```
import numpy as np
import matplotlib.pyplot as plt

'''
Hyperparameter Configurations
'''
# Prior weights are assumed to be i.i.d, so they all have the same prior_std
prior_mean = [0, 0]
prior_std = 0.0001
noise_mean = 0.0
noise_std = 1.0
# Can either by Gaussian or Laplacian Prior
options = ["Gaussian", "Laplacian"]
option = options[1]

# Plot and Randomness Configurations
fig, ax = plt.subplots()
ax.axhline(y=0, color='k')
ax.axvline(x=0, color='k')
lower_range = -5
upper_range = 5
# DO NOT Change the Seed and Num Points for Easy and Consistent Grading
np.random.seed(7)
num_data_points = 10


'''
Point Estimate Functions
'''
def MLE_Point_Est(X, Y):
    '''
    Input: X - nx2; Y - nx1
    Return: w - 2x1
    Returns the OLS Solution
    '''
    return np.linalg.solve(X.T@X, X.T@Y)

def Gaussian_MAP_Point_Est(X, Y, noise_std, prior_std, prior_mean):
    '''
    Input: X - nx2; Y - nx1, Prior and Noise Staistics (All scalars)
    Return: w - 2x1
    Returns the Ridge Regression Solution
    '''
    lambda_ = (noise_std/prior_std)**2
    return np.linalg.solve(X.T@X+lambda_*np.identity(X.shape[1]),
        X.T@Y+lambda_*np.array(prior_mean).reshape((2,1)))

def LASSO_Point_Est(w1, w2, map_contour):
    '''
    This part of the code is filled out. Since in most cases there is no
     closed form for the LASSO equation, we directly find the center
     through a linear search.
    DON'T DO THIS.
    '''
    min_candidates = np.where(map_contour == np.amin(map_contour))
```

```
    processed_points = list(zip(min_candidates[0], min_candidates[1]))

    x_ind = processed_points[0][0]
    y_ind = processed_points[0][1]
    return np.array([w1[x_ind][y_ind], w2[x_ind][y_ind]]).reshape((-1,1))

'''
Functions for Contour Plotting
'''

def Gaussian_Prior(w1, w2, prior_mean):
    '''
    Returns the Gaussian Prior or L2 norm squared for weights
    '''
    return
     (1/(2*np.pi*(prior_std**2)))*np.exp(-0.5*((((w1-prior_mean
     [0])/prior_std)**2)+(((w2-prior_mean[1])/prior_std)**2)))

def Laplacian_Prior(w1, w2, prior_mean):
    '''
    Returns the Laplacian Prior for weights
    '''
    return
     ((np.abs(w1-prior_mean[0])+np.abs(w2-prior_mean
     [1]))/prior_std)*(2*(noise_std**2))

def Gaussian_MLE_Contour(w1,w2, X, Y):
    '''
    Returns the MLE estimate value
    '''
    x = np.array([0]*200)
    y = np.array([0]*200)
    Z, z = np.meshgrid(x,y)
    for i in range(200):
        for j in range(200):
            Z[i][j] = np.sum(((Y-X@(np.array([w1[i][j],
                w2[i][j]]).reshape((2,1))))**2)/(2*(noise_std**2))) +
                 num_data_points*np.log(np.sqrt(2*np.pi)*noise_std)
    return Z

def MAP_Contour(w1, w2, X, Y, noise_std, prior_mean, prior_std,
 option="Gaussian"):
    '''
        Hint 1: Use the methods above to compute the MAP. Ideally one line
         of code.
        Hint 2: Use two for loops to go through the (x,y) coordinates for
         w1, w2
    '''
    x = np.array([0]*200)
    y = np.array([0]*200)
    Z, z = np.meshgrid(x,y)
    for i in range(200):
        for j in range(200):
            if option == "Laplacian":
```

```
                Z[i][j] = np.linalg.norm(Y-X@(np.array([w1[i][j],
                 w2[i]
                 [j]])
                 .reshape((2,1))))**2+(2*(noise_std**2)/prior_std)*((np
                 .abs(w1[i][j]-prior_mean[0])+np.abs(w2[i][j]-prior_mean
                 [1])))
            Z[i][j] = np.sum(((Y-X@(np.array([w1[i][j],
             w2[i][j]]).reshape((2,1))))**2)/(2*(noise_std**2))) +
             (np.array([w1[i][j],
                w2[i][j]]) - np.array(prior_mean))@((np.array([w1[i][j],
                 w2[i][j]]) -
                 np.array(prior_mean)).reshape(2,1))/(2*(prior_std**2))
    return Z

'''
Plotting Functions
'''
def plot_point(w, txt):
    plt.plot(w[0][0], w[1][0], 'ro')
    ax.annotate(txt, (w[0][0], w[1][0]))

# Generating Data and Labels with Random Gaussian Noise
w = np.array([[2.0], [1.0]])
Xf= np.random.uniform(-1, 1, num_data_points)
X1 = np.array([1.0]*num_data_points)
X = np.array([Xf, X1]).T
Y = X.dot(w).reshape((-1, 1)) + np.random.normal(noise_mean, noise_std,
 num_data_points).reshape((-1,1))

w1 = np.linspace(lower_range, upper_range, 200)
w2= np.linspace(lower_range, upper_range, 200)
w1, w2 = np.meshgrid(w1, w2)

prior = Gaussian_Prior(w1, w2, prior_mean) if option=="Gaussian" else
 Laplacian_Prior(w1,w2, prior_mean)
mle_contour = Gaussian_MLE_Contour(w1,w2, X, Y)
map_contour = MAP_Contour(w1, w2, X, Y, noise_std, prior_mean, prior_std,
 option)

w_mle = MLE_Point_Est(X,Y)
w_map = Gaussian_MAP_Point_Est(X,Y, noise_std, prior_std, prior_mean) if
 option=="Gaussian" else LASSO_Point_Est(w1, w2, map_contour)

plot_point(w, "Truth")
plot_point(w_mle, "MLE")
plot_point(w_map, "MAP")

# Choose which contours to plot [prior, mle, or map]
plt.contour(w1, w2, map_contour, 7, cmap='Reds_r')
plt.contour(w1, w2, mle_contour, 7, cmap='Blues_r')
plt.contour(w1, w2, prior, 7, cmap='gray')

plt.show()
```