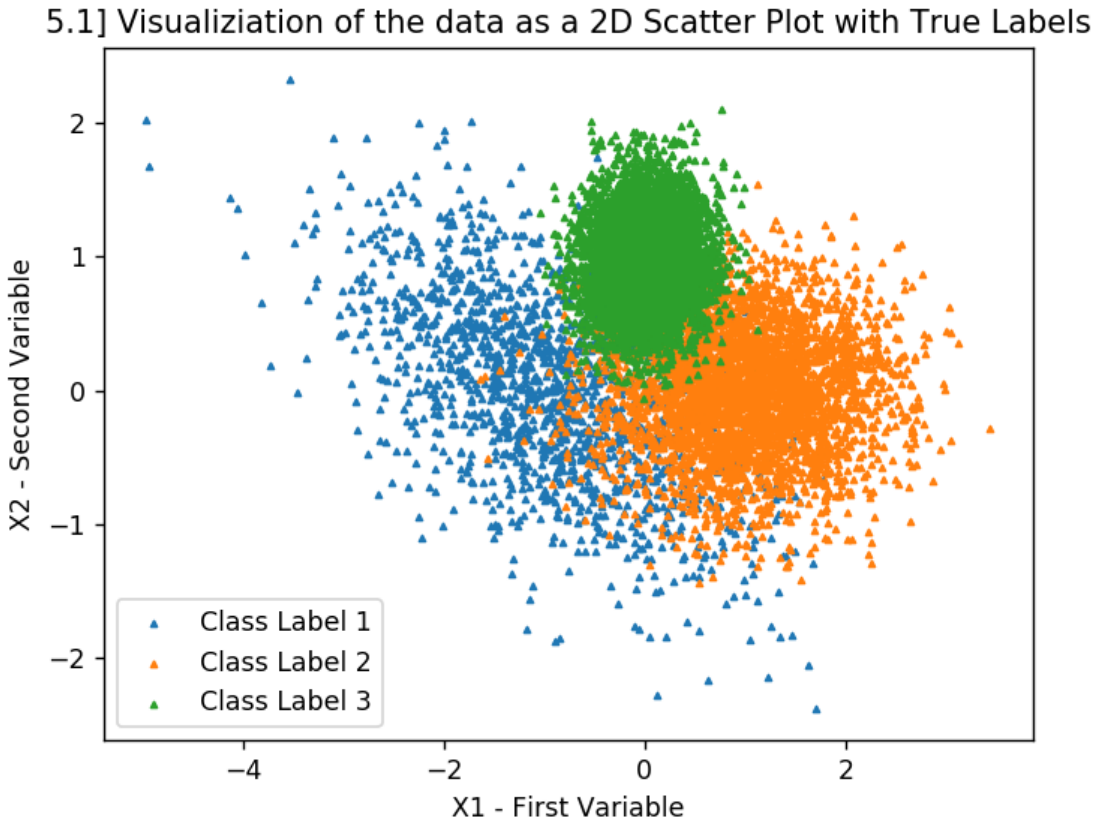
In [4]:
```
# Generating data for Class Label 2:
mean2 = np.array([1, 0])
cov2 = np.array([[0.5, 0], [0, 0.2]])
p2 = 0.35
```

In [5]:
```
# Generating data for Class Label 3:
mean3 = np.array([0, 1])
cov3 = np.array([[0.1, 0], [0, 0.1]])
p3 = 0.5
```

In [6]:
```
data1, data2, data3 = sample_data_generator(mean1, cov1, mean2, cov2, mean3, c
ov3, p1, p2, p3)
```

```
1.] The number of samples generated for each class are as follows:
Label 1: 1453
Label 2: 3498
Label 3: 5049
```

In [7]:
```
# Visualizing the samples generated for each class as a Scatter Plot
# Answer for the 5th Subquestion part 1
%matplotlib notebook
plt.scatter(data1[:, 0], data1[:, 1], marker='^', label='Class Label 1', s=5)
plt.scatter(data2[:, 0], data2[:, 1], marker='^', label='Class Label 2', s=5)
plt.scatter(data3[:, 0], data3[:, 1], marker='^', label='Class Label 3', s=5)
plt.xlabel('X1 - First Variable')
plt.ylabel('X2 - Second Variable')
plt.title('5.1] Visualiziation of the data as a 2D Scatter Plot with True Labe
ls')
plt.legend()
```



Out[7]: <matplotlib.legend.Legend at 0x210651d24e0>

Initializing the MAP Classifier

We are using the MAP classifier because it aims at achieving the minimum probability of error

In [15]:
```
def map_classifier(mean1, mean2, mean3, cov1, cov2, cov3, p1, p2, p3, x1, x2):
    px = []
    px.append((math.pow(2*math.pi, -1.5)*math.pow(np.linalg.det(cov1), -0.5)*m
ath.exp((-0.5)*([x1, x2] - mean1).T.dot(np.linalg.inv(cov1)).dot([x1, x2] - me
an1)))*p1)
    px.append((math.pow(2*math.pi, -1.5)*math.pow(np.linalg.det(cov2), -0.5)*m
ath.exp((-0.5)*([x1, x2] - mean2).T.dot(np.linalg.inv(cov2)).dot([x1, x2] - me
an2)))*p2)
    px.append((math.pow(2*math.pi, -1.5)*math.pow(np.linalg.det(cov2), -0.5)*m
ath.exp((-0.5)*([x1, x2] - mean3).T.dot(np.linalg.inv(cov2)).dot([x1, x2] - me
an3)))*p3)
#     print(np.argmax(px) + 1)
    return np.argmax(px) + 1
```

In [16]:
```
data1_correct = []
data1_incorrect = []
data2_correct = []
data2_incorrect = []
data3_correct = []
data3_incorrect = []
data1_incorrect2 = []
data1_incorrect3 = []
data2_incorrect1 = []
data2_incorrect3 = []
data3_incorrect1 = []
data3_incorrect2 = []
[data1_correct.append([a, b]) if map_classifier(mean1, mean2, mean3, cov1, cov
2, cov3, p1, p2, p3, a, b) == 1 else data1_incorrect.append([a, b]) for a, b i
n data1]
[data2_correct.append([a, b]) if map_classifier(mean1, mean2, mean3, cov1, cov
2, cov3, p1, p2, p3, a, b) == 2 else data2_incorrect.append([a, b]) for a, b i
n data2]
[data3_correct.append([a, b]) if map_classifier(mean1, mean2, mean3, cov1, cov
2, cov3, p1, p2, p3, a, b) == 3 else data3_incorrect.append([a, b]) for a, b i
n data3]
[data1_incorrect2.append([a, b]) if map_classifier(mean1, mean2, mean3, cov1,
cov2, cov3, p1, p2, p3, a, b) == 2 else data1_incorrect3.append([a, b]) for a,
b in data1_incorrect]
[data2_incorrect1.append([a, b]) if map_classifier(mean1, mean2, mean3, cov1,
cov2, cov3, p1, p2, p3, a, b) == 1 else data2_incorrect3.append([a, b]) for a,
b in data2_incorrect]
[data3_incorrect1.append([a, b]) if map_classifier(mean1, mean2, mean3, cov1,
cov2, cov3, p1, p2, p3, a, b) == 1 else data3_incorrect2.append([a, b]) for a,
b in data3_incorrect]

print('Correct Classifications of Label 1: {}'.format(len(data1_correct)))
print('Incorrect Classifications of Label 1: {}'.format(len(data1_incorrect)))
print('Correct Classifications of Label 2: {}'.format(len(data2_correct)))
print('Incorrect Classifications of Label 2: {}'.format(len(data2_incorrect)))
print('Correct Classifications of Label 3: {}'.format(len(data3_correct)))
print('Incorrect Classifications of Label 3: {}'.format(len(data3_incorrect)))
```

```
Correct Classifications of Label 1: 1010
Incorrect Classifications of Label 1: 443
Correct Classifications of Label 2: 2975
Incorrect Classifications of Label 2: 523
Correct Classifications of Label 3: 4984
Incorrect Classifications of Label 3: 65
```

In [17]:
```
# Determining the Confusion Matrix
#Answer for the second sub-question
np.set_printoptions(suppress=True)
confusion_mat = np.zeros([3, 3])
confusion_mat[0][0] = len(data1_correct)
data1_incorrect_labels = [map_classifier(mean1, mean2, mean3, cov1, cov2, cov3
, p1, p2, p3, a, b) for a, b in data1_incorrect]
confusion_mat[1][0] = data1_incorrect_labels.count(2)
confusion_mat[2][0] = data1_incorrect_labels.count(3)
confusion_mat[1][1] = len(data2_correct)
data2_incorrect_labels = [map_classifier(mean1, mean2, mean3, cov1, cov2, cov3
, p1, p2, p3, a, b) for a, b in data2_incorrect]
confusion_mat[0][1] = data2_incorrect_labels.count(1)
confusion_mat[2][1] = data2_incorrect_labels.count(3)
confusion_mat[2][2] = len(data3_correct)
data3_incorrect_labels = [map_classifier(mean1, mean2, mean3, cov1, cov2, cov3
, p1, p2, p3, a, b) for a, b in data3_incorrect]
confusion_mat[0][2] = data3_incorrect_labels.count(1)
confusion_mat[1][2] = data3_incorrect_labels.count(2)
print('2.] Following is the Confusion Matrix: \n {}'.format(confusion_mat))
```

```
2.] Following is the Confusion Matrix:
 [[1010.  120.    2.]
 [ 221. 2975.   63.]
 [ 222.  403. 4984.]]
```

In the Confusion Matrix above, the columns represent the True Class Labels {1, 2, 3}, and the rows represent the Decision Class Labels {1, 2, 3}. Therefore, all the diagonal elements of the Matrix represent the correctly classified samples, whereas the other elements represent the sample numbers that were incorrectly classified into the other class.

In [18]:
```
# colLabels = ['True Label 1', 'True Label 2', 'True Label 3']
# rowLabels = ['Decision Label 1', 'Decision Label 2', 'Decision Label 3']
# plt.table(cellText=confusion_mat, colLabels=colLabels, rowLabels=rowLabels,
 loc='center')
```

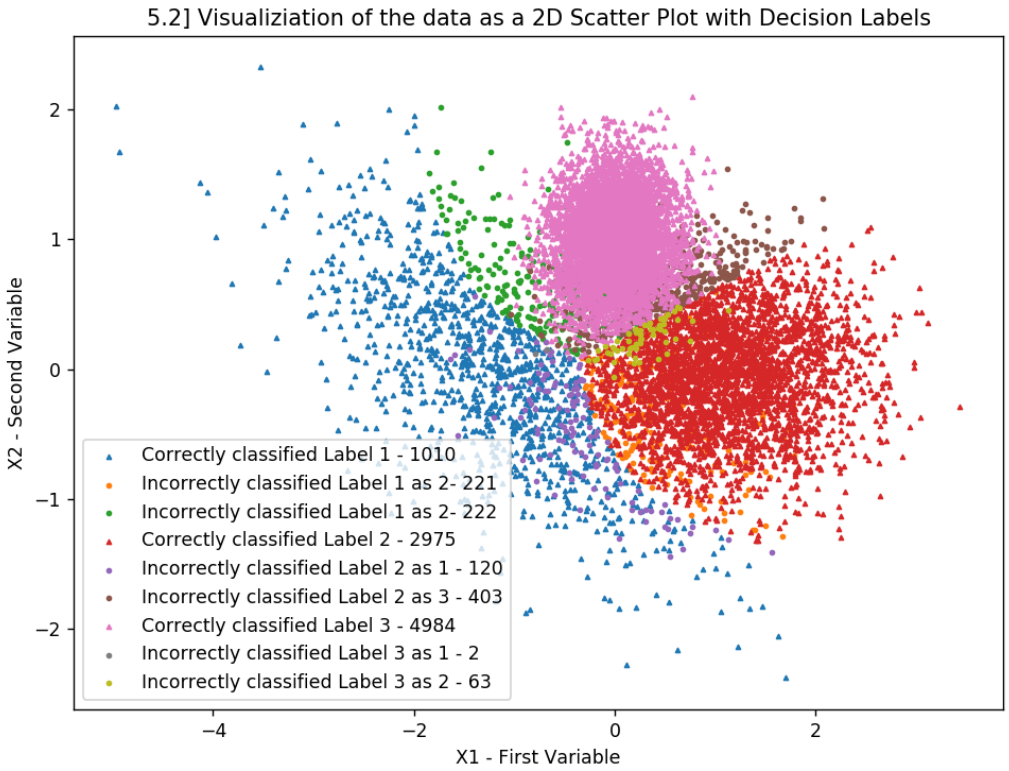
In [19]:
```
# Calculating the number of misclassified samples by the MAP classifier
total_incorrect = len(data1_incorrect) + len(data2_incorrect) + len(data3_inco
rrect)
print('3.] The total number of samples misclassified by the MAP classifier: {}
'.format(total_incorrect))
```

```
3.] The total number of samples misclassified by the MAP classifier: 1031
```

In [20]:
```
# Calculating the Estimate of the Probability of Error for the classifier
# That would be the total number of misclassified samples divided by the total
number of samples
p_error = total_incorrect/10000
print('4.] The Probability of Error for our MAP classifier is : {}'.format(p_e
rror))
```

```
4.] The Probability of Error for our MAP classifier is : 0.1031
```

In [21]:
```
# Visualizing the Classified labels by the MAP classifier for each label as a
 Scatter Plot
# Answer for the 5th Subquestion part 2
%matplotlib notebook
plt.scatter(np.array(data1_correct)[:, 0], np.array(data1_correct)[:, 1], mark
er='^', label='Correctly classified Label 1 - {}'.format(len(data1_correct)),
s=5)
plt.scatter(np.array(data1_incorrect2)[:, 0], np.array(data1_incorrect2)[:, 1
], marker='o', label='Incorrectly classified Label 1 as 2- {}'.format(len(data
1_incorrect2)), s=5)
plt.scatter(np.array(data1_incorrect3)[:, 0], np.array(data1_incorrect3)[:, 1
], marker='o', label='Incorrectly classified Label 1 as 2- {}'.format(len(data
1_incorrect3)), s=5)
plt.scatter(np.array(data2_correct)[:, 0], np.array(data2_correct)[:, 1], mark
er='^', label='Correctly classified Label 2 - {}'.format(len(data2_correct)),
s=5)
plt.scatter(np.array(data2_incorrect1)[:, 0], np.array(data2_incorrect1)[:, 1
], marker='o', label='Incorrectly classified Label 2 as 1 - {}'.format(len(dat
a2_incorrect1)), s=5)
plt.scatter(np.array(data2_incorrect3)[:, 0], np.array(data2_incorrect3)[:, 1
], marker='o', label='Incorrectly classified Label 2 as 3 - {}'.format(len(dat
a2_incorrect3)), s=5)
plt.scatter(np.array(data3_correct)[:, 0], np.array(data3_correct)[:, 1], mark
er='^', label='Correctly classified Label 3 - {}'.format(len(data3_correct)),
s=5)
plt.scatter(np.array(data3_incorrect1)[:, 0], np.array(data3_incorrect1)[:, 1
], marker='o', label='Incorrectly classified Label 3 as 1 - {}'.format(len(dat
a3_incorrect1)), s=5)
plt.scatter(np.array(data3_incorrect2)[:, 0], np.array(data3_incorrect2)[:, 1
], marker='o', label='Incorrectly classified Label 3 as 2 - {}'.format(len(dat
a3_incorrect2)), s=5)
plt.xlabel('X1 - First Variable')
plt.ylabel('X2 - Second Variable')
plt.title('5.2] Visualiziation of the data as a 2D Scatter Plot with Decision
 Labels')
plt.legend()
```



Out[21]: <matplotlib.legend.Legend at 0x2106573ef98>

6.] Description of the results gathered above

I have been presented with 3 Class Labels, each sampled at a given Gaussian.

1.] I randomly generated the data for the respective class labels as per the priors provided to me.

By using the Maximum-A-Posteriori (MAP) Classifier I found the decision labels of my classifier for each of the 10000 samples. MAP classifier dictates that for any given sample, the maximum product of the prior with the probability that the sample is part of a given label is classified to be that Label's sample.

2.] Through the MAP Classifier I found the Confusion Matrix, which gives a comparative chart of how many samples of each class were correctly and incorrectly classified as part of another class.

3.] By adding the number of misclassified labels, I was able to determine the total number of Misclassified Labels.

4.] The ratio of the misclassified labels to the total number of samples gives us the Probability of Error for our classifier.

5.] Through two scatter plots, I have visualized the data with the True Labels and with the Decision Labels. In the second plot you can clearly see the number of correctly classified labels in each class with the triangular markers, along with the number of misclassified labels in each class with the circular markers.

In [ ]:

2.] Given :- $r_i = d_{T_i} + n_i$

Here r_i is the range measurement from i^{th} Landmark point to the object's true position(d_{T_i}) corrupted with a gaussian noise (n_i) having zero mean and σ_i variance

$$d_{T_i} = \| [x_T, y_T]^T - [x_i, y_i]^T \|$$

$$= \| [x_T - x_i, y_T - y_i]^T \|$$

$$= \sqrt{(x_T - x_i)^2 + (y_T - y_i)^2} \quad \left[\begin{matrix}\text{Norm of} \\ \text{the vector}\end{matrix}\right]$$

$$n_i \sim \mathcal{N}(0, \sigma_i^2)$$

$[x_T, y_T]$ → Object's true location

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

Prior for the location of the object for a given point [x y] [candidate]

OBJECTIVE :- Given $K \in \{1, 2, 3, 4\}$, and that position of K can lie anywhere on the unit circle radius, find the optimization problem that needs to be solved to determine the MAP estimate of the object position.

Solution: We have r_i which is the range measurement of Object position from i^{th} Landmark point.

With the linearity property

$$x = AZ + b \quad \text{where } Z \sim \mathcal{N}(0, \Sigma) \text{ and } x \sim \mathcal{N}(b, AA^T)$$

we know, that r_i is also a gaussian distribution

we have, $r_i = d_{Ti} + n_i$

$$\Rightarrow r_i = n_i + d_{Ti}$$

Comparing with the property equation, we ~~know~~ get $r_i \sim \mathcal{N}(d_{Ti}, \sigma_i^2)$

To find the MAP estimate of the Object position $[x, y]$, we have to find the maximum ~~likelihood~~ posterior of position $[x, y]$ given r_i.

i.e.

$$\begin{bmatrix} x \\ y \end{bmatrix}_{MAP} = \underset{[x\ y]^T}{\operatorname{argmax}} \; p\left(\begin{bmatrix} x \\ y \end{bmatrix} / r_i\right)$$

Applying Bayes Rule:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{MAP} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \; p\left(r_i / \begin{bmatrix} x \\ y \end{bmatrix}\right) \times p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) / p(r_i)$$

(likelihood) prior evidence [Irrelevant to Estimation]

$$\therefore \begin{bmatrix} x \\ y \end{bmatrix}_{MAP} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \sum_{i=1}^{K} P\left(r_i \mid \begin{bmatrix} x \\ y \end{bmatrix}\right) \times P\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)$$

Taking log on both sides

$$\ln \begin{bmatrix} x \\ y \end{bmatrix}_{MAP} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \sum_{i=1}^{K} \ln\left[P\left(r_i \mid [x\ y]^T\right)\right] + \ln\left[P([x\ y])\right] \rightarrow (1)$$

Solving each term on R.H.S. individually;

Part 1:

$$\ln\left[P\left(r_i \mid [x\ y]^T\right)\right] = \ln \frac{1}{\sqrt{2\pi}} * \Sigma^{+1/2} \times e^{\frac{-1}{2} \frac{(r_i - \mu)^2}{\sigma_i^2}}$$

[For Parameter Estimation of a Normal]

$$= -\frac{1}{2}\ln(2\pi) + \ln \Sigma^{1/2} \times e^{-\frac{1}{2} \frac{(r_i - \mu)^2}{\sigma_i^2}}$$

As we know, Here $\mu = d_{Ti}$ & $\Sigma = 1$ for r_i

$$\Rightarrow -\frac{1}{2}\ln(2\pi) + \ln(1) \times e^{\frac{-1}{2} \frac{(r_i - d_{Ti})^2}{\sigma_i^2}}$$

$$\Rightarrow \underbrace{-\frac{1}{2}\ln(2\pi)}_{\text{constant}} + \left(-\frac{1}{2}\right) \frac{(r_i - d_{Ti})^2}{\sigma_i^2}$$

↓ Does not affect the maximization problem

$$\Rightarrow \frac{-1}{2} \frac{(r_i - d_{Ti})^2}{\sigma_i^2} \rightarrow (2)$$

Part 2 :

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

$$= \underbrace{-\ln(2\pi) - \ln\sigma_x - \ln\sigma_y}_{\text{Constant} \rightarrow \text{Irrelevant}} - \frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} 1/\sigma_x^2 & 0 \\ 0 & 1/\sigma_y^2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$$

$$= -\frac{1}{2}\left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right] \rightarrow (3)$$

From (1), (2) & (3)

$$\ln\begin{bmatrix} x \\ y \end{bmatrix}_{MAP} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\text{argmax}} \sum_{i=1}^{K} -\frac{1}{2}\left(\frac{r_i - d_{Ti}}{\sigma_i}\right)^2 - \frac{1}{2}\left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right]$$

$$\Rightarrow \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\text{argmax}} -\frac{1}{2}\left[\sum_{i=1}^{K}\frac{(r_i - d_{Ti})^2}{\sigma_i^2} + \left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right]\right]$$

For a negative value inside argmax, we can take the argmin and remove the constant, giving us our Objective function as :-

$$\ln\begin{bmatrix} x \\ y \end{bmatrix}_{MAP} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\text{argmin}} \underbrace{\sum_{i=1}^{K}\frac{(r_i - d_{Ti})^2}{\sigma_i^2}}_{\text{LIKELIHOOD}} + \underbrace{\left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right]}_{\text{PRIOR}} \rightarrow (4)$$

Question 2 (35%)

An object at true position [xT , yT] T in 2-dimensional space is to be localized using distance (range) measurements to K reference (landmark) coordinates {[x1, y1] T ,...,[xi , yi] T ,...,[xK, yK] T}. These range measurements are ri = dTi +ni for i ∈ {1,...,K}, where dTi = k[xT , yT] T −[xi , yi] Tk is the true distance between the object and the i th reference point, and ni is a zero mean Gaussian distributed measurement noise with known variance σ 2 i . The noise in each measurement is independent from the others. Assume that we have the following prior knowledge regarding the position of the object: p x y ! = (2πσxσy) −1 e − 1 2 h x yi " σ 2 x 0 0 σ 2 y

−1"

x y (1) where [x, y] T indicates a candidate position under consideration. Express the optimization problem that needs to be solved to determine the MAP estimate of the object position. Simplify the objective function so that the exponentials and additive/multiplicative terms that do not impact the determination of the MAP estimate [xMAP, yMAP] T are removed appropriately from the objective function for computational savings when evaluating the objective. Implement the following as computer code: Set the true object location to be inside the circle with unit radious centered at the origin. For each K ∈ {1,2,3,4} repeat the following. Place evenly spaced K landmarks on a circle with unit radius centered at the origin. Set measurement noise standard deviation to 0.3 for all range measurements. Generate K range measurements according to the model specified above (if a range measurement turns out to be negative, reject it and resample; all range measurements need to be nonnegative). Plot the equilevel contours of the MAP estimation objective for the range of horizontal and vertical coordinates from −2 to 2; superimpose the true location of the object on these equilevel contours (e.g. use a + mark), as well as the landmark locations (e.g. use a o mark for each one). Provide plots of the MAP objective function contours for each value of K. When preparing your final contour plots for different K values, make sure to plot contours at the same function value across each of the different contour plots for easy visual comparison of the MAP objective landscapes. Supplement your plots with a brief description of how your code works. Comment on the behavior of the MAP estimate of position (visually assessed from the contour plots; roughly center of the innermost contour) relative to the true position. Does the MAP estimate get closer to the true position as K increases? Doe is get more certain? Explain how your contours justify your conclusions. Suggestion: For σx and σy consider values around 0.25, and for the noise variance values σ 2 i consider values around 0.1 for posterior functions that are illustrative; you may choose different values than what is suggested here, so make sure to specify what your values are in the numerical results presented. Note: The additive Gaussian distributed noise used in this question is actually not appropriate, since it could lead to negative measurements, which are not legitimate for a proper distance sensor. However, in this question, we will ignore this issue and proceeding with this noise model for the sake of illustration. In practice, a multiplicative log-normal distributed noise may be more appropriate than an additive normal distributed noise.

In [1]:
```
# Importing the necessary libraries
import numpy as np
import math
import matplotlib.pyplot as plt
```

Below is the MAP Estimator Function It calculates the Likelihood and the Prior of a given candidate point and returns the MAP Estimate for that point

In [314]:
```
# Initializing the Map Estimator function
def map_estimator(stddev1, stddev2, x, y, R, Dti, sigma_i):
#     map_estimate = []
    likelihood_term = []
    for i, (r, dti) in enumerate(zip(R, Dti)):
        likelihood_term.append(((r - dti)**2)/sigma_i**2)
    prior_x_y_term = (((x**2) / (stddev1**2)) + ((y**2) / (stddev2**2)))
    map_estimate = sum(likelihood_term) + prior_x_y_term
    return map_estimate
```

Below I set the Object's true location inside the Unit circle centered at Origin

In [506]:
```
# Setting the Object's True location inside the unit circle
O_loc = [0.8, 0.8]
```

Here I am setting the Sigma values for the Gaussian Noise and the Prior

In [475]:
```
# Setting noise variance
sigma_i = 0.04

# Setting Sigma Values
sigma_x = 0.05
sigma_y = 0.05

# Setting levels for contours
levels = 40
```

Below I am calculating the Distance of the Landmark point from the True Location of the object and corrupting it with a Gaussian Noise

In [476]:
```
# Calculating ri for landmark points
def cal_ri(KX, KY):
    R = []
    Dti = []
    for (kx, ky) in zip(KX, KY):
        ni = np.random.normal(loc=0, scale=sigma_i)
        dti = math.sqrt(((O_loc[0] - kx)**2) + ((O_loc[1] - ky)**2))
        R.append(dti + ni)
        Dti.append(dti)
    return R, Dti
```

Below I am finding the distance of the Landmark point from the candidate location

In [520]:
```
# Calculating Dti for candidate point
def cal_dti(KX, KY, xref, yref):
    Dti = []
    for (kx, ky) in zip(KX, KY):
        Dti.append(math.sqrt(((xref - kx)**2) + ((yref - ky)**2)))
    return Dti
```

In [320]:
```
X = np.arange(-2, 2.2, 0.2)
Y = np.arange(-2, 2.2, 0.2)
```

In [321]:
```
X1, Y1 = np.meshgrid(X,Y)
```

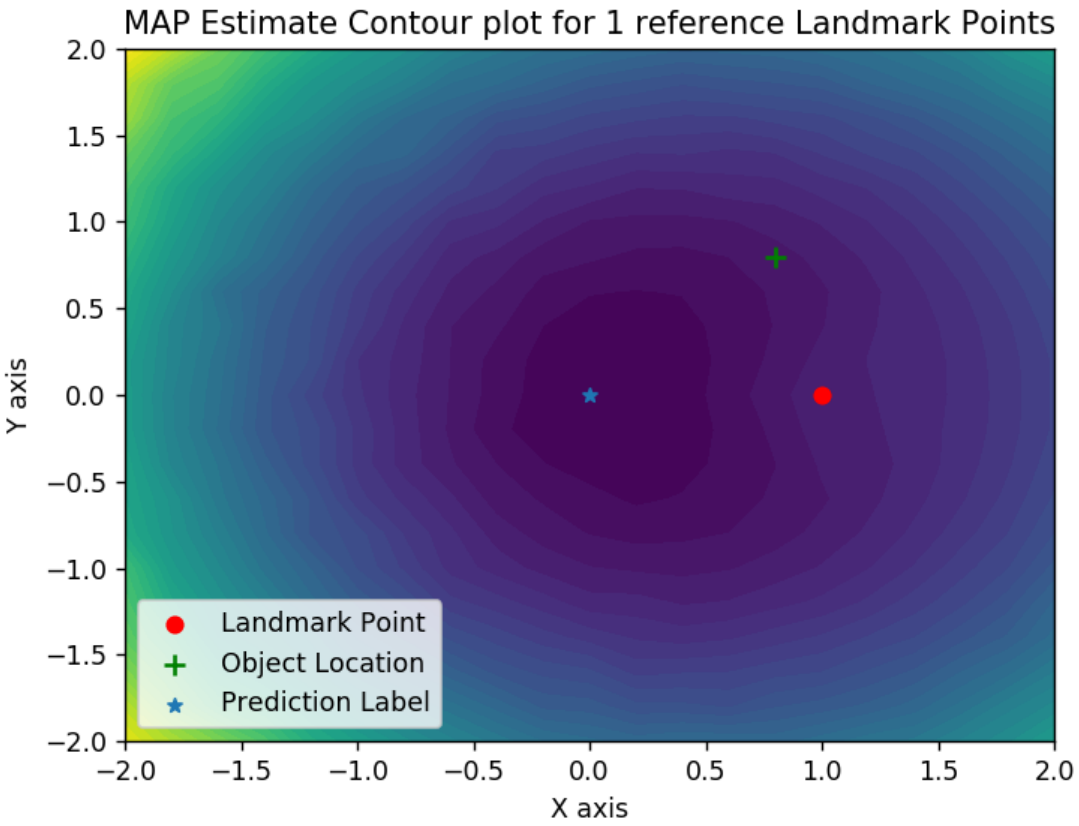
The function below takes every value inside the meshgrid ranging from -2 to +2 and runs it through the MAP estimator and then stores it in the variable Z1. I then plot the contour of X1, Y1 and Z1 points.

In [503]:
```
def generate_contour_plot_for_K(sigma_x, sigma_y, K, X1, Y1, sigma_i):
    Z1 = np.zeros(list(X1.shape))
    predicted_point = []
    for i in range(X1.shape[0]):
        for j in range(X1.shape[1]):
            Z1[i][j] = map_estimator(sigma_x, sigma_y, X1[i][j], Y1[i][j], cal
_ri(K[:, 0], K[:, 1])[0], cal_dti(K[:, 0], K[:, 1], X1[i][j], Y1[i][j]), sigma
_i)
    contour = plt.contourf(X1, Y1, Z1, levels=levels)
    for (x, y) in K:
        plt.scatter(x, y, color='r', marker='o',label='Landmark Point')
    plt.scatter(O_loc[0], O_loc[1], color='g', marker='+', label='Object Locat
ion', s=70)
    plt.scatter(X1.reshape(X1.size, 1)[np.argmin(Z1)], Y1.reshape(Y1.size, 1)[
np.argmin(Z1)], marker='*', label='Prediction Label', s=30)
    plt.xlabel('X axis')
    plt.ylabel('Y axis')
    plt.title('MAP Estimate Contour plot for {} reference Landmark Points'.for
mat(K.shape[0]))
    plt.legend(loc='lower left')
```

Case 1: K = {1}

In [507]:
```
K = np.array([[1, 0]])
```

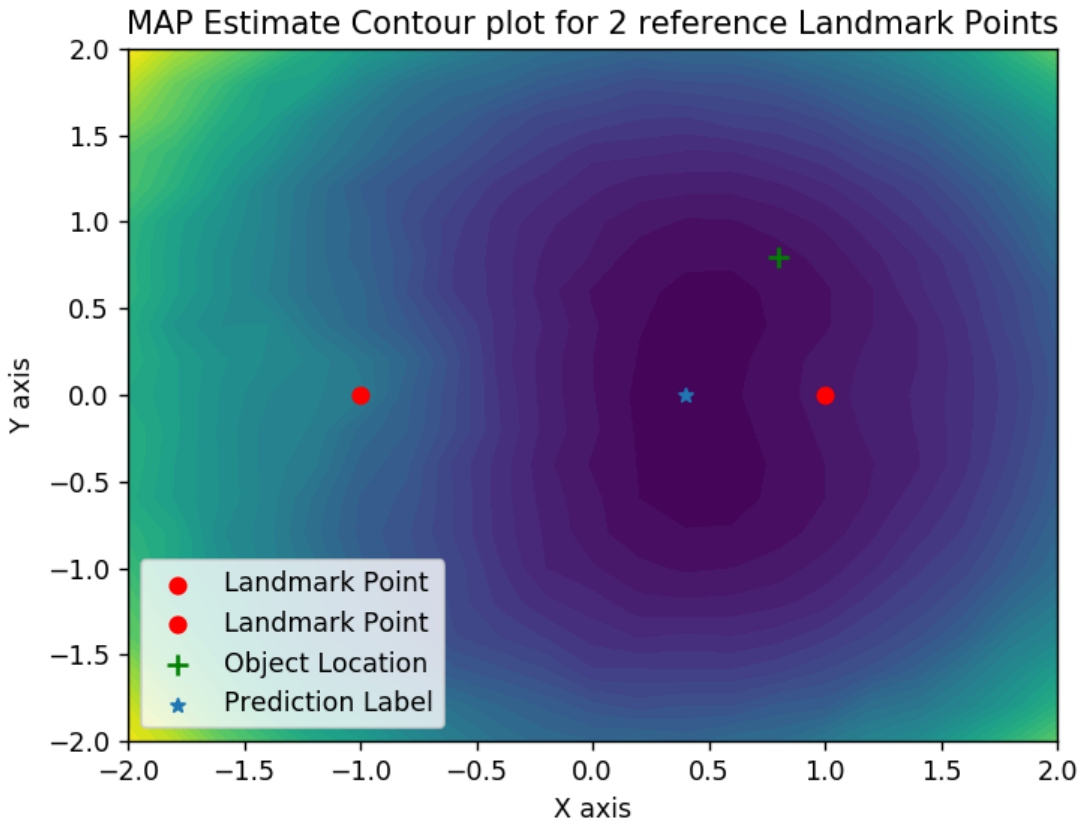
In [508]:
```
%matplotlib notebook
generate_contour_plot_for_K(sigma_x, sigma_y, K, X1, Y1, sigma_i)
```



Case 2: K = {2}

In [509]:
```
K = np.array([[1, 0], [-1, 0]])
```

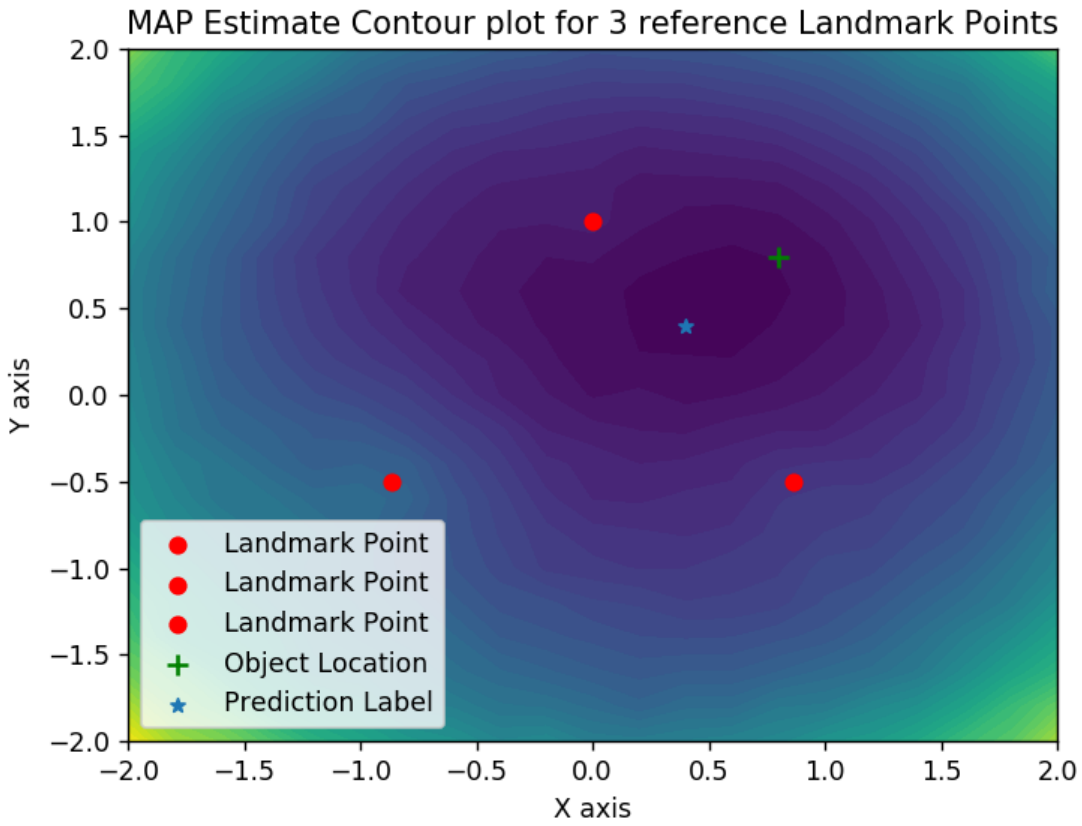
In [510]:
```
%matplotlib notebook
generate_contour_plot_for_K(sigma_x, sigma_y, K, X1, Y1, sigma_i)
```



Case 2: K = {3}

In [511]:
```
K = np.array([[-0.866, -0.5], [0, 1], [0.866, -0.5]])
```

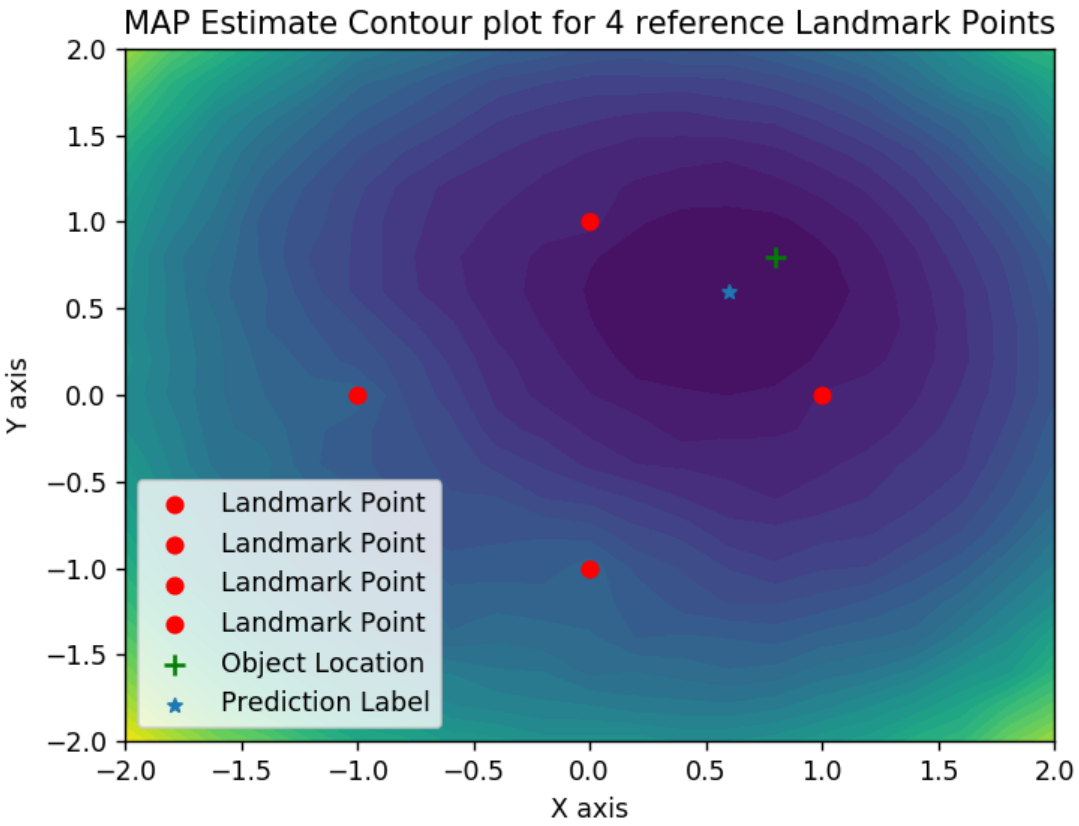
In [512]:
```
%matplotlib notebook
generate_contour_plot_for_K(sigma_x, sigma_y, K, X1, Y1, sigma_i)
```



Case 2: K = {4}

In [513]:
```
K = np.array([[1, 0], [0, 1], [-1, 0], [0, -1]])
```

In [517]:
```
%matplotlib notebook
generate_contour_plot_for_K(sigma_x, sigma_y, K, X1, Y1, sigma_i)
```



Conclusion

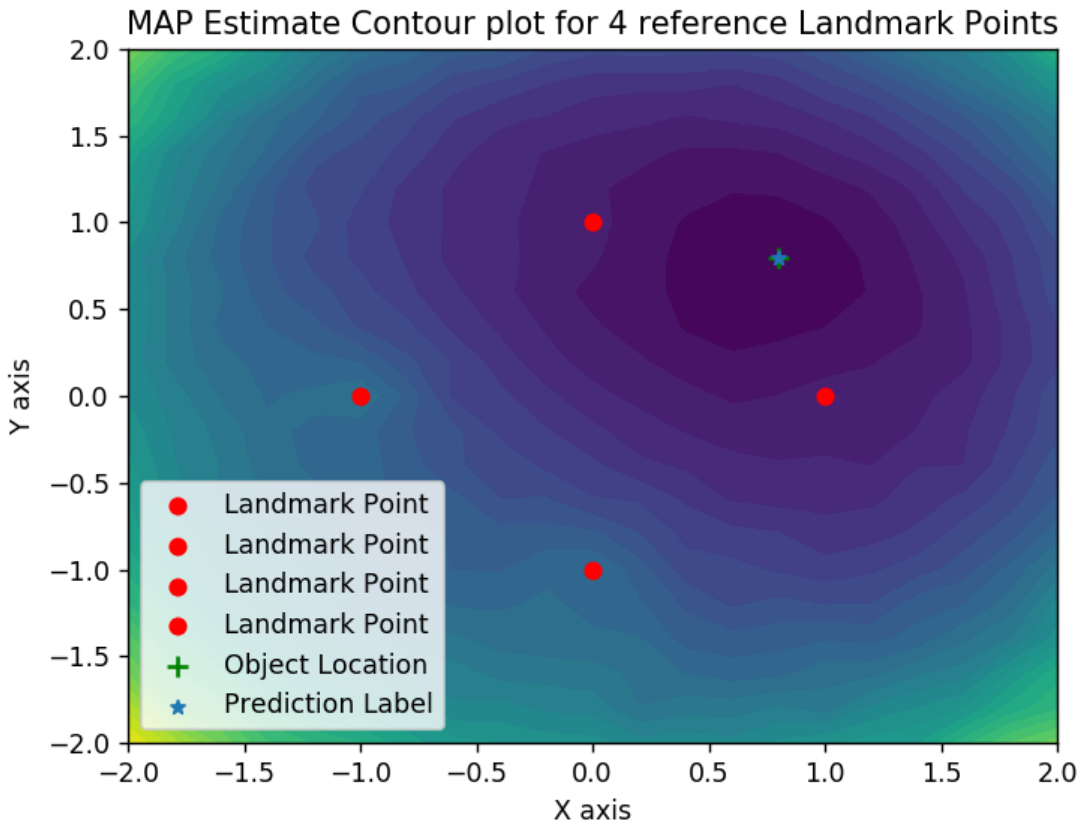
We derived the MAP estimation function for localizing the object position. The likelihood we achieved along with the prior term have been coded above in the MAP estimation function. Considering 1 referene landmark point, the distance measure from the object position along with the noise is the value of ri. Now we don't know where the object lies and we have to find it based on the distance we have measured. So that results in increasing the likelihood of our object to lie inside a circle centered around the Landmark point, with a radius of the distance measured. The prior that we have is Zero mean with Sigma X and Sigma Y being the values that control the variance of that point. That means it is centered at origin and its job is to pull the localization towards the center. Hence the convergence of the Prior and the Likelihood will tilt the contour shape in a direction of where the object position is likely to be, but not exactly depending upon the intensity of the noise.

As we reduce the noise, the maximum MAP estimate likelihood as per the contour converges towards the True Object position.

Along with this, as is evident from the plots above, as the number of Landmark points increase, the MAP estimate prediction i.e the center of the contour region starts getting closer to the True Object position.

Infact, if we reduce the noise further, for the case of 4 landmark points, we can actually see that the predicted point converges with the actual location of the object.

In [519]:
```
%matplotlib notebook
K = np.array([[1, 0], [0, 1], [-1, 0], [0, -1]])
generate_contour_plot_for_K(sigma_x, sigma_y, K, X1, Y1, 0.02)
```



So I would say that the localization gets more certain with the increasing number of Landmark Points

Along with that, we can observe that the innermost contour shape and area reduces as the number of landmark points increase. This goes to say that as we get a better idea of the likelihood of where the object lies, we start getting more certain and the MAP estimation reflects that with the shape of the contour.

In [ ]:

3.] Given :- We have a third degree polynomial

$$y = ax^3 + bx^2 + cx + d + v$$

$$\Rightarrow y = W^T b(x) + v$$

where, $W = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$ $b(x) = \begin{bmatrix} x^3 \\ x^2 \\ x \\ 1 \end{bmatrix}$

and $v \sim \mathcal{N}(0, \sigma^2)$

Prior of $W \Rightarrow W \sim \mathcal{N}(0, \gamma^2 I)$

W_{True} are the true parameters of y.

→ Estimate the MAP function parameter W.

Solution :- $y = W^T b(x) + v$

From the linearity property $x = Az + b$,

we know $y \sim \mathcal{N}(W^T b(x), \sigma^2)$

To estimate W, we have our MAP estimator:

$$W_{MAP} = \underset{W}{\operatorname{argmax}}\ p(W; y|x)$$

Applying Bayes Rule and disregarding the evidence;

$$W_{MAP} = \underset{W}{\operatorname{argmax}}\ \underbrace{p(y|x; W)}_{\text{likelihood}} * \underbrace{p(W)}_{\text{prior}}$$

Taking log on both sides:

$$\ln W_{MAP} = \underset{W}{\operatorname{argmax}} \sum_{i=1}^{K} \ln p(y_i | x_i; W) + \ln p(W)$$

(Considering i samples)

Solving each term independently :-

$$\ln p(y_i | x_i; W) = \ln \frac{1}{\sqrt{2\pi}\,\sigma} \times e^{-\frac{1}{2}\frac{(y_i - W^T b(x_i))^2}{\sigma^2}}$$

$$= \underbrace{\ln \frac{\sigma^{-1}}{\sqrt{2\pi}}}_{\text{Constant}} + \frac{1}{2} \frac{(y_i - W^T b(x_i))^2}{\sigma^2}$$

$$\ln p(W) = \ln \frac{|\gamma^2 I|^{-1/2}}{2\pi} \times e^{-\frac{1}{2} W^T |\gamma^2 I|^{-1} W}$$

$$= \underbrace{\ln \frac{|\gamma^2 I|^{-1/2}}{2\pi}}_{\text{Constant}} + \left(-\frac{1}{2}\right) W^T |\gamma^2 I|^{-1} W$$

$$\therefore \ln W_{MAP} = \underset{W}{\operatorname{argmax}} \sum_{i=1}^{K} \left[-\frac{1}{2} \frac{(y_i - W^T b(x_i))^2}{\sigma^2} \right] - \frac{1}{2} W^T |\gamma^2 I|^{-1} W$$

$$\Rightarrow \ln W_{MAP} = \underset{W}{\operatorname{argmin}} \sum_{i=1}^{K} \frac{(y_i - W^T b(x_i))^2}{\sigma^2} - W^T |\gamma^2 I|^{-1} W$$

To minimize the RHS, we take a gradient at W_{MAP}.

$$\nabla_{w=w_{MAP}} \Rightarrow \frac{2}{\sigma^2}\sum_{i=1}^{K}\left(y_i - w_{MAP}^T b(x_i)\right) b(x_i)^T + (-2)\left(\gamma^2 I\right)^{-1} w_{MAP} = 0$$

$$\Rightarrow \frac{1}{\sigma^2}\sum_{i=1}^{K} b(x_i)\, y_i - \frac{1}{\sigma^2}\sum_{i=1}^{K} w_{MAP}^T\, b(x_i)\, b(x_i)^T +$$

$$(-1)\left(\gamma^2 I\right)^{-1} w_{MAP} = 0$$

Multiplying by σ^2:

$$\sum_{i=1}^{K} b(x_i)\, y_i = \sum_{i=1}^{K} b(x_i)\, b(x_i)^T w_{MAP} + \sigma^2\left(\gamma^2 I\right)^{-1} w_{MAP}$$

$$\therefore \boxed{w_{MAP} = \left[\sum_{i=1}^{K} b(x_i) b(x_i)^T + \sigma^2\left(\gamma^2 I\right)^{-1}\right]^{-1} \times \sum_{i=1}^{K} b(x_i)\, y_i}$$

This is the MAP estimate of the parameter w

Question 3 (35%)

We have two dimensional real-valued data (x;y) that is generated by the following procedure, where all polynomial coefficients are real-valued and v □N (0;s2): y = ax3+bx2+cx+d+v (2) Let w = [a;b;c;d]T be the parameter vector for this polynomial relationship. Given the knowledge of s and that the relationship between x and y is a cubic polynomial corrupted by additive noise as shown above, iid samples D = (x1;y1); : : : ; (xN;yN) generated by the procedure using the true value of the parameters (say wtrue), and a Gaussian prior w □N (0; g2I), where I is the 4□4 identity matrix, determine the MAP estimate for the parameter vector. Write code to generate N = 10 samples according to the model, draw iid x □Uni f orm[□1;1] and choose the true parameters to place the real roots (for simplicity) for the polynomial in the interval [□1;1]. Pick a value for s (that makes the noise level sufficiently large), and keep it constant for the experiments. Repeat the following for different values of g (note that as g increases the MAP estimates approach the ML estimate). Generate samples of x and v, then determine the corrsponding values of y. Given this particular realization of the dataset D, for each value of g, find the MAP estimate of the parameter vector and calculate the squared L2 distance between the true parameter vector and this estimate. For each value of g perform at least 100 experiments, where the data is independently generated according to the procedure, while keeping the true parameters fixed. Report the minimum, 25%, median, 75%, and maximum values of these squared-error values, kwtrue□wMAPk22 , for the MAP estimator for each value of g in a single plot. How do these curves behave as this parameter for the prior changes? Note: Make sure to change gamma to cover a sufficiently broad range to see its effects at multiple scales. To achieve this, you might want to select values for this hyperparameter as power of 10 linearly spaced from □B to +B, so that you cover the interval [10□B;10B] logarithmically. Choose B > 0 well.

In [1]:
```
# Importing the necessary packages
import numpy as np
import math
import matplotlib.pyplot as plt
```

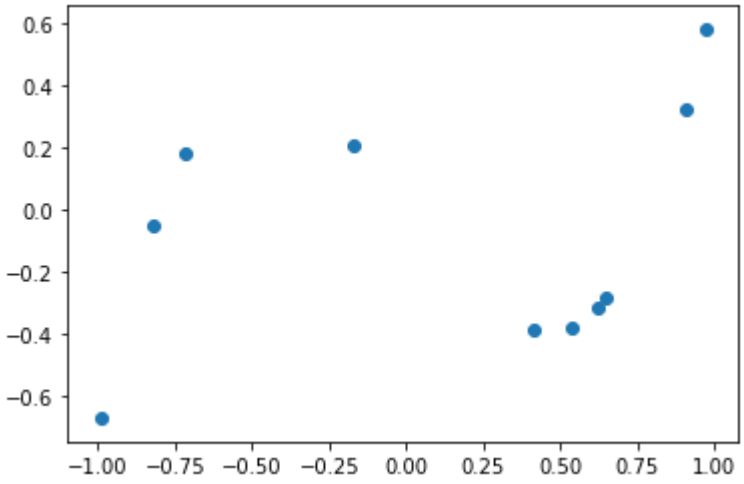
In [2]:
```
# Defining the polynomial function
def b(x):
    return [x**3, x**2, x, 1]
```

In [51]:
```
# Generating the samples without noise
W_true = [2, 0, -1.28, 0]
X = np.random.uniform(-1, 1, 10)
Y_pure = []
for x in X:
    Y_pure.append(np.array(W_true).dot(b(x)))
```

Plotting the third degree polynomial having roots between -1, 1 without any noise

In [52]:
```
plt.scatter(X,Y_pure)
```

Out[52]:
```
<matplotlib.collections.PathCollection at 0x25e737e4a90>
```

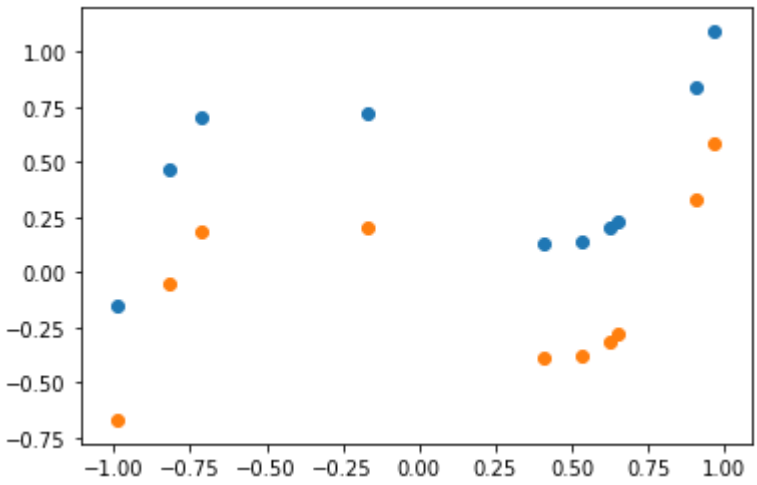


In [55]:
```
# Corrupting the samples with noise
Y = []
v = np.random.normal(loc=0, scale=0.4)
for y in Y_pure:
    Y.append(y + v)
```

Plotting the same third degree polynomial with noise

In [101]:
```
plt.scatter(X, Y)
plt.scatter(X, Y_pure)
```

Out[101]:
```
<matplotlib.collections.PathCollection at 0x25e73f86cf8>
```



In [76]:
```
def calculate_BX(X):
    BX = []
    [BX.append(b(x)) for x in X]
    return BX
```

In [93]:
```
def wmap(Y, BX, gamma, sigma):
    gamma_square_matrix = (sigma**2)*np.linalg.inv(gamma**2*np.identity(4))
    term2 = []
    [term2.append(np.matmul(np.reshape(bx, (4,1)), np.reshape(bx, (1,4)))) for
bx in BX]
    left_term = gamma_square_matrix + sum(term2)
    BY_term = []
    [BY_term.append(np.reshape(bx, (4,1))*y) for y,bx in zip(Y, BX)]
    wmap = np.matmul(np.linalg.inv(left_term),sum(BY_term))
    return wmap
```

In [95]:
```
W_MAP = wmap(Y, calculate_BX(X), 10**(-1), 0.8)
```

In [97]:
```
L2_squared_distance = np.linalg.norm(np.reshape(W_true, (4,1)) - W_MAP)
```

In [98]:
```
def generate_dataset(sigma, W_true):
    # Generating the samples without noise
    X = np.random.uniform(-1, 1, 10)
    Y_pure = []
    for x in X:
        Y_pure.append(np.array(W_true).dot(b(x)))
    # Corrupting the samples with noise
    Y = []
    v = np.random.normal(loc=0, scale=0.4)
    for y in Y_pure:
        Y.append(y + v)
    return X, Y
```

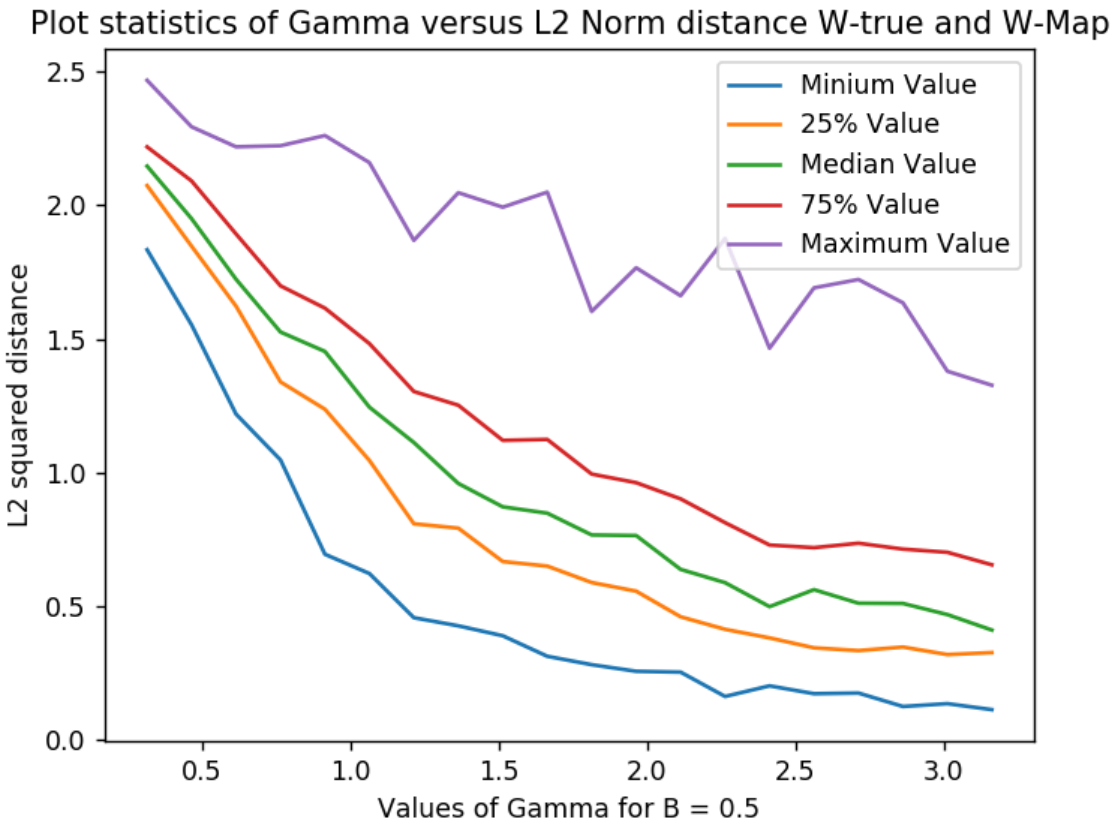
In [178]:
```
W_true = [2, 0, -1.28, 0]
```

In [223]:
```
# Generating W-Map for a range of gamma values by performing 100 experiments f
or each value
# Sampling individually in every experiment
def get_gamma_l2list_dict(gamma, sigma):
    gamma_l2list_dict = {}
    for gamma in np.linspace(10**(-gamma), 10**(gamma), 20):
        L2_squared_distance_list = []
        WMAP_list = []
        for i in range(100):
            X, Y = generate_dataset(sigma, W_true)
            W_MAP = (wmap(Y, calculate_BX(X), gamma, sigma))
            L2_squared_distance_list.append(np.linalg.norm(np.reshape(W_true,
(4,1)) - W_MAP))
            L2_squared_distance_list.sort()
        gamma_l2list_dict[gamma] = L2_squared_distance_list
    return gamma_l2list_dict
```

In [229]:
```
def generate_statistical_plots_for_gamma(gamma, sigma):
    # Initializing the lists that will store the statistical data for each val
ue of gamma
    Min_report = []
    twenty_five_percent = []
    fifty_percent = []
    seventy_five_percent = []
    Max_report = []

    # Getting the L2-list values for the given Gamma through 100 experiments
    gamma_l2list_dict = get_gamma_l2list_dict(gamma, sigma)

    # Populating the lists with appropriate values
    [Min_report.append(l2_list[0]) for l2_list in gamma_l2list_dict.values()]
    [twenty_five_percent.append(l2_list[24]) for l2_list in gamma_l2list_dict.
values()]
    [fifty_percent.append(l2_list[49]) for l2_list in gamma_l2list_dict.values
()]
    [seventy_five_percent.append(l2_list[74]) for l2_list in gamma_l2list_dict
.values()]
    [Max_report.append(l2_list[99]) for l2_list in gamma_l2list_dict.values()]

    # Generating the plots
    plt.plot(list(gamma_l2list_dict.keys()), Min_report, label='Minium Value')
    plt.plot(list(gamma_l2list_dict.keys()), twenty_five_percent, label='25% V
alue')
    plt.plot(list(gamma_l2list_dict.keys()), fifty_percent, label='Median Valu
e')
    plt.plot(list(gamma_l2list_dict.keys()), seventy_five_percent, label='75%
 Value')
    plt.plot(list(gamma_l2list_dict.keys()), Max_report, label='Maximum Value'
)
    plt.xlabel('Values of Gamma for B = {}'.format(gamma))
    plt.ylabel('L2 squared distance')
    plt.title('Plot statistics of Gamma versus L2 Norm distance W-true and W-M
ap')
    plt.legend()
```

Case 1: Gamma = 0.5

In [231]:
```
%matplotlib notebook
generate_statistical_plots_for_gamma(0.5, 0.3)
```

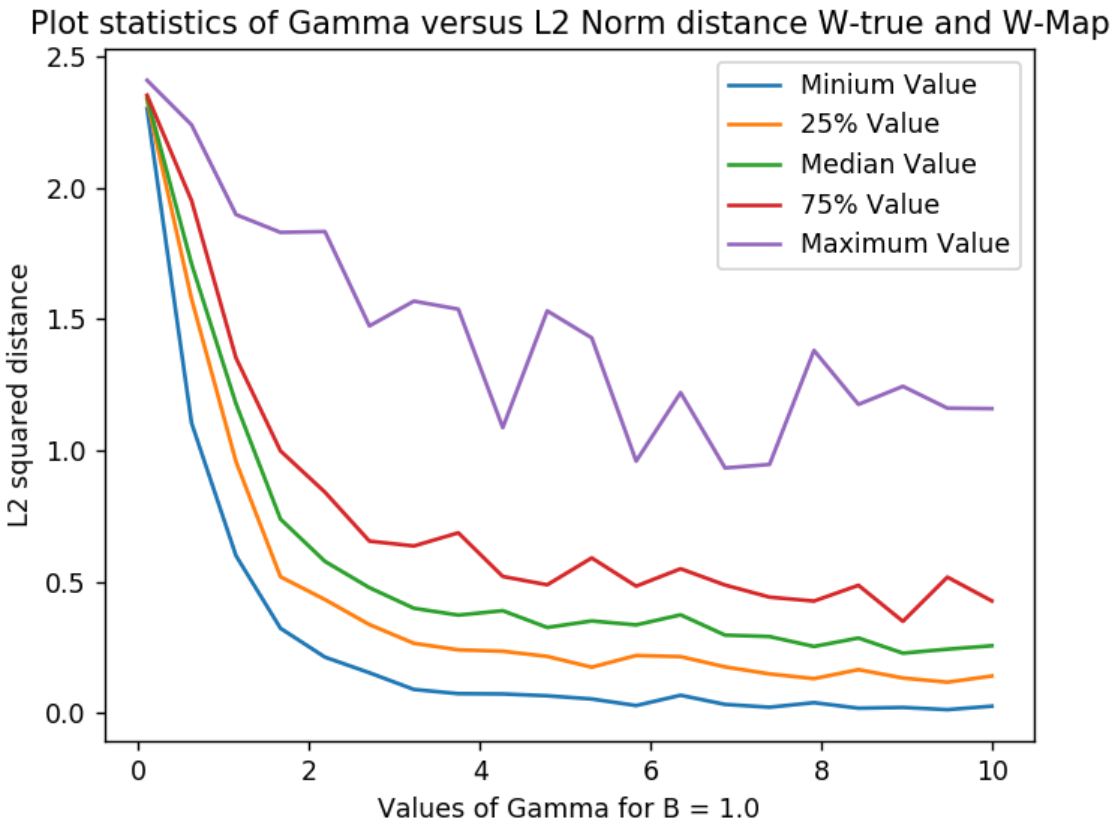


Inference

When the Gamma is really small, there is a high difference between the estimated parameter and the true parameters of W. The Maximum, Median, 25th, 75th and the Minimum values for every Gamma are very far from the True values when the gamma is small. As Gamma starts increasing the values slowly start converging towards the True parameters making the L2 difference less. But the Maximum L2 difference between the Estimated and True parameters still remains high depending upon the noise that we consider for our experiment. And after a certain point, all our curves remain constant and stay at the same level regardless of the change in Gamma.

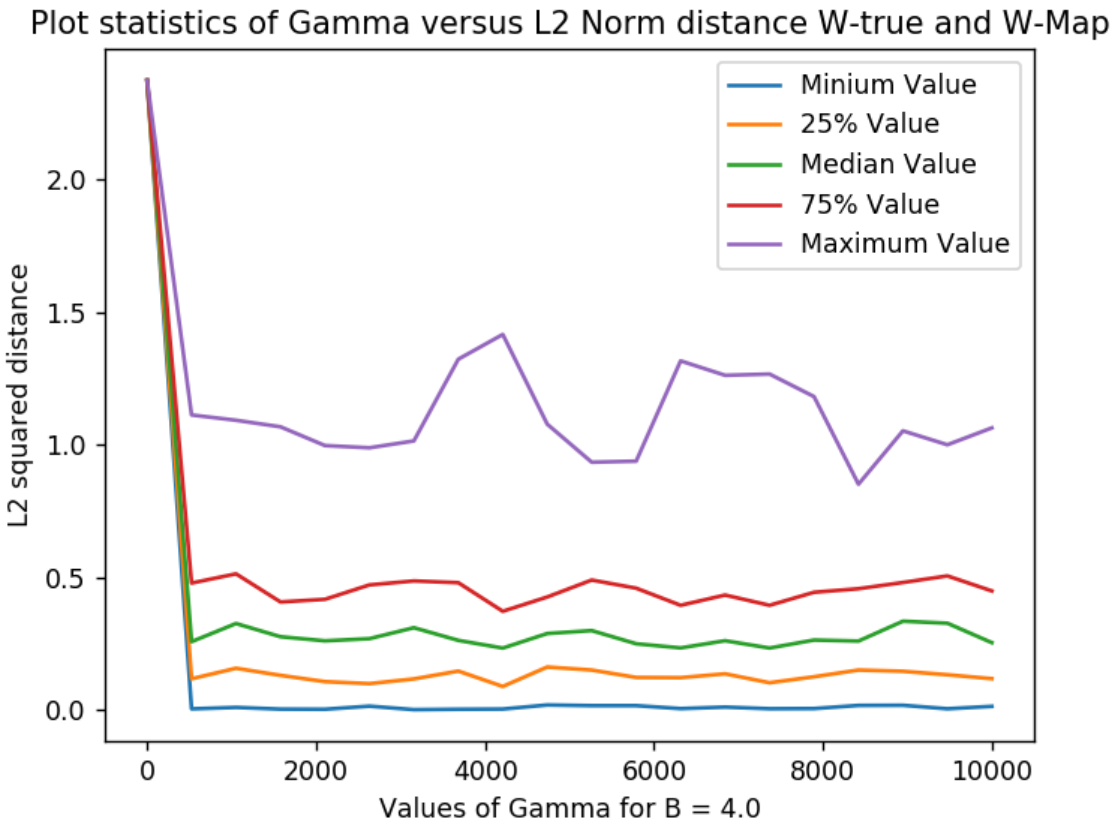
Case 2: Gamma = 1.0

In [232]:
```
%matplotlib notebook
generate_statistical_plots_for_gamma(1.0, 0.3)
```



Case 3: Gamma = 4.0

In [233]:
```
%matplotlib notebook
generate_statistical_plots_for_gamma(4.0, 0.3)
```



In [ ]:
```
```

All codes and original plots and text are present on github for reference in the following folder:

https://github.com/rishabhgks/EECE_5644/tree/exam1/exam_1