**4b**

step\_size = 0.1 is most efficient

step\_size = 0.1 (in 1000 steps):

The real object location is

[[ 44.38632327 33.36743274]]

The estimated object location with zero initialization is

[[ 43.07188433 32.71217817]]

The estimated object location with random initialization is

[[ 43.07188433 32.71217817]]

step\_size = 0.01 (in 1000 steps):

The real object location is

[[ 44.38632327 33.36743274]]

The estimated object location with zero initialization is

[[ 43.07188433 32.71217817]]

The estimated object location with random initialization is

[[ 43.07188433 32.71217817]]

step\_size = 0.001(in 1000 steps):

The real object location is

[[ 44.38632327 33.36743274]]

The estimated object location with zero initialization is

[[ 42.57128857 32.04116473]]

The estimated object location with random initialization is

[[ 44.42031211 34.37381408]]

-----------------code-------------------------------------------------------------

from common import \*

from math import sqrt

import numpy as np

from numpy.linalg import norm

########################################################################

######### Part b ###################################

########################################################################

########################################################################

######### Gradient Computing and MLE ###################################

########################################################################

def compute\_gradient\_of\_likelihood(single\_obj\_loc, sensor\_loc,

single\_distance):

"""

Compute the gradient of the loglikelihood function for part a.

Input:

single\_obj\_loc: 1 \* d numpy array.

Location of the single object.

sensor\_loc: k \* d numpy array.

Location of sensor.

single\_distance: k dimensional numpy array. (k: number of sensors, d: dimensionality)

Observed distance of the object.

Output:

grad: d-dimensional numpy array.

"""

#Your code: implement the gradient of loglikelihood

#grad = np.zeros\_like(single\_obj\_loc)

#print(single\_obj\_loc)

#print(np.diag(single\_obj\_loc[0, :]))

#print(np.ones\_like(sensor\_loc))

sensor\_loc\_diff = sensor\_loc - np.ones\_like(sensor\_loc) @ np.diag(single\_obj\_loc[0, :])

#print(sensor\_loc)

#print(sensor\_loc\_diff)

sensor\_loc\_diff\_norm = norm(sensor\_loc\_diff, axis = 1)

#print('\n')

#print(sensor\_loc\_diff\_norm.shape)

#print(single\_distance.shape)

second\_term = 1 - single\_distance/sensor\_loc\_diff\_norm

#print(second\_term)

grad = -2\*sensor\_loc\_diff.T @ second\_term

#print(grad)

return grad

def find\_mle\_by\_grad\_descent\_part\_b(initial\_obj\_loc,

sensor\_loc, single\_distance, lr=0.001, num\_iters = 10000):

"""

Compute the gradient of the loglikelihood function for part a.

Input:

initial\_obj\_loc: 1 \* d numpy array.

Initialized Location of the single object.

sensor\_loc: k \* d numpy array. Location of sensor.

single\_distance: k dimensional numpy array.

Observed distance of the object.

Output:

obj\_loc: 1 \* d numpy array. The mle for the location of the object.

"""

# Your code: do gradient descent

obj\_loc = initial\_obj\_loc

for i in range(num\_iters):

obj\_loc = obj\_loc - lr \* compute\_gradient\_of\_likelihood(obj\_loc, sensor\_loc,

single\_distance)

return obj\_loc

if \_\_name\_\_ == "\_\_main\_\_":

########################################################################

######### MAIN ########################################################

########################################################################

# Your code: set some appropriate learning rate here

lr = 0.001

np.random.seed(0)

sensor\_loc = generate\_sensors()

obj\_loc, distance = generate\_data(sensor\_loc)

single\_distance = distance[0]

print('The real object location is')

print(obj\_loc)

# Initialized as [0,0]

initial\_obj\_loc = np.array([[0.,0.]])

estimated\_obj\_loc = find\_mle\_by\_grad\_descent\_part\_b(initial\_obj\_loc,

sensor\_loc, single\_distance, lr=lr, num\_iters = 1000)

print('The estimated object location with zero initialization is')

print(estimated\_obj\_loc)

# Random initialization.

initial\_obj\_loc = np.random.randn(1,2)\*100+100

#initial\_obj\_loc = np.array([[44.38, 33.36]])

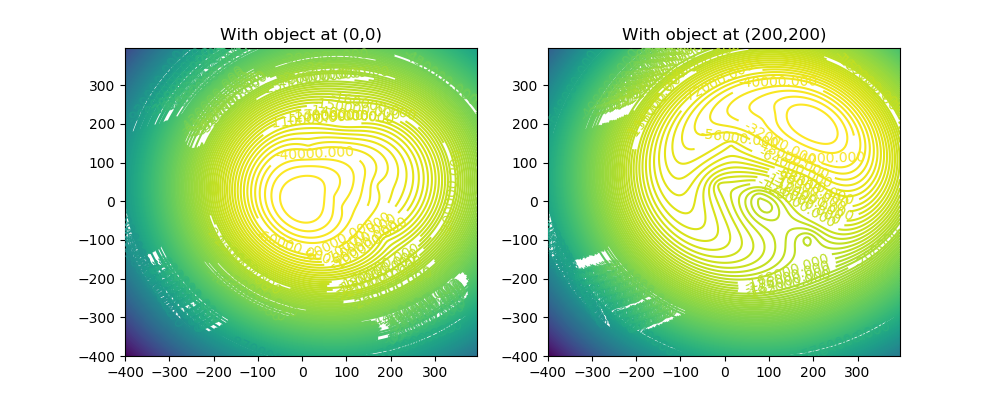
estimated\_obj\_loc = find\_mle\_by\_grad\_descent\_part\_b(initial\_obj\_loc,

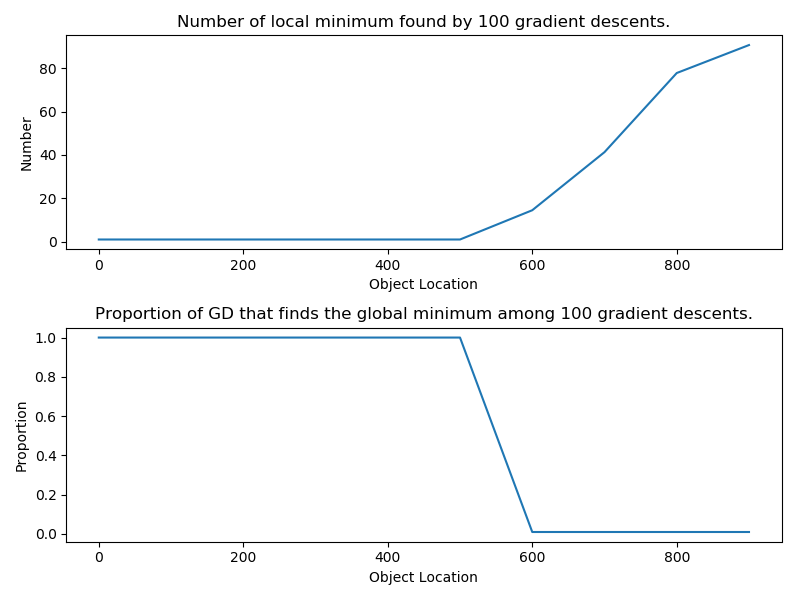
sensor\_loc, single\_distance, lr=lr, num\_iters = 1000)

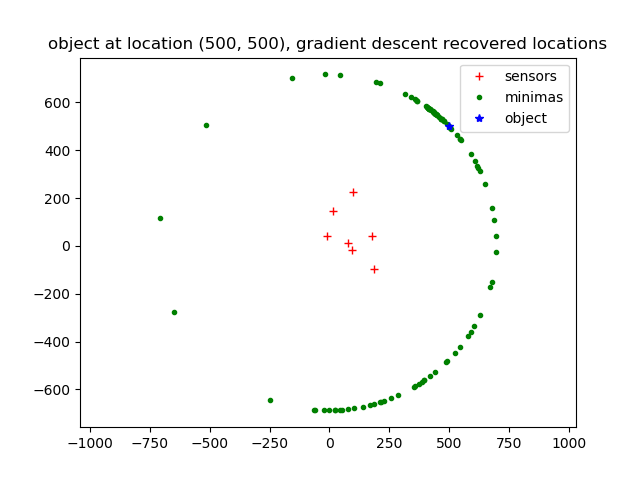
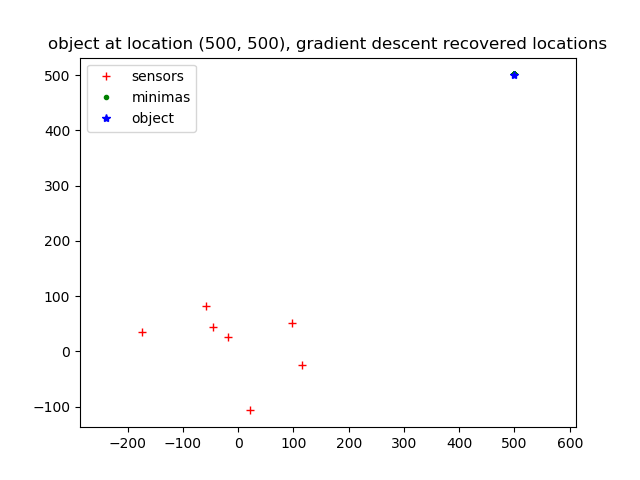
print('The estimated object location with random initialization is')

print(estimated\_obj\_loc)

**4c**

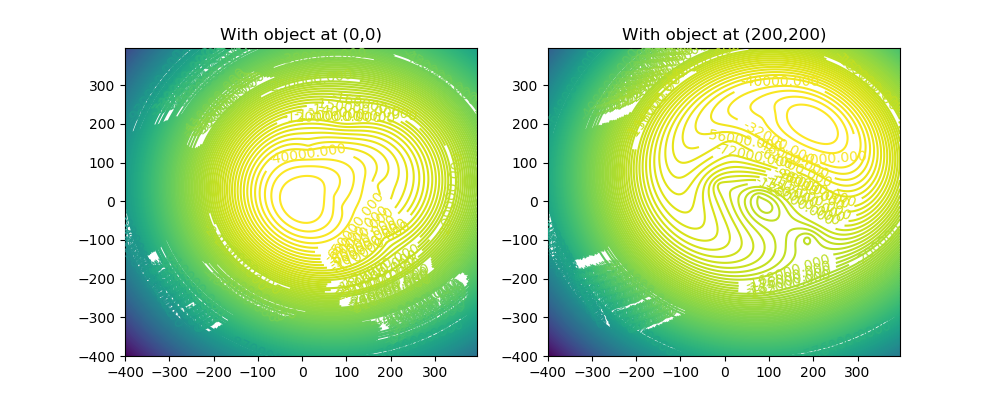
****

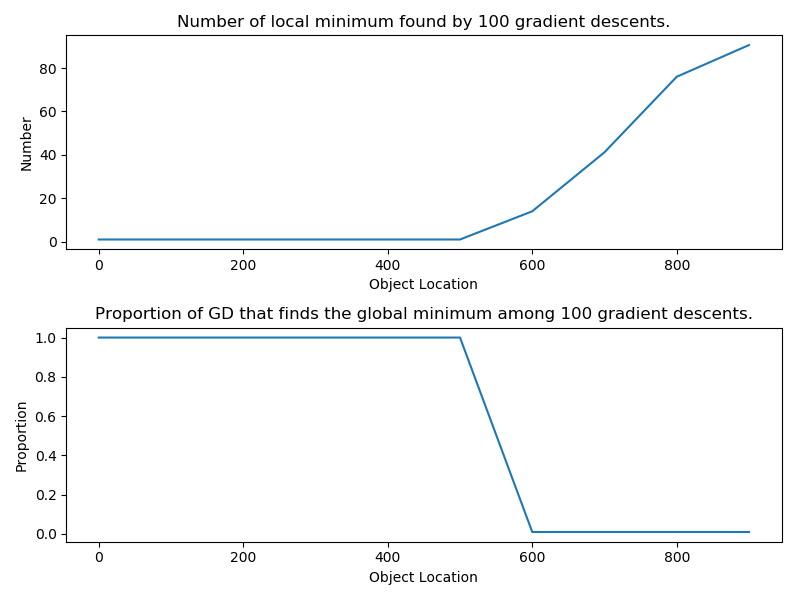
****

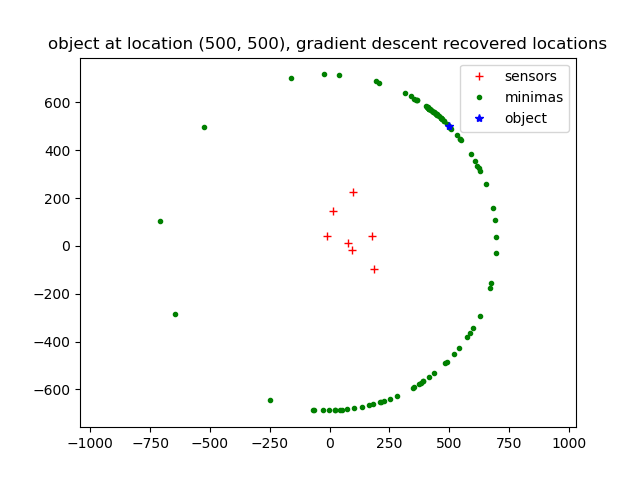
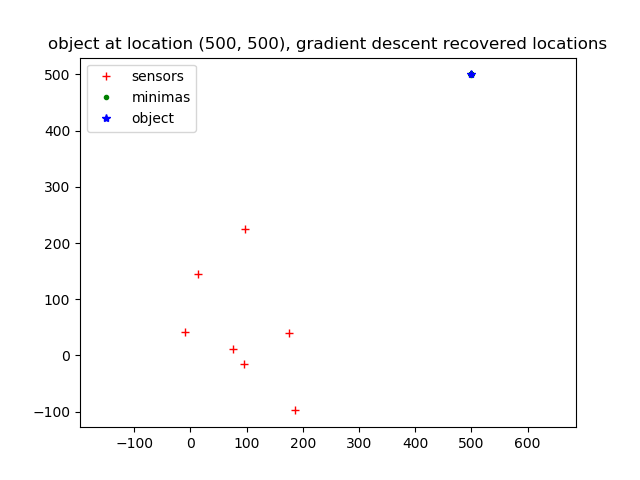
****

step\_size = 0.1 step\_size = 0.01

**4d**

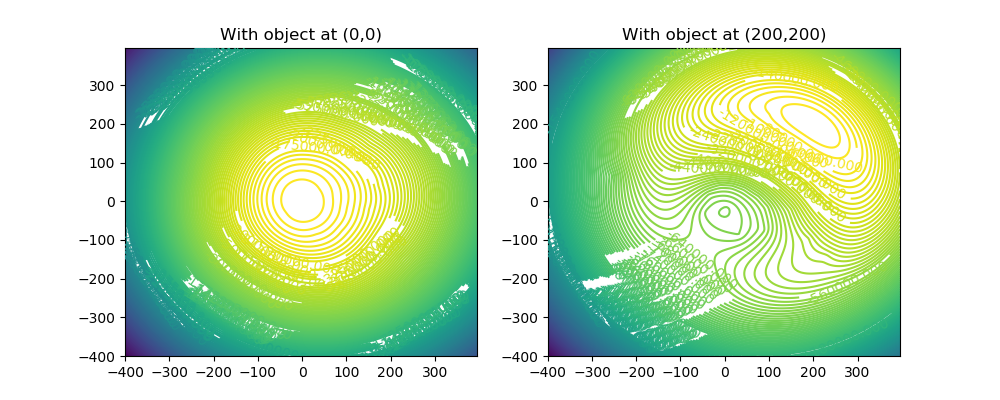
****

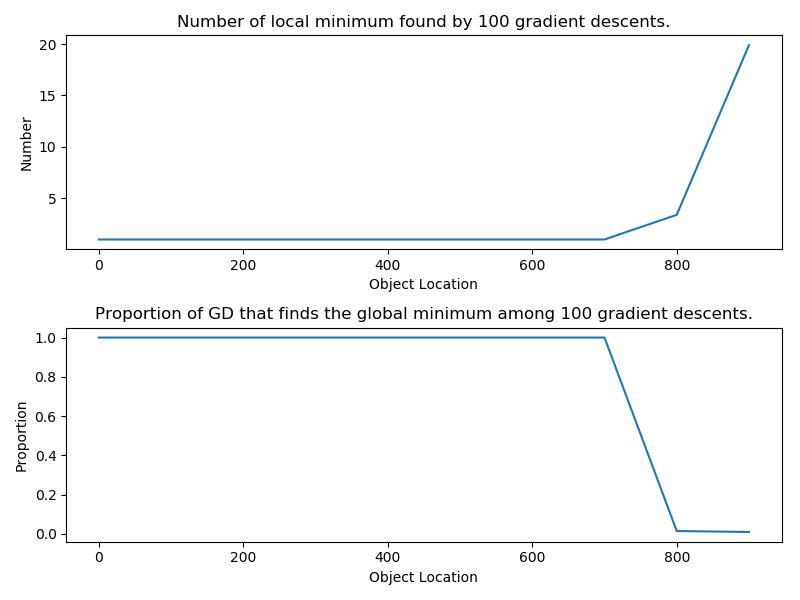
****

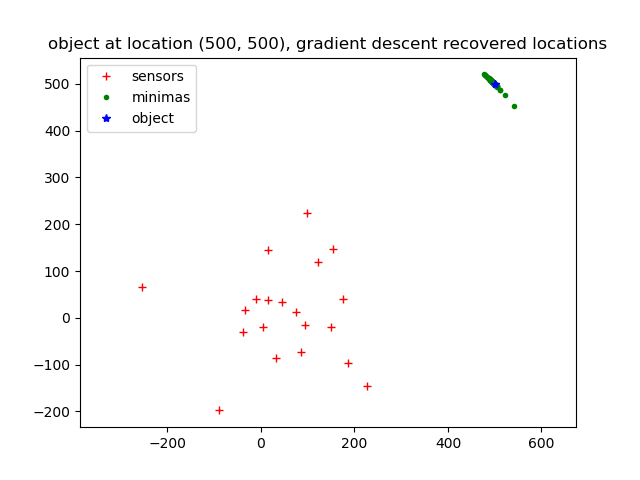
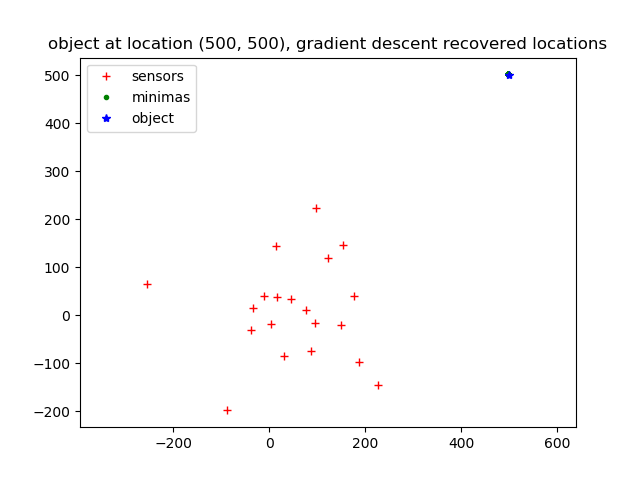
****

step\_size = 0.1 step\_size = 0.01

**4e**

****

****

****

step\_size = 0.05 step\_size = 0.01

**4f**

The MSE for Case 1 is 5.927908379315981

The MSE for Case 2 is 226.22539833527367

The MSE for Case 2 (if we knew mu is [300,300]) is 1.306100883985725

------------------------code------------------------

from common import \*

from part\_b\_starter import find\_mle\_by\_grad\_descent\_part\_b

from part\_b\_starter import compute\_gradient\_of\_likelihood

from part\_c\_starter import log\_likelihood

from numpy.linalg import norm

########################################################################

######### Gradient Computing and MLE ##################################

########################################################################

def compute\_grad\_likelihood(sensor\_loc, obj\_loc, distance):

"""

Compute the gradient of the loglikelihood function for part f.

Input:

sensor\_loc: k \* d numpy array.

Location of sensors.

obj\_loc: n \* d numpy array.

Location of the objects.

distance: n \* k dimensional numpy array.

Observed distance of the object.

Output:

grad: k \* d numpy array.

"""

grad = np.zeros(sensor\_loc.shape)

# Your code: finish the grad loglike

for i in range(sensor\_loc.shape[0]):

sensor\_loc\_diff = np.ones\_like(obj\_loc) @ np.diag(sensor\_loc[i, :]) - obj\_loc

sensor\_loc\_diff\_norm = norm(sensor\_loc\_diff, axis = 1)

second\_term = 1 - distance[:, i]/sensor\_loc\_diff\_norm

grad[i, :] = (2\*sensor\_loc\_diff.T @ second\_term).T

return grad

def find\_mle\_by\_grad\_descent(initial\_sensor\_loc,

obj\_loc, distance, lr=0.001, num\_iters = 1000):

"""

Compute the gradient of the loglikelihood function for part f.

Input:

initial\_sensor\_loc: k \* d numpy array.

Initialized Location of the sensors.

obj\_loc: n \* d numpy array. Location of the n objects.

distance: n \* k dimensional numpy array.

Observed distance of the n object.

Output:

sensor\_loc: k \* d numpy array. The mle for the location of the object.

"""

sensor\_loc = initial\_sensor\_loc

# Your code: finish the gradient descent

for i in range(num\_iters):

sensor\_loc = sensor\_loc - lr \* compute\_grad\_likelihood(sensor\_loc, obj\_loc, distance)

return sensor\_loc

########################################################################

######### Gradient Computing and MLE ##################################

########################################################################

np.random.seed(0)

sensor\_loc = generate\_sensors()

obj\_loc, distance = generate\_data(sensor\_loc, n = 100)

print('The real sensor locations are')

print(sensor\_loc)

# Initialized as zeros.

initial\_sensor\_loc = np.random.randn(7,2)\*100

estimated\_sensor\_loc = find\_mle\_by\_grad\_descent(initial\_sensor\_loc,

obj\_loc, distance, lr=0.01, num\_iters = 10000)

print('The predicted sensor locations are')

print(estimated\_sensor\_loc)

print('\n')

########################################################################

######### Estimate distance given estimated sensor locations. #########

########################################################################

def compute\_distance\_with\_sensor\_and\_obj\_loc(sensor\_loc, obj\_loc):

"""

stimate distance given estimated sensor locations.

Input:

sensor\_loc: k \* d numpy array.

Location of the sensors.

obj\_loc: n \* d numpy array. Location of the n objects.

Output:

distance: n \* k dimensional numpy array.

"""

estimated\_distance = scipy.spatial.distance.cdist(obj\_loc,

sensor\_loc,

metric='euclidean')

return estimated\_distance

########################################################################

######### MAIN #######################################################

########################################################################

np.random.seed(100)

########################################################################

######### Case 1. #####################################################

########################################################################

mse =0

for i in range(100):

obj\_loc, distance = generate\_data(sensor\_loc, k = 7, d = 2, n = 1, original\_dist = True)

obj\_loc, distance = generate\_data\_given\_location(estimated\_sensor\_loc, obj\_loc, k = 7, d = 2)

l = float('-inf')

initial\_obj\_loc = np.array([[0.,0.]])

mse += 1.0/100 \* norm(find\_mle\_by\_grad\_descent\_part\_b(initial\_obj\_loc,

estimated\_sensor\_loc, distance[0], lr=0.01, num\_iters = 1000) - obj\_loc)

# Your code: compute the mse for this case

print('The MSE for Case 1 is {}'.format(mse))

########################################################################

######### Case 2. #####################################################

########################################################################

mse =0

for i in range(100):

obj\_loc, distance = generate\_data(sensor\_loc, k = 7, d = 2, n = 1, original\_dist = False)

obj\_loc, distance = generate\_data\_given\_location(estimated\_sensor\_loc, obj\_loc, k = 7, d = 2)

l = float('-inf')

# Your code: compute the mse for this case

initial\_obj\_loc = np.array([[0.,0.]])

mse += 1.0/100 \* norm(find\_mle\_by\_grad\_descent\_part\_b(initial\_obj\_loc,

estimated\_sensor\_loc, distance[0], lr=0.01, num\_iters = 1000) - obj\_loc)

print('The MSE for Case 2 is {}'.format(mse))

########################################################################

######### Case 3. #####################################################

########################################################################

mse =0

for i in range(100):

obj\_loc, distance = generate\_data(sensor\_loc, k = 7, d = 2, n = 1, original\_dist = False)

obj\_loc, distance = generate\_data\_given\_location(estimated\_sensor\_loc, obj\_loc, k = 7, d = 2)

l = float('-inf')

# Your code: compute the mse for this case

initial\_obj\_loc = np.array([[300.,300.]])

mse += 1.0/100 \* norm(find\_mle\_by\_grad\_descent\_part\_b(initial\_obj\_loc,

estimated\_sensor\_loc, distance[0], lr=0.01, num\_iters = 1000) - obj\_loc)

print('The MSE for Case 2 (if we knew mu is [300,300]) is {}'.format(mse))