Jason Teng
January 26, 2018
Programming Assignment 1

1. Implement Gradient Descent

I used three different functions to test my gradient descent procedure:

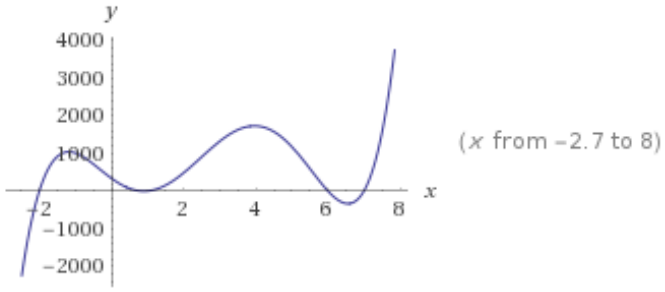
$$f(x, y) = 2x^2 + 4y^2$$
$$f(x) = -5 + x^2$$
$$f(x) = 5 + (x + 2)(x - 1)(5x - 4)(x - 6)(x - 7)$$

For the first two functions, since they are convex functions with exactly one global minimum, the starting guess does not matter much except to change the number of iterations until the solution is found. For the third function, which has the shape as shown in figure 1 (courtesy of Wolfram Alpha), it is a non-convex function with two local minima. Thus, a starting guess that is too low will result in the procedure entering an endless loop, and a step size that is too large can also cause the algorithm to jump to that side of the function. Smaller step sizes and/or more strict convergence criteria will result in requiring more iterations before the procedure terminates.

My implementation of calculating the numerical gradient is analogous to partial derivatives, and I used the identity matrix to streamline the code a bit. Testing my code, the results are extremely close to those found by the analytic gradient.

When comparing to scipy.optimize.fmin_bfgs, my code has similar iteration counts. In fact, I believe that the difference in interation counts (6 for mine vs. 5 for scipy) is due to having stricter step_size/convergence thresholds, as the answer given by my procedures is more precise than that given by scipy. A readout of my functions run against scipy is below, with the third function, and initial guess of x=0. The first line is generated using the analytic gradient, the second line through the numeric gradient, and the remaining lines are printed by the scipy optimizer.

```
([0.8999223107137192], 6)
([0.89992231071800433], 6)
Optimization terminated successfully.
         Current function value: 0.489047
         Iterations: 5
         Function evaluations: 18
         Gradient evaluations: 6
[ 0.89992414]
```



(x from −2.7 to 8)

2. Linear Basis Function Regression

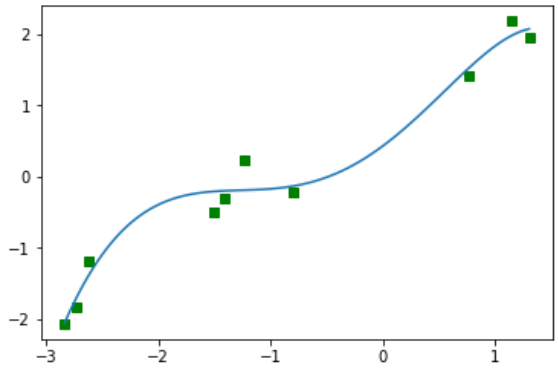
Lines 87-93 in homework1.py generate the graphs, which are very close to the graphs given in the appendix of the assignment. Visually inspecting the weight parameters also confirms similar weight values.

3. Ridge Regression

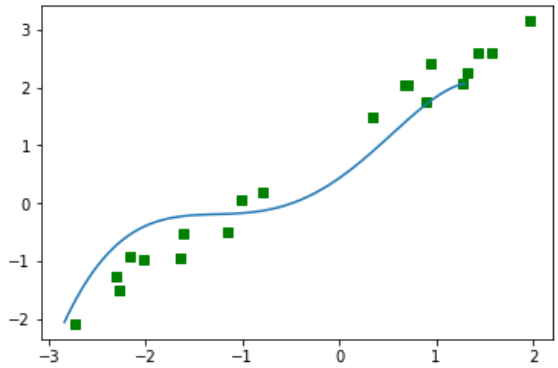
When testing various values of M and lambda, I found that raising lambda would flatten the curve dramatically, even if M was made to be very large.

Below are some examples of testing different values of lambda and M on the test data, and checking the result against the validation data.

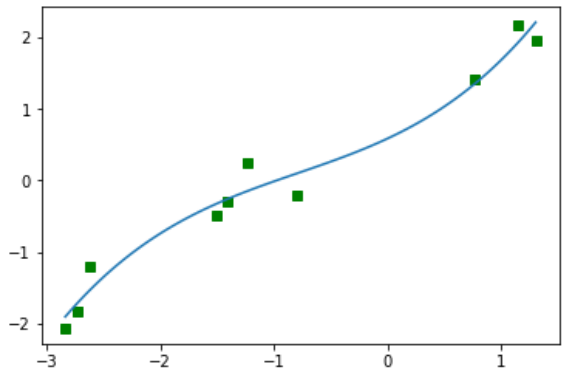
With M = 5 and lambda = 0.001:
Training Data A:



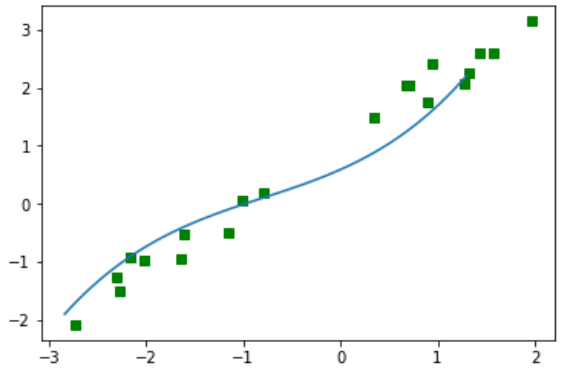
Validation Data:



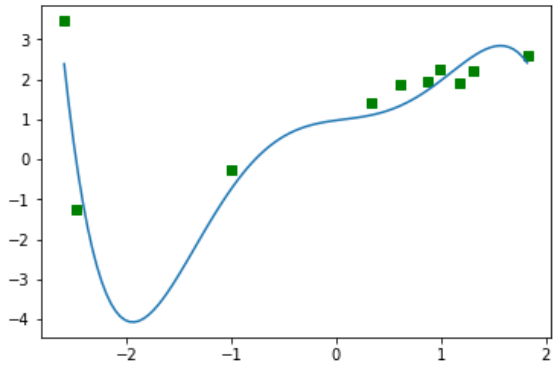
With M = 3 and lambda = 0.001
Training Data A:



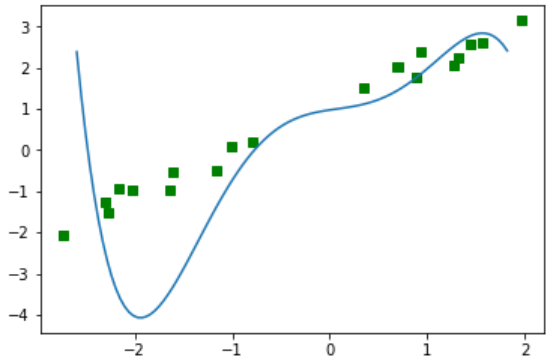
Validation Data:



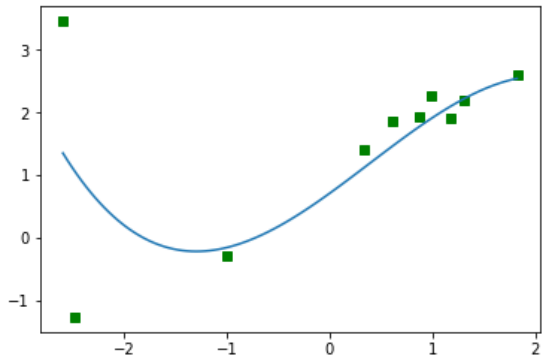
With M = 5 and lambda = 1:
Training Data B:



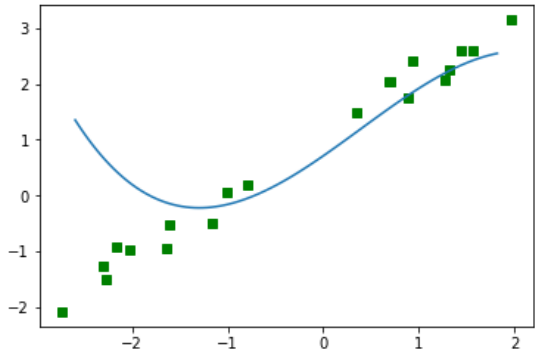
Validation Data:



With M = 3 and lambda = 1:
Training Data B:



Validation Data:



Based on these results, choosing a lower M and a higher lambda seem to prevent overfitting. Also, it appears that training set A overall better models the validation data.