HW3

October 21, 2018

1 Cogs 118a Homework 3:

1.0.1 Broderick Higby

1.1 Q1: Convex

$ f(a) $ = convex
$ f(b) $ = non-convex
$ f(c) $ = convex
$ f(d) $ = non-convex
$ f(e) $ = convex
$ f(f) $ = non-convex

1.2 Q2 Single Variable Linear Regression

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %config InlineBackend.figure_format = 'retina'
```

```
In [2]: # Import packages and load data
        X_and_Y = np.load('./q2-least-square.npy')
        X = X_and_Y[:, 0]  # Shape: (300,)
        Y = X_and_Y[:, 1]  # Shape: (300,)
```

1.2.1 2.1: 2D Scatterplot

```
In [3]: # TODO: Plot the a scatter graph of data.
        plt.scatter(X, Y, color='b', label='X and Y points')
        plt.legend(loc = 'lower right')
        plt.title("Single variable Linear Regression")
        plt.show()
```

(2.) $S = \{(x_i, y_i), i = 1 \dots n\}$. Here $x_i, i = 1, \dots, n$

$y_i = w_1 x_i \leftarrow$ line

$y_i = w_1 x_i + w_2 x_i^2 + w_3 \leftarrow$ parabola

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

X_i is a feature vector corresponding to the data x_i

X_i is $\begin{bmatrix} 1 \\ x_i \end{bmatrix}$ or $\begin{bmatrix} 1 \\ x_i \\ x_i^2 \end{bmatrix}$

$$g(W) = ||XW - Y||_2^2 = (XW - Y)^T (XW - Y)$$

(a) Compute gradient of $g(w)$ w/r/t W

$$g(w) = ||XW - Y||_2^2 = (XW - Y)^T (XW - Y)$$

$$g(w) = W^T X^T X W - W^T X^T Y - Y^T X W + Y^T Y$$

$$\frac{dg(W)}{dW} = 2(X^T X)W - X^T Y - X^T Y + 0$$

$$\frac{dg(W)}{dW} = 2X^T X W - 2X^T Y = 0$$

2(b) By setting the answer of part (a) to **0**, prove the following:

$$W^* = \arg\min_W g(W) = (X^TX)^{-1}X^TY.$$

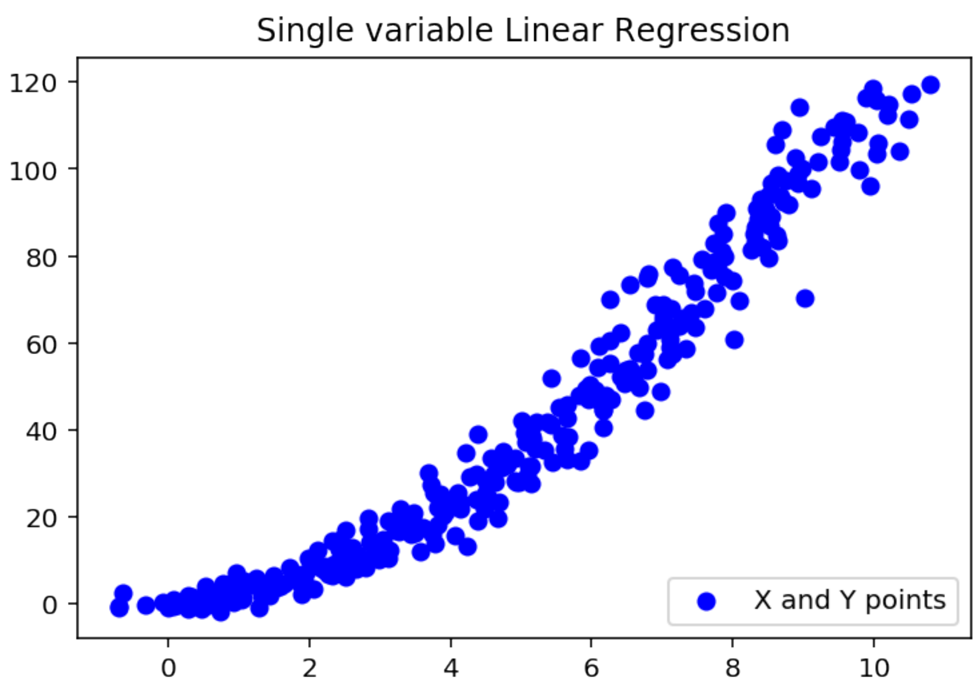
$$2X^TXW - 2X^TY = 0$$

$$\Rightarrow \quad 2X^TXW + 2X^TY = 2X^TY$$

$$\Rightarrow \quad \frac{2X^TXW}{2X^TX} = \frac{2X^TY}{2X^TX}$$

$$\Rightarrow \quad W = \frac{\cancel{2}X^TY}{\cancel{2}X^TX}$$

$$W^* = \arg\min_W g(W) = (X^TX)^{-1}X^TY$$



1.2.2 2.2: Compute the Least Square Line Using the Closed Form (Example Code)

```
In [4]: from numpy.linalg import inv
        from numpy import dot, multiply, matrix, hstack, ones, reshape
        # You might find the following functions useful: np.matrix, np.hstack, np.ones, reshap
        # Compute the least square line over the given data
        # Assume Y = w0 + w1 * X = (w0, w1).(1, X) = W.X1
        square_X = np.hstack((np.ones((len(X),1)), (np.matrix(X)).T))

        W = dot(dot(inv(dot(square_X.T,square_X)),square_X.T),Y)


        w0, w1 = W[0,0], W[0,1]
        # least_squares_Y = w0 + w1 * square_X
        # W_X1 = dot(dot(w0,w1),square_X)
        # w0, w1 = W_X1[0,0], W_X1[0,1]

        print('Y = {:.2f} + {:.2f}*X'.format(w0, w1))
```

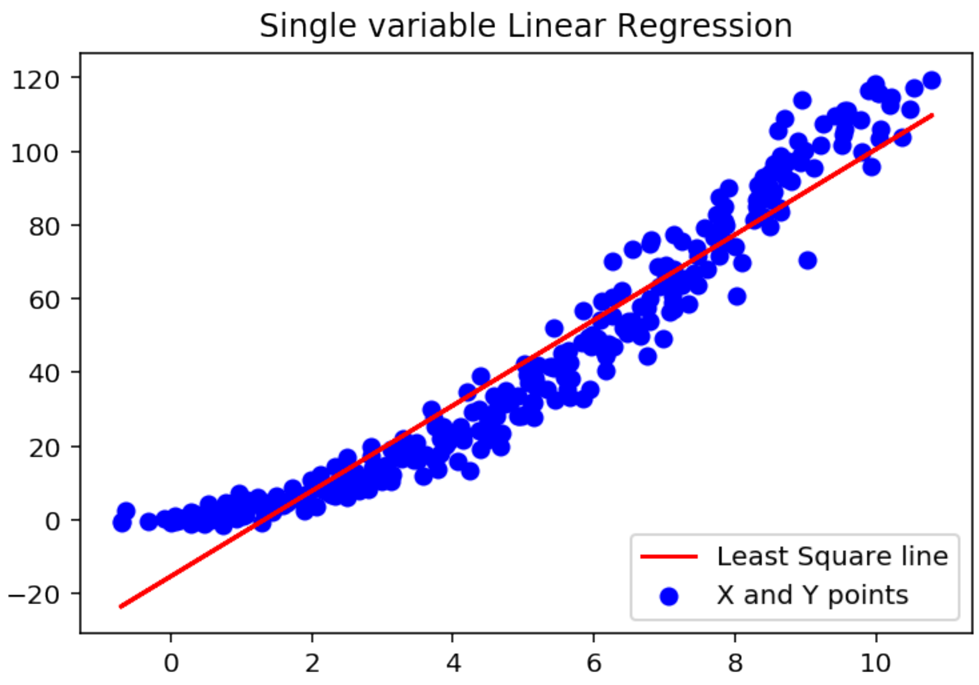
```
Y = -15.47 + 11.61*X
```

1.2.3 2.3: 2D Scatterplot & the Estimated Least Square Line

```
In [5]: # TODO 3.  Plot the the estimated least square line on top of the scatter plot in (2).
        # The scatterplot and the line should be in the same figure.
```

```
plt.scatter(X, Y, color='b', label='X and Y points')
plt.plot(X, w0 + w1 * X, 'r', label='Least Square line')

plt.title("Single variable Linear Regression")
plt.legend(loc = 'lower right')
plt.show()
```



1.2.4 2.4: Compute the Least Square Parabola Using the Closed Form

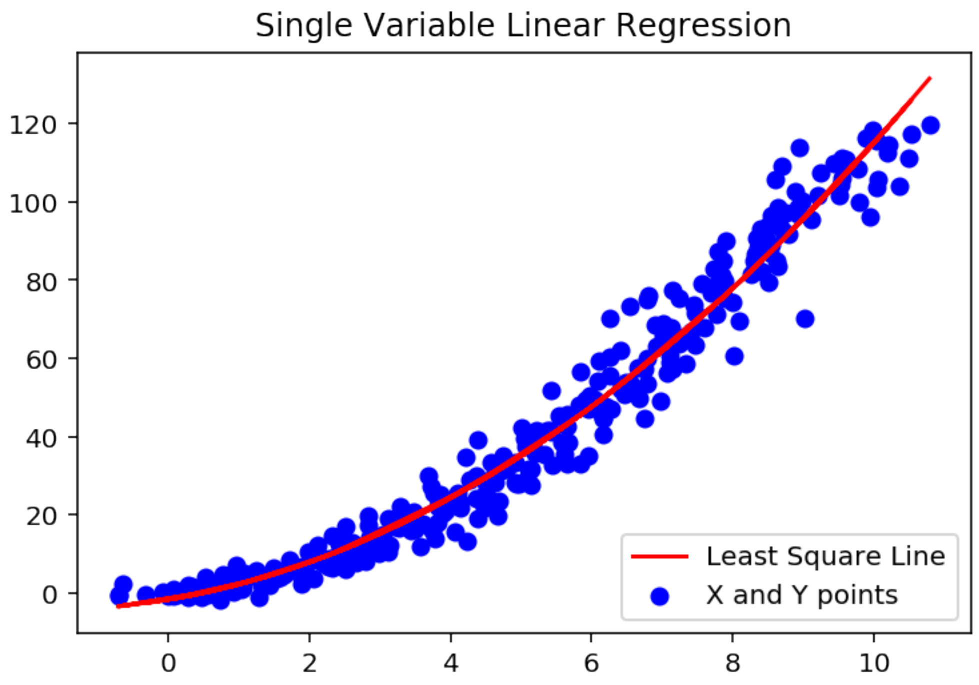
```
In [6]: # TODO 4. Compute the least square parabola over the given data
        # Assume Y = w0 + w1 * X + w2 * X^2 = (w0, w1, w2).(1, X, X^2) = W.X2
        X = X.reshape(300,1)
        Y = Y.reshape(300,1)
        X2d = np.hstack((np.ones((len(X), 1)), (np.matrix(X)), (np.matrix(np.square(X)))))
        W2d = dot(dot(inv(dot(X2d.T,X2d)),X2d.T),Y)
        print(X2d.shape)
        print(W2d.shape)
        w0, w1, w2 = W2d[0,0], W2d[1,0], W2d[2,0]
        print('Y = {:.2f} + {:.2f}*X + {:.2f}*X^2'.format(w0, w1, w2))
```

```
(300, 3)
(3, 1)
Y = -1.71 + 3.02*X + 0.87*X^2
```

1.2.5 2.5: 2D Scatterplot & the Estimated Parabola

```
In [7]: # TODO 5.  Plot the the estimated parabola on top of the scatter plot in (2).
        # The scatter plot and the parabola should be in the same figure
        # x1 = np.linspace(300,3)
        # parabola = np.polyfit(X,W,2)

        x_parabola = (w0 + dot(w1, X2d[:,1]) + dot(w2,X2d[:,2]))
        plt.scatter(X,Y, color='b', label='X and Y points')
        plt.plot(X, x_parabola, 'r', label='Least Square Line')
        plt.title("Single Variable Linear Regression")
        plt.legend(loc = 'lower right')
        plt.show()
```



2 Q3 Multi-Variable Linear Regression

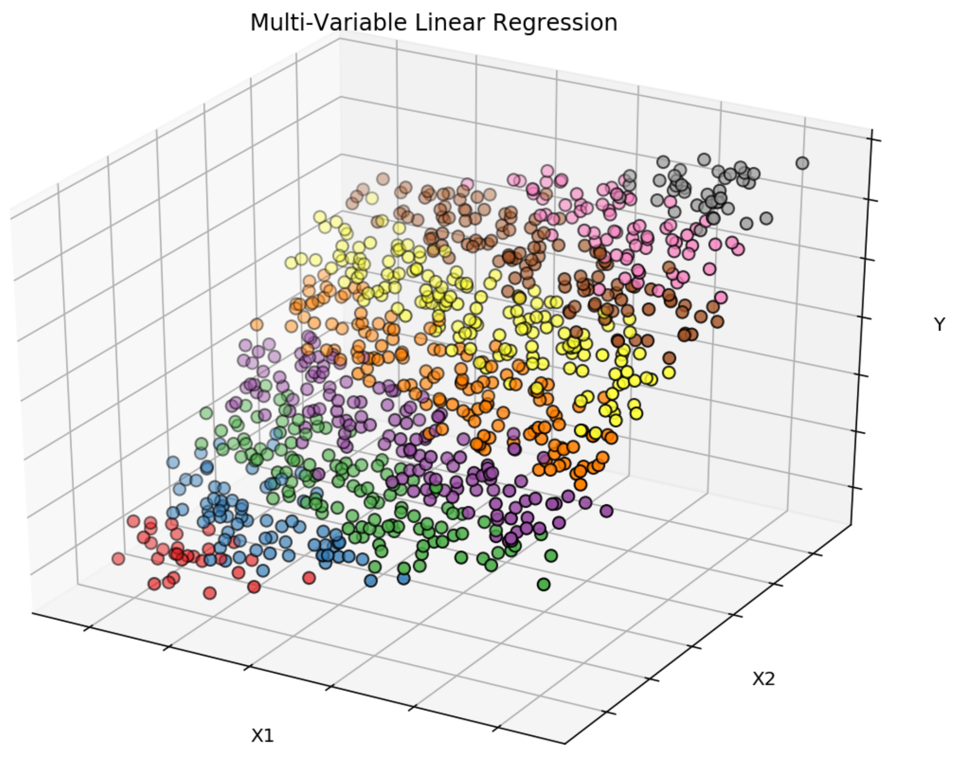
```
In [8]: import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        %config InlineBackend.figure_format = 'retina'

In [9]: # Import packages and load data
        X_and_Y = np.load('./q3-gradient-descent.npy')
        X1 = X_and_Y[:, 0]     # Shape: (900,)
        X2 = X_and_Y[:, 1]     # Shape: (900,)
        Y  = X_and_Y[:, 2]     # Shape: (900,)
        print(X1.shape, X2.shape, Y.shape)
```

```
(900,) (900,) (900,)
```

2.0.1 3.1: 3D Scatterplot

```
In [10]: # TODO: Plot the a scatter graph of data.
         fig = plt.figure(1, figsize=(8, 6))
         ax = Axes3D(fig)
         ax.scatter(X1,X2,Y, c=Y, cmap=plt.cm.Set1, edgecolor='k', s=40)
         ax.set_title("Multi-Variable Linear Regression")
         ax.set_xlabel("X1")
         ax.w_xaxis.set_ticklabels([])
         ax.set_ylabel("X2")
         ax.w_yaxis.set_ticklabels([])
         ax.set_zlabel("Y")
         ax.w_zaxis.set_ticklabels([])
         plt.show()
```



2.0.2 3.2 Compute the Least Square Plane Using the Closed Form

```
In [11]: # TODO: Compute the least square Plane over the given data
         # Assume Y = w0 + w1 * X1 + W2 * X2 = (w0, w1, w2).(1, X1, X2) = W.X
         X1 = X1.reshape(900,1)
         X2 = X2.reshape(900,1)
         Y = Y.reshape(900,1)
         one = ones(shape = Y.shape)
         # one = one_shape.reshape(900,1)
         print(X1.shape)
         print(X2.shape)
         print(one.shape)
         print(Y.shape)
         # X1, X2 = matrix(X1.reshape(900,1)), matrix(X2.reshape(900,1))
         # one = matrix(ones(len(X1),1).reshape(900,1))
         # X = np.hstack((np.ones((len(X1), 1)), (np.matrix(X1)), (np.matrix(X2))))
         X = hstack((one, matrix(X1), matrix(X2)))

         W = dot(dot(inv(dot(X.T,X)),X.T),Y)
         w0, w1, w2 = W[0,0], W[1,0], W[2,0]
         print('Y = {:.2f} + {:.2f}*X1 + {:.2f}*X2'.format(w0, w1, w2))
```

```
(900, 1)
(900, 1)
(900, 1)
(900, 1)
Y = -0.70 + 0.98*X1 + 1.94*X2
```

2.0.3 3.3: 3D Scatterplot & the Estimated Least Square Plane

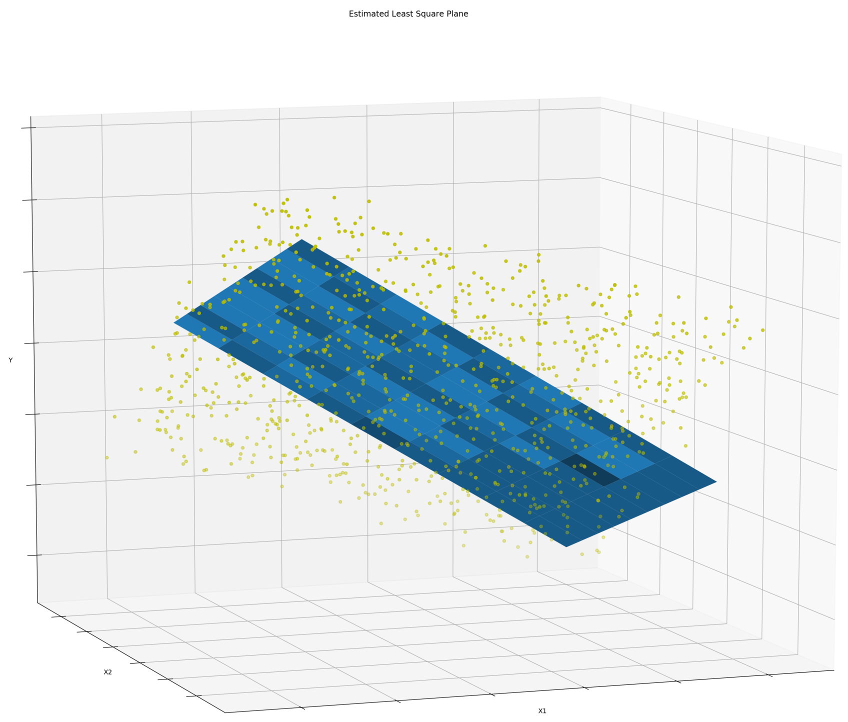
```
In [12]: # TODO: Plot the scatter graph of data and estimated plane using the closed form solu
         fig = plt.figure(1, figsize=(30, 25))
         ax = fig.gca(projection='3d')
         x, y = np.meshgrid(range(10), range(10))

         Z = w2*x + w1*y + w0

         ax.plot_surface(x, y, Z)

         ax.scatter(X1,X2,Y, c='y')
         ax.view_init(10,70)

         ax.set_title("Estimated Least Square Plane")
         ax.set_xlabel("X1")
         ax.w_xaxis.set_ticklabels([])
         ax.set_ylabel("X2")
         ax.w_yaxis.set_ticklabels([])
         ax.set_zlabel("Y")
         ax.w_zaxis.set_ticklabels([])
         plt.show()
```



2.0.4 3.4: Compute the gradient of g(W) with respect to W.

Hint: You have computed the analytic solution in problem 3

```
In [13]: # TODO: g'(W) - The equation from 2A
         def g_prime_W(X, Y, W):
         #     XT_X = dot(X.T,X)
         #     XT_X_W = dot(XT_X,W)
         #     print(XT_X_W.shape)
         #     XT_Y = dot(X.T,Y)
         #     print(XT_X_W.shape)
         #     print(XT_Y.shape)
         #     return (2 * (XT_X_W - XT_Y))
             return 2*((X.transpose().dot(X)).dot(W) - X.transpose().dot(Y))
         g_prime_W(X,Y,W)
         print(g_prime_W(X,Y,W))
```

```
[[  1.45519152e-11]
 [  2.91038305e-11]
 [  0.00000000e+00]]
```

2.0.5 3.5 Compute the Least Squares Plane Using Gradient Descent

```
In [15]: # TODO: Compute the least square Plane over the given data
         # print('Y = {:.2f} + {:.2f}*X1 + {:.2f}*X2'.format(w0, w1, w2))

         def linear_regression(X, W, epochs=10000, learning_rate=0.0001):
             W_new = [0,0,0]
             w_old = W
             while sum(abs(W_new - W_old) > learning_rate && count < epoch:
                 gradient = g_prime_w(X,Y,W)
                 W_new = W_old - learning_rate * gradient
                 count + 1

             return W_new
```

```
      File "<ipython-input-15-f3c7672d6680>", line 7
    while sum(abs(W_new - W_old) > learning_rate && count < epoch:
                                                  ^
SyntaxError: invalid syntax
```

2.0.6 3.6 Plot the training curve

```
In [16]: # TODO: Plot the training curve
```

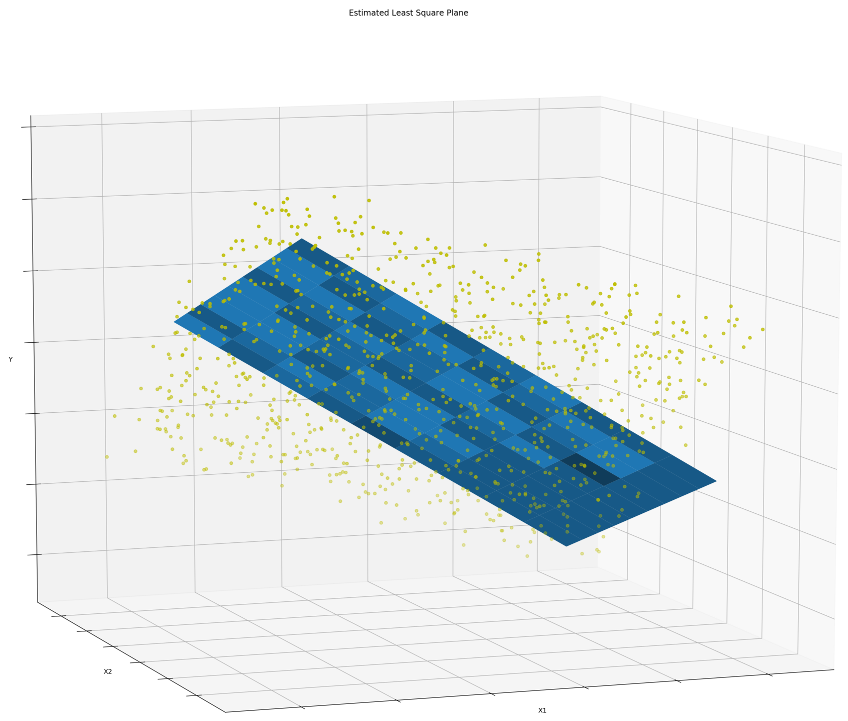
2.0.7 3.7 Plot the scatter graph of data and estimated plane using the gradient descent solution

```
In [17]: # TODO: Plot the scatter graph of data and estimated plane
         # TODO: Plot the a scatter graph of data.
         fig = plt.figure(1, figsize=(30, 25))
         ax = fig.gca(projection='3d')
         x, y = np.meshgrid(range(10), range(10))
         Z = w2*x + w1*y + w0

         ax.plot_surface(x, y, Z)

         ax.scatter(X1,X2,Y, c='y')
         ax.view_init(10,70)

         ax.set_title("Estimated Least Square Plane")
         ax.set_xlabel("X1")
         ax.w_xaxis.set_ticklabels([])
         ax.set_ylabel("X2")
         ax.w_yaxis.set_ticklabels([])
         ax.set_zlabel("Y")
         ax.w_zaxis.set_ticklabels([])
         plt.show()
```



2.1 Q4: Concepts

1. What are the most significant difference between regression and classification?

```
B. prediction of continuous values vs. prediction of class labels

D. convex vs. non-convex problem
```

2. What are true about solving regression problem with gradient descent compared to closed-form solution?

```
A. matrix inverse could be expensive when the dataset is large
```

3. Is gradient descent guaranteed to find the global optimal in a convex problem? What about non-convex problem?

   `B. no for a convex problem`

   `C. yes for a non-convex problem D. no for a non-convex problem`

4. What are true about local optimal and global optimal?

   `B. There can exist multiple local optimal`

   `C. gradient descent is able to find the global optimal`