EE 1146
Prof. Dolecek
Dis 1B

Joshua Liu
10513603I

Homework 2

① $X^T X w = X^T y, \quad X \in \mathbb{R}^{N \times m}, \quad y \in \mathbb{R}^N$

Prove: The Gram matrix $X^T X$ is nonsingular if and only if X has linearly independent columns.

X has linearly indep. columns if $Ax = 0$ only for $x = 0$.

$Ax = 0$ has only the trivial solution $x = 0$ iff A is nonsingular.

If A is singular, there exists $z \neq 0$ s.t. $Az = 0$.

If X has linearly indep. columns, then by definition it only has the trivial solution, where $Xy = 0$ only if $y = 0$.

Let z be a vector.

$X^T X z = 0$ ← show that z can only be zero.

$$z^T X^T X z = (z^T X^T) X z$$
$$= (Xz)^T Xz$$
$$= \|Xz\|^2$$
$$= 0$$

then $Xz = 0$.

Since the columns of X are lin. indep,

$z = 0$.

Thus, $X^T X$ is nonsingular. ■.

If the columns of X are linearly dependent, then

$Xz = 0$ for some vector $z \neq 0$.

then $X^T X z = 0$ where $z \neq 0$,

implying that $X^T X$ is thus singular if its columns are linearly dependent. As a result, $X^T X$ is nonsingular if and only if X has linearly independent columns.

(2) $H = X(X^TX)^{-1}X^T$, $X \in \mathbb{R}^{N\times M}$ and X^TX is invertible.

(a) Show that H is symmetric.

$H \stackrel{?}{=} H^T$

$H^T = [X(X^TX)^{-1}X^T]^T$

$= [(X^TX)^{-1}X^T]^T X^T$

$= X[(X^TX)^{-1}]^T X^T$

$= X(X^TX)^{-1}X^T$

$H^T = H$ ∎.

(b) Show that $H^K = H$ for any positive integer K.

$k=2$: $H^2 = X(X^TX)^{-1}X^TX(X^TX)^{-1}X^T$

$= X(X^TX)^{-1}X^T$ → I

$= H$.

Suppose that $H^{k-1} = H$. Then, $H^k = H^{k-1}H = H\cdot H = H$.

By induction, $H^K = H$ for any positive integer K.

(c) Show that $(I-H)^K = I-H$ for any positive integer K.

$k=2$: $(I-H)^2 = I^2 - 2H + H^2$

$= I - 2H + H$ from part B

$= I - H$

Suppose that $(I-H)^{k-1} = (I-H)$. Then, $(I-H)^k = (I-H)^{k-1}(I-H) = (I-H)(I-H) = I-H$

By induction, $(I-H)^k = (I-H)$ for any positive integer K.

(d) Show that $tr(H) = M$. Trace (AB) = Trace (BA)

$tr(H) = tr(X(X^TX)^{-1}X^T)$ $X \in \mathbb{R}^{N\times M}$

$= tr(X^TX(X^TX)^{-1})$ $(M\times N)\times(N\times M) = (M\times M)$

$= tr(I_M)$

$= M$ ∎.

(3) $J(w_0, w_1) = \sum_{n=1}^{N} \alpha_n (w_0 + w_1 x_{n,1} - y_n)^2, \quad \alpha_n > 0.$

$$\frac{\partial}{\partial w_0} J(w_0, w_1) = \sum_{n=1}^{N} 2\alpha_n (w_0 + w_1 x_{n,1} - y_n)$$

$$\frac{\partial}{\partial w_1} J(w_0, w_1) = \sum_{n=1}^{N} 2\alpha_n x_{n,1} (w_0 + w_1 x_{n,1} - y_n)$$

$$\nabla_w J(w_0, w_1) = \begin{bmatrix} \sum_{n=1}^{N} 2\alpha_n (w_0 + w_1 x_{n,1} - y_n) \\ \sum_{n=1}^{N} 2\alpha_n x_{n,1} (w_0 + w_1 x_{n,1} - y_n) \end{bmatrix}$$

If $\alpha_i = 0$ for some i, then the data (x_i, y_i) will have no impact to the gradient, since the whole $(w_0 + w_1 x_{i,1} - y_i)$ term will be zero.

If α_j is much greater than $\alpha_{j'}$, $j' \neq j$, then this α_j term will be amplified, where a small value of $w_0 + w_1 x_{j,1} - y_j$ will result in a large step. In other words, data (x_j, y_j) will have a larger impact since α_j, the coefficient, is larger, and thus overall tend towards and converges to the line through (x_j, y_j).

As a result, the values α_n affect how strongly a point (x_n, y_n) is factored into the gradient descent algorithm.

④ (a) $\frac{\partial J(w)}{\partial w}$ for $J(w) = -\sum_{i \in \mathcal{M}} w^T x_i y_i$, $w \in \mathbb{R}^N$, $x_i \in \mathbb{R}^N$, $i = 1, \dots, M$, $y_i \in \{-1, 1\}$

misclassified points → $\mathcal{M} = \{ i \mid \text{sign}(w^T x_i) \neq y_i \}$. Assume $w^T x_i \neq 0, \forall i$

$$\frac{\partial J(w)}{\partial w} = -\sum_{i \in \mathcal{M}} x_i y_i$$

(b) $\frac{\partial J(w)}{\partial w}$ for $J(w) = \sum_{i=1}^{M} \max [0, -w^T x_i y_i]$, $w \in \mathbb{R}^N$, $x_i \in \mathbb{R}^N$, $y_i \in \{-1, 1\}$, $w^T x_i \neq 0, \forall i$

$$\frac{\partial J(w)}{\partial w} = \sum_{i=1}^{M} \max [0, -x_i y_i]$$

$$= -\sum_{i \in \mathcal{M}} x_i y_i$$

(c) The answers are equivalent, since they both sum up the product $x_i y_i$ of the misclassified points. (a) is clear since it sums up the points in $\mathcal{M}$, the set of misclassified points. In (b), we are summing up values of $-x_i y_i$ that are positive, which is another way of saying misclassified points ($\text{sign}(x_i) \neq y_i$, so product is negative, and negative sign makes it positive, and we sum up the positive values due to the max function).

Using SGD, we find that

$$w_{t+1} = w_t - \eta (-x_i y_i)$$

and when $\eta = 1$:

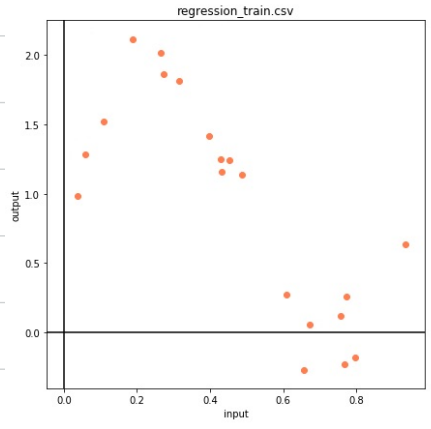
$$w_{t+1} = w_t - (-x_i y_i)$$

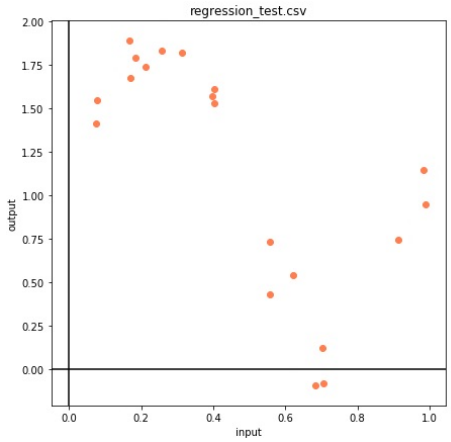
$$w_{t+1} = w_t + x_i y_i$$

which matches the perceptron algorithm update

$$w^{(k+1)} = w^{(k)} + x_i y_i$$

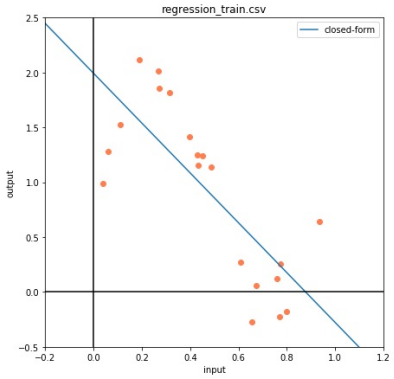
(5) (a) The plots look pretty similar overall, which is expected since one is the training data and the other is testing based on the training data. The points have a negative relationship, but this relationship isn't too strong due to the points having a curve towards the left and right. Linear regression will work, but a polynomial regression will fit much better.





(b)

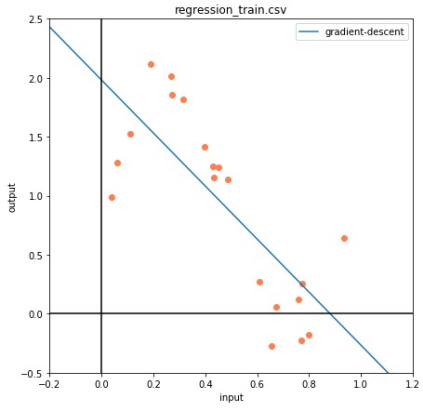
```
w: [1.9919616255921748, -2.270483716936368]
J(w): 4.603634906406957
```



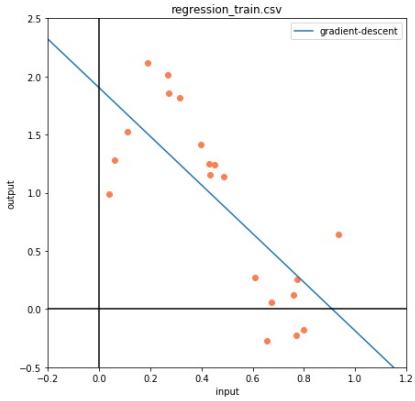
* loss is result of objective function J(w).

(c)

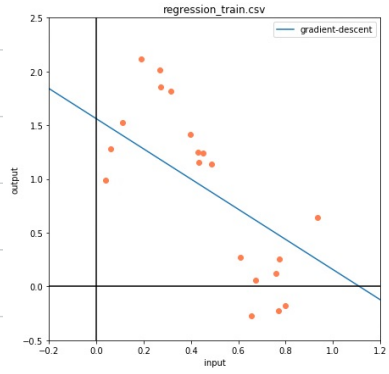
learning rate: 0.05
number of iterations: 83
loss: 4.6044056136703855
w: [1.9801678867515258, -2.2468145728931086]



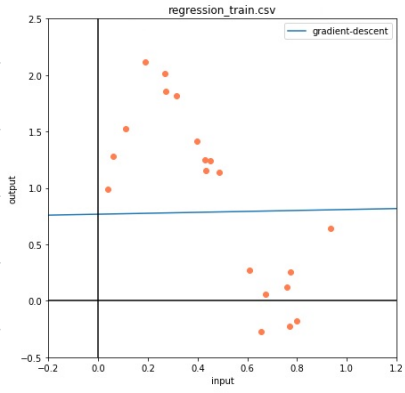
learning rate: 0.001
number of iterations: 2420
loss: 4.648852786068523
w: [1.9016254132187878, -2.089185754358247]



learning rate: 0.0001
number of iterations: 10000
loss: 5.640772178144737
w: [1.5593233872472827, -1.402211414149804]



learning rate: 1e-05
number of iterations: 10000
loss: 12.249367268440261
w: [0.7670318619787567, 0.04168682986585101]

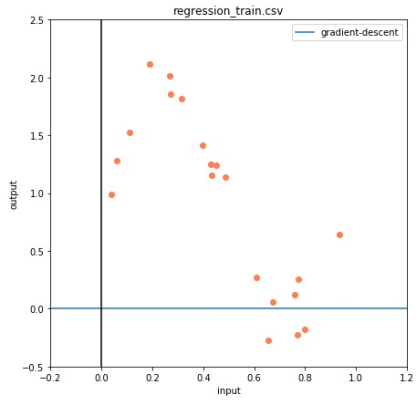


As the learning rate decreases, the number of iterations increases until it reaches our maximum of 10,000. For $\eta = 0.05$ and 0.001, the algorithm converges before 10,000, but the others do not, showing that it is "learning" slower, depicted by the prediction functions shown (this is because the weights are updated with a smaller change given a smaller η).

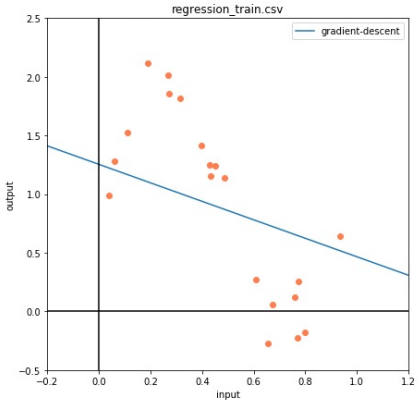
n (learning rate)	0.05	0.001	0.0001	0.00001
W_0	1.980168	1.901625	1.559323	0.767032
W_1	-2.246815	-2.089186	-1.402211	0.041687
Iterations	83	2420	10000	10000
J(w)	4.604406	4.6488532	5.640772	12.249367

(d) * loss is result of objective function J(w)

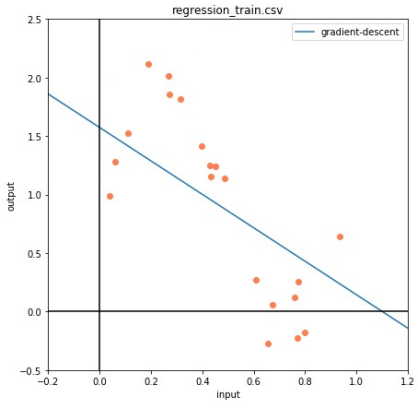
learning rate: 0.05
number of iterations: 0
loss: 28.655524112125004
w: [0, 0]



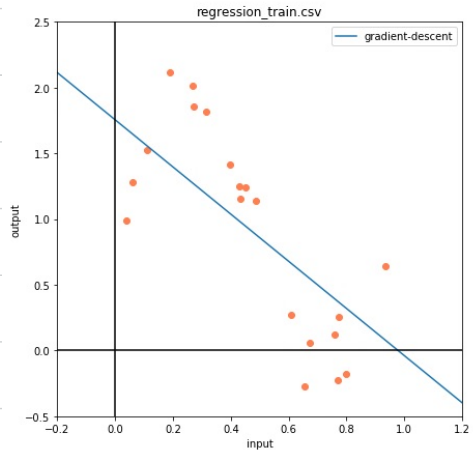
learning rate: 0.05
number of iterations: 10
loss: 7.629319845754809
w: [1.2530051879906214, -0.7874547713071378]



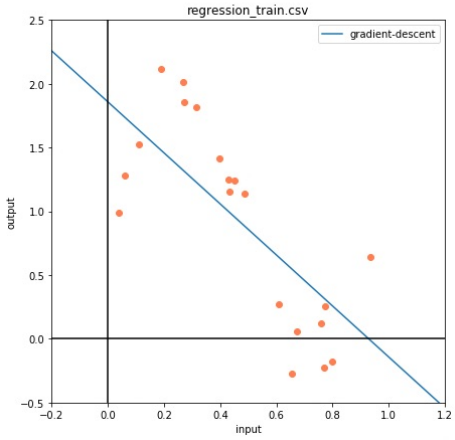
learning rate: 0.05
number of iterations: 20
loss: 5.5775115336820615
w: [1.5727254484963118, -1.4291083398913116]



```
learning rate: 0.05
number of iterations: 30
loss: 4.917096388407238
w: [1.7541139717364527, -1.7931414206953504]
```



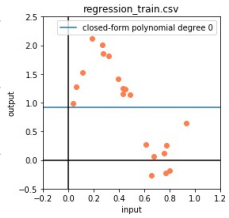

```
learning rate: 0.05
number of iterations: 40
loss: 4.7045286931001815
w: [1.8570221582099042, -1.9996703867663537]
```



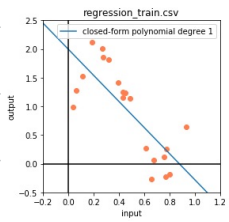
We notice that as the number of iterations increase, J(w) decreases. As we saw in (b), a learning rate of 0.05 converges around 83 iterations, and in the diagrams above, we can see the line quickly approaching the line in (b) after 83 iterations. Also, we can notice that a learning rate of 0.05 after 20 iterations has a J(w) close to a learning rate of 0.0001 after 10,000 iterations, so we can see a contrast here.

(d)

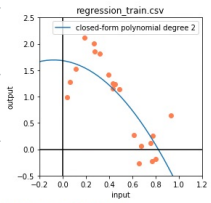
```
0 degree polynomial
w: [[0.92297725]]
J(w): 11.617784031773748
E_rms: 0.7621608764484618
```



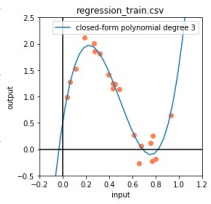

```
1 degree polynomial
w: [[ 1.99196163]
 [-2.27048372]]
J(w): 4.603634906406957
E_rms: 0.47977259750880713
```



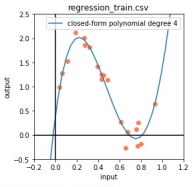

```
2 degree polynomial
w: [[ 1.6816842 ]
 [-0.33324919]
 [-2.07734139]]
J(w): 4.248652349320661
E_rms: 0.460904130450176 25
```



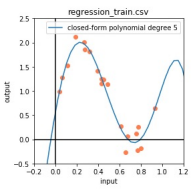

```
3 degree polynomial
w: [[  0.54820174]
 [ 14.03353094]
 [-40.37363261]
 [ 27.34103421]]
J(w): 0.4609092769957789
E_rms: 0.1518073247567091
```



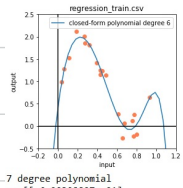

```
4 degree polynomial
w: [[  0.42743584]
 [ 16.40010412]
 [-50.88915257]
 [ 43.73252946]
 [ -8.30664945]]
J(w): 0.44207209888513116
E_rms: 0.14867281171840585
```



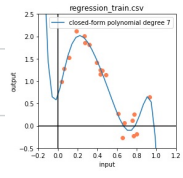

```
5 degree polynomial
w: [[  0.58823909]
 [ 12.14323138]
 [-21.99918653]
 [-33.94214576]
 [ 81.15037714]
 [-36.85890533]]
J(w): 0.4279047306120238
E_rms: 0.14627110627393639
```




```
6 degree polynomial
w: [[  0.44287806]
 [ 16.82527465]
 [ -65.12869574]
 [ 135.8313542 ]
 [-242.12889404]
 [ 256.83536528]
 [-101.9357041 ]]
J(w): 0.4230414214102506
E_rms: 0.14543751603528068
```

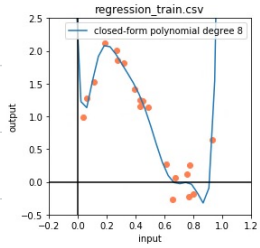



```
7 degree polynomial
w: [[ 6.90308807e-01]
 [ 6.93030070e+00]
 [ 6.02805747e+01]
 [-5.73959540e+02]
 [ 1.81252920e+03]
 [-2.90299276e+03]
 [ 2.35112573e+03]
 [-7.55196752e+02]]
J(w): 0.4173738852191 62
E_rms: 0.14446000921001667
```

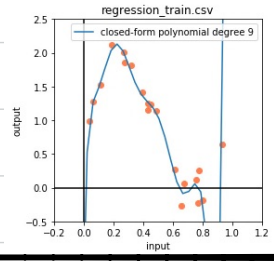


on train data

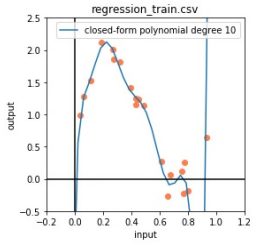
8 degree polynomial
w: [[1.82173713e+00]
 [-4.40542329e+01]
 [8.32033260e+02]
 [-6.03588459e+03]
 [2.26578403e+04]
 [-4.83838109e+04]
 [5.90340338e+04]
 [-3.82402031e+04]
 [1.01871451e+04]]
J(w): 0.3607800313421655
E_rms: 0.13430935025942264



9 degree polynomial
w: [[-6.55287706e-01]
 [8.31556823e+01]
 [-1.48627486e+03]
 [1.44768423e+04]
 [-7.83368910e+04]
 [2.46944021e+05]
 [-4.64924591e+05]
 [5.14451347e+05]
 [-3.08255149e+05]
 [7.70891552e+04]]
J(w): 0.24564218557751769
E_rms: 0.1108246781131165

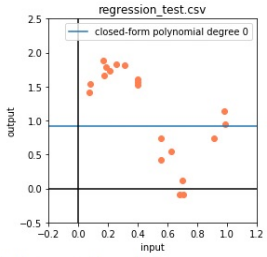


10 degree polynomial
w: [[-5.02007698e-01]
 [7.44444232e+01]
 [-1.30184042e+03]
 [1.25223512e+04]
 [-6.64961584e+04]
 [2.02976821e+05]
 [-3.61561975e+05]
 [3.60241215e+05]
 [-1.66894749e+05]
 [4.53981408e+03]
 [1.59448507e+04]]
J(w): 0.2455550578747174
E_rms: 0.11080399538706116

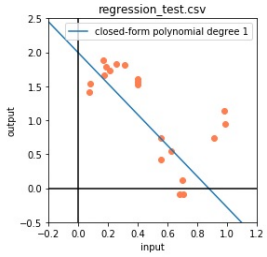


on train data (continued)

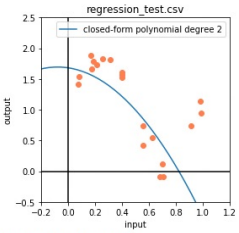
0 degree polynomial
w: [[0.92297725]]
J(w): 9.58573780830925
E_rms: 0.6923054892281748



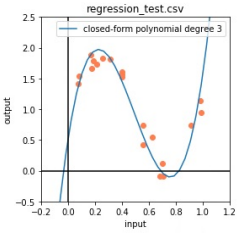
1 degree polynomial
w: [[1.99196163]
 [-2.27048372]]
J(w): 6.311234017725415
E_rms: 0.5617487880594587



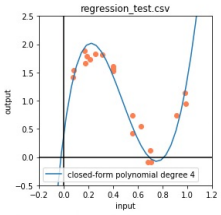
2 degree polynomial
w: [[1.6816842]
 [-0.33324919]
 [-2.07734139]]
J(w): 8.949024707521396
E_rms: 0.6689179586287617



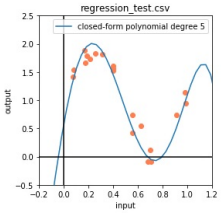
3 degree polynomial
w: [[0.54820174]
 [14.03353094]
 [-40.37363261]
 [27.34103421]]
J(w): 0.6465501347554546
E_rms: 0.17979851706221808



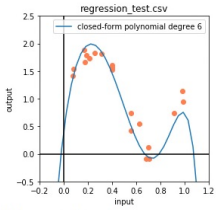
4 degree polynomial
w: [[0.42743584]
 [16.40010412]
 [-50.88915257]
 [43.73252946]
 [-8.30664945]]
J(w): 0.6356367708212918
E_rms: 0.1782746155263407



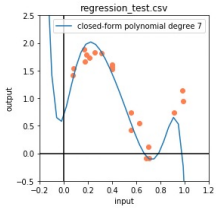
5 degree polynomial
w: [[0.58823909]
 [12.14323138]
 [-21.99918653]
 [-33.94214576]
 [81.15037714]
 [-36.85890533]]
J(w): 0.5812951929063455
E_rms: 0.17048389849284087



6 degree polynomial
w: [[0.44287806]
 [16.82527465]
 [-65.12869574]
 [135.8313542]
 [-242.12889404]
 [256.83536528]
 [-101.9357041]]
J(w): 0.7131498838197792
E_rms: 0.1888319204768859

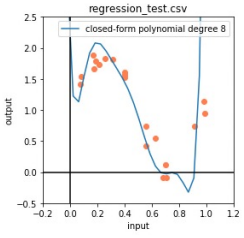


7 degree polynomial
w: [[6.90308807e-01]
 [6.93030070e+00]
 [6.02805747e+01]
 [-5.73959540e+02]
 [1.81252920e+03]
 [-2.90299276e+03]
 [2.35112573e+03]
 [-7.55196752e+02]]
J(w): 3.224907655005114
E_rms: 0.40155371091580727

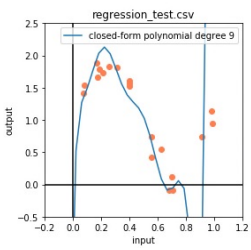


on test data

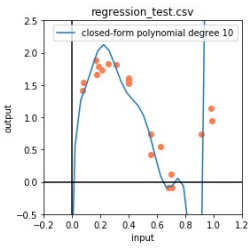
8 degree polynomial
w: [[1.82173713e+00]
 [-4.40542329e+01]
 [8.32033260e+02]
 [-6.03588459e+03]
 [2.26578403e+04]
 [-4.83838109e+04]
 [5.90340338e+04]
 [-3.82402031e+04]
 [1.01871451e+04]]
J(w): 48.35874292393426
E_rms: 1.5549717509320589

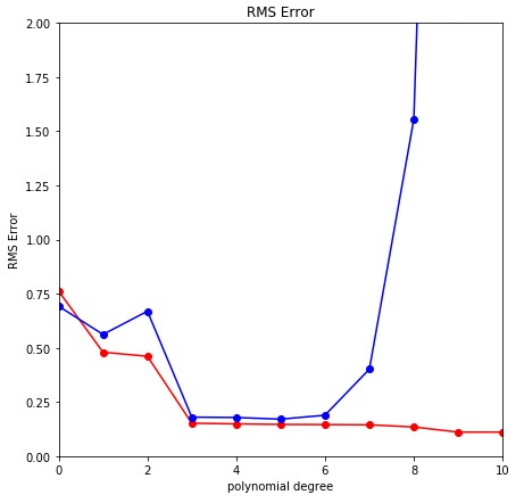


9 degree polynomial
w: [[-6.55287706e-01]
 [8.31556823e+01]
 [-1.48627486e+03]
 [1.44768423e+04]
 [-7.83368910e+04]
 [2.46944021e+05]
 [-4.64924591e+05]
 [5.14451347e+05]
 [-3.08255149e+05]
 [7.70891552e+04]]
J(w): 1182.0411074242536
E_rms: 7.687786116380494

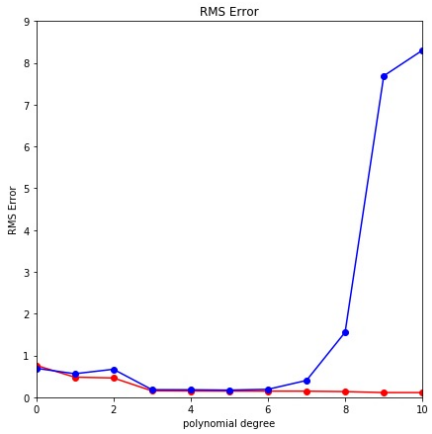


10 degree polynomial
w: [[-5.02007698e-01]
 [7.44444232e+01]
 [-1.30184042e+03]
 [1.25223512e+04]
 [-6.64961584e+04]
 [2.02976821e+05]
 [-3.61561975e+05]
 [3.60241215e+05]
 [-1.66894749e+05]
 [4.53981408e+03]
 [1.59448507e+04]]
J(w): 1377.6392182444129
E_rms: 8.29951570347455





larger scope →



m	Training E_{RMS}	Testing E_{RMS}
0	0.762160876	0.692305489
1	0.479772598	0.561748788
2	0.46090413	0.668917959
3	0.151807325	0.179798517
4	0.148672812	0.178274616
5	0.146271106	0.170483898
6	0.145437516	0.18883192
7	0.144460009	0.401553711
8	0.13430935	1.554971751
9	0.110824678	7.687786116
10	0.110803995	8.299515703

too good

bad

A polynomial of $m = 3$ to 6 seems to best fit the training and testing data. We can see that overfitting begins happening after $m=6$, indicated by the great performance on the training data and the terrible results on the testing data. We can also see in the graphs above that after $m=6$, the function generation seems to try and reach all the training points, and this is an indication of overfitting since we want the model to learn the patterns and not memorize what it is given in order to predict unseen data. The slightly higher values for $m = 1$ and 2 indicate that the model does not perform optimally when fit to a 1 or 2 degree polynomial due to the general shape/location of the training and testing graphs/points.