1] Prove that the Gram matrix X^TX is nonsingular iff X has linearly independent columns.

Zach Berger

($\Rightarrow$) Suppose X^TX is nonsingular.
Then, the equation $X^TXx = 0$ has only the trivial solution $x = 0$.
Suppose $\exists\ x \neq 0 \in \mathbb{R}^n$ s.t. $Xx = 0$.
This would imply $X^T(Xx) = X^T(0) = 0$ for $x \neq 0$.
But, this contradicts the fact that $X^TXx = 0$ only for $x = 0$.
$\therefore Xx = 0$ only for $x = 0$.
$\Rightarrow X$ has linearly columns.

($\Leftarrow$) Suppose that X has linearly independent columns.
Then, $Xx = 0$ only for $x = 0$, so $\ker X = \{0\}$.
Claim: the kernel of X, X^TX, are equivalent.*
$\Rightarrow \ker(X^TX) = \{0\}$
$\Rightarrow X^TX$ is invertible
$\Rightarrow X^TX$ is nonsingular

□

* Proof of claim.
Let x be some arbitrary vector in $\ker(X^TX)$.

$$x \in \ker(X^TX) \Rightarrow X^TXx = 0 \Rightarrow x^TX^TXx = 0 \Rightarrow (Xx)^T(Xx) = 0$$
$$\Rightarrow \|Xx\|^2 = 0 \Rightarrow Xx = 0 \Rightarrow x \in \ker(X)$$

$\therefore \ker(X^TX) \subseteq \ker(X)$.

Now let x be some arbitrary vector in $\ker(X)$.

$$x \in \ker(X) \Rightarrow Xx = 0 \Rightarrow \|Xx\|^2 = 0 \Rightarrow (Xx)^TXx = 0$$
$$\Rightarrow x^TX^TXx = 0 \Rightarrow X^TXx = 0 \Rightarrow x \in \ker(X^TX)$$

$\therefore \ker(X) \subseteq \ker(X^TX)$.
$\Rightarrow \ker(X) = \ker(X^TX)$

□

2| Consider the hat matrix $H = X(X^TX)^{-1}X^T$ where $X \in \mathbb{R}^{N \times M}$, X^TX is invertible.

a) Show that H is symmetric.

We must show that $H^T = H$ in order to prove the claim. Recall that H is square $(N \times N)$ so this is indeed possible.

$$H^T = \left(X(X^TX)^{-1}X^T\right)^T = (X^T)^T\left((X^TX)^{-1}\right)^T X^T$$
$$= X\left((X^TX)^T\right)^{-1}X^T$$
$$= X\left(X^T(X^T)^T\right)^{-1}X^T$$
$$= X(X^TX)^{-1}X^T = H$$

□

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

We proceed by induction on K.

Base cases ($K=1$, $K=2$): Obviously $H^1 = H$ is true ✓

Now, $H^2 = \left(X(X^TX)^{-1}X^T\right)\left(X(X^TX)^{-1}X^T\right)$
$$= X\left[(X^TX)^{-1}(X^TX)\right](X^TX)^{-1}X^T$$
$$= X\,I\,(X^TX)^{-1}X^T = X(X^TX)^{-1}X^T$$
$$= H \quad \checkmark$$

Induction step: Suppose $H^K = H$ is true.

Now, show that $H^K = H$ implies $H^{K+1} = H$ is true.

$$H^{K+1} = H^KH = (H)H \quad \text{by inductive hypothesis}$$
$$= H^2$$
$$= H \quad \checkmark$$

Hence by the principle of mathematical induction, $H^K = H \;\forall\; K \in \mathbb{Z}^+$.

□

c) If I denotes identity matrix size $N \times N$, show that $(I-H)^K = I-H$ for any positive integer K.

↳ Proceed by induction on K. Note that H is $N \times N$.

Base cases ($K=1, K=2$): The statement for $K=1$, $(I-H)^1 = I-H$ is obvious. ✓

For $K=2$, compute $(I-H)^2 = (I-H)(I-H) = II - IH - HI + HH$

$= I^2 - H - H + H^2$

$= I - 2H + H \quad (H^2 = H, I^2 = I)$

$= I - H$ ✓

Induction step: Suppose $(I-H)^K = (I-H)$ is true.

Show that $(I-H)^K = I-H$ implies $(I-H)^{K+1} = (I-H)$ is true.

$(I-H)^{K+1} = (I-H)^K (I-H)$

$= (I-H)(I-H)$ by inductive hypothesis

$= (I-H)^2 = (I-H)$ ✓

Hence by the principle of mathematical induction, $(I-H)^K = (I-H)$ $\forall K \in \mathbb{Z}^+$.

□

d) Show that $\text{Trace}(H) = M$.

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

$= \text{Trace}[X^T(X(X^TX)^{-1})]$ b/c $\text{Tr}(AB) = \text{Tr}(BA)$

↳ Note X^T is $M \times N$, X is $N \times M$, $(X^TX)^{-1}$ is $M \times M$

so dimensionally the product $X^TX(X^TX)^{-1}$ makes sense.

$= \text{Trace}[(X^TX)(X^TX)^{-1}]$

$= \text{Trace}[I]$ where $I \in \mathbb{R}^{M \times M}$

$= \sum_{i=1}^{M} 1 = M$

□

3] Consider the function $J(w_0, w_1) = \sum_{n=1}^{N} d_n (w_0 + w_1 x_{n,1} - y_n)^2$, $d_n > 0$. calculate $\nabla_w (J(w_0, w_1))$. Comment on d_n's effect on linear regression problem. Qualitatively describe what happens in gradient descent if $d_j >> d_{j'}$, $j \neq j'$.

↳ note, d_n amounts to a weight on the training examples.

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

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

$$\Rightarrow \nabla_w (J(w_0, w_1)) = \left\langle \frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1} \right\rangle = \left\langle \sum_{n=1}^{N} 2 d_n (w_0 + w_1 x_{n,1} - y_n), \sum_{n=1}^{N} 2 d_n (w_0 + w_1 x_{n,1} - y_n) x_{n,1} \right\rangle$$

$$= \sum_{n=1}^{N} 2 d_n \left\langle (w_0 + w_1 x_{n,1} - y_n), (w_0 + w_1 x_{n,1} - y_n) x_{n,1} \right\rangle$$

$$\boxed{\nabla_w (J(w_0, w_1)) = \sum_{n=1}^{N} 2 d_n \left\langle (w_0 + w_1 x_{n,1} - y_n), x_{n,1} (w_0 + w_1 x_{n,1} - y_n) \right\rangle}$$

Truly, d_n is a weight that adds a measure of importance to each training instance. A larger d_i for a training instance (x_i, y_i) means that training instance is more important, and smaller d_i indicates less importance.

↳ For example, if $d_i = 0$ for some i, the corresponding training instance (x_i, y_i) will not be counted in J, so will not affect the optimization of the model for w. This is b/c the penalty incurred by (x_i, y_i) will equal $0 (w_0 + w_1 x_{n,1} - y_n)^2 = 0$.

Hence, the minimum of the loss function will bias to minimize the error contributed by the traing instance with relatively large values of d_i.

If $d_j >> d_{j'}$, $j \neq j'$, then qualitatively speaking, gradient descent will yield a minimization result, w, that is greatly biased in minimizing the j^{th} traning instance. *visually, the regression line parameterized by w will get closer to the j^{th} training instance with larger d_j.

4. Develop Stochastic Gradient Descent view of perceptron...

a) Find the gradient $\frac{\partial J(w)}{\partial w}$ for the loss function $J(w) = -\sum_{i \in M} w^T x_i y_i$ where $w \in \mathbb{R}^N$, $x_i \in \mathbb{R}^N$, $i = 1, \ldots, M$, $y_i \in \{-1, 1\}$. $M = \{i \mid \text{sign}(w^T x_i) \neq y_i\}$. Assume $w^T x_i \neq 0$.

↳ Note that $\frac{\partial J}{\partial w_j} = \frac{\partial}{\partial w_j}\left(-\sum_{i \in M} w^T x_i y_i\right)$ for $1 \leq j \leq N$.

$$= -\sum_{i \in M} \frac{\partial}{\partial w_j}(w_{i1}x_{i1} + \cdots + w_{ij}x_{ij} + \cdots + w_{iN}x_{iN})y_i$$

$$= -\sum_{i \in M} x_{ij} y_i$$

Then, $\frac{\partial J}{\partial w} = \nabla_w(J(w)) = \langle -\sum_{i \in M} x_{i1} y_i, -\sum_{i \in M} x_{i2} y_i, \ldots, -\sum_{i \in M} x_{iN} y_i \rangle$

$$\Rightarrow \boxed{\nabla_w(J(w)) = -\sum_{i \in M} x_i y_i}$$

b) Find the gradient $\frac{\partial J(w)}{\partial w}$ for the loss function $J(w) = \sum_{i=1}^{M} \max[0, -w^T x_i y_i]$ where $w \in \mathbb{R}^N$, $x_i \in \mathbb{R}^N$, $y_i \in \{-1, +1\}$. Assume $w^T x_i \neq 0$.

↳ If a point x_i is misclassified, then $\text{sign}(w^T x_i) \neq y_i$.

$\Rightarrow -w^T x_i y_i > 0 \Rightarrow \max(0, -w^T x_i y_i) = -w^T x_i y_i$

If a point is correctly classified, then $\text{sign}(w^T x_i) = y_i$

$\Rightarrow -w^T x_i y_i < 0, \Rightarrow \max(0, -w^T x_i y_i) = 0$.

From part a, recall that $\frac{\partial}{\partial w}(-w^T x_i y_i) = -x_i y_i$

Accordingly, $\frac{\partial J(w)}{\partial w} = \frac{\partial}{\partial w}\left(\sum_{i=1}^{M} \max(0, -w^T x_i y_i)\right)$

$$= \sum_{i=1}^{M} \begin{cases} -x_i y_i, & x_i \text{ is improperly classified} \\ 0, & x_i \text{ is properly classified} \end{cases}$$

$$\boxed{\nabla_w(J(w)) = \sum_{i=1}^{M} \begin{cases} -x_i y_i, & x_i \text{ misclassified} \\ 0, & x_i \text{ correctly classified} \end{cases}}$$

c) The answers in parts a and b are equivalent.
The answer in b can be recast as $\nabla_w(J(w)) = -\sum_{i \in M} x_i y_i$
where M = index set of misclassified points, illustrating this equivalence.

The update rule to minimize J, using SGD would be...
→ assuming x_i is normalized
→ assuming x_i is misclassified

$$w_{k+1} = w_k - \eta(-x_i y_i) \implies \boxed{\text{Update: } w_{k+1} = w_k + \eta x_i y_i}$$

With a learning rate of $\eta = 1$, the update becomes $w_{k+1} = w_k + x_i y_i$.
While this update is the same update as in the Perceptron algorithm, SGD and Perceptron are not equivalent.
This amounts to the difference in how the two algorithms iterate through the misclassified data points.

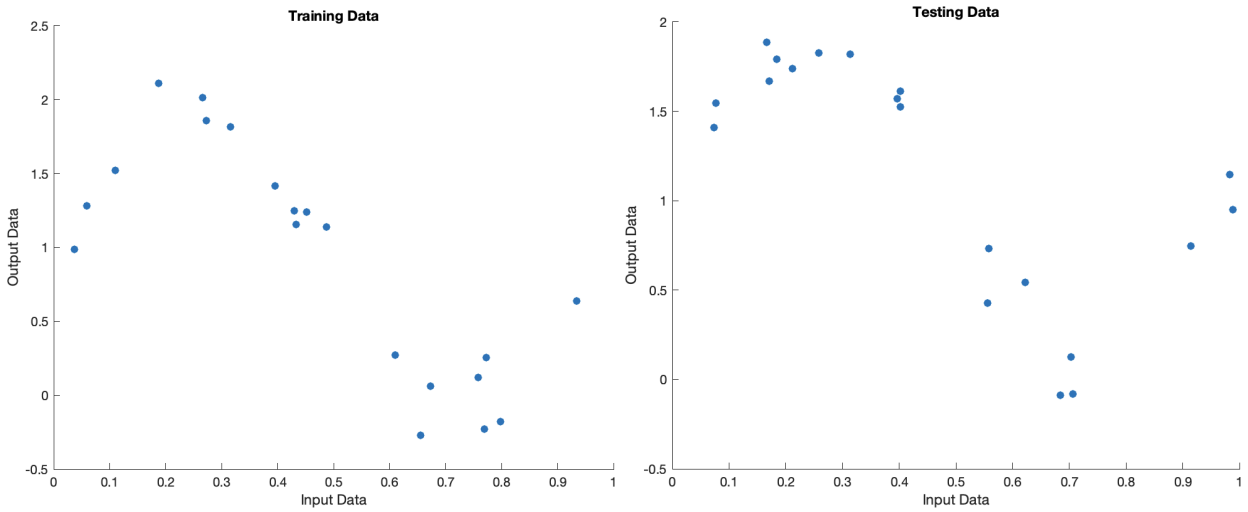
- Perceptron iterates through all of the data points in order over and over again, updating at each misclassified point until all points are correctly classified.
- SGD randomly selects a point from the set of misclassified and updates until all points are correctly classified.
  ⟹ SGD's ordering is more randomized than Perceptron.

Zack Berger

Question 5

Part A, Visualization

The following is a plot for both the training data and test data. It shows outputs plotted on the y-axis (dependent variable) and inputs plotted on the x-axis (independent variable).



I observe that the training data is roughly shaped like a line with a negative slope. The testing data follows a similar pattern of correlation between input and output. That said, the data is scattered more erratically in the testing set.

My guess is that due to the linear nature of the scattered data, linear regression should be a fairly effective predictor of the data. Based on the shape of the two sets, I expect there to be significantly more error in predicting the testing set over predicting the training set.

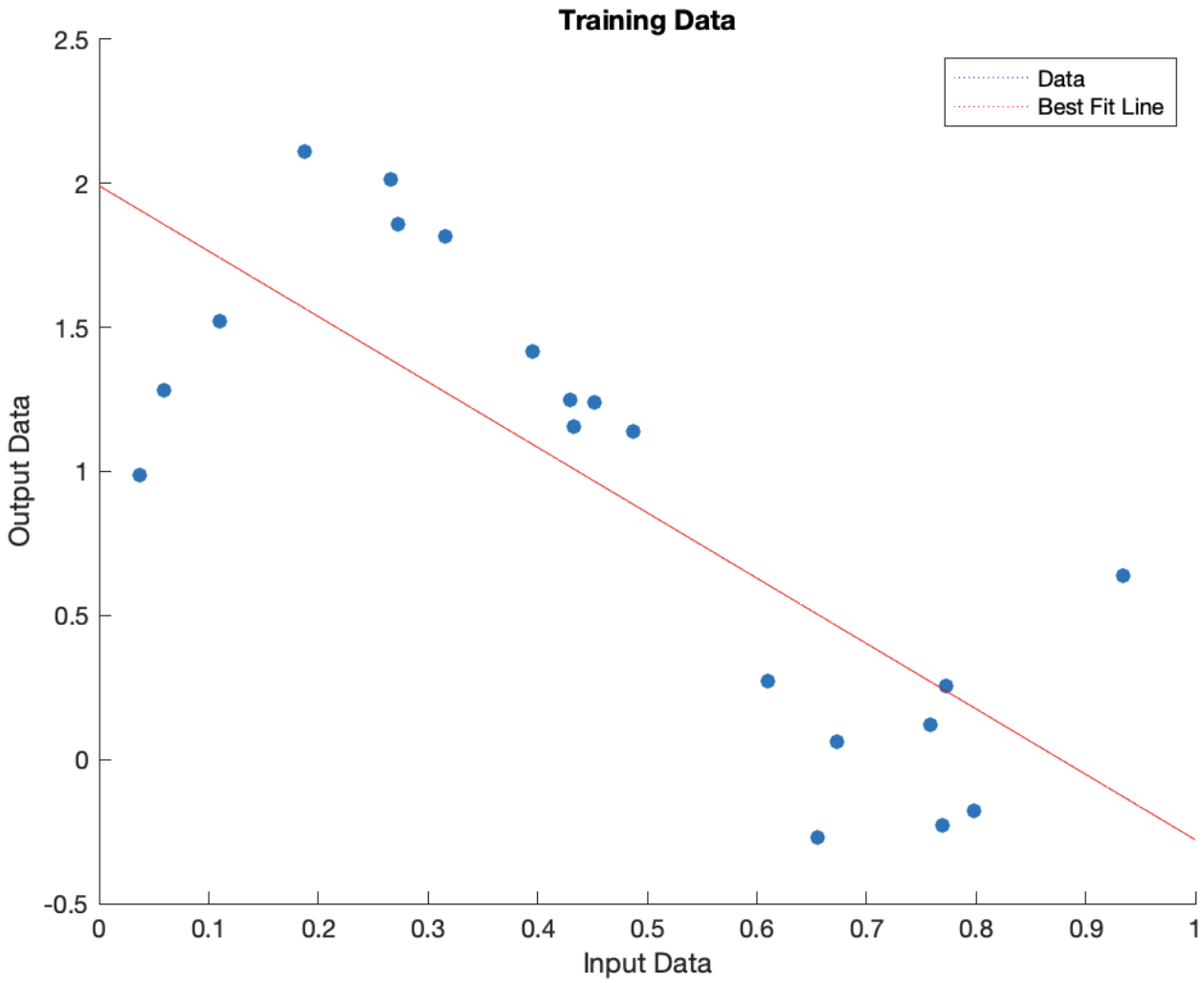
Part B, Closed Form Linear Regression

The closed form solution to linear regression is $w = (X^TX)^{-1}X^Ty$. After running this solution on the training data, I found the following:

$w = [1.9920, -2.2705]$

$J(w) = 4.6036$

Here is a plot of the training data and fitted line…



Part C, Gradient Descent
Report of J(w) and number of iterations for different learning rates η:

Step Size	J(w) -- Measure of Error	# of Iterations
0.05	4.6044	83
0.001	4.6489	2420
0.0001	5.6408	10000
0.00001	12.2494	10000

Discussion of learning rate η
As η decreases in size, gradient descent requires more iterations to converge. This can be observed for the case of η = 0.05 and η = 0.001. By decreasing η, over two thousand extra iterations were required to reach convergence.

If η is smaller, more precise shifts in w are caused. This is not necessarily a good thing -- in the case of η = 0.001, J(w) was worse than for η = 0.05 in large part because J(w) wasn't changing enough (i.e. the condition $|J(w)_t - J(w)_{t-1}| < 0.0001$ was triggered).

If η is very small, it takes significantly many more iterations for gradient descent to actually achieve the minimum of the objective function. This can be seen when η = 0.0001 and η = 0.00001, where η was so small that the algorithm didn't converge to the minimum within 10,000 iterations. As a consequence, the obtained value for w was very poor, as illustrated in the abysmal J(w) values of 5.6408 and 12.2494 respectively.

Part D, Gradient Descent Visualization

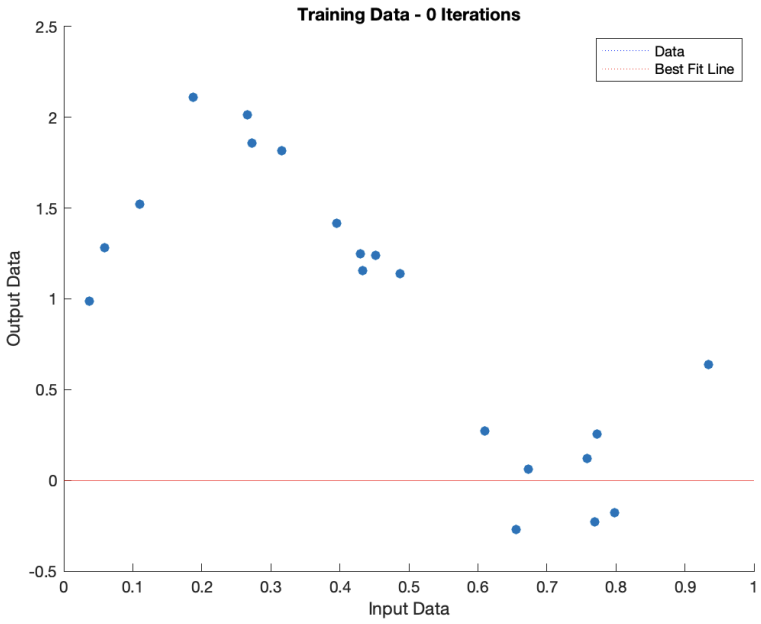
After repeating exercise C with η = 0.05, running gradient descent for 40 iterations, I found:

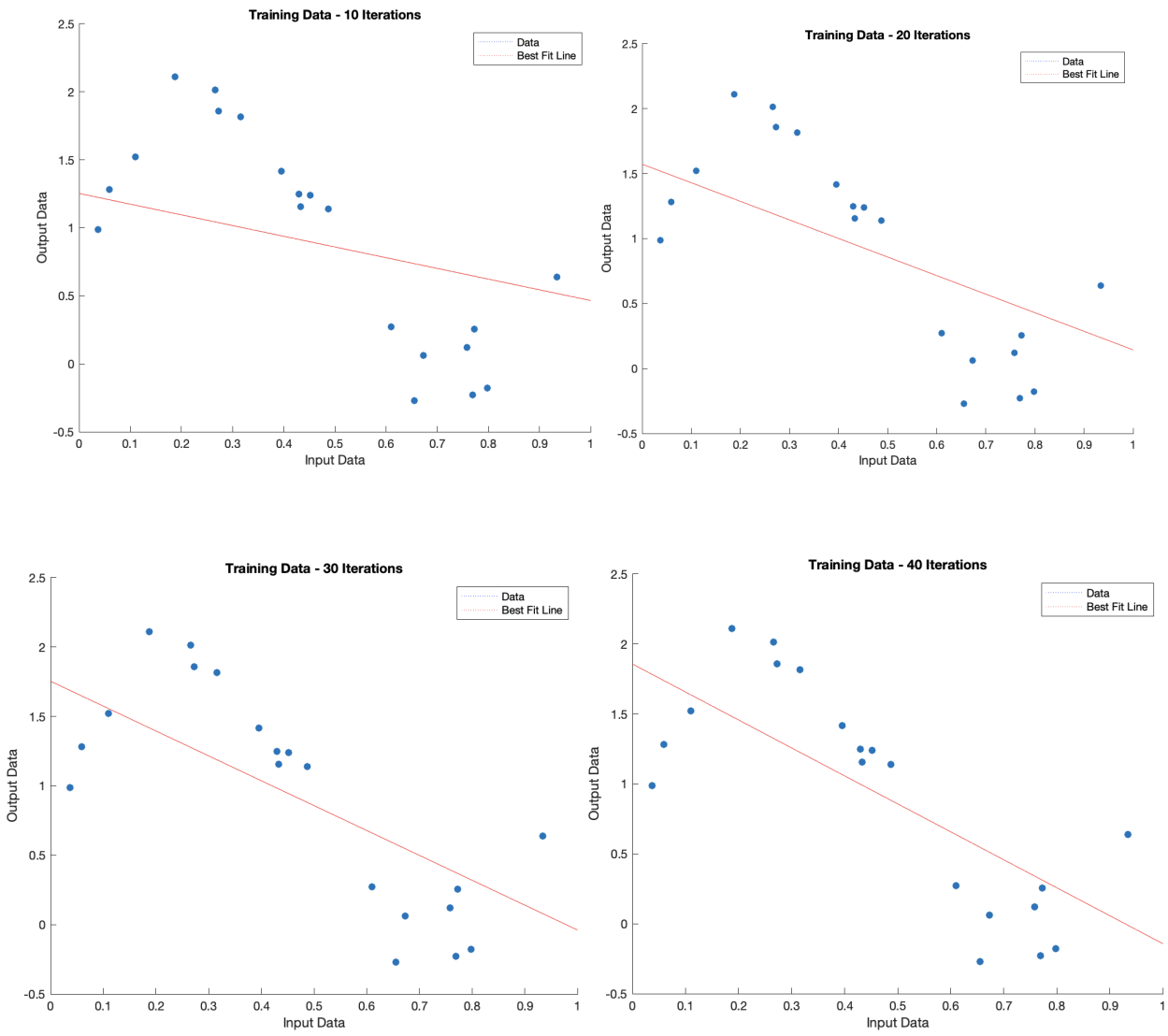
w = [1.8570, -1.9997]

J(w) = 4.7045

The data is recorded and fitted line plotted here for 0, 10, 20, 30, and 40 iterations:

# of Iterations	w	J(w)
0	[0, 0]	28.6555
10	[1.2530, -0.7875]	7.6293
20	[1.5727, -1.4291]	5.5775
30	[1.7541, -1.7931]	4.9171
40	[1.8570, -1.9997]	4.7045





The J(w) values are higher than the J(w) value obtained in part b. Additionally, the fitted lines are more horizontal than in b, and thus appear to fit the plotted data worse.

Over different iterations, it is clear that the J(w) values get smaller. Additionally, the fitted lines become increasingly vertical, and thus fit the data progressively better with each iteration.

Part E, Polynomial Regression

The following table includes the best fit model w for the training data using polynomials degree 0, 1, …, 10 and the closed form solution. Additionally, given that w, it shows the root mean square error for the training and test data.

Note that by degree 7, Matlab's default precision suppressed some details about the elements of w. I report exactly what Matlab returned for brevity, as I do not think reporting the precise value of w here is necessary to answer the provided questions…

Additionally note that each entity in the *w* column is a vector of values. So, for instance, the vector in degree 10, column w more properly could be represented as

1.0e+05 * [-0.0000 0.0007 -0.0131 0.1258 -0.6681 2.0415 -3.6431 3.6431 -1.7060 0.0643 0.1553]

Degree	w	RMS Training	RMS Testing
0	[0.9230]	0.7622	0.6923
1	[1.9920 -2.2705]	0.4798	0.5617
2	[1.6817 -0.3332 -2.0773]	0.4609	0.6689
3	[0.5482 14.0335 -40.3736 27.3410]	0.1518	0.1798
4	[0.4274 16.4001 -50.8892 43.7325 -8.3066]	0.1487	0.1783
5	[0.5882 12.1432 -21.9992 -33.9421 81.1504 -36.8589]	0.1463	0.1705
6	[0.4429 16.8253 -65.1287 135.8314 -242.1289 256.8354 -101.9357]	0.1454	0.1888
7	1.0e+03 * [0.0007 0.0069 0.0603 -0.5740 1.8125 -2.9030 2.3511 -0.7552]	0.1445	0.4016
8	1.0e+04 * [0.0002 -0.0044 0.0832 -0.6036 2.2658 -4.8384 5.9034 -3.8240 1.0187]	0.1343	1.5550
9	1.0e+05 * [-0.0000 0.0008 -0.0149 0.1448 -0.7835 2.4699 -4.6500 5.1453 -3.0830 0.7710]	0.1108	7.6890
10	1.0e+05 * [-0.0000 0.0007 -0.0131 0.1258 -0.6681 2.0415 -3.6431 3.6431 -1.7060 0.0643 0.1553]	0.1114	8.2655

The following plot shows how RMSE varies with model complexity:

Based on the results, I think that polynomial degree 3 best fits the data. This is because the RMSE scores for both the training and testing data are small (RMSE < 0.2) and using a reasonably simple model prevents the case of overfitting the data.

According to the plot, there is *no* significant evidence of underfitting the data. For relatively simple models (i.e. polynomial degree 0, 1 and 2), the RMSE for both the training and testing data was well under 1. Furthermore, for a given degree, the RMSE score for the training data was very close to the RMSE score for the testing data.

According to the plot, there *is* evidence of overfitting the data. For the more complex models (i.e. polynomial degree 7, 8, 9, 10), we see very small errors for the testing data (RMSE < 0.2), but very large errors for the training data (RMSE up to 8.2). This indicates that the model was overfit to the training data, thus performing poorly on new test data points.