HW7 Yao Cai

2. (a) We need to prove that $\lambda f(x_1) + (1-\lambda) f(x_2) \geq f(\lambda x_1 + (1-\lambda) x_2)$ (where $\lambda \in [0,1]$)

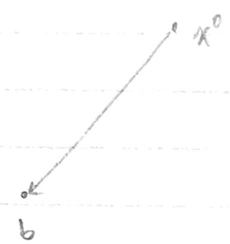
$$\lambda f(x_1) + (1-\lambda) f(x_2) = \lambda \|x_1 - b\|_2 + (1-\lambda)\|x_2 - b\|_2$$
$$= \|\lambda x_1 - \lambda b\|_2 + \|(1-\lambda)x_2 - (1-\lambda)b\|_2$$
$$\geq \|\lambda x_1 - \lambda b + (1-\lambda)x_2 - (1-\lambda)b\|_2 \quad \text{(by triangle inequality)}$$
$$= \|\lambda x_1 + (1-\lambda)x_2 - b\|_2$$
$$= f(\lambda x_1 + (1-\lambda)x_2)$$

(b). No. Because the gradients are constant, step size is also constant and large, so the result is jumping around the minimum but not approaching it. Proof below:

$$f(x) = \sqrt{(x_1-b_1)^2 + (x_2-b_2)^2} \qquad \vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}. \quad \vec{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$\frac{\partial f(x)}{\partial x_1} = \frac{1}{2} \frac{2(x_1-b_1)}{\sqrt{(x_1-b_1)^2+(x_2-b_2)^2}} = \frac{x_1-b_1}{\sqrt{(x_1-b_1)^2+(x_2-b_2)^2}}$$

$$\frac{\partial f(x)}{\partial x_2} = \frac{x_2-b_2}{\sqrt{(x_1-b_1)^2+(x_2-b_2)^2}}$$



so the direction of GD is always along the same line as illustrated on the right

- If we plug in x^0 and b, then $\frac{\partial f(x)}{\partial x_1} = 0.6$, $\frac{\partial f(x)}{\partial x_2} = 0.8$ or $\frac{\partial f(x)}{\partial x_1} = -0.6$, $\frac{\partial f(x)}{\partial x_2} = -0.8$

Constant and large gradient and step size, there is no way we can landed on true b

- Only plug in x^0, then $\frac{\partial f(x)}{\partial x_1} = \frac{\text{sign}(b_1)}{\sqrt{b_1^2+b_2^2}}$, $\frac{\partial f(x)}{\partial x_2} = \frac{\text{sign}(b_2)}{\sqrt{b_1^2+b_2^2}}$

or negate both

We can see that $\left(\frac{\partial f(x)}{\partial x_1}\right)^2 + \left(\frac{\partial f(x)}{\partial x_2}\right)^2 = 1$, so we cannot approach b using constant step size

No.

(c) from (b) we know that x_i is always on the same line and x_i move forward by $(\frac{5}{6})^i \times 1$ along the line

$$\|x_i\|_2 = 1 + (\tfrac{5}{6})^1 + (\tfrac{5}{6})^2 + (\tfrac{5}{6})^3 + \cdots + (\tfrac{5}{6})^{i-1}$$

when $i \to \infty$, this is $\frac{1}{1-\frac{5}{6}} = 6$

$\|b\|_2 = 7.5$

so we will never get to b.

if $\|b\|_2 < 6$, then we can get to it in finite steps

(d) Yes

$$\|x_i\|_2 = 1 + \tfrac{1}{2} + \tfrac{1}{3} + \cdots + \tfrac{1}{i} \sim \log i$$

$\|b\|_2 = 7.5 = \log i$

$i \sim e^{7.5} = 1808$

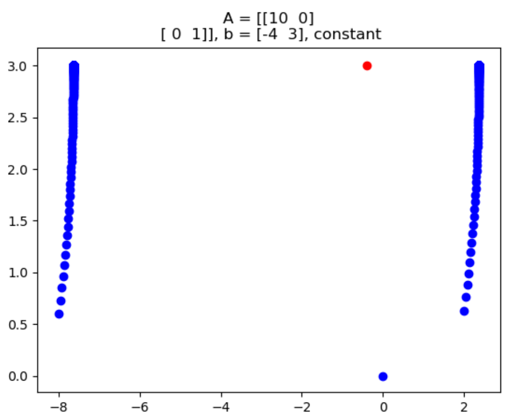
In about 1800 or so steps, we can get within 0.01

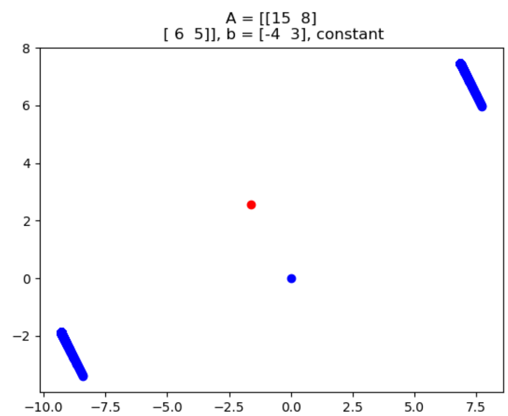
for general b, $i \sim e^{\|b\|_2}$

now if $\|b\|_2$ is large, say ~ 100, then $i \sim 10^{43}$ which is not practical

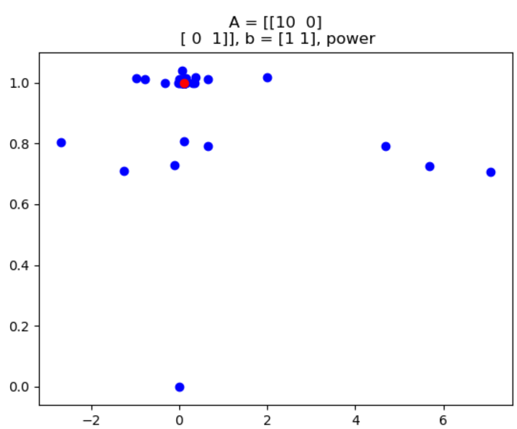
2e

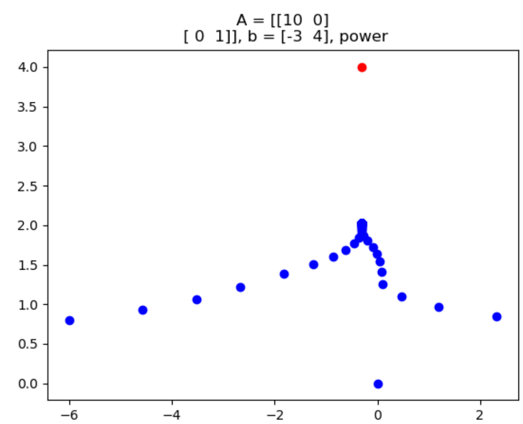
constant step size:

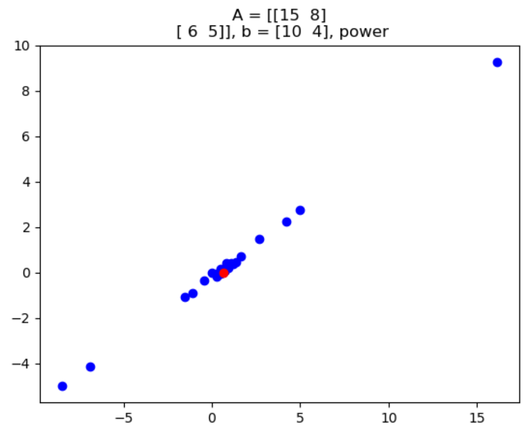


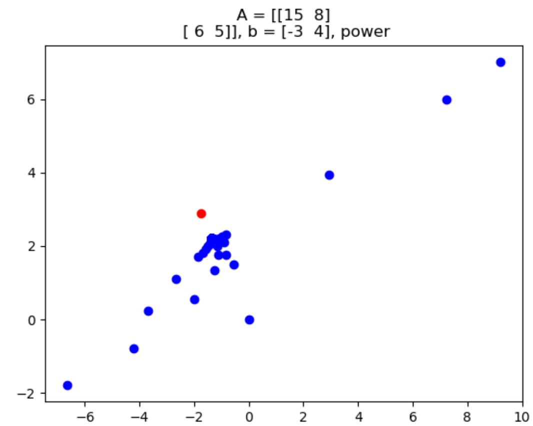


power step size:

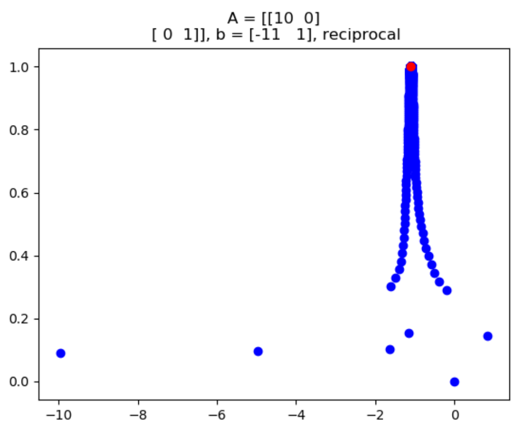


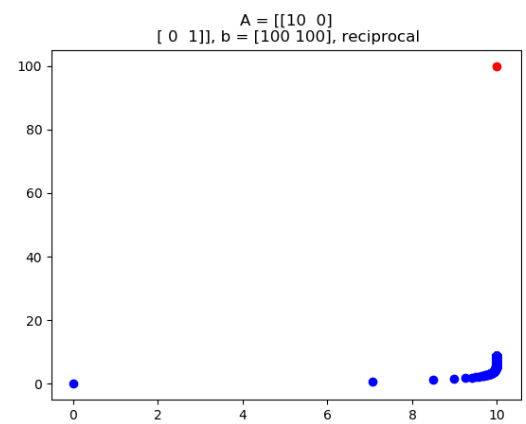


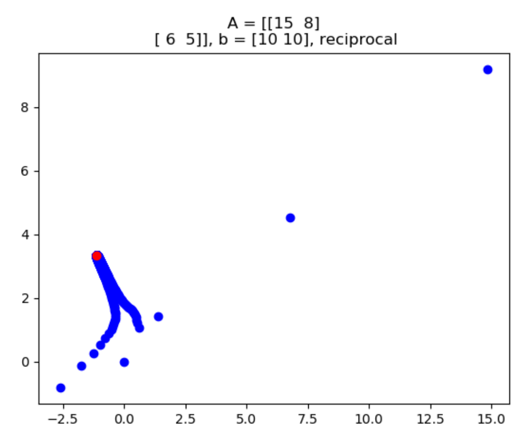


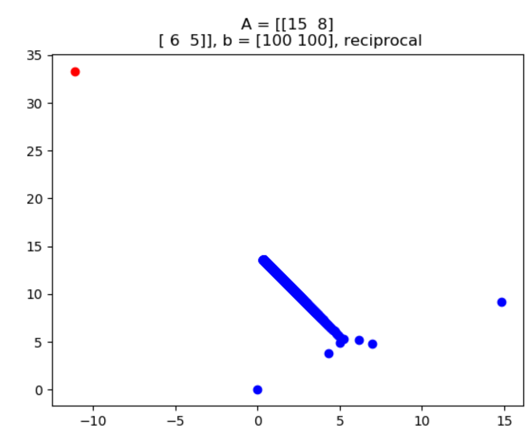


reciprocal step size:









(e) None of these step sizes work for all choices of A and b.
Since Ax is linear, Using GD on $f(x) = \|Ax - b\|_2$ is similar to using GD on $f(y) = \|y - b\|_2$.
then the same reasoning as in (b), (c), (d) applies.

constant step size, will jump back and forth

power step size, converge to a certain constant, and will not converge to any constant

reciprocal step size, always converges but can be very slow and not practical.

3. (a) $f(x) = \frac{1}{2}(Ax-b)^T(Ax-b)$

$$= \frac{1}{2}(x^TA^TAx + b^Tb - 2x^TA^Tb)$$

$$= \frac{1}{2}x^TA^TAx - x^TA^Tb + \frac{1}{2}b^Tb$$

$$\nabla f(x) = \frac{1}{2}(A^TA + A^TA)x - A^Tb$$
$$= A^TAx - A^Tb$$

$$x_1 = x_0 - r\nabla f(x_0)$$
$$= x_0 - r(A^TAx_0 - A^Tb)$$
$$= (I - rA^TA)x_0 + rA^Tb$$

$$x_2 = (I - rA^TA)x_1 + rA^Tb$$
$$= (I - rA^TA)\left[(I - rA^TA)x_0 + rA^Tb\right] + rA^Tb$$
$$= (I - rA^TA)^2x_0 + (I - rA^TA)rA^Tb + rA^Tb$$

$$x_3 = (I - rA^TA)x_2 + rA^Tb$$
$$= (I - rA^TA)\left[(I - rA^TA)^2x_0 + (I - rA^TA)rA^Tb + rA^Tb\right] + rA^Tb$$
$$= (I - rA^TA)^3x_0 + (I - rA^TA)^2rA^Tb + (I - rA^TA)rA^Tb + rA^Tb$$

$$x_n = (I - rA^TA)^nx_0 + \sum_{i=0}^{n-1}(I - rA^TA)^i rA^Tb$$

when $b = 0$

$$x_n = (I - rA^TA)^n x_0$$

(b) $B = I - rA^TA$

$$|\lambda_{max}(B)| = |\lambda_{max}(I - rA^TA)|$$

$$= |1 - r\lambda_{min}(A^TA)| \leq 1$$

$$|\lambda_{min}(B)| = |1 - r\lambda_{max}(A^TA)| \leq 1$$

when $|1 - r\lambda_{min}(A^TA)|$ and $|1 - r\lambda_{max}(A^TA)|$ both ≤ 1,

we have stable dynamical system.

(d) $\psi(x_k) = x_k - r\nabla f(x_k) = x_{k+1}$

$$\psi(x^*) = x^* - r\nabla f(x^*) = x^*$$

$$\|x_{k+1} - x^*\|_2 = \|\psi(x_k) - \psi(x^*)\|_2 \leq \beta\|x_k - x^*\|_2$$

$$\leq \beta^2\|x_{k-1} - x^*\|_2$$
$$\leq \beta^3\|x_{k-2} - x^*\|_2$$

$$\leq \beta^{k+1}\|x_0 - x^*\|_2$$

(e) $f(x) - f(x^*) = f(x) = \frac{1}{2}\|Ax - b\|_2^2$

$$= \frac{1}{2}\|Ax - Ax^*\|_2^2$$

$$= \frac{1}{2}\|A(x - x^*)\|_2^2$$

(g) $\beta = \max\{|1-r\lambda_{max}(A^TA)|, |1-r\lambda_{min}(A^TA)|\}$

① $r \le \frac{1}{\lambda_{max}}$

$$\beta = \max\{1-r\lambda_{max}, 1-r\lambda_{min}\} = 1-r\lambda_{min} \ge 1-\frac{\lambda_{min}}{\lambda_{max}} = \frac{\lambda_{max}-\lambda_{min}}{\lambda_{max}}$$

② $\frac{1}{\lambda_{max}} \le r \le \frac{2}{\lambda_{max}+\lambda_{min}}$

$$\beta = \max\{r\lambda_{max}-1, 1-r\lambda_{min}\} = r\lambda_{max}-1 \le \frac{2\lambda_{max}}{\lambda_{max}+\lambda_{min}}-1 = \frac{\lambda_{max}-\lambda_{min}}{\lambda_{max}+\lambda_{min}}.$$

③ $\frac{2}{\lambda_{max}+\lambda_{min}} \le r \le \frac{1}{\lambda_{min}}$

$$\beta = \max\{r\lambda_{max}-1, 1-r\lambda_{min}\} = 1-r\lambda_{min} \le 1-\frac{2\lambda_{min}}{\lambda_{max}+\lambda_{min}} = \frac{\lambda_{max}-\lambda_{min}}{\lambda_{max}+\lambda_{min}}.$$

④ $r \ge \frac{1}{\lambda_{min}}$

$$\beta = \max\{r\lambda_{max}-1, r\lambda_{min}-1\} = r\lambda_{max}-1 \ge \frac{\lambda_{max}}{\lambda_{min}}-1 = \frac{\lambda_{max}-\lambda_{min}}{\lambda_{min}}$$

since $\frac{\lambda_{max}-\lambda_{min}}{max}, \frac{\lambda_{max}-\lambda_{min}}{\lambda_{min}} > \frac{\lambda_{max}-\lambda_{min}}{\lambda_{max}+\lambda_{min}}$

We should choose $\boxed{\frac{1}{\lambda_{max}} \le r \le \frac{1}{\lambda_{min}}}$, and

$$\beta \le \frac{\lambda_{max}-\lambda_{min}}{\lambda_{max}+\lambda_{min}} = \frac{k-1}{k+1} \quad \left(k = \frac{\lambda_{max}}{\lambda_{min}}\right)$$

then $f(x_k) - f(x^*) \le \frac{\alpha}{2}\beta^{2k}\|x_0-x^*\|_2^2 \le \frac{\alpha}{2}\left(\frac{k-1}{k+1}\right)^{2k}\|x_0-x^*\|_2^2$

Now we can see that the condition number of A^TA (or A) really matters a lot in optimization.

And we can tune step-size to make the convergence faster

4. (a) $$\frac{\partial L}{\partial x_1} = +\sum_{i=1}^{7} 2\left(\sqrt{(a_i-x_1)^2+(b_i-y_1)^2} - d_i\right)\frac{1}{2}\frac{2(a_i-x_1)}{\sqrt{(a_i-x_1)^2+(b_i-y_1)^2}}$$

$$= \sum_{i=1}^{7} 2(a_i-x_1)\left(1-\frac{d_i}{\sqrt{(a_i-x_1)^2+(b_i-y_1)^2}}\right)$$

$$\frac{\partial L}{\partial y_1} = \sum_{i=1}^{7} 2(b_i-y_1)\left(1-\frac{d_i}{\sqrt{(a_i-x_1)^2+(b_i-y_1)^2}}\right)$$

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
```

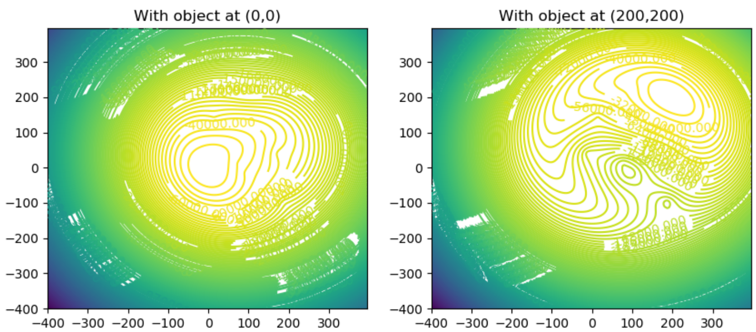
```
from numpy.linalg import norm
####################################################################
#####
######### Part b ###################################
####################################################################
#####

####################################################################
#####
#########  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)
```

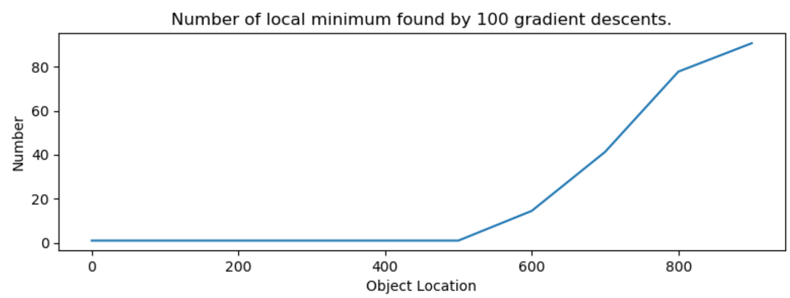
```
        #print(single_obj_loc)
        #print(np.diag(single_obj_loc[0, :]))
        #print(np.ones_like(sensor_loc))
        sensor_loc_diff = sensor_loc - np.ones_like(sensor_loc) @
np.diag(single_obj_loc[0, :])
        #print(sensor_loc)
        #print(sensor_loc_diff)
        sensor_loc_diff_norm = 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.

        """
```

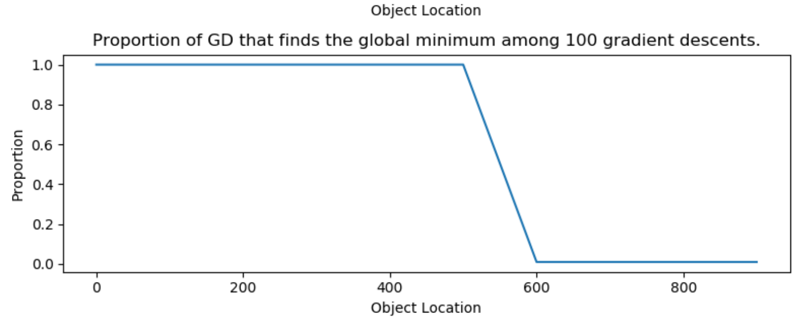
```
    # Your code: do gradient descent
    obj_loc = initial_obj_loc
    for i in range(num_iters):
        obj_loc = obj_loc - lr * compute_gradient_of_likelihood(obj_loc,
sensor_loc,
                                                        single_distance)
    return obj_loc

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)
```

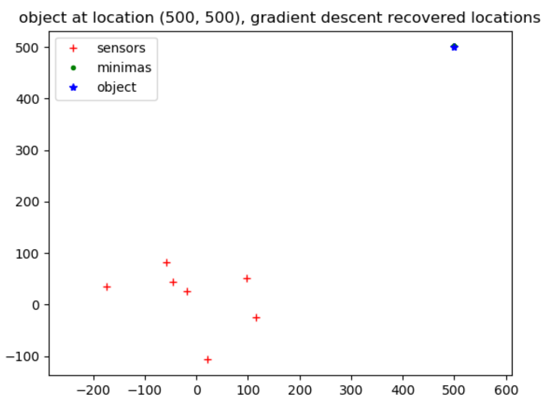
```
print('The estimated object location with random initialization is')
print(estimated_obj_loc)
```

4c

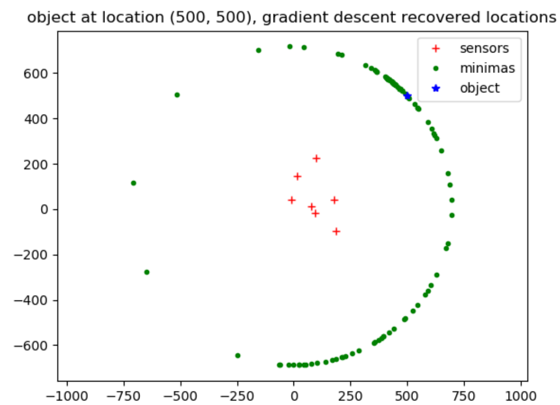






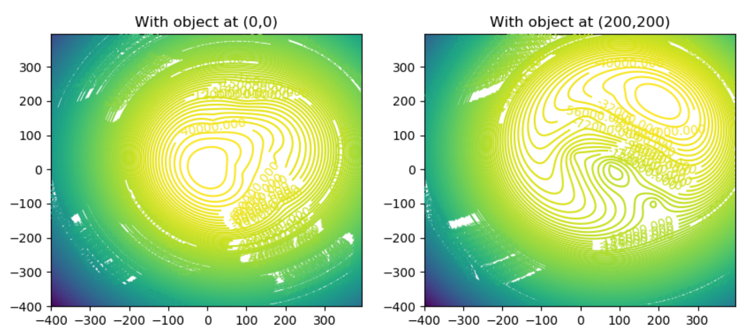


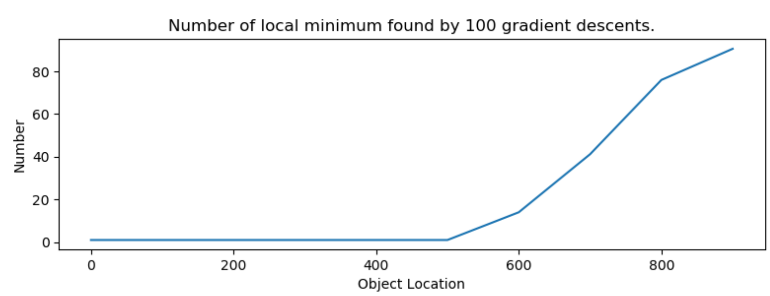
step_size = 0.1

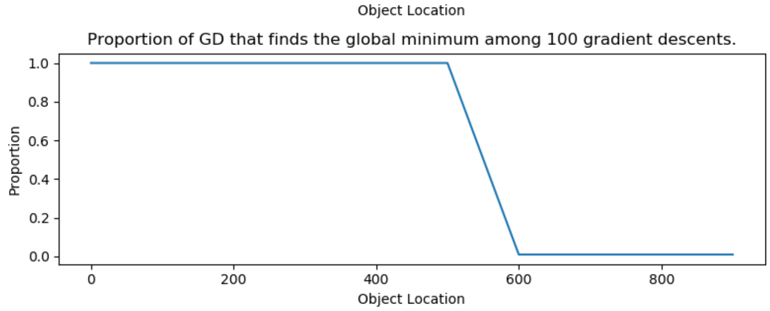


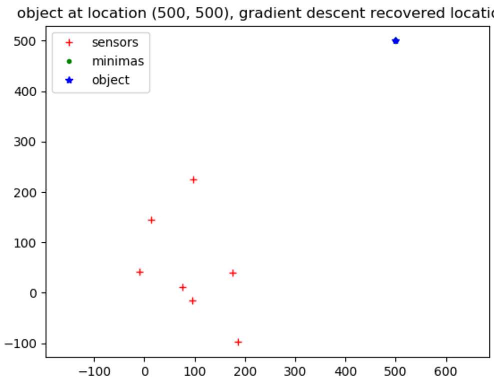
step_size = 0.01

4d

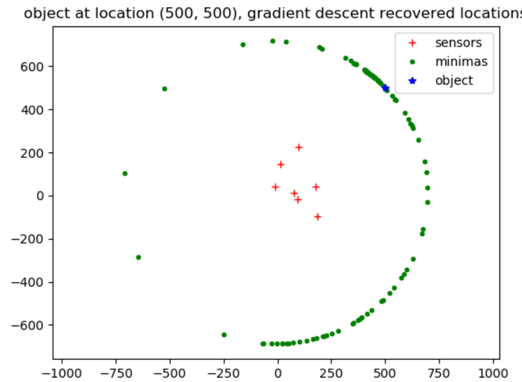






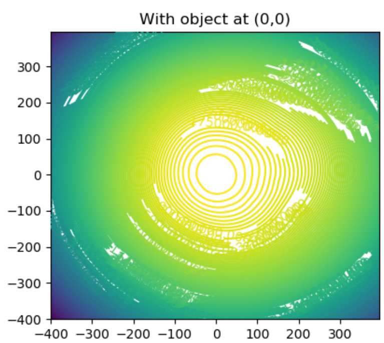


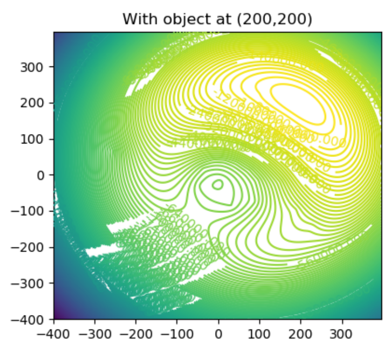
step_size = 0.1

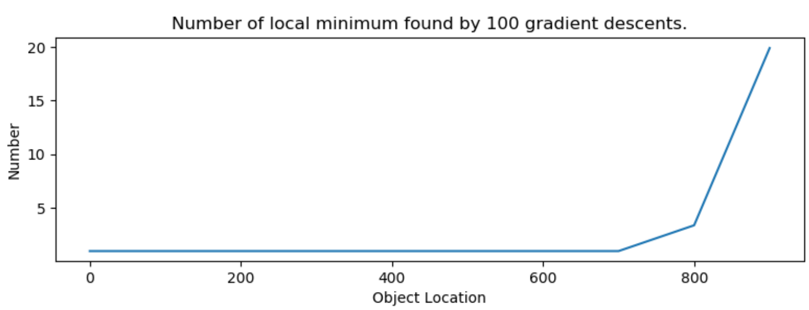


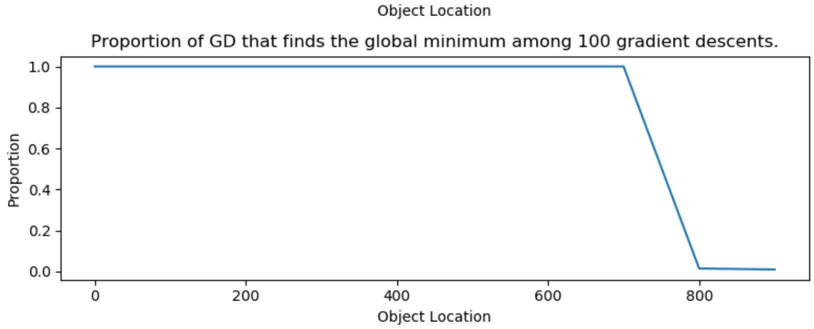
step_size = 0.01

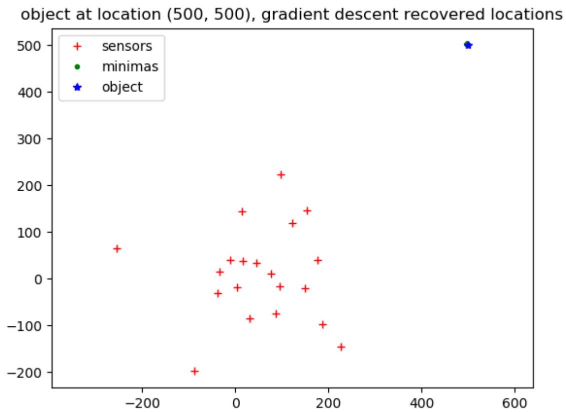
4e



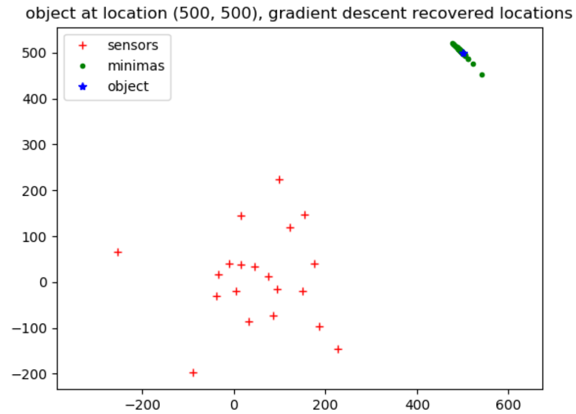








step_size = 0.05



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))
```