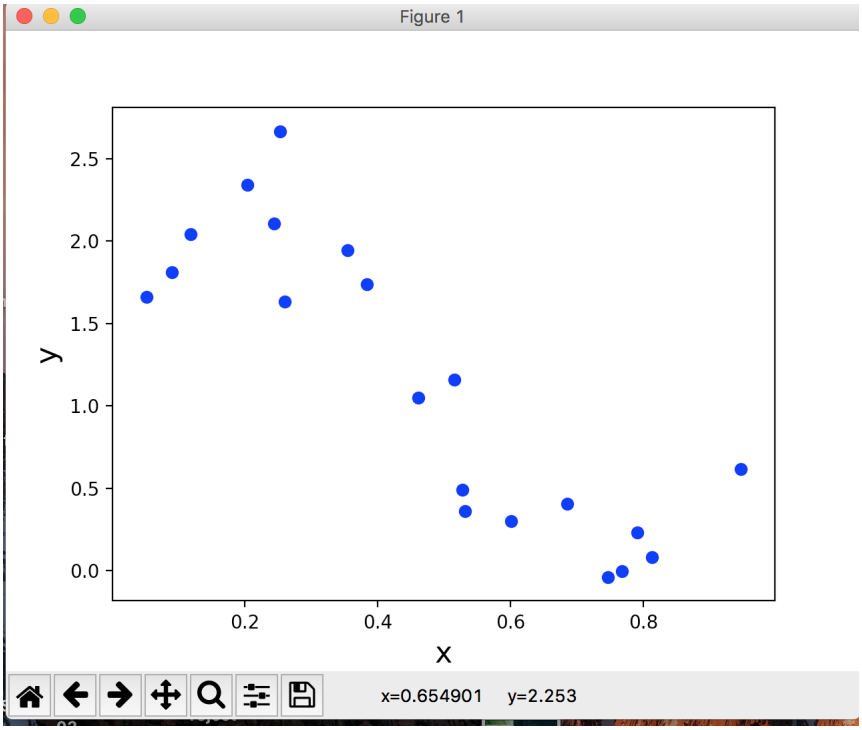
CS188, Winter 2017
Problem Set 2: programming
Due 2/16,2017

1 Problem 4



(a) Problem 4a

Solution: In general, we see that the data does appear roughly linear, but with a noticeable amount of noise. For the most part, we see that y decreases with x, but there are exceptions, such as y increasing with x for x in teh range of 0 to 0.3. From the very limited data that we have, I would guess that a linear regression could be appropriate since the data appears roughly linear. A polynomial regression may also be appropriate since the data resemble a curved plot.

(b) Problem 4b

Solution: I changed the code to do the tasks.

(c) Problem 4c **Solution:** I changed the code to do the tasks.

(d) Problem 4d
Cost refers to the final value of the obj. func

```
eta: 0.0001 | coefficient: [ 1.91573585 -1.74358989] | cost: 5.49356558874 | num_iterations: 10000 | time: 0.215529
eta: 0.001 | coefficient: [ 2.4463815  -2.81630184] | cost: 3.91257640947 | num_iterations: 10000 | time: 0.206215
eta: 0.01 | coefficient: [ 2.44640698 -2.81635335] | cost: 3.91257640579 | num_iterations: 1480 | time: 0.030502
eta: 0.0407 | coefficient: [ 2.44640705 -2.8163535 ] | cost: 3.91257640579 | num_iterations: 378 | time: 0.009235
```

Solution: The coefficients are almost exactly the same (up to the 3rd decimal place), besides the ones generated by the smallest eta, which did not reach as low of a cost as the others had. For etas of 0.0001 and 0.001, the algorithm iterated for the same number of times (the maximum, 10k), but for the eta of 0.0001, we didn't get as low of a cost. Next, the eta of 0.01 took 1480 iterations (about 10x less than the previous etas), while the eta of 0.0407 took 378 iterations, so it converged much quicker. I used python's time.clock() to also time each function.

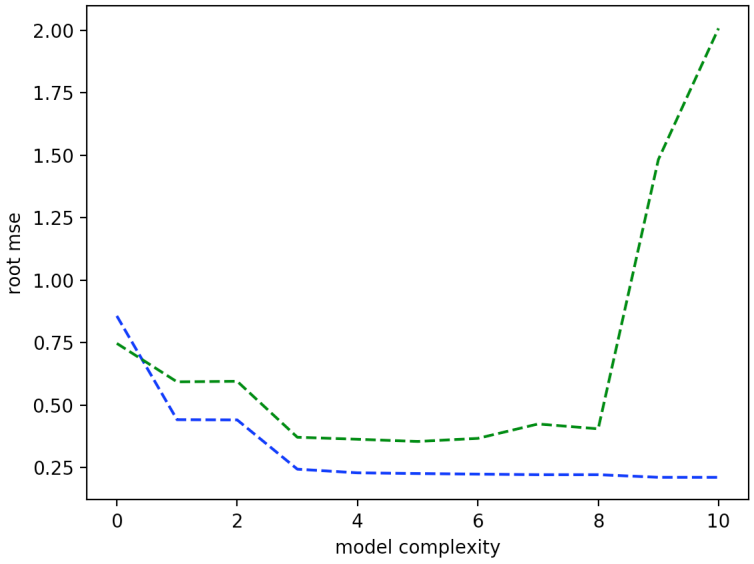
(e) Problem 4e
Solution: the closed form solution is: [2.44640709 -2.81635359] the coefficients are almost exactly the same as what we got when we did GD with 0.001. 0.01, and 0.0407 etas, but different from the 0.0001 eta.
for the closed form solution, the cost was 3.91257640579, which is almost exactly what we got we did GD with 0.001. 0.01, and 0.0407 etas, but different from the 0.0001 eta.
Since there's no sense of iteration in the closed form solution algorithm, I used python's time.clock() to get a sense of the running speed. It ran in 0.000271 on my computer. The fastest GD ran in 0.009 seconds, so this closed form solution is about 45x faster. I think this is because the dataset was relatively small and numpy has very efficient matrix operations implemented in optimized C. For very large datasets, gradient descent may be more appropriate.

(f) problem 4f **Solution:** With the learning rate that's a function of the iterations, the fit gd algorithm took 0.227652 seconds to converge, but it iterated until the limited (10k).
The full data for the learning rate as a function is:
iters: 10000,time: 0.227652,cost: 3.91257642432 decay coefs: [2.44634965 -2.81623746]

(g) problem 4g **Solution:** i wrote the code for this.

(h) problem 4h **Solution:** RMSE, or root mean squared error, measures how good the fit is, while J measures how strong the correlation is. So since RMSE indicates how good our fit is, this may be a preferable metric because we want to know how well our learned parameters are fitting to the dataset, this can be more indicative of a model's quality than knowing about correlation.

(i) problem 4i



The green plot indicates test error, the blue plot indicates training

error (not sure why matplotlib didn't show it). The best model complexity is a degree 8 polynomial. There is overfitting after degree 8 since training error continues to decrease but test error goes up quickly.

2 Problem 2

Solution: Solution to problem 2