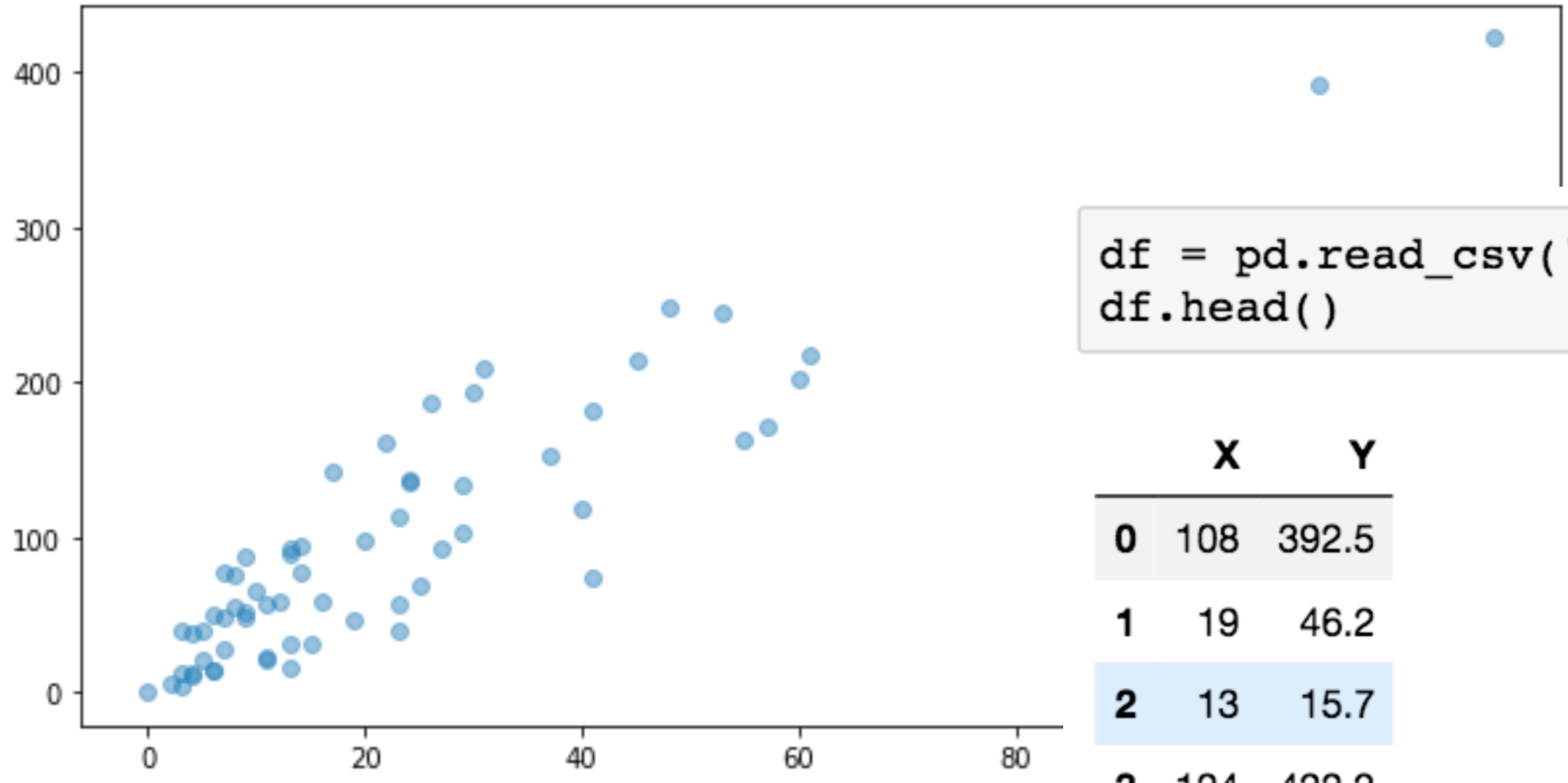
Implementation

Linear Regression

Director of TEAMLAB
Sungchul Choi

Gradient Descent로 Linear Regression 구해보기




```
df = pd.read_csv("data/slr06.csv")
df.head()
```

	X	Y
0	108	392.5
1	19	46.2
2	13	15.7
3	124	422.2
4	40	119.4

```
raw_X = df["X"].values.reshape(-1, 1)
y = df["Y"].values
```

```
plt.figure(figsize=(10,5))
plt.plot(raw_X,y, 'o', alpha=0.5)
```

```python
raw_X[:5], y[:5]
```

```
(array([[108],
        [ 19],
        [ 13],
        [124],
        [ 40]]), array([ 392.5,   46.2,   15.7,  422.2,  119.4]))
```

```python
np.ones((len(raw_X),1))[:5]
```

```
array([[ 1.],
       [ 1.],
       [ 1.],
       [ 1.],
       [ 1.]])
```

```python
X = np.concatenate( (np.ones((len(raw_X),1)), raw_X ), axis=1)
X[:5]
```

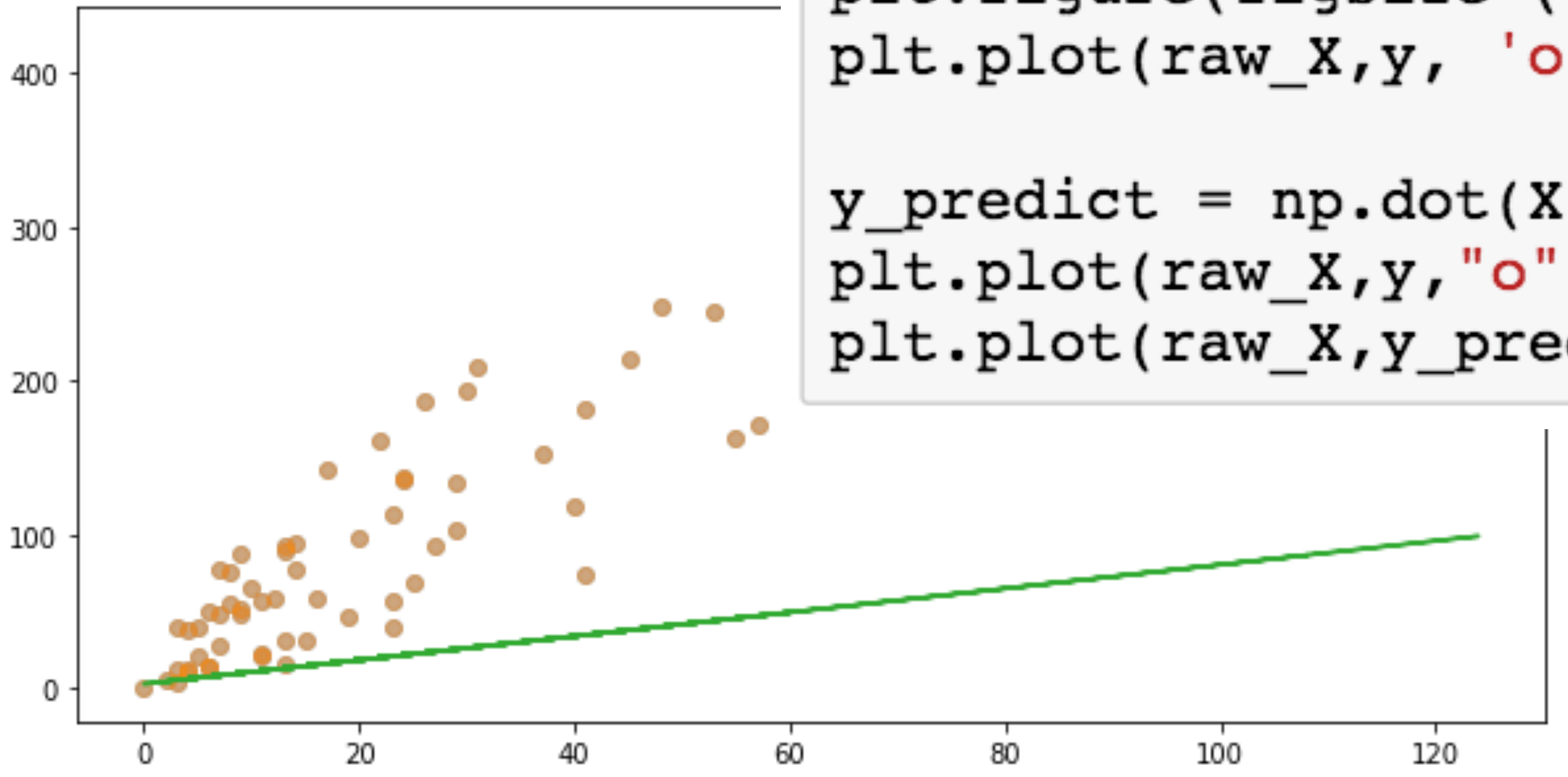
```
array([[   1.,  108.],
       [   1.,   19.],
       [   1.,   13.],
       [   1.,  124.],
       [   1.,   40.]])
```

```
w = np.random.normal((2,1))
# w = np.array([5,3])
w
```

```
array([ 3.19571562,  0.77429904])
```

```
plt.figure(figsize=(10,5))
plt.plot(raw_X,y, 'o', alpha=0.5)

y_predict = np.dot(X, w)
plt.plot(raw_X,y,"o", alpha=0.5)
plt.plot(raw_X,y_predict)
```

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$$



```python
def hypothesis_function(X, theta):
    return X.dot(theta)
```

$$f(x) = h_\theta(x)$$

```python
hypothesis_function(X,w)[:5]
```

```
array([ 86.82001149,  17.9073973 ,  13.26160309,  99.20879607,  34.16767706])
```

```python
def cost_function(h, y):
    return (1/(2*len(y))) * np.sum((h-y)**2)
```

```python
h = hypothesis_function(X,w)
cost_function(h, y)
```

$$J(w_0, w_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

```
5479.121377852004
```

```python
def gradient_descent(X, y, w, alpha, iterations):
    theta = w
    m = len(y)

    theta_list = [theta.tolist()]
    cost = cost_function(hypothesis_function(X, theta), y)
    cost_list = [cost]

    for i in range(iterations):
        t0 = theta[0] - (alpha / m) * np.sum(np.dot(X, theta) - y)
        t1 = theta[1] - (alpha / m) * np.sum((np.dot(X, theta) - y) * X[:,1])
        theta = np.array([t0, t1])

        if i % 10== 0:
            theta_list.append(theta.tolist())
            cost = cost_function(hypothesis_function(X, theta), y)
            cost_list.append(cost)


    return theta, theta_list, cost_list
```

$$f(x) = h_\theta(x) \quad J(w_0, w_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$\frac{\partial J}{\partial w_0} = \frac{1}{m}\sum_{i=1}^{m}(w_1 x^{(i)} + w_0 - y^{(i)})$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m}(w_1 x^{(i)} + w_0 - y^{(i)})x^{(i)}$$

```python
def gradient_descent(X, y, w, alpha, iterations):
    theta = w
    m = len(y)

    theta_list = [theta.tolist()]
    cost = cost_function(hypothesis_function(X, theta), y)
    cost_list = [cost]

    for i in range(iterations):
        t0 = theta[0] - (alpha / m) * np.sum(np.dot(X, theta) - y)
        t1 = theta[1] - (alpha / m) * np.sum((np.dot(X, theta) - y) * X[:,1])
        theta = np.array([t0, t1])

                              .tolist())
                              pothesis_function(X, theta), y)

    return theta, theta_list, cost_list
```

$$f(x) = h_\theta(x) \quad J(w_0, w_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$\frac{\partial J}{\partial w_0} = \frac{1}{m}\sum_{i=1}^{m}(w_1 x^{(i)} + w_0 - y^{(i)})$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m}(w_1 x^{(i)} + w_0 - y^{(i)})x^{(i)}$$

loop until convergence{

do $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

}

```python
def gradient_descent(X, y, w, alpha, iterations):
    theta = w
    m = len(y)

    theta_list = [theta.tolist()]
    cost = cost_function(hypothesis_function(X, theta), y)
    cost_list = [cost]

    for i in range(iterations):
        t0 = theta[0] - (alpha / m) * np.sum(np.dot(X, theta) - y)
        t1 = theta[1] - (alpha / m) * np.sum((np.dot(X, theta) - y) * X[:,1])
        theta = np.array([t0, t1])

        if i % 10== 0:
            theta_list.append(theta.tolist())
            cost = cost_function(hypothesis_function(X, theta), y)
            cost_list.append(cost)


    return theta, theta_list, cost_list
```

$$f(x) = h_\theta(x)$$

$$J(w_0, w_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

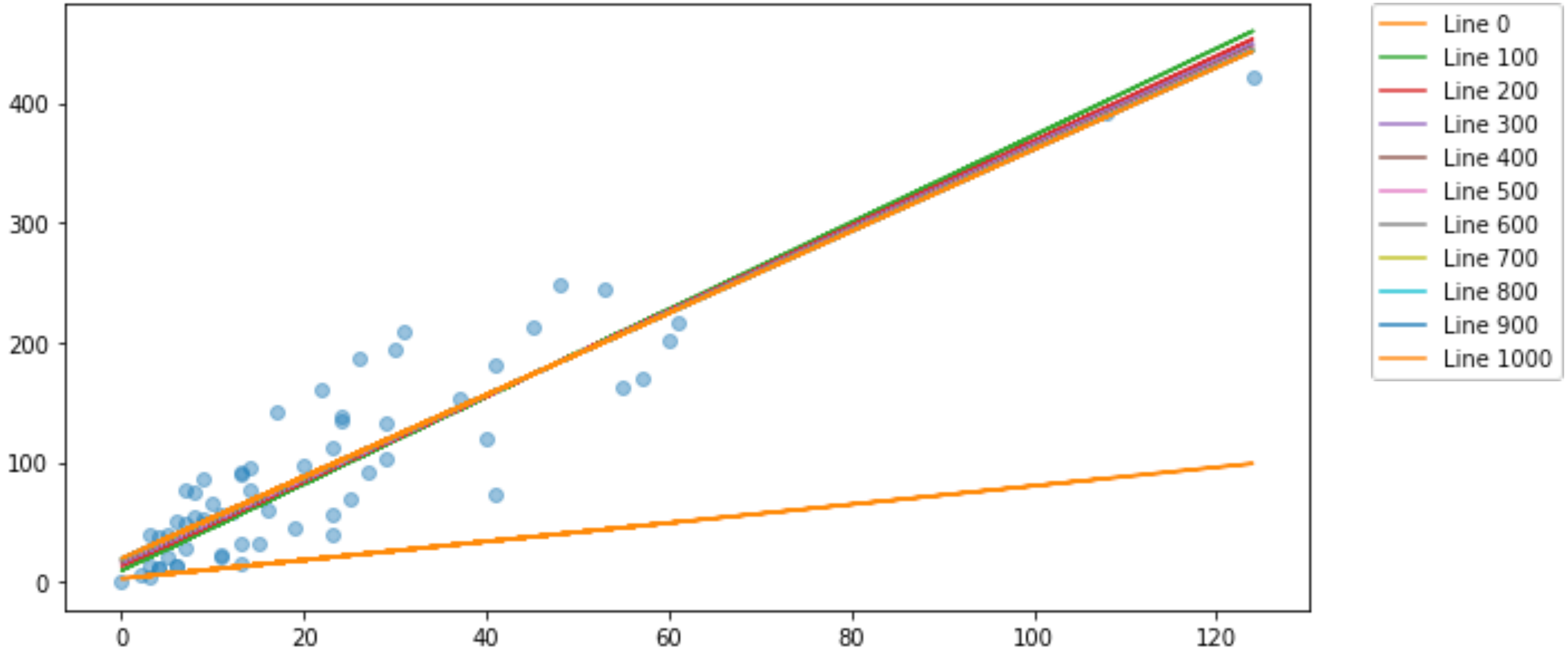
$$\frac{\partial J}{\partial w_0} = \frac{1}{m}\sum_{i=1}^{m}(w_1 x^{(i)} + w_0 - y^{(i)})$$

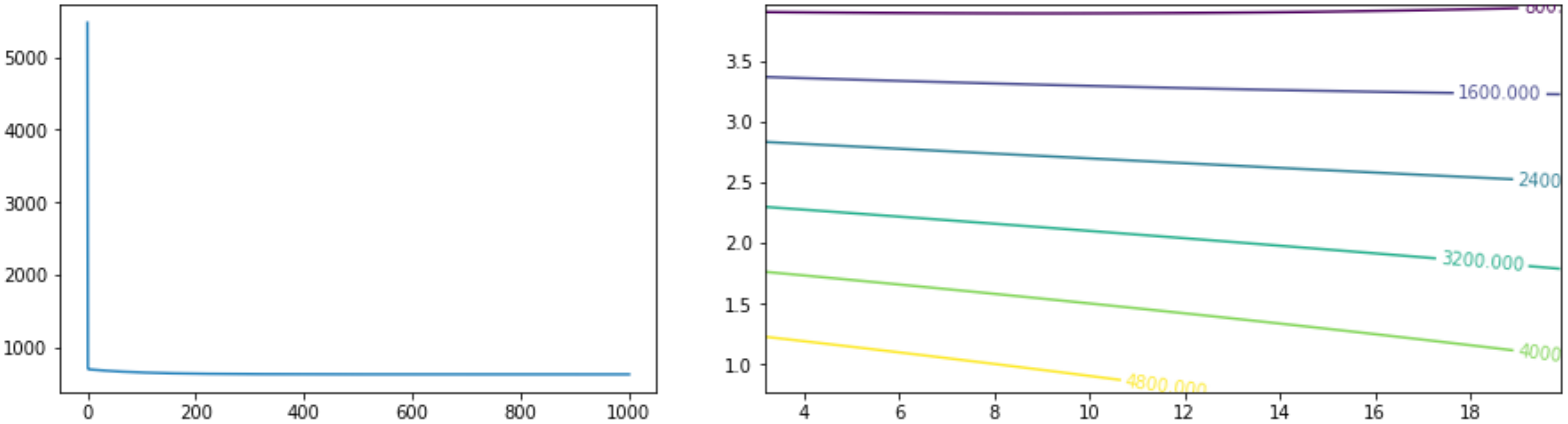
$$\frac{\partial J}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m}(w_1 x^{(i)} + w_0 - y^{(i)})x^{(i)}$$

```python
iterations = 10000
alpha = 0.001

theta, theta_list, cost_list = gradient_descent(X, y, w, alpha, iterations)
cost = cost_function(hypothesis_function(X, theta), y)

print("theta:", theta)
print('cost:', cost_function(hypothesis_function(X, theta), y))
```

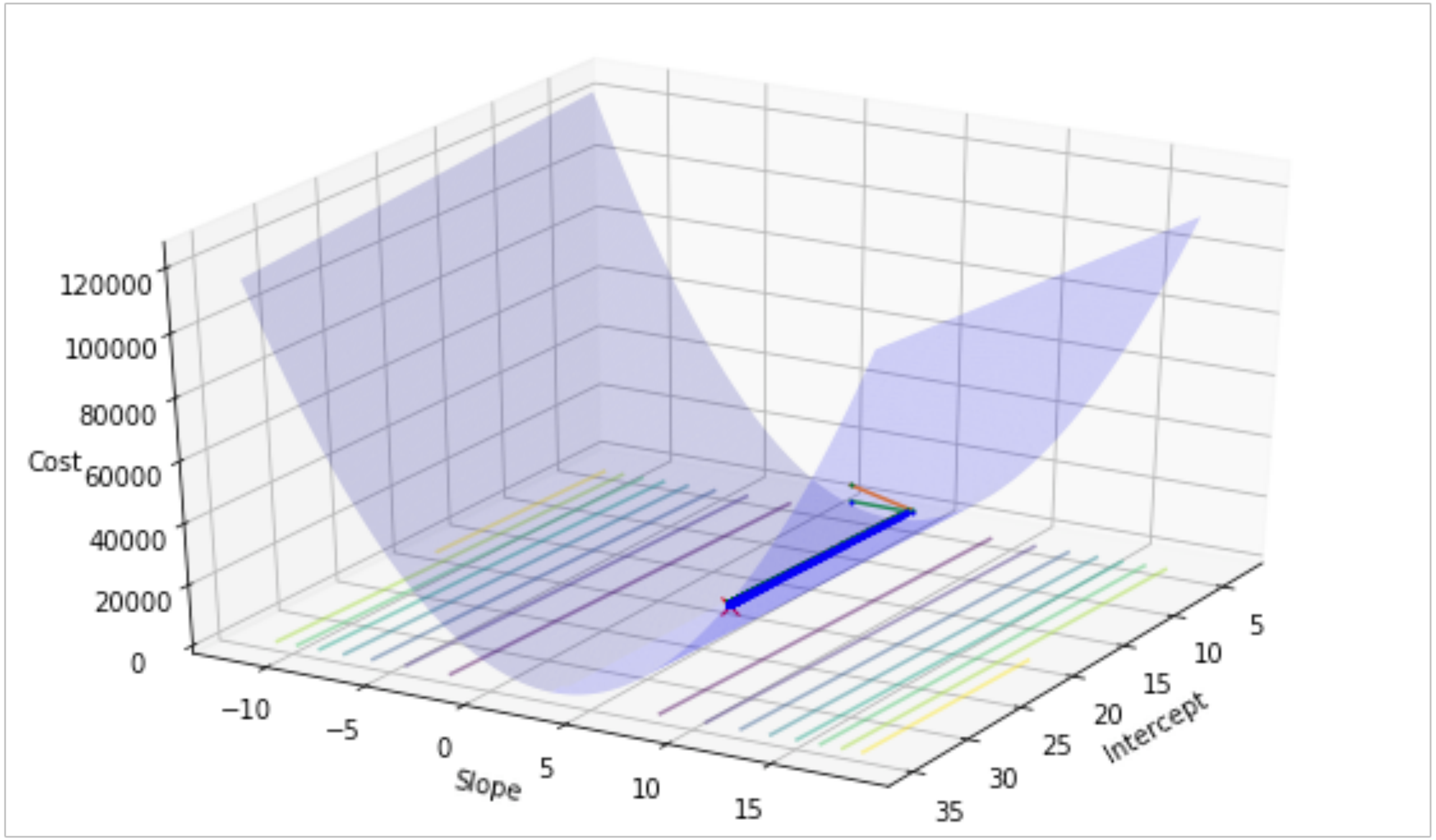
```
theta: [ 19.8878307   3.4161265]
cost: 625.373840751
```

```python
plt.figure(figsize=(10,5))

y_predict_step= np.dot(X, theta_list.transpose())
plt.plot(raw_X,y,"o", alpha=0.5)
for i in range (0,len(cost_list),100):
    plt.plot(raw_X,y_predict_step[:,i], label='Line %d'%i)

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```





5000
4000
3000
2000
1000
0
200
400
600
800
1000
3.5
3.0
2.5
2.0
1.5
1.0
4
6
8
10
12
14
16
18
1600.000
2400
3200.000
4000
4800.000



120000
100000
80000
Cost
60000
40000
20000
0
−10
−5
0
5
10
15
Slope
5
10
15
20
25
30
35
Intercept

TEAMLAB

Human knowledge belongs to the world.