Homework 4 Problem 2

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

In class, we considered standard linear regression where we sought parameters $w \in \mathbb{R}^d$ so that we minimized the following loss

$$L(w) = \frac{1}{2}\|y - Xw\|_2^2.$$

As we know, from the very first homework problem,

$$-\frac{\partial L}{\partial w_i} = x_i^\top (y - Xw) \cdot \frac{\partial w_i}{\partial w_i}$$

where x_i is the ith column of X. We argued in class that following this gradient from an initialization close to 0 gave w with minimum ℓ_2 norm.

In this homework problem, you'll get to see the dynamic in action. Of course, we'll need a small example to visualize it. Consider $y \in \mathbb{R}$, $w \in \mathbb{R}^2$, and $X \in \mathbb{R}^{1\times 2}$. Do check that all the operations in the loss make sense with these dimensions.

Now let's plot what's going on.

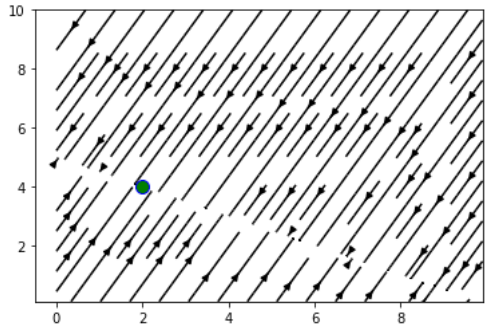
We first write a function to compute the gradient.

```
def compute_grad(w1, w2, X, y):
  w = np.array([[w1], [w2]])
  return X.T @ (y-X @ w) # this is the equation above in vector form
```

Now we initialize y and X. We'll visualize the parameter space of w by considering w_1 on the horizontal access and w_2 on the vertical access.

For each point (w_1, w_2), we'll use our handy `compute_grad` function to compute the gradient. Then we'll plot it!

```
y = 20
X = np.array([[2, 4]])
w = X.T * (1/(X @ X.T)) * y

W1, W2, W1_grad, W2_grad = np.zeros((100,100)), np.zeros((100,100)), np.zeros((100,100)), np.zeros((100,100))

for i in range(100):
  for j in range(100):
    w1 = j / 10
    w2 = (100-i) / 10
    W1[i,j], W2[i,j] = w1, w2
    W1_grad[i,j], W2_grad[i,j] = compute_grad(w1, w2, X, y)

plt.streamplot(W1, W2, W1_grad, W2_grad, color='k', density=1)
plt.plot(w[0], w[1], marker="o", markersize=10, markeredgecolor="blue", markerfacecolor="green")
plt.show()
```



So the set of w that achieve 0 loss forms a line. This is slightly difficult to see but the gradient arrows point directly to it.

Part 1

In class, we argued that the solution to the loss function with minimum ℓ_2 norm is

$$w = X^\top (XX^\top)^{-1} y.$$

Compute this w with the y and X defined above and draw it on the plot. Comment on what initialization of the weights will allow gradient descent to take us to this point.

Hint: In our case, $XX^\top$ is a scalar (single number, not a matrix) so its inverse is simply $1/XX^\top$.

Part 2

In class, we discussed another parameterization of this problem. We called it a neural network with parallel layers. The loss is

$$L(w) = \frac{1}{2}\|y - X(w \odot w)\|_2^2$$

where $w \odot w$ denotes entry-wise multiplication between w and w.

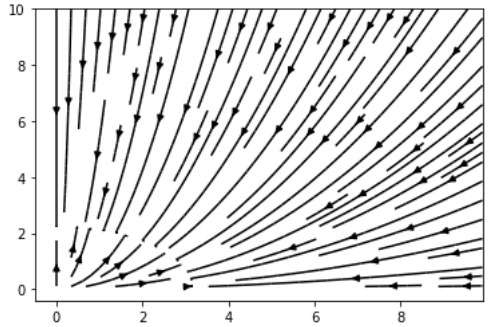
Then we have

$$-\frac{\partial L}{\partial w_i} = x_i^\top (y - X(w \odot w)) \frac{\partial w_i^2}{\partial w_i} = x_i^\top (y - X(w \odot w)) 2w_i.$$

I told you in class that following this gradient with an initialization close to 0 would give w with minimum ℓ_1 norm.

Using the code above with a slight modification, visualize the gradient at each point. Comment on whether the figure backs up my claim from class.

```
def compute_gradient(w1, w2, X, y):
  w = np.array([[w1], [w2]])
  #w_entwise = np.array([[w1], [w1]])

  return X.T @ (y-X @ np.square(w)) * 2*w


y = 20
X = np.array([[2, 4]])
w = X.T * (1/(X @ X.T)) * y

W1, W2, W1_grad, W2_grad = np.zeros((100,100)), np.zeros((100,100)), np.zeros((100,100)), np.zeros((100,100))

for i in range(100):
  for j in range(100):
    w1 = j / 10
    w2 = (100-i) / 10
    W1[i,j], W2[i,j] = w1, w2
    W1_grad[i,j], W2_grad[i,j] = compute_gradient(w1, w2, X, y)

plt.streamplot(W1, W2, W1_grad, W2_grad, color='k', density=1)
plt.show()
```



✓ 1s completed at 4:37 PM