20180129_COGS118a_Hw3

January 28, 2018

1 Minimizers and Maximizers

1.1 Probability

1. $(i^*, j^*) = (0, 1)$
2. $j^* = 0$
3. $i^* = 0$

1.2 Function

Because the function $G(\theta)$ and the ln operator is monotonic, $10 - 3 \times \ln(G(\theta))$ will be minimized when $G(\theta)$ is maximized, thus $\theta^* = 67$.

2 Convex

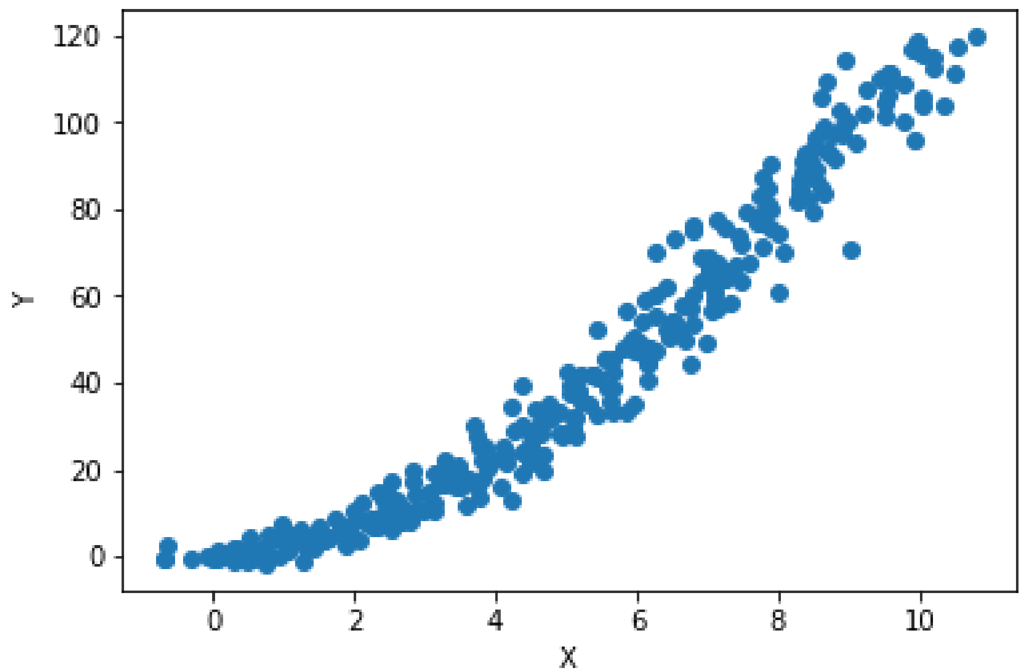
a. Convex
b. Not Convex
c. Convex
d. Not Convex
e. Not Convex
f. Convex

3 Least Square Estimation

1. By the chain rule,
$$\nabla g(W) = X^T \cdot 2(X \cdot W - Y)$$
2. Setting equal to
$$\nabla g(W) = 0$$
and solving for W,
$$\begin{aligned} 0 &= X^T \cdot 2(X \cdot W - Y) \\ 0 &= 2((X \cdot X^T)W - X^T \cdot Y) \\ 2((X \cdot X^T)W) &= 2(X^T \cdot Y) \\ W &= (X \cdot X^T)^{-1} X^T Y \end{aligned}$$

Because $g(W)$ is convex, the value for W found above will produce the minimum for $g(W)$, hence will equal argmin_W

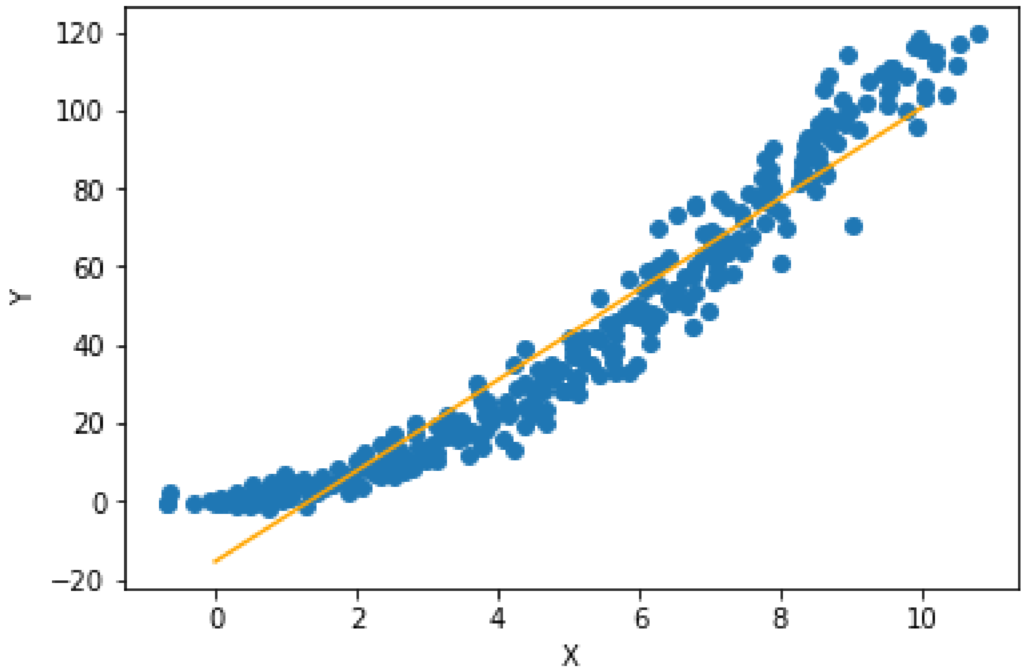
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
X_and_Y = np.load('./q3-least-square.npy')
X = X_and_Y[:, 0] # Shape: (300,)
Y = X_and_Y[:, 1] # Shape: (300,)

plt.scatter(X, Y)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



In [2]:
```
# Assume Y = w0 + w1*X = (w0, w1).(1, X) = W.X1
# X1 contains 1 and X.
X1 = np.matrix(np.hstack((np.ones((len(X),1)),
X.reshape(-1,1))))
W = X1.T.dot(X1).I.dot(X1.T).dot(Y)

print (np.array(W).reshape(-1))
w0, w1 = np.array(W).reshape(-1)
print('Y = {:.2f} + {:.2f}*X'.format(w0, w1))
```

```
[-15.47063382  11.60892201]
Y = -15.47 + 11.61*X
```

```
In [3]: X_line = np.linspace(0,10,300)
        Y_line = w0 + w1 * X_line
        plt.scatter(X, Y)
        plt.plot(X_line, Y_line, color='orange')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.show()
```

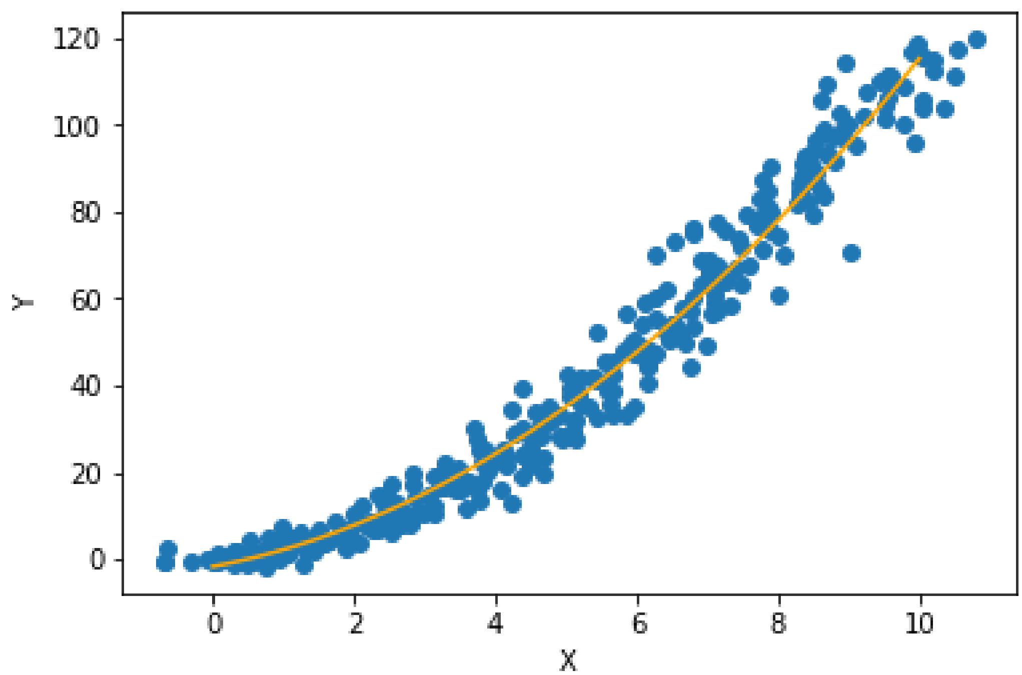



```
In [4]: # Assume Y = w0 + w1*X + w2*X2 = (w0, w1, w2).(1, X, X2) = W.X2
        # X2 contains 1, X and X2.

        X2 = np.matrix(np.hstack((X1, (X**2).reshape(-1,1))))
        W = X2.T.dot(X2).I.dot(X2.T).dot(Y)
        w0, w1, w2 = np.array(W).reshape(-1)
        print('Y = {:.2f} + {:.2f}*X + {:.2f}*X2'.format(w0, w1, w2))
```

```
Y = -1.71 + 3.02*X + 0.87*X2
```

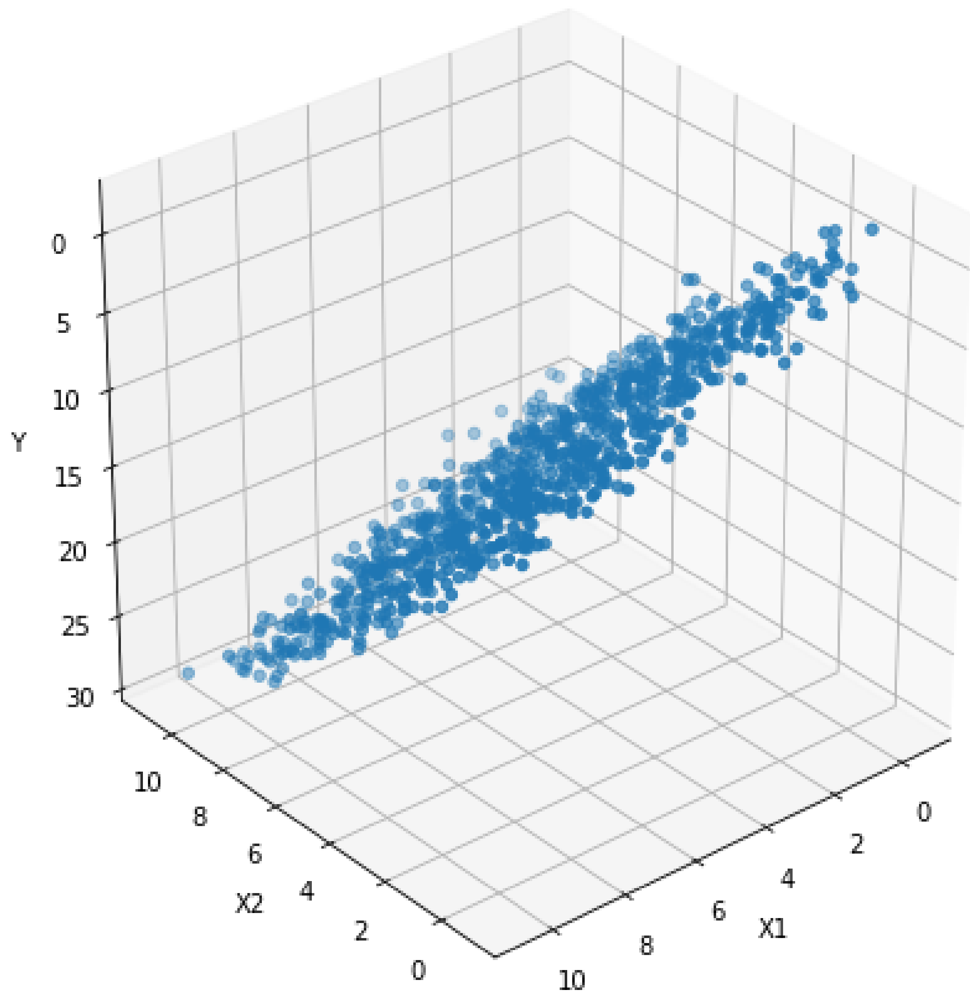
```
In [5]: X_line = np.linspace(0,10,300)
        Y_line = w0 + w1 * X_line + w2 * (X_line**2)
```

```
plt.scatter(X, Y)
plt.plot(X_line, Y_line, color='orange')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



4 Least Square Estimation via Gradient Descent

```
In [6]: import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        X_and_Y = np.load('./q4-gradient-descent.npy')
        X1 = X_and_Y[:, 0] # Shape: (900,)
        X2 = X_and_Y[:, 1] # Shape: (900,)
        Y = X_and_Y[:, 2] # Shape: (900,)

        fig = plt.figure(figsize = (6, 6))
        ax = Axes3D(fig, elev = -150, azim = 130)
        ax.scatter(X1, X2, Y)
        ax.set_xlabel('X1')
        ax.set_ylabel('X2')
        ax.set_zlabel('Y')
        plt.show()
```

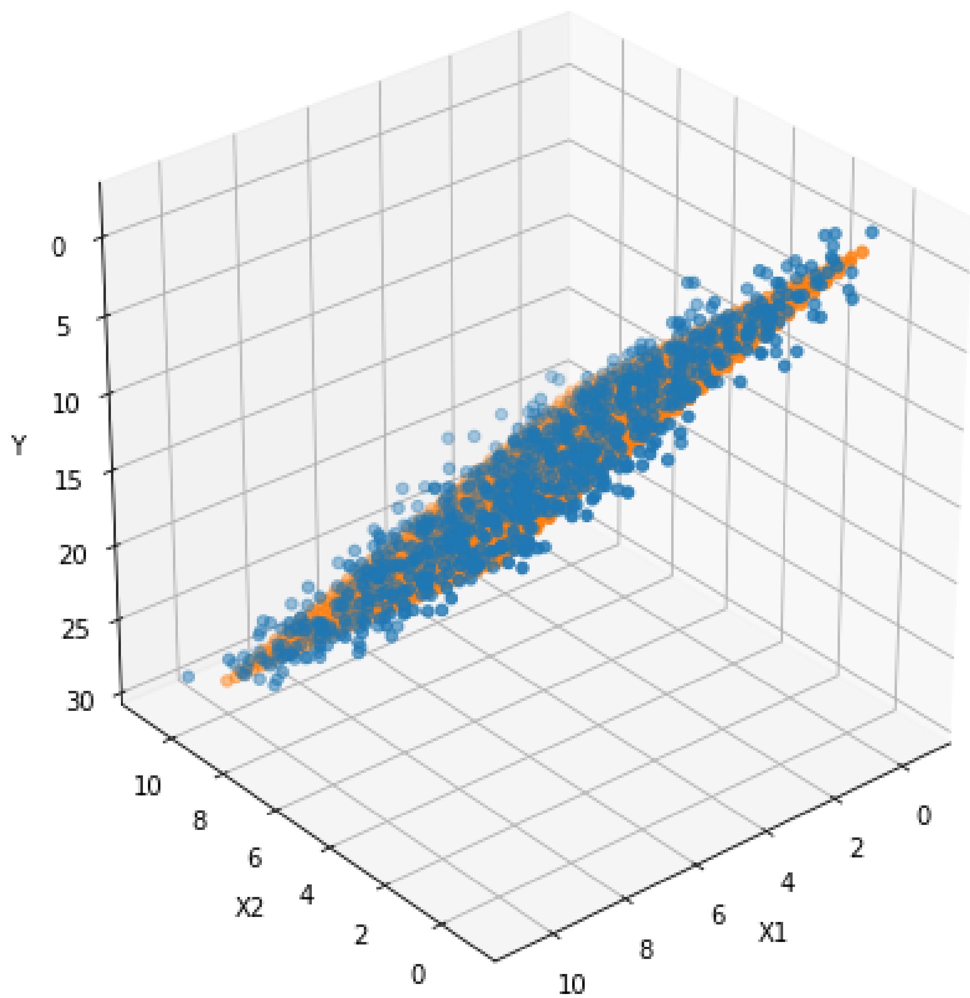



```
In [7]: # Assume Y = w0 + w1 * X1 + W2 * X2
        # = (w0, w1, w2).(1, X1, X2) = W.X
        x1 = np.matrix(np.hstack((np.ones((len(X1),1)),
        X1.reshape(-1,1))))
        X = np.matrix(np.hstack((x1, X2.reshape(-1,1))))
        W = X.T.dot(X).I.dot(X.T).dot(Y)
        w0, w1, w2 = np.array(W).reshape(-1)
        print('Y = {:.2f} + {:.2f}*X1 + {:.2f}*X2'.format(w0, w1, w2))
```

```
Y = -0.70 + 0.98*X1 + 1.94*X2
```

```
In [8]: num = 30
        X_plane_range = np.linspace(0,10,num)
```

```
X_plane_range = np.linspace(0,10,num)
X1_plane, X2_plane = np.meshgrid(X_plane_range, X_plane_range)
Y_plane = w0 + w1 * X1_plane + w2 * X2_plane
fig = plt.figure(figsize = (6, 6))
ax = Axes3D(fig, elev = -150, azim = 130)
ax.scatter(X1, X2, Y)
ax.scatter(X1_plane, X2_plane, Y_plane)
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('Y')
plt.show()
```

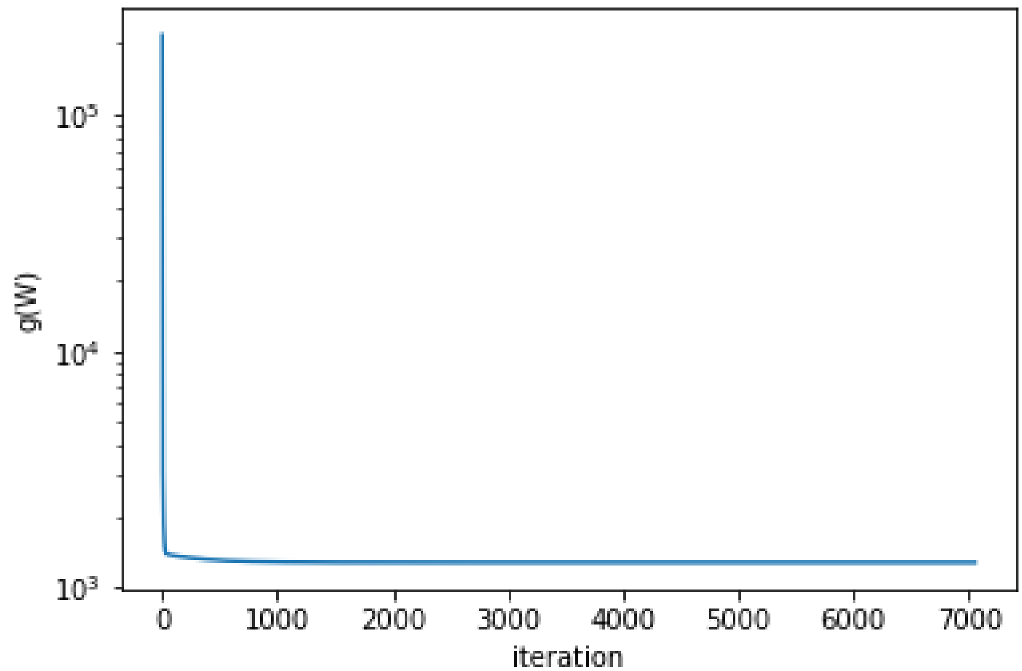



```
In [15]: # g'(W)
         def g_prime_W(X, Y, W):
```

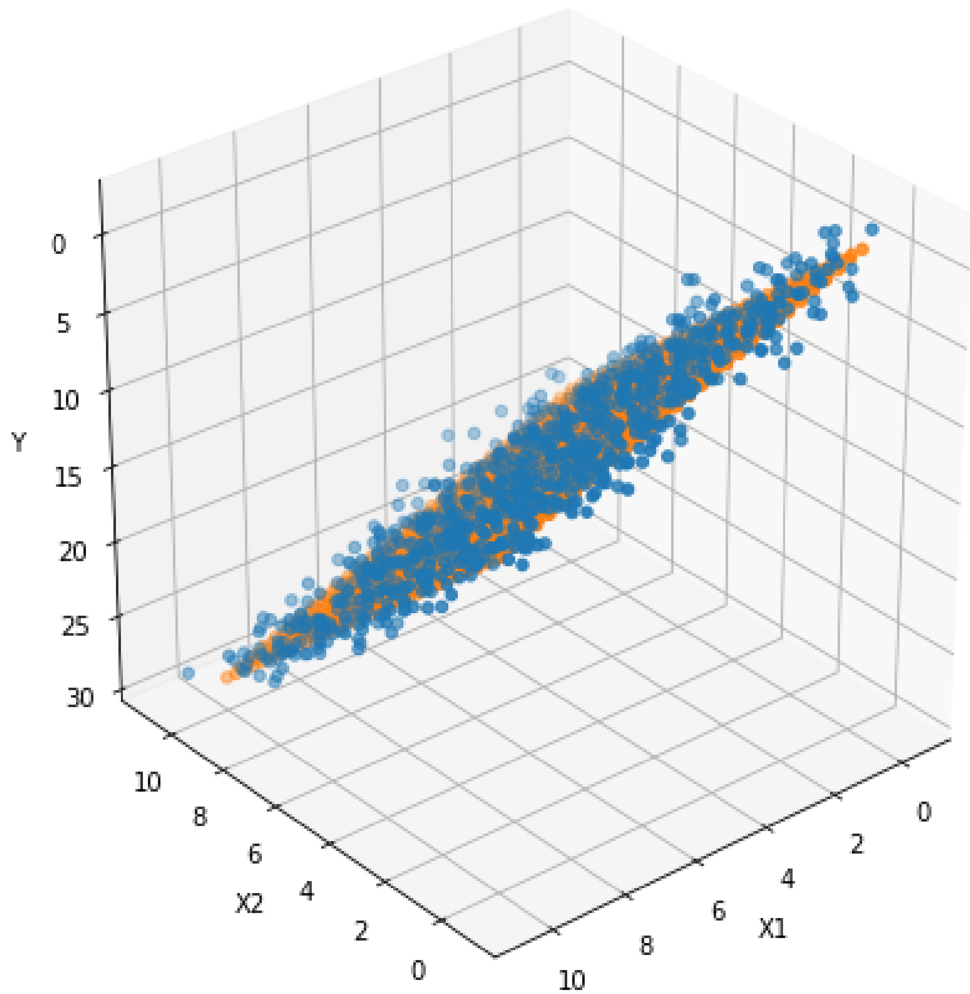
```
        return X.T.dot(2 * ( X.dot(W) - Y))

    # Assume Y = w0 + w1 * X1 + W2 * X2
    # = (w0, w1, w2).(1, X1, X2) = W.X
    W = np.matrix(np.zeros((3,1)))
    Y = Y.reshape(-1, 1)
    # We will keep track of training loss over iterations
    iterations = [0]
    g_W = [(X.dot(W) - Y).T.dot(X.dot(W) - Y)]
    for i in range(10000):
        grad = g_prime_W(X, Y, W)
        W_new = W - 0.000005 * grad
        iterations.append(i+1)
        g_W.append((X.dot(W_new) - Y).T.dot(X.dot(W_new) - Y))
        if np.linalg.norm(W_new - W, ord = 1) < 0.0000001:
            print("gradient descent terminated after " + str(i) + " iterations")
            break
        W = W_new
    w0, w1, w2 = np.array(W).reshape(-1)
    print('Y = {:.2f} + {:.2f}*X1 + {:.2f}*X2'.format(w0, w1, w2))
```

```
gradient descent terminated after 7049 iterations
Y = -0.70 + 0.98*X1 + 1.94*X2
```

```
In [10]: plt.xlabel('iteration')
         plt.ylabel('g(W)')
         plt.semilogy(iterations, np.array(g_W).reshape(-1, 1))
         plt.show()
```




```
In [11]: num = 30
         X_plane_range = np.linspace(0,10,num)
         X_plane_range = np.linspace(0,10,num)
         X1_plane, X2_plane = np.meshgrid(X_plane_range, X_plane_range)
         Y_plane = w0 + w1 * X1_plane + w2 * X2_plane
         fig = plt.figure(figsize = (6, 6))
         ax = Axes3D(fig, elev = -150, azim = 130)
         ax.scatter(X1, X2, Y)
         ax.scatter(X1_plane, X2_plane, Y_plane)
         ax.set_xlabel('X1')
         ax.set_ylabel('X2')
         ax.set_zlabel('Y')
         plt.show()
```



0
5
10
Y
15
20
25
30
10
8
6
X2
4
2
0
0
2
4
6
X1
8
10