Task-C: Regression outlier effect.

Objective:Visualization best fit linear regression line for different scenarios

In [24]:

```
# you should not import any other packages
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import numpy as np
from sklearn.linear_model import SGDRegressor
```

In [25]:

```
import numpy as np
import scipy as sp
import scipy.optimize

def angles_in_ellipse(num,a,b):
    assert(num > 0)
    assert(a < b)
    angles = 2 * np.pi * np.arange(num) / num
    if a != b:
        e = (1.0 - a ** 2.0 / b ** 2.0) ** 0.5
        tot_size = sp.special.ellipeinc(2.0 * np.pi, e)
        arc_size = tot_size / num
        arcs = np.arange(num) * arc_size
        res = sp.optimize.root(
            lambda x: (sp.special.ellipeinc(x, e) - arcs), angles)
        angles = res.x
    return angles
```

In [26]:

```
a = 2
b = 9
n = 50

phi = angles_in_ellipse(n, a, b)
e = (1.0 - a ** 2.0 / b ** 2.0) ** 0.5
arcs = sp.special.ellipeinc(phi, e)

fig = plt.figure()
ax = fig.gca()
ax.axes.set_aspect('equal')
ax.scatter(b * np.sin(phi), a * np.cos(phi))
plt.show()
```

In [27]:

```
X= b * np.sin(phi)
Y= a * np.cos(phi)
```

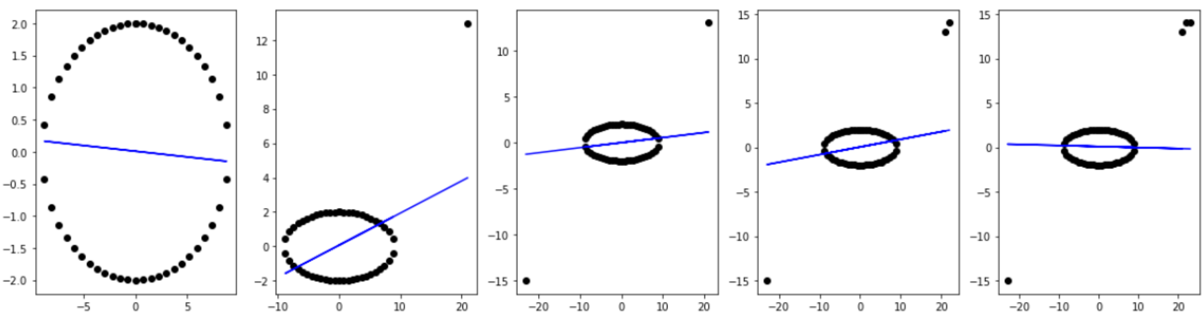
1. As a part of this assignment you will be working the regression problem and how regularization helps to get rid of outliers

2. Use the above created X, Y for this experiment.

3. to do this task you can either implement your own SGDRegression(prefered) excatly similar to "SGD assignment" with mean sequared error or you can use the SGDRegression of sklearn, for example "SGDRegressor(alpha=0.001, eta0=0.001, learning_rate='constant',random_state=0)" note that you have to use the constant learning rate and learning rate ***eta0*** *initialized.*

4. as a part of this experiment you will train your linear regression on the data (X, Y) with different regularizations alpha=[0.0001, 1, 100] and observe how prediction hyper plan moves with respect to the outliers

5. This the results of one of the experiment we did (title of the plot was not metioned intentionally)



in each iteration we were adding single outlier and observed the movement of the hyper plane.

6. please consider this list of outliers: [(0,2),(21, 13), (-23, -15), (22,14), (23, 14)] in each of tuple the first elemet is the input feature(X) and the second element is the output(Y)

7. for each regularizer, you need to add these outliers one at time to data and then train your model again on the updated data.

*8. you should plot a 3*5 grid of subplots, where each row corresponds to results of model with a single regularizer.*

9. Algorithm:

for each regularizer:

for each outlier:
#add the outlier to the data
#fit the linear regression to the updated data
#get the hyper plane
#plot the hyperplane along with the data points

10. MAKE SURE YOU WRITE THE DETAILED OBSERVATIONS, PLEASE CHECK THE LOSS FUNCTION IN THE SKLEARN DOCUMENTATION (please do search for it).

In [28]:

```
print(X.shape)
print(Y.shape)
```

```
(50,)
(50,)
```

In [29]:

```
reg = SGDRegressor()
reg.fit(X.reshape(-1,1),Y.reshape(-1,1))
reg.coef_,reg.intercept_
```

Out[29]:

```
(array([0.01327063]), array([0.00974604]))
```

In [30]:

```
pred = reg.predict(X.reshape(-1,1))
pred
```

Out[30]:

```
array([ 0.00974604,  0.01962923,  0.0295107 ,  0.03938858,  0.04926066,
        0.05912418,  0.0689754 ,  0.07880881,  0.08861568,  0.09838049,
        0.10807106,  0.11760034,  0.12652804,  0.12652804,  0.11760034,
        0.10807106,  0.09838049,  0.08861568,  0.07880881,  0.0689754 ,
        0.05912418,  0.04926066,  0.03938858,  0.0295107 ,  0.01962923,
```

```
        0.00974604, -0.00013716, -0.01001863, -0.01989651, -0.02976859,
       -0.03963211, -0.04948333, -0.05931674, -0.06912361, -0.07888842,
       -0.08857899, -0.09810827, -0.10703597, -0.10703597, -0.09810827,
       -0.08857899, -0.07888842, -0.06912361, -0.05931674, -0.04948333,
       -0.03963211, -0.02976859, -0.01989651, -0.01001863, -0.00013716])
```

In [31]:

```
import matplotlib.pyplot as plt
plt.scatter(X,Y)
plt.plot(X,pred,color='red')
plt.show()
```

In [32]:

```
reg_stren = [0.0001, 1, 100]
outliers = [(0,2),(21, 13), (-23, -15), (22,14), (23, 14)]
plt.figure(figsize=(20,15))
k = 0
for stren in reg_stren:
    X= b * np.sin(phi)
    Y= a * np.cos(phi)
    reg = SGDRegressor(alpha=stren,penalty='l1') #l1 works better than l2 in case of outliers as l2 blo
ws up the error after squaring it
    for outlier in outliers:
        k+=1
        X=np.append(X,[outlier[0]])
        Y=np.append(Y,[outlier[1]])
        #print(X,Y)
        reg.fit(X.reshape(-1,1),Y.reshape(-1,1))
        pred = reg.predict(X.reshape(-1,1))
        plt.subplot(3,5,k)
        plt.scatter(X,Y)
        plt.plot(X,pred,color='red')
        plt.title('Alpha={}'.format(stren))
plt.suptitle('Regularization(L1) and Outliers Impact')
plt.show()
```

In [33]:

```
reg_stren = [0.0001, 1, 100]
outliers = [(0,2),(21, 13), (-23, -15), (22,14), (23, 14)]
plt.figure(figsize=(20,15))
k = 0
for stren in reg_stren:
    X= b * np.sin(phi)
    Y= a * np.cos(phi)
    reg = SGDRegressor(alpha=stren,penalty='l2')
    for outlier in outliers:
        k+=1
        X=np.append(X,[outlier[0]])
        Y=np.append(Y,[outlier[1]])
        #print(X,Y)
        reg.fit(X.reshape(-1,1),Y.reshape(-1,1))
        pred = reg.predict(X.reshape(-1,1))
        plt.subplot(3,5,k)
        plt.scatter(X,Y)
        plt.plot(X,pred,color='red')
        plt.title('Alpha={}'.format(stren))
plt.suptitle('Regularization(L2) and Outliers Impact')
plt.show()
```

Below are the observations:

- When Alpha is too low(0.0001), the algorithm tries to fit to the training set as well as it can and gives little importance to regulariz
- When Alpha is a little considerable(1), the single outlier does not change the decision boundary much but adding more outliers le
- When Alpha is high(100), the algorithm gives appropriate focus on regularization as well and hence adding multiple outliers also
- l1 regularization works better than l2 in case of outliers as l2 blows up the error after squaring it