Question 1:

In [54]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
```

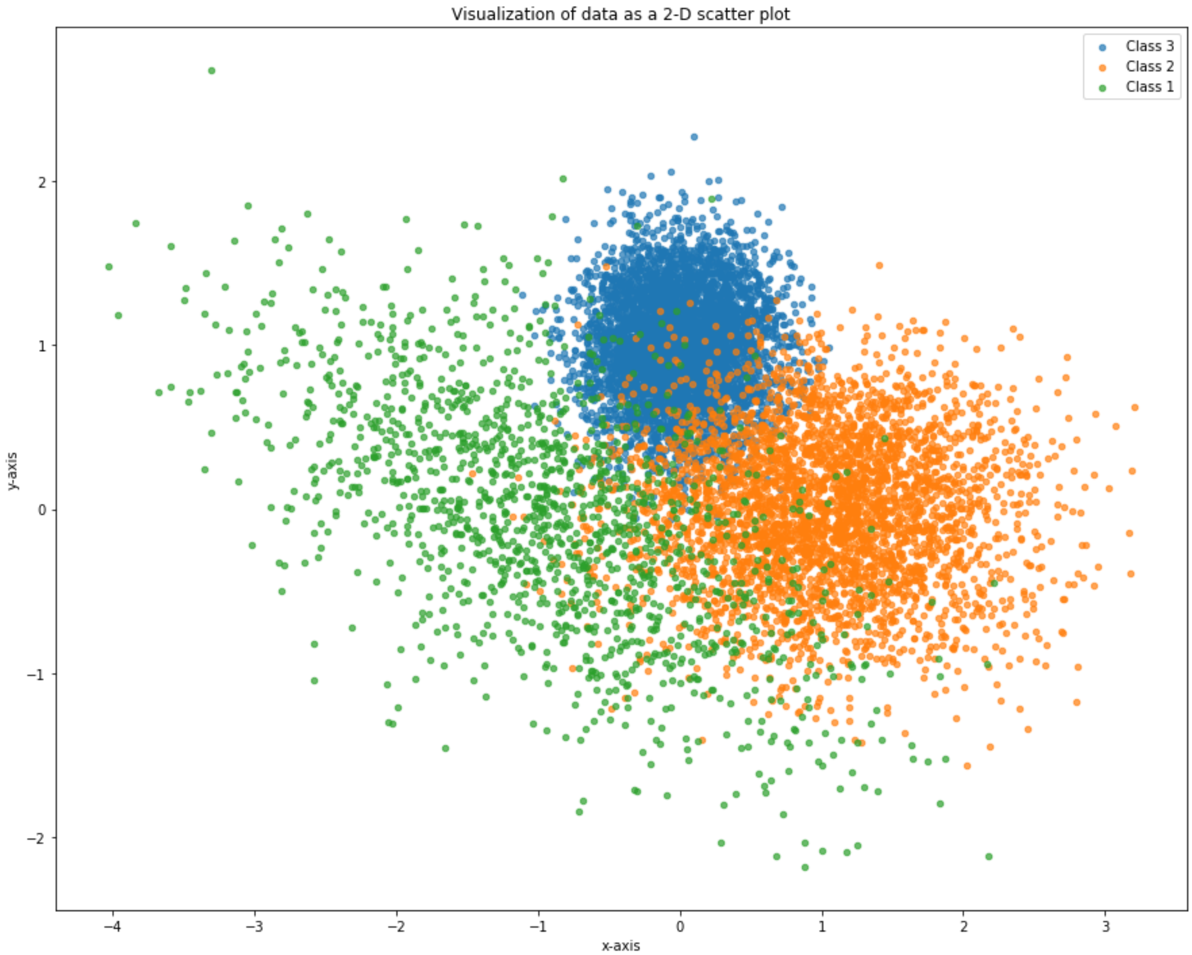
In [55]:
```
data_x = []
data_y = []
data_z = []

N = 10000
prior = [0.15, 0.35, 0.5]
l_1 = 0
l_2 = 0
l_3 = 0

mu_x = [-1, 0]
variance_x = [[1, -0.4], [-0.4, 0.5]]

mu_y = [1, 0]
variance_y = [[0.5, 0], [0, 0.2]]

mu_z = [0, 1]
variance_z = [[0.1, 0], [0, 0.1]]
```

In [56]:
```
#generating sample and checking for prior values
for i in range(N):
    r = np.random.uniform(0, 1, 1)
    if r <= prior[0]:
        l_1 = l_1 + 1
    elif r <= prior[1] + prior[0]:
        l_2 = l_2 + 1
    else:
        l_3 = l_3 + 1

l_1, l_2, l_3
```

Out[56]:
```
(1448, 3517, 5035)
```

In [57]:
```
#generating data according to label
data_x = np.random.multivariate_normal(mu_x, variance_x, l_1)

data_y = np.random.multivariate_normal(mu_y, variance_y, l_2)

data_z = np.random.multivariate_normal(mu_z, variance_z, l_3)
```

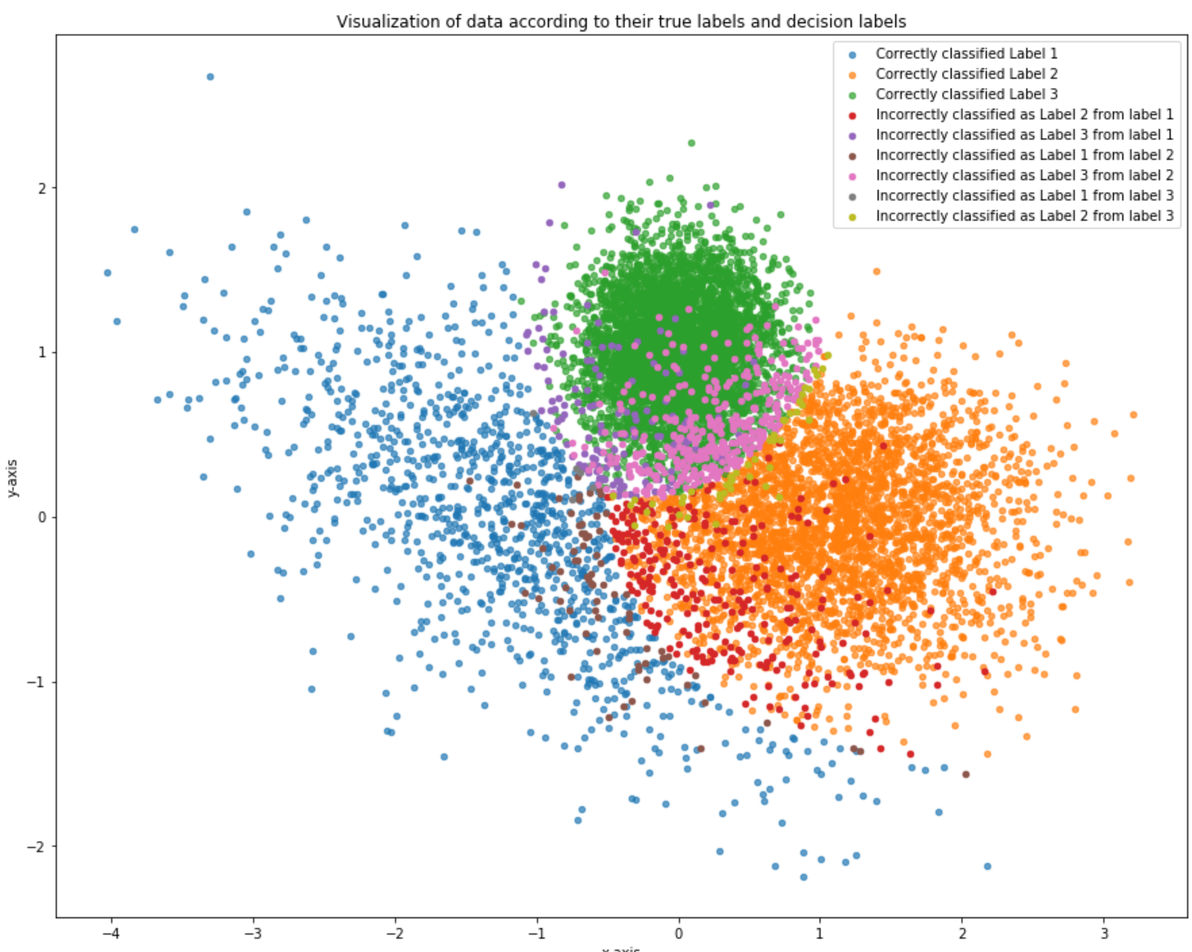
In [58]:
```
fig = plt.figure(figsize=(15, 12))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(data_z)[:, 0], np.array(data_z)[:, 1], s = 20, alpha=0.7,
label='Class 3')
ax.scatter(np.array(data_y)[:, 0], np.array(data_y)[:, 1], s = 20, alpha=0.7,
label='Class 2')
ax.scatter(np.array(data_x)[:, 0], np.array(data_x)[:, 1], s = 20, alpha=0.7,
label='Class 1')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Visualization of data as a 2-D scatter plot')
ax.legend()
plt.show()
```



In [59]:
```
def normal_prob(x, m, v):
    x_t = x
    m_t = m
    np.reshape(x, (2,1))
    np.reshape(m, (2,1))
    p = math.exp(-0.5*np.matmul(np.matmul((x_t-m_t), np.linalg.inv(v)), (x-m
)))/(2*math.pi*np.linalg.det(v))
    return p
```

In [60]:
```
x_right = []
x_error_y = []
x_error_z = []

y_right = []
y_error_x = []
y_error_z = []

z_right = []
z_error_x = []
z_error_y = []

for i in range(l_1):
    p_1 = normal_prob(np.array(data_x)[i, :], mu_x, variance_x)
    p_2 = normal_prob(np.array(data_x)[i, :], mu_y, variance_y)
    p_3 = normal_prob(np.array(data_x)[i, :], mu_z, variance_z)
    if (p_1*prior[0] >= p_2*prior[1]) and (p_1*prior[0] >= p_3*prior[2]):
        x_right.append(np.array(data_x)[i, :])
    elif p_2*prior[1] >= p_3*prior[2]:
        x_error_y.append(np.array(data_x)[i, :])
    else:
        x_error_z.append(np.array(data_x)[i, :])

for i in range(l_2):
    p_1 = normal_prob(np.array(data_y)[i, :], mu_x, variance_x)
    p_2 = normal_prob(np.array(data_y)[i, :], mu_y, variance_y)
    p_3 = normal_prob(np.array(data_y)[i, :], mu_z, variance_z)
    if (p_2*prior[1] >= p_1*prior[0]) and (p_2*prior[1] >= p_3*prior[2]):
        y_right.append(np.array(data_y)[i, :])
    elif p_1*prior[0] >= p_3*prior[2]:
        y_error_x.append(np.array(data_y)[i, :])
    else:
        y_error_z.append(np.array(data_y)[i, :])

for i in range(l_3):
    p_1 = normal_prob(np.array(data_z)[i, :], mu_x, variance_x)
    p_2 = normal_prob(np.array(data_z)[i, :], mu_y, variance_y)
    p_3 = normal_prob(np.array(data_z)[i, :], mu_z, variance_z)
    if (p_3*prior[2] >= p_1*prior[0]) and (p_3*prior[2] >= p_2*prior[1]):
        z_right.append(np.array(data_z)[i, :])
    elif p_1*prior[0] >= p_2*prior[1]:
        z_error_x.append(np.array(data_z)[i, :])
    else:
        z_error_y.append(np.array(data_z)[i, :])
```

In [61]:
```
fig = plt.figure(figsize=(15, 12))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(x_right)[:, 0], np.array(x_right)[:, 1], s = 20, alpha=0.7
, label='Correctly classified Label 1')
ax.scatter(np.array(y_right)[:, 0], np.array(y_right)[:, 1], s = 20, alpha=0.7
, label='Correctly classified Label 2')
ax.scatter(np.array(z_right)[:, 0], np.array(z_right)[:, 1], s = 20, alpha=0.7
, label='Correctly classified Label 3')
ax.scatter(np.array(x_error_y)[:, 0], np.array(x_error_y)[:, 1], s = 20, alpha
=1, label='Incorrectly classified as Label 2 from label 1')
ax.scatter(np.array(x_error_z)[:, 0], np.array(x_error_z)[:, 1], s = 20, alpha
=1, label='Incorrectly classified as Label 3 from label 1')
ax.scatter(np.array(y_error_x)[:, 0], np.array(y_error_x)[:, 1], s = 20, alpha
=1, label='Incorrectly classified as Label 1 from label 2')
ax.scatter(np.array(y_error_z)[:, 0], np.array(y_error_z)[:, 1], s = 20, alpha
=1, label='Incorrectly classified as Label 3 from label 2')
ax.scatter(np.array(z_error_x)[:, 0], np.array(z_error_x)[:, 1], s = 20, alpha
=1, label='Incorrectly classified as Label 1 from label 3')
ax.scatter(np.array(z_error_y)[:, 0], np.array(z_error_y)[:, 1], s = 20, alpha
=1, label='Incorrectly classified as Label 2 from label 3')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Visualization of data according to their true labels and decision l
abels')
ax.legend()
plt.show()
```



In [62]:
```
c_mat = [[len(x_right), len(y_error_x), len(z_error_x)], [len(x_error_y), len(
y_right), len(z_error_y)], [len(x_error_z), len(y_error_z), len(z_right)]]

print("The actual number of samples that were generated from class 1 = {}, cla
ss 2 = {}, and class 3 = {}".format(l_1, l_2, l_3))

print("\nThe confusion matrix is:\n {}".format(np.array(c_mat)))

print("\nThe total number of samples misclassified by the classifier are {}".f
ormat(len(x_error_y) + len(x_error_z) + len(y_error_x) + len(y_error_z) + len(
z_error_x) + len(z_error_y)))

print("\nThe error probability is {}.".format((len(x_error_y) + len(x_error_z)
+ len(y_error_x) + len(y_error_z) + len(z_error_x) + len(z_error_y))/N))
```

```
The actual number of samples that were generated from class 1 = 1448, class 2
= 3517, and class 3 = 5035

The confusion matrix is:
 [[1051   72    5]
 [ 264 3077   46]
 [ 133  368 4984]]

The total number of samples misclassified by the classifier are 888

The error probability is 0.0888.
```

Description of Results:

1) The number of samples for each class have been distributed randomly by a uniform distribution with parameters (0,1). The samples themselves have been distributed randomly in a Gaussian distribution of the respectives parameters given in the question.

2) The confusion matrix shows all the classifications done by the classifier, which includes all the correct classifications as well as the mis-classifications. The confusion matrix is sufficient for us to derive most of the basic inferences about the samples and the classifier. Here, we can see that a majority of the samples are covered in the diagonal elements which shows that a major portion of the data has been classified correctly.

3) The total number of samples misclassified by the classifier are 873.

4) The probability of error is given by: Probability of error = Total number of misclassified sample / Total number of samples = 873/10000 = 0.0873

5) The scatter plot helps us infer numerous properties regarding the classifications of various samples by the classifier. There are total of 3 classes and hence 6 misclassifications possible. All the 9 (3+6) cases of markings have been covered in the above graph.

In [ ]:
```

```

Question 2

Given: True position of object $:= \begin{bmatrix} x_T \\ y_T \end{bmatrix}$

Coordinates of the landmarks $\rightarrow \left\{ \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, \ldots, \begin{bmatrix} x_K \\ y_K \end{bmatrix} \right\}$

Range measurements: $r = d_{T_i} + n_i$ for $i \in \{1, \ldots, K\}$

where $d_{T_i} = \left\| \begin{bmatrix} x_T \\ y_T \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right\|$

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

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

To note:

1) Since $r_i = d_{T_i} + n_i$, and $n \sim \mathcal{N}(0, \sigma_i^2)$,
therefore, $r_i \sim \mathcal{N}(d_{T_i}, \sigma_i^2)$

Objective fn.:

The objective function to determine the MAP estimate of the object position is given by:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{MAP} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \; P\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| r\right)$$

Applying Bayes Theorem,

$$\begin{bmatrix} x \\ y \end{bmatrix}_{MAP} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \; P\left(r \middle| \begin{bmatrix} x \\ y \end{bmatrix}\right) \cdot P\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \left(\sum_{i=1}^{K} P\left(r_i \middle| \begin{bmatrix} x \\ y \end{bmatrix}\right)\right) P\begin{bmatrix} x \\ y \end{bmatrix}$$

Let a variable $X = \begin{bmatrix} x \\ y \end{bmatrix}$ for simpler derivation understanding.

$\therefore$ Substituting the pdf's of each function,

$$X_{MAP} = \underset{X}{\operatorname{argmax}} \left(\sum_{i=1}^{K} (2\pi)^{-n/2} |\sigma_i^2|^{-1/2} \cdot e^{-\frac{1}{2}\frac{(r_i - d_{T_i})^2}{\sigma_i^2}} \right) \cdot \left((2\pi\sigma_x\sigma_y)^{-1} \right. \\ \left. e^{-\frac{1}{2} \{X^T \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} X} \right)$$

$\longrightarrow$

Applying ~~log~~ ln(log) function to the RHS (Since it will not affect the maximum point),

$$X_{MAP} = \underset{X}{\operatorname{argmax}} \left(\sum_{i=1}^{K} \underline{-\frac{n}{2}\ln(2\pi)} - \underline{\frac{1}{2}\ln(\sigma_i^2)} - \frac{1}{2}\frac{(r_i - d_{Ti})^2}{\sigma_i^2} \right)$$

$$- \underline{\ln(2\pi\sigma_x\sigma_y)} - \frac{1}{2}X^T \cdot \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} X$$

All the constant values (underlined) can be ignored since they do not ~~the~~ affect the optimization point.

$$\therefore X_{MAP} = \underset{X}{\operatorname{argmax}} \sum_{i=1}^{K} -\frac{1}{2}\frac{(r_i - d_{Ti})^2}{\sigma_i^2} - \frac{1}{2}X^T \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} X$$

$$= \underset{X}{\operatorname{argmax}} -\frac{1}{2}\left(\sum_{i=1}^{K} \frac{(r_i - d_{Ti})^2}{\sigma_i^2} + X^T \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} X \right)$$

Maximizing a negative value is equivalent to minimising the positive coefficient of the negative sign. Therefore,

$$X_{MAP} = \underset{X}{\operatorname{argmin}} \sum_{i=1}^{K} \frac{(r_i - d_{Ti})^2}{\sigma_i^2} + X^T \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} X$$

Substituting the value of $X = \begin{bmatrix} x \\ y \end{bmatrix}$,

$$\begin{bmatrix} x \\ y \end{bmatrix}_{MAP} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmin}} \sum_{i=1}^{K} \frac{(r_i - d_{Ti})^2}{\sigma_i^2} + \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmin}} \sum_{i=1}^{K} \frac{(r_i - d_{Ti})^2}{\sigma_i^2} + \frac{1}{\sigma_x^2\sigma_y^2} \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \sigma_y^2 & 0 \\ 0 & \sigma_x^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmin}} \sum_{i=1}^{K} \frac{(r_i - d_{Ti})^2}{\sigma_i^2} + \frac{1}{\sigma_x^2 \cdot \sigma_y^2} \cdot \begin{bmatrix} x \cdot \sigma_y^2 & y \cdot \sigma_x^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmin}} \sum_{i=1}^{K} \frac{(r_i - d_{Ti})^2}{\sigma_i^2} + \frac{1}{\sigma_x^2 \cdot \sigma_y^2} \cdot (x^2 \cdot \sigma_y^2 + y^2 \cdot \sigma_x^2)$$

$\longrightarrow$

$$\therefore \begin{bmatrix} x \\ y \end{bmatrix}_{MAP} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmin}} \sum_{i=1}^{K} \frac{(r_i - d_{T_i})^2}{\sigma_i^2} + \frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}$$

This is the objective function (simplified).

Solution 2: K=1

In [967]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
```

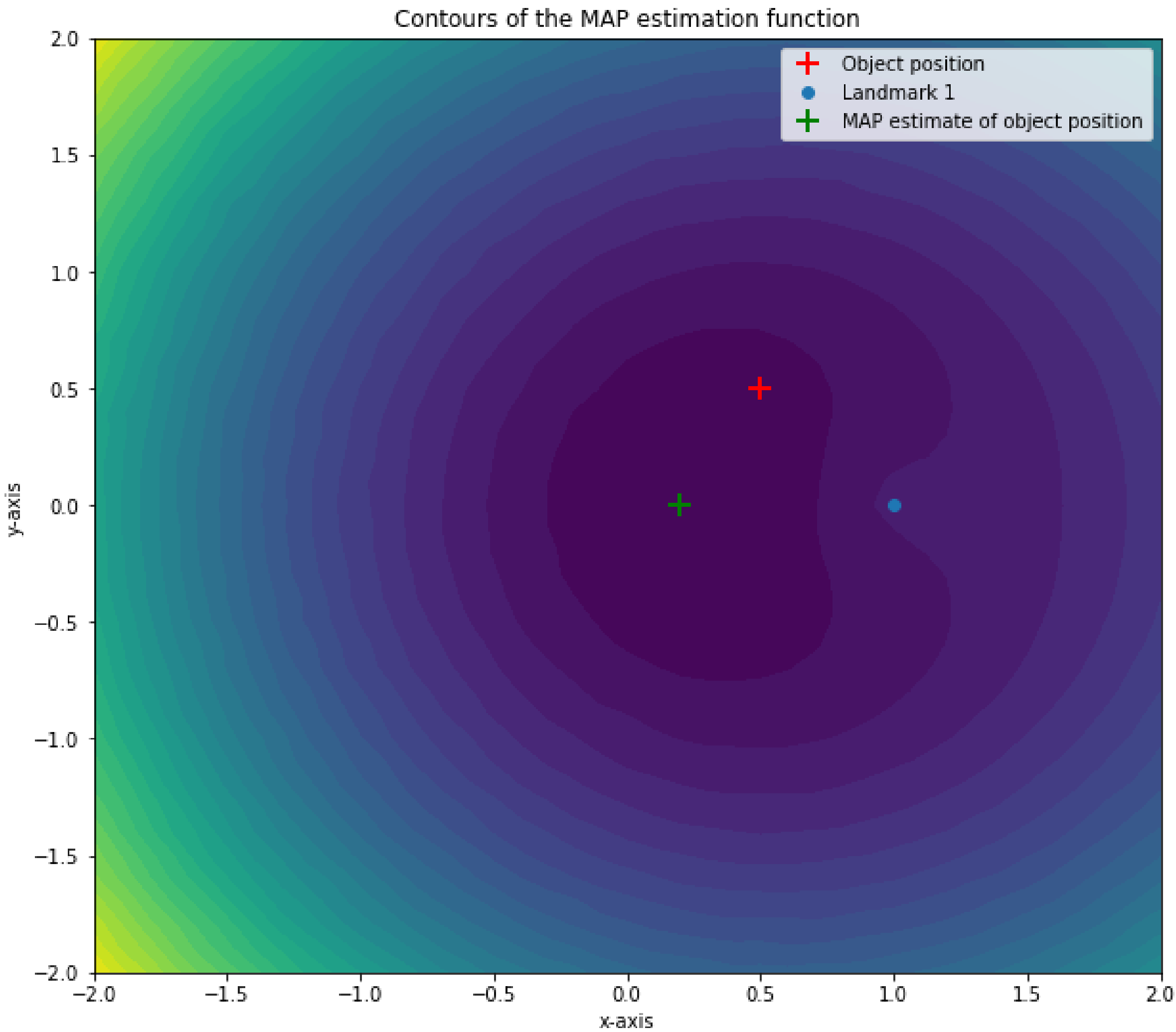
In [968]:
```
x_points = np.arange(-2, 2.05, 0.1)
y_points = np.arange(-2, 2.05, 0.1)
x_mesh, y_mesh = np.meshgrid(x_points, y_points)
```

In [969]:
```
xy_map = np.zeros((len(x_mesh), len(y_mesh)))
landmark = np.zeros((2,4))
check = 1

x_t = 0.5
y_t = 0.5
land_x = [1, -1, 0, 0]
land_y = [0, 0, 1, -1]
K = 1

sig_i = 0.1
sig_x = 0.15
sig_y = 0.15
```

In [970]:
```
def cal_dti(x_t_d, y_t_d, x_i_d, y_i_d):
    dti = math.sqrt((x_t_d - x_i_d)**2 + (y_t_d - y_i_d)**2)
    return dti
```

In [971]:
```
for i in range(len(x_mesh)):
    for j in range(len(y_mesh)):
        likelihood = 0
        for q in range(K):
            n_i = np.random.normal(0, sig_i**2)
            r_i = cal_dti(x_t, y_t, land_x[q], land_y[q]) + n_i
            likelihood = likelihood + ((r_i - cal_dti(land_x[q], land_y[q], x_
mesh[i,j], y_mesh[i,j]))**2)/(sig_i**2)
        prior = (x_mesh[i,j]**2)/(sig_x**2) + (y_mesh[i,j]**2)/(sig_y**2)
        xy_map[i, j] = likelihood + prior
        if check == 1:
            mini = xy_map[i,j]
            check = 2
        if xy_map[i, j] < mini:
            mini = xy_map[i,j]
            mini_x = x_mesh[i,j]
            mini_y = y_mesh[i,j]
```

In [972]:
```
fig = plt.figure(figsize=(10, 9))
ax = fig.add_subplot(1, 1, 1)
cs = ax.contourf(x_mesh, y_mesh, xy_map, levels = 30)
ax.plot(x_t, y_t, 'r+', markeredgewidth = 2, markersize = 12, label = 'Object
 position')
for q in range(K):
    ax.plot(land_x[q], land_y[q], 'o', label = 'Landmark {}'.format(q+1))
ax.plot(mini_x, mini_y, 'g+', markeredgewidth = 2, markersize = 12, label = 'M
AP estimate of object position')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Contours of the MAP estimation function')
ax.legend()
plt.show()
```



In [ ]:

K=2

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
```

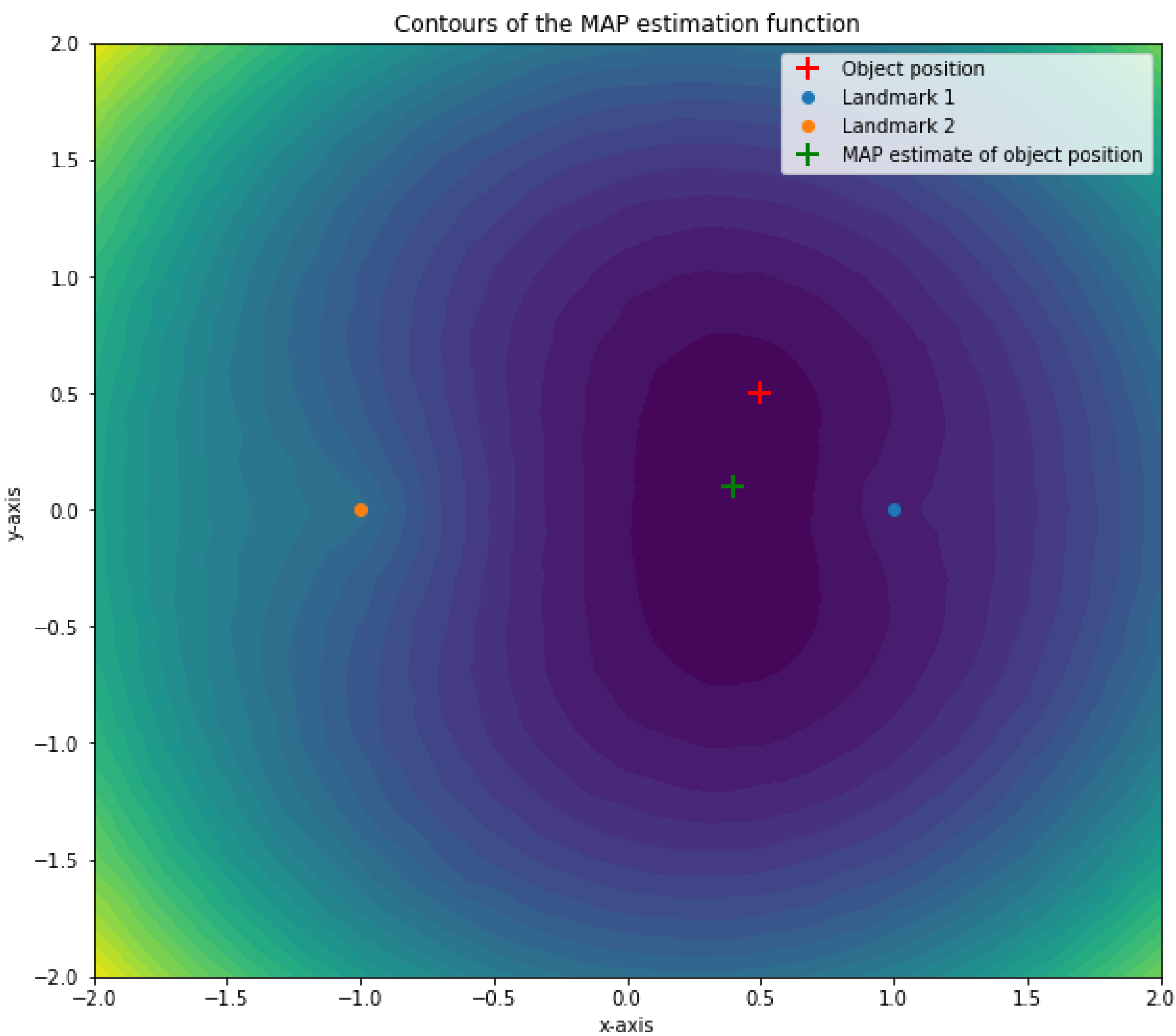
In [39]:
```
x_points = np.arange(-2, 2.05, 0.1)
y_points = np.arange(-2, 2.05, 0.1)
x_mesh, y_mesh = np.meshgrid(x_points, y_points)
```

In [40]:
```
xy_map = np.zeros((len(x_mesh), len(y_mesh)))
landmark = np.zeros((2,4))
check = 1

x_t = 0.5
y_t = 0.5
land_x = [1, -1, 0, 0]
land_y = [0, 0, 1, -1]
K = 2

sig_i = 0.1
sig_x = 0.15
sig_y = 0.15
```

In [41]:
```
def cal_dti(x_t_d, y_t_d, x_i_d, y_i_d):
    dti = math.sqrt((x_t_d - x_i_d)**2 + (y_t_d - y_i_d)**2)
    return dti
```

In [42]:
```
for i in range(len(x_mesh)):
    for j in range(len(y_mesh)):
        likelihood = 0
        for q in range(K):
            n_i = np.random.normal(0, sig_i**2)
            r_i = cal_dti(x_t, y_t, land_x[q], land_y[q]) + n_i
            likelihood = likelihood + ((r_i - cal_dti(land_x[q], land_y[q], x_
mesh[i,j], y_mesh[i,j]))**2)/(sig_i**2)
        prior = (x_mesh[i,j]**2)/(sig_x**2) + (y_mesh[i,j]**2)/(sig_y**2)
        xy_map[i, j] = likelihood + prior
        if check == 1:
            mini = xy_map[i,j]
            check = 2
        if xy_map[i, j] < mini:
            mini = xy_map[i,j]
            mini_x = x_mesh[i,j]
            mini_y = y_mesh[i,j]
```

In [43]:
```
fig = plt.figure(figsize=(10, 9))
ax = fig.add_subplot(1, 1, 1)
cs = ax.contourf(x_mesh, y_mesh, xy_map, levels = 30)
ax.plot(x_t, y_t, 'r+', markeredgewidth = 2, markersize = 12, label = 'Object
 position')
for q in range(K):
    ax.plot(land_x[q], land_y[q], 'o', label = 'Landmark {}'.format(q+1))
ax.plot(mini_x, mini_y, 'g+', markeredgewidth = 2, markersize = 12, label = 'M
AP estimate of object position')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Contours of the MAP estimation function')
ax.legend()
plt.show()
```



In [ ]:

K=3

In [34]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
```

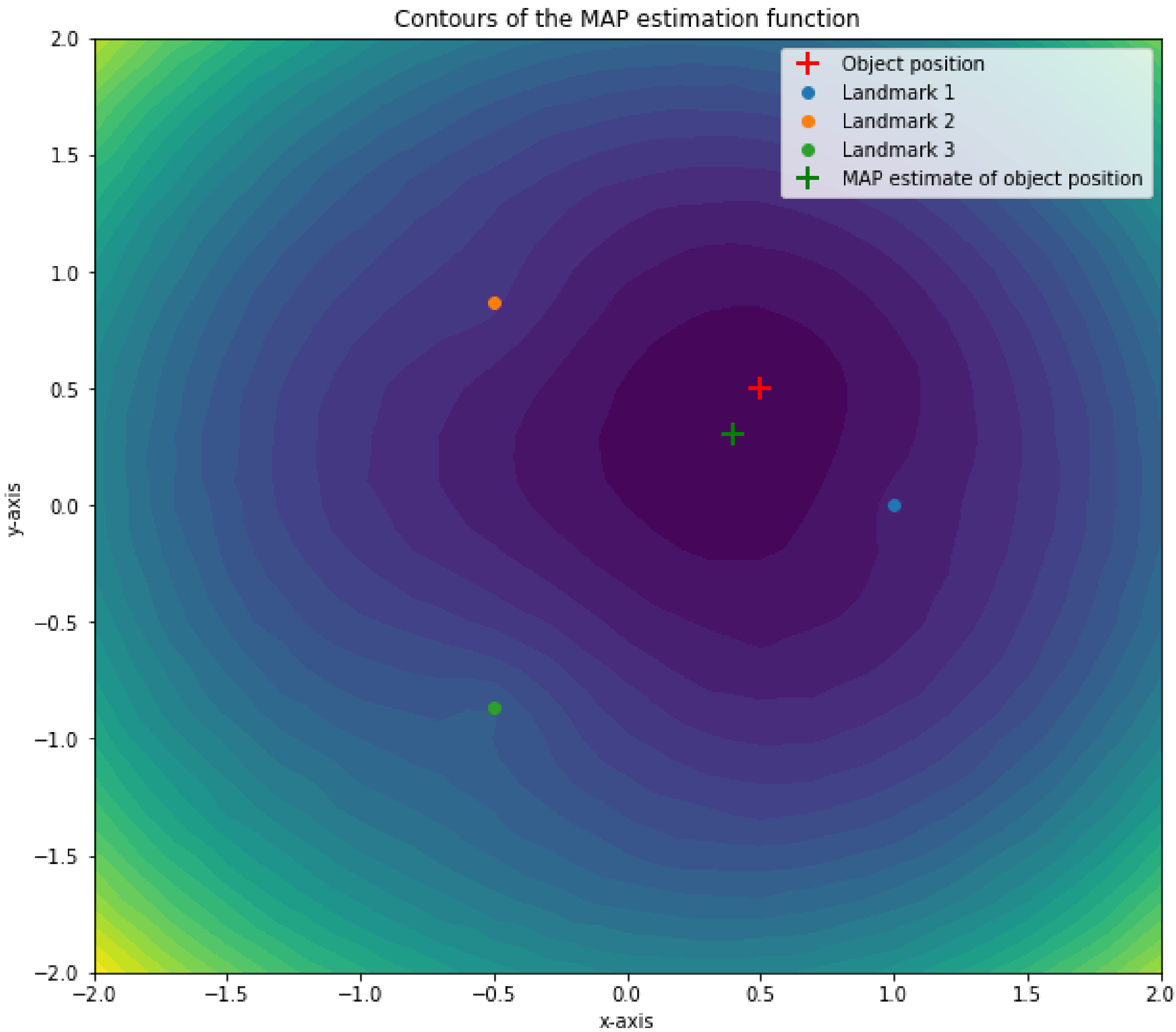
In [35]:
```
x_points = np.arange(-2, 2.05, 0.1)
y_points = np.arange(-2, 2.05, 0.1)
x_mesh, y_mesh = np.meshgrid(x_points, y_points)
```

In [36]:
```
xy_map = np.zeros((len(x_mesh), len(y_mesh)))
landmark = np.zeros((2,4))
check = 1

x_t = 0.5
y_t = 0.5
land_x = [1, -0.5, -0.5]
land_y = [0, 0.866, -0.866]
K = 3

sig_i = 0.1
sig_x = 0.15
sig_y = 0.15
```

In [37]:
```
def cal_dti(x_t_d, y_t_d, x_i_d, y_i_d):
    dti = math.sqrt((x_t_d - x_i_d)**2 + (y_t_d - y_i_d)**2)
    return dti
```

In [38]:
```
for i in range(len(x_mesh)):
    for j in range(len(y_mesh)):
        likelihood = 0
        for q in range(K):
            n_i = np.random.normal(0, sig_i**2)
            r_i = cal_dti(x_t, y_t, land_x[q], land_y[q]) + n_i
            likelihood = likelihood + ((r_i - cal_dti(land_x[q], land_y[q], x_
mesh[i,j], y_mesh[i,j]))**2)/(sig_i**2)
        prior = (x_mesh[i,j]**2)/(sig_x**2) + (y_mesh[i,j]**2)/(sig_y**2)
        xy_map[i, j] = likelihood + prior
        if check == 1:
            mini = xy_map[i,j]
            check = 2
        if xy_map[i, j] < mini:
            mini = xy_map[i,j]
            mini_x = x_mesh[i,j]
            mini_y = y_mesh[i,j]
```

In [39]:
```
fig = plt.figure(figsize=(10, 9))
ax = fig.add_subplot(1, 1, 1)
cs = ax.contourf(x_mesh, y_mesh, xy_map, levels = 30)
ax.plot(x_t, y_t, 'r+', markeredgewidth = 2, markersize = 12, label = 'Object
 position')
for q in range(K):
    ax.plot(land_x[q], land_y[q], 'o', label = 'Landmark {}'.format(q+1))
ax.plot(mini_x, mini_y, 'g+', markeredgewidth = 2, markersize = 12, label = 'M
AP estimate of object position')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Contours of the MAP estimation function')
ax.legend()
plt.show()
```



In [ ]:

K=4

In [35]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
```

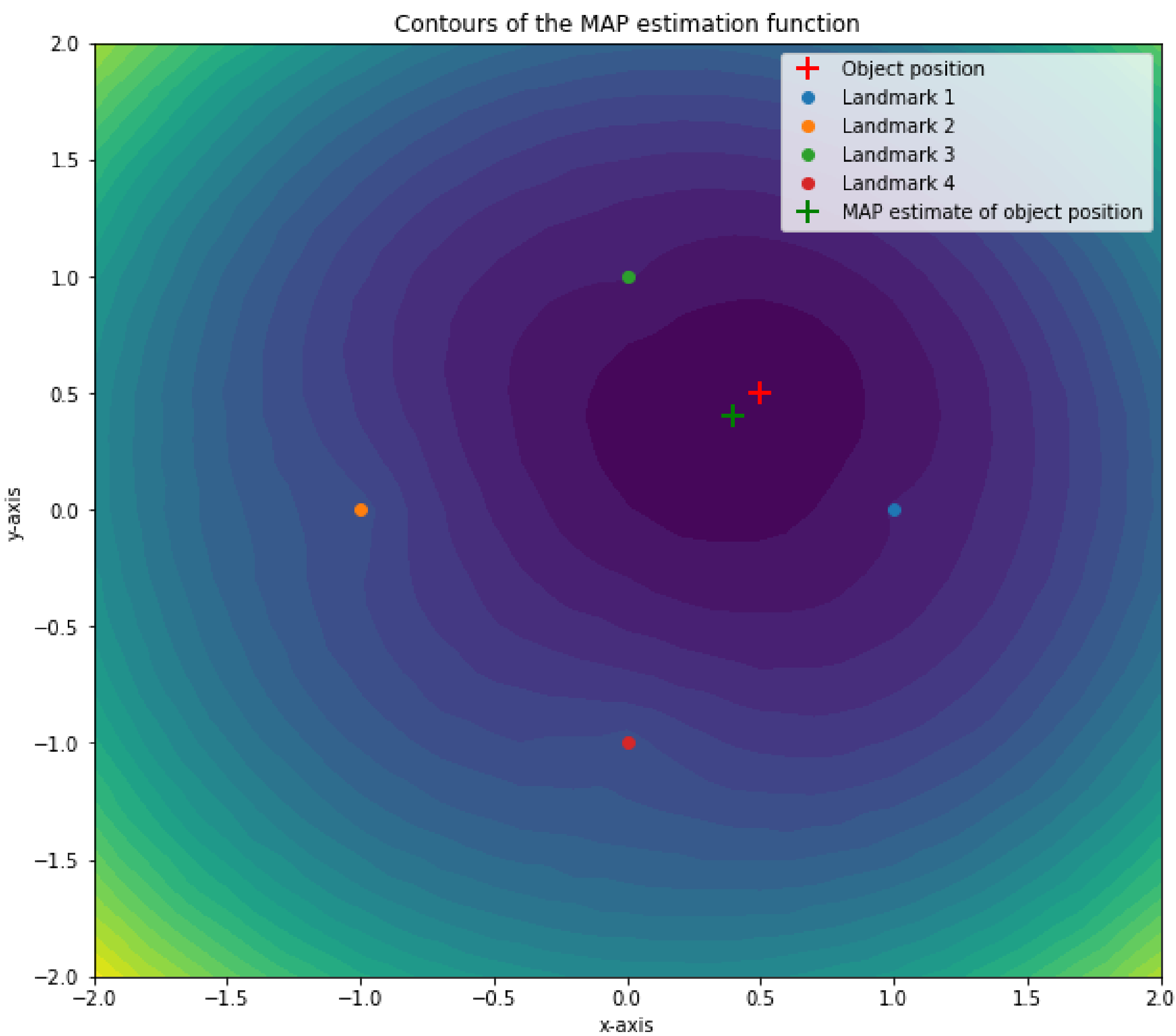
In [36]:
```
x_points = np.arange(-2, 2.05, 0.1)
y_points = np.arange(-2, 2.05, 0.1)
x_mesh, y_mesh = np.meshgrid(x_points, y_points)
```

In [37]:
```
xy_map = np.zeros((len(x_mesh), len(y_mesh)))
landmark = np.zeros((2,4))
check = 1

x_t = 0.5
y_t = 0.5
land_x = [1, -1, 0, 0]
land_y = [0, 0, 1, -1]
K = 4

sig_i = 0.1
sig_x = 0.15
sig_y = 0.15
```

In [38]:
```
def cal_dti(x_t_d, y_t_d, x_i_d, y_i_d):
    dti = math.sqrt((x_t_d - x_i_d)**2 + (y_t_d - y_i_d)**2)
    return dti
```

In [39]:
```
for i in range(len(x_mesh)):
    for j in range(len(y_mesh)):
        likelihood = 0
        for q in range(K):
            n_i = np.random.normal(0, sig_i**2)
            r_i = cal_dti(x_t, y_t, land_x[q], land_y[q]) + n_i
            likelihood = likelihood + ((r_i - cal_dti(land_x[q], land_y[q], x_
mesh[i,j], y_mesh[i,j]))**2)/(sig_i**2)
        prior = (x_mesh[i,j]**2)/(sig_x**2) + (y_mesh[i,j]**2)/(sig_y**2)
        xy_map[i, j] = likelihood + prior
        if check == 1:
            mini = xy_map[i,j]
            check = 2
        if xy_map[i, j] < mini:
            mini = xy_map[i,j]
            mini_x = x_mesh[i,j]
            mini_y = y_mesh[i,j]
```

In [40]:
```
fig = plt.figure(figsize=(10, 9))
ax = fig.add_subplot(1, 1, 1)
cs = ax.contourf(x_mesh, y_mesh, xy_map, levels = 30)
ax.plot(x_t, y_t, 'r+', markeredgewidth = 2, markersize = 12, label = 'Object
 position')
for q in range(K):
    ax.plot(land_x[q], land_y[q], 'o', label = 'Landmark {}'.format(q+1))
ax.plot(mini_x, mini_y, 'g+', markeredgewidth = 2, markersize = 12, label = 'M
AP estimate of object position')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Contours of the MAP estimation function')
ax.legend()
plt.show()
```



In [ ]:

Working of the code:

The code first divides the contour ~~into~~ into numerous points and applies the ~~app~~ objective function for each point. ~~so as~~ The contour is plotted based on the value of the MAP estimate at each point of the contour. The values of σ_i, σ_x, and σ_y are taken such that they balance the prior and likelihood in the function. ~~All~~ Once all the MAP estimates are calculated, the ~~lowest~~ point that gives the minimum value of the MAP estimate objective function is assumed to be the object position based on the MAP estimator function.

Note: In the program, $\sigma_i = 0.1$, $\sigma_x = 0.15$, $\sigma_y = 0.15$

Behaviour of the MAP estimate of position:

We can see that the ~~ed~~ object position estimated by the MAP estimator is near the centre of the contour when $k=1$. This is because, the prior plays a vital role at this stage.

As the value of k increases from 1 to 4, we can see that the estimated position of the object nears the true position of the object. This is due to the increasing number of landmarks which increase the likelihood of the objective function.

Hence, as K increases, the estimated object position nears the true object position.

Question 3

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

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

$$w = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, \quad w \sim \mathcal{N}(0, \gamma^2 I) \qquad I \rightarrow 4\times4$$

$$\therefore\ y = w^T . b(x) + v \qquad \text{where } b(x) = \begin{bmatrix} x^3 \\ x^2 \\ x \\ 1 \end{bmatrix}$$

Also, since $v \sim \mathcal{N}(0, \sigma^2)$, therefore, $y \sim \mathcal{N}(w^T . b(x), \sigma^2)$

MAP estimation objective fn.:

$$W_{MAP} = \underset{w}{\operatorname{argmax}}\ P(W|D) \qquad \text{(Applying Bayes theorem)}$$

$$= \underset{w}{\operatorname{argmax}}\ P(D|W) \cdot P(W)$$

$$= \underset{w}{\operatorname{argmax}} \sum_{i=1}^{N} P(y|W) \cdot P(W)$$

$$= \underset{w}{\operatorname{argmax}} \left(\sum_{i=1}^{N} (2\pi)^{-n/2} (\sigma^2)^{-1/2} . e^{-\frac{1}{2}\frac{(y_i - w^T b(x_i))^2}{\sigma^2}} \right) . \left((2\pi)^{-n/2} . |\gamma^2 I|^{-1/2} \right.$$

$$. e^{-\frac{1}{2}(w)^T (\gamma^2 I)^{-1} w}$$

Applying ln (log) (to simplify eqn.) since it does not affect the maximum value,

$$W_{MAP} = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{N} \underline{-\frac{n}{2}\ln(2\pi)} - \underline{\frac{1}{2}\ln(\sigma^2)} - \frac{1}{2}\frac{(y_i - w^T . b(x_i))^2}{\sigma^2}$$

$$\underline{-\frac{n}{2}\ln(2\pi)} - \underline{\frac{1}{2}\ln(\gamma^2)} - \frac{1}{2} w^T (\gamma^2 I)^{-1} w$$

Neglecting the constants in the equation, (since they do not affect the point of maximum),

$$W_{MAP} = \underset{W}{\operatorname{argmax}} -\frac{1}{2}\left(\sum_{i=1}^{N} \frac{(y_i - W^T . b(x_i))^2}{\sigma^2} + W^T(\gamma^2 I)^{-1} W\right)$$

Maximising a negative function is equivalent to minimising the positive coefficient of the negative sign. Hence,

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

To find the minimum ~~p~~ point, we need to differentiate the equation w.r.t. W and equate it to zero,

$$\nabla_W^T (W_{MAP})\Big|_{W = W_{MAP}} = 0$$

$$\therefore \cancel{2}(\gamma^2 I)^{-1} W_{MAP} - \sum_{i=1}^{N} \cancel{2} \frac{(y_i - W_{MAP}^T b(x_i)) \cdot b^T(x_i)}{\sigma^2} = 0$$

$$\therefore (\gamma^2 I)^{-1} W_{MAP} - \sum_{i=1}^{N} \frac{b(x_i)}{\sigma^2} (y_i - b^T(x_i) \cdot W_{MAP}) = 0$$

$$\therefore \sigma^2 (\gamma^2 I)^{-1} W_{MAP} - \sum_{i=1}^{N} b(x_i) \cdot y_i + b(x_i) \cdot b^T(x_i) W_{MAP} = 0$$

$$\therefore \left(\sigma^2 (\gamma^2 I)^{-1} + \sum_{i=1}^{N} b(x_i) \cdot b^T(x_i)\right) W_{MAP} = \sum_{i=1}^{N} b(x_i) \cdot y_i$$

$$\therefore W_{MAP} = \left[\sigma^2 (\gamma^2 I)^{-1} + \sum_{i=1}^{N} b(x_i) \cdot b^T(x_i)\right]^{-1} \left[\sum_{i=1}^{N} b(x_i) \cdot y_i\right]$$

Note: To calculate W_{true} such that they lie in the interval $[-1, 1]$, ~~that~~ I have chosen a set of roots in this interval for x. These values are: $(-0.5, 0, 0.5)$

Substituting them in $a(x-r_1)(x-r_2)(x-r_3)$

$$= ax^3 - 0.25\,ax$$

we get the value of W_{true} as $W_{true} = \begin{bmatrix} a \\ 0 \\ -0.25a \\ 0 \end{bmatrix}$

Then, we have ~~calculated~~ assumed a value for a such that it best suits our equations. In this program $a = 1$.

Question 3:

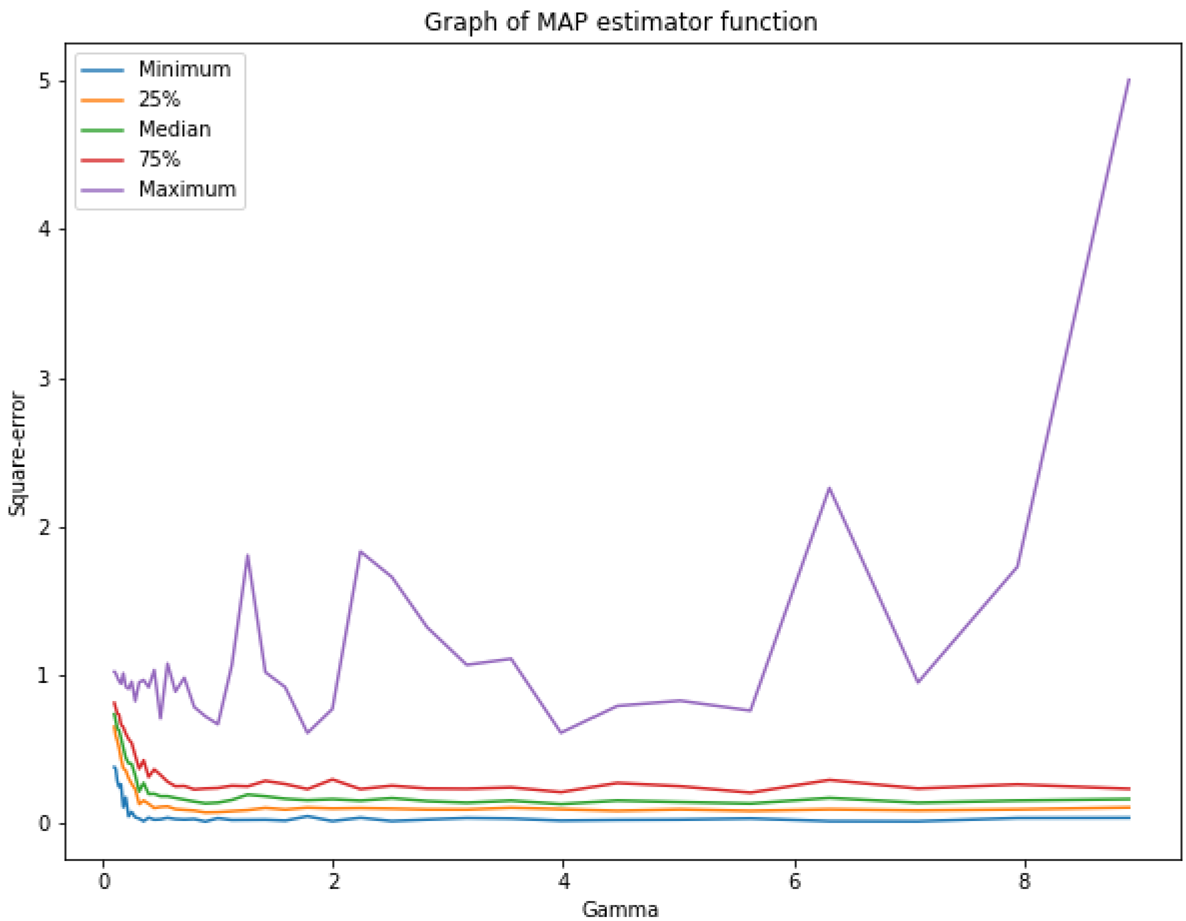
In [197]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
```

In [198]:
```
N = 10
a = 1
v = 0
sig = 0.05
gamma = np.zeros((40))
y_i = np.zeros((10,1))
w_true = [a, 0, -0.25*a, 0]

med_0 = []
med_25 = []
med_50 = []
med_75 = []
med_100 = []
```

In [199]:
```
def b_x_i(x):
    return [x**3, x**2, x, 1]
```

In [200]:
```
for i in range(40):
    gamma[i] = math.pow(10, -1+0.05*i)
    L2 = []

    for j in range(100):
        x_i = np.random.uniform(-1, 1, N)

        for m in range(N):
            v = np.random.normal(0, sig)
            y_i[m] = np.matmul(np.array(w_true).reshape((1,4)), np.array(b_x_i
(x_i[m]))) + v

        h1_1 = (sig**2)*np.linalg.inv((gamma[i]**2)*np.identity(4))
        h1_2 = 0
        h2 = 0

        for n in range(10):
            b_temp = b_x_i(x_i[n])
            h1_2 = h1_2 + np.matmul(np.array(b_temp).reshape((4,1)), np.array(
b_temp).reshape((1,4)))
            h2 = h2 + np.matmul(np.reshape(b_temp, (4,1)), y_i[n])

        w_map = np.matmul(np.linalg.inv(h1_1 + h1_2), h2)

        L2.append(np.linalg.norm(w_true - w_map))
    L2.sort()
    med_0.append(L2[0])
    med_25.append(L2[24])
    med_50.append(L2[49])
    med_75.append(L2[74])
    med_100.append(L2[99])
```

In [204]:
```
fig = plt.figure(figsize=(10, 7.5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(gamma, med_0, label='Minimum')
ax.plot(gamma, med_25, label='25%')
ax.plot(gamma, med_50, label='Median')
ax.plot(gamma, med_75, label='75%')
ax.plot(gamma, med_100, label='Maximum')
plt.xlabel('Gamma')
plt.ylabel('Square-error')
plt.title('Graph of MAP estimator function')
ax.legend()
plt.show()
```



The curve shows that the squared-error values are very high for very low values of gamma. As the value of gamma increases (goes towards infinity), the error stabilises.

Gamma gives confidence to the prior. If gamma is near 0, it will be very confident that weight should be 0. As gamma increases, it will become less confident as the data will begin to influence the MAP estimate and results in a performance increase.

As gamma tends to infinity, the priors will have minimal or no effect and hence, the estimate will go towards Maximum Likelihood i.e. it will begin to behave as if there is no prior.

In [ ]: