Stat 6910, Section 002
Deep Learning: Theory and Applications
Spring 2019

Homework 1

Jared Hansen

A-number: A01439768

e-mail: jrdhansen@gmail.com

Due: 5:00 PM, Friday 01/18/2019

Homework I

STAT 6910/7810-002 - Spring semester 2019

Due: Friday, January 18, 2019 - 5:00 PM

Please put all relevant files & solutions into a single folder titled `<lastname and initials>_assignment1` and then zip that folder into a single zip file titled `<lastname and initials>_assignment1.zip`, e.g. for a student named Tom Marvolo Riddle, `riddletm_assignment1.zip`. Include a single PDF titled `<lastname and initals>_assignment1.pdf` and any Python scripts specified. Any requested plots should be sufficiently labeled for full points.

Unless otherwise stated, programming assignments should use built-in functions in Python, Tensorflow, and PyTorch. In general, you may use the `scipy` stack; however, exercises are designed to emphasize the nuances of machine learning and deep learning algorithms - if a function exists that trivially solves an entire problem, please consult with the TA before using it.

Problem 1

Provide an example application where each of the following architectures or techniques would be useful. Do not reuse examples from class.

1. Multilayer perceptron
   Since the output of a multi-layer perceptron is binary, these could be used for binary classification modeling. One example of this might be fraud detection: banks and credit card companies work to determine which of their customers' transactions are fraudulent. A caveat though: input data would require some pre-processing since the inputs to MLPs must also be binary. (Other methods might typically work better, but the structure of an MLP would technically work for these types of problems.)

2. Convolutional Neural Network
   Since CNNs (convolutional neural networks) involve filtering, they might also be applied to audio inputs rather than the visual inputs they're often used to process. While CNNs are well-known for their abilities to classify images, maybe they'd be useful for classifying who is speaking during an audio clip. The network could be trained on previous clips of the speakers, and then would determine who is speaking during certain portions of the new clip(s) playing.

3. Recurrent Neural Network
   RNNs would be useful for modeling time series data as an alternative to traditional statistical models, or any other data in which time is a very important feature. One example of this is the research I'm working on in trade sign classification in high frequency trading. A lot of information about the whether the next trade will be a buy or a sell is contained in the previous observation.

4. Autoencoder
   These would be useful in any application in which compressing data while still retaining (near) the same level of information is crucial. JPEG images were mentioned aloud in class, but I found that PNG images use it too. I also found an article talking about how they can be used to remove watermarks from images.

5. Generative Adversarial Network
   These are useful in situations where new examples of something are to be created and then testing these new examples against actual data. For instance, we could train a network on all of the songs of Stevie Ray Vaughan, use that network to create new songs, and then use a separate network to determine which songs are authentic and which are GAN-generated.

6. Deep reinforcement learning
   In talking with one of my supervisory committee members, he mentioned a paper in which deep reinforcement learning was used for hedging in options. Essentially, this method is used to select a portfolio to hedge against certain risk and optimize returns. https://arxiv.org/pdf/1802.03042.pdf

Problem 2

1. For a matrix A, we write $A \succeq 0$ and $A \succ 0$ when A is positive semi-definite (PSD) or positive definite (PD), respectively. Using the definition of a PD matrix, prove that the sum of two PD matrices is also PD. A very similar approach can be used to prove the sum of two PSD matrices is also PSD (although you don't have to prove it).

   - By definition, a matrix $\boldsymbol{A}$ is PD if $\left[\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} > 0 \ \forall \boldsymbol{x} \in \mathbb{R}^d\right]$ where $\boldsymbol{A}$ is a $d \times d$ matrix and $\boldsymbol{x}$ is a $d \times 1$ vector.
   - Let $\boldsymbol{A}$ and $\boldsymbol{B}$ both be PD matrices. Since $\boldsymbol{A}$ and $\boldsymbol{B}$ are both PD we know that $\left[\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} > 0 \ \forall \boldsymbol{x} \in \mathbb{R}^d\right]$ and $\left[\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x} > 0 \ \forall \boldsymbol{x} \in \mathbb{R}^d\right]$.
   - Also, since the question is asking about "the sum of two PD matrices", we will now be working with the matrix $\big(\boldsymbol{A} + \boldsymbol{B}\big)$. Let's multiply it by $\boldsymbol{x}^T$ on the left and $\boldsymbol{x}$ on the right and see if it is indeed a PD matrix.
   - Using properties of matrix-vector multiplication as well as facts from the previous bulleted item we can rewrite and manipulate this as follows:
     $\left[\boldsymbol{x}^T\big(\boldsymbol{A}+\boldsymbol{B}\big)\boldsymbol{x}\right] = \left[\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}\right] + \left[\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x}\right] = \left[\text{“} > 0\text{”}\right] + \left[\text{“} > 0\text{”}\right] = \left[\text{“} > 0\text{”}\right] \implies \left[\boldsymbol{x}^T\big(\boldsymbol{A}+\boldsymbol{B}\big)\boldsymbol{x}\right] > 0.$
     Using the definition in the first bulleted item, this must mean that the matrix $\big(\boldsymbol{A}+\boldsymbol{B}\big)$ is necessarily a PD matrix, proving the desire result.

2. Is the sum of a PD matrix and a PSD matrix necessarily PD, PSD, or neither? Explain why.

   - Let $\boldsymbol{A}$ be a PD matrix and $\boldsymbol{B}$ be a PSD matrix. We are trying to determine whether the matrix $\big(\boldsymbol{A} + \boldsymbol{B}\big)$ is PD, PSD, or neither.
   - Using properties of PD and PSD matrices as well as properties of matrix-vector multiplication we can make the following string of deductions:
     $\left[\boldsymbol{x}^T\big(\boldsymbol{A}+\boldsymbol{B}\big)\boldsymbol{x}\right] = \left[\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}\right] + \left[\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x}\right] = \left[\text{“} > 0\text{”}\right] + \left[\text{“} \geq 0\text{”}\right] = \left[\text{“} > 0\text{”}\right] \implies \left[\boldsymbol{x}^T\big(\boldsymbol{A}+\boldsymbol{B}\big)\boldsymbol{x}\right] > 0.$
   - From the conclusion of the above bulleted item, we know that the matrix $\big(\boldsymbol{A}+\boldsymbol{B}\big)$ must necessarily be a PD matrix. More generally, the sum of a PD matrix and a PSD matrix is necessarily a PD matrix.

3. Consider the following matrices:

$$A = \begin{bmatrix} 1 & -2 \\ 3 & 4 \\ -5 & 6 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, C = \begin{bmatrix} 2 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 2 \end{bmatrix}$$

Determine whether the following matrices are PD, PSD, or neither. Briefly explain why. You may use the NumPy library for this problem.
Hint: A symmetric matrix is PD if and only if all of its eigenvalues are greater than zero. A symmetric matrix is PSD if and only if all of its eigenvalues are greater than or equal to zero.

*****NOTE***: based on the mathematical definition, any PD matrix is also PSD (but PSD matrices aren't PD). So anywhere I've found a PD matrix I also say that it is PSD.

(a) A is **NEITHER**. In order to be either PD or PSD, a matrix must be square. A isn't a square matrix $\implies$ A cannot be either PD or PSD.

(b) A^TA is **PD** (and also **PSD**): has eigenvalues {22.91, 68.09}, both of which are > 0.

(c) AA^T is **PSD**: has eigenvalues {68.09, 0, 22.91}, all of which are ≥ 0.

(d) B is **PD** (and also **PSD**): has eigenvalues {1, 1, 1}, all of which are > 0.

(e) $-B$ is **NEITHER**: has eigenvalues {-1, -1, -1}, none of which are ≥ 0.

(f) C is **PD** (and also **PSD**): has eigenvalues {5.83, 0.17}, both of which are > 0.

(g) $C - 0.1 \times B$ is **PD** (and also **PSD**): has eigenvalues {5.73, 0.9, 0.07}, all of which are > 0.

(h) $C - 0.01 \times AA^T$ is **NEITHER**: has eigenvalues {5.61, 0.52, -0.04}. Since (-0.04) is <u>not</u> ≥ 0, the eigenvalues of this matrix don't satisfy the requirements for being either PD or PSD.

Problem 3

Consider the function $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$ as defined below:

$$\mathbf{f}(\mathbf{v}) = \begin{bmatrix} f_1(\mathbf{v}) \\ f_2(\mathbf{v}) \end{bmatrix} = \begin{bmatrix} v_1^2 + 3v_1 e^{v_2} \\ 4v_1^3 v_2 - v_1 v_2 \log v_2 \end{bmatrix}.$$

1. Compute the gradient and Hessian of f_1.

   $\nabla f_1 = \begin{bmatrix} \frac{\partial f_1}{\partial v_1} \\ \frac{\partial f_1}{\partial v_2} \end{bmatrix} = \begin{bmatrix} 2v_1 + 3e^{v_2} \\ 3v_1 e^{v_2} \end{bmatrix}$. (This is the gradient of f_1.)

   To obtain the Hessian we take the gradient of the gradient ("second derivative") to get the matrix:

   $\nabla^2 f_1 = \begin{bmatrix} \frac{\partial^2 f_1}{\partial^2 v_1} & \frac{\partial^2 f_1}{\partial v_1 v_2} \\ \frac{\partial^2 f_1}{\partial v_2 v_1} & \frac{\partial^2 f_1}{\partial^2 v_2} \end{bmatrix} = \begin{bmatrix} 2 & 3e^{v_2} \\ 3e^{v_2} & 3v_1 e^{v_2} \end{bmatrix}$. (This is the Hessian of f_1.)

2. Compute the gradient and Hessian of f_2.

   $\nabla f_2 = \begin{bmatrix} \frac{\partial f_2}{\partial v_1} \\ \frac{\partial f_2}{\partial v_2} \end{bmatrix} = \begin{bmatrix} 12{v_1}^2 v_2 - v_2 log(v_2) \\ 4{v_1}^3 - v_1(1 + log(v_2)) \end{bmatrix}$. (This is the gradient of f_2.)

   To obtain the Hessian we take the gradient of the gradient ("second derivative") to get the matrix:

   $\nabla^2 f_2 = \begin{bmatrix} \frac{\partial^2 f_2}{\partial^2 v_1} & \frac{\partial^2 f_2}{\partial v_1 v_2} \\ \frac{\partial^2 f_2}{\partial v_2 v_1} & \frac{\partial^2 f_2}{\partial^2 v_2} \end{bmatrix} = \begin{bmatrix} 24v_1 v_2 & 12{v_1}^2 - 1 - log(v_2) \\ 12{v_1}^2 - 1 - log(v_2) & -\frac{v_1}{v_2} \end{bmatrix}$. (This is the Hessian of f_2.)

3. Compute the Jacobian of $\mathbf{f}$.

   By definition (in the notes) $J_f = \begin{bmatrix} \nabla f_1 & \nabla f_2 \end{bmatrix} \implies J_f = \begin{bmatrix} 2v_1 + 3e^{v_2} & 12{v_1}^2 v_2 - v_2 log(v_2) \\ 3v_1 e^{v_2} & 4{v_1}^3 - v_1(1 + log(v_2)) \end{bmatrix}$

Problem 4

1. Consider two arbitrary random variables X and Y. For the following equations, describe the relationship between them. Write one of four answers to replace the question mark: "=", "$\leq$", "$\geq$", or "depends". Choose the most specific relation that always holds and briefly explain why. Assume all probabilities are non-zero.

   (a) $\Pr(X = x, Y = y) \boxed{\leq} \Pr(X = x)$

   If we think if $P(X = x)$ and $P(Y = y)$ as constraints, we know that imposing the additional constraint of $P(Y = y)$ to $P(X = x)$ can result only in a lower probability if $Y \not\subset X$. However, if $Y \subset X$ then $P(X = x, Y = y) = P(X = x)$ since all events $\in Y$ are also $\in X$. Hence the correct answer is $\boxed{\leq}$.

   (b) $\Pr(X = x|Y = y) \boxed{\textit{depends}} Pr(X = x)$

   If X and Y are independent: $P(X = x|Y = y) = P(X = x)$ by definition.
   If X and Y are not independent: we know $P(X = x|Y = y) = \frac{P(X=x,Y=y)}{P(Y=y)}$. So we can rewrite the original expression to be $\frac{P(X=x,Y=y)}{P(Y=y)} \boxed{?} P(X = x)$. We know from part (a) that $P(X = x, Y = y) \leq P(X = x)$, so we can rewrite to get $\frac{\text{"}\leq P(X=x)\text{"}}{P(Y=y)} \boxed{?} P(X = x)$. Therefore, the wiggle room in the "$\leq P(X = x)$" makes the answer $\boxed{\textit{depends}}$.

   (c) $\Pr(X = x|Y = y) \boxed{\geq} Pr(Y = y|X = x) \Pr(X = x)$

   Let's consider two cases: (1.) X and Y are independent, (2.) X and Y are not independent.

   (1.) If X and Y are independent then the original expression becomes $P(X = x) \boxed{?} P(Y = y)P(X = x)$. Since $0 \leq P(Y = y) \leq 1$ and $0 \leq P(X = x) \leq 1$ we know that $P(X = x) = P(X = x)P(Y = y)$ if $P(Y = y) = 1$, and $P(X = x) > P(X = x)P(Y = y)$ otherwise.

   (2.) Write out the facts that $P(X = x|Y = y) = \left[\frac{P(X=x,Y=y)}{P(X=x)}\right]$ and $P(Y = y|X = x) = \left[\frac{P(Y=y,X=x)}{P(X=x)}\right] \implies P(Y = y, X = x) = P(Y = y|X = x)P(X = x)$.
   Now use these to rewrite the original expression as $\left[\frac{P(X=x,Y=y)}{P(Y=y)}\right] \boxed{?} \left[\frac{P(Y=y,X=x)}{1}\right]$.
   Since $P(X = x, Y = y) = P(Y = y, X = x)$ this implies that the sign must be $\geq$. If $P(Y = y) = 1$ then the two expressions are equal, but if $P(Y = y) < 1$ then the quantity $\left[\frac{P(X=x,Y=y)}{P(Y=y)}\right]$ will be larger than $P(X = x, Y = y)$. Hence the correct answer is $\boxed{\geq}$.

Problem 5

1. Suppose we have d-dimensional data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and corresponding real-valued response variables $y_1, \ldots, y_n$. In regression, we are trying to learn a function $f(\mathbf{x})$ such that $y \approx f(\mathbf{x})$. For linear regression, we assume that f is a linear function: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$ where w_0 is an offset term.

   One approach to approximating y is to minimize the empirical mean-square-error (MSE). The empirical MSE can be written as
   $$\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i + w_0 - y_i)^2.$$

   Now let X be a $n \times d+1$ matrix where the ith row corresponds to $[\mathbf{x}_i^T, 1]$ where the 1 term is added to include the offset term in the regression model. Also let y be a n-dimensional vector of the response variables where the ith entry corresponds to y_i. Show that the linear regression solution that minimizes the empirical MSE is
   $$\mathbf{w}^* = (X^TX)^{-1}X^T\mathbf{y}.$$

   <u>NOTE:**</u> I used the original defintion of MSE given in the prompt: $\sum_{i=1}^{n}\left(\boldsymbol{w}^T\boldsymbol{x}_i + w_0 - y_i\right)^2$ which is not scaled by $\left(\frac{1}{n}\right)$. However, finding $\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$ will minimize both this quantity and typical MSE since one is just a scalar multiple of the other.
   That being said, let's go through the problem.

   - First, let's absorb the w_0 term such that $\boldsymbol{w} = \begin{bmatrix} w_1 & w_2 \ldots w_n & w_0 \end{bmatrix}^T$. Also, let each $\boldsymbol{x_i} = \begin{bmatrix} x_i{}^{(1)} x_i{}^{(2)} \ldots x_i{}^{(n)} & 1 \end{bmatrix}$.
     Thus we'll rewrite MSE from $\sum_{i=1}^{n}\left(\boldsymbol{w}^T\boldsymbol{x}_i + w_0 - y_i\right)^2$ as given to now be $\sum_{i=1}^{n}\left(\boldsymbol{w}^T\boldsymbol{x}_i - y_i\right)^2$.
   - Now, let's expand the term $\sum_{i=1}^{n}\left(\boldsymbol{w}^T\boldsymbol{x}_i - y_i\right)^2$ to be $\sum_{i=1}^{n}\left(\boldsymbol{w}^T\boldsymbol{x}_i - y_i\right)\left(\boldsymbol{w}^T\boldsymbol{x}_i - y_i\right) =$
     $\sum_{i=1}^{n}\boldsymbol{w}^T\boldsymbol{x}_i\boldsymbol{w}^T\boldsymbol{x}_i - \boldsymbol{w}^T\boldsymbol{x}_i y_i - y_i\boldsymbol{w}^T\boldsymbol{x}_i + y_i y_i$. This is just summing this new expression "one i (row) at a time." To make this more efficient we can rewrite in terms of matrices, and will get:
     $\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\boldsymbol{w} + \boldsymbol{y}^T\boldsymbol{y} = MSE_{matrix}$ (this is the matrix form of un-scaled MSE).
   - The goal is to minimize this quantity (MSE) relative to the weights $\boldsymbol{w}$ since they're the only thing that isn't fixed. We will do this by taking $\nabla_{\boldsymbol{w}}\left(MSE_{matrix}\right)$ set it $= 0$ and then solve for $\boldsymbol{w}^*$ which will give the optimal weights for minimizing MSE.
   - Using rules for matrix/vector differentiation (https://atmos.washington.edu/ dennis/MatrixCalculus.pdf) we will calculate that $\nabla_{\boldsymbol{w}}\left(MSE_{matrix}\right) = 2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X} + 0$. By examination, we can see that the two middle terms are equivalent expressions, so we can condense them to get: $\nabla_{\boldsymbol{w}} = 2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - 2\boldsymbol{X}^T\boldsymbol{y}$.
   - Now let's set $\nabla_{\boldsymbol{w}} = 0$ and solve for $\boldsymbol{w}^*$:
     $\left[2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w}^* - 2\boldsymbol{X}^T\boldsymbol{y} = 0\right] \rightarrow \left[2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w}^* = 2\boldsymbol{X}^T\boldsymbol{y}\right] \rightarrow \left[\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w}^* = \boldsymbol{X}^T\boldsymbol{y}\right]$
     $\rightarrow \left[\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\left(\boldsymbol{X}^T\boldsymbol{X}\right)\boldsymbol{w}^* = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\left(\boldsymbol{X}^T\boldsymbol{y}\right)\right] \rightarrow \boxed{\left[\boldsymbol{w}^* = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\left(\boldsymbol{X}^T\boldsymbol{y}\right)\right]}$
   - Therefore we have shown that the linear regression solution $\boldsymbol{w}^* = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\left(\boldsymbol{X}^T\boldsymbol{y}\right)$ minimizes empirical MSE, proving the desired result.

2. Write code in Python with file name `prob2_2.py` that randomly generates N points sampled uniformly in the interval $x \in [-1, 3]$. Then output the function $y = x^2 - 3x + 1$ for each of the points generated. Then write code that adds zero-mean Gaussian noise with standard deviation σ to y. Make plots of x and y with $N \in \{15, 100\}$ and $\sigma \in \{0, .05, .2\}$ (there should be six plots in total). Save the point sets for the following question.
Hint: You may want to check the NumPy library for generating noise.

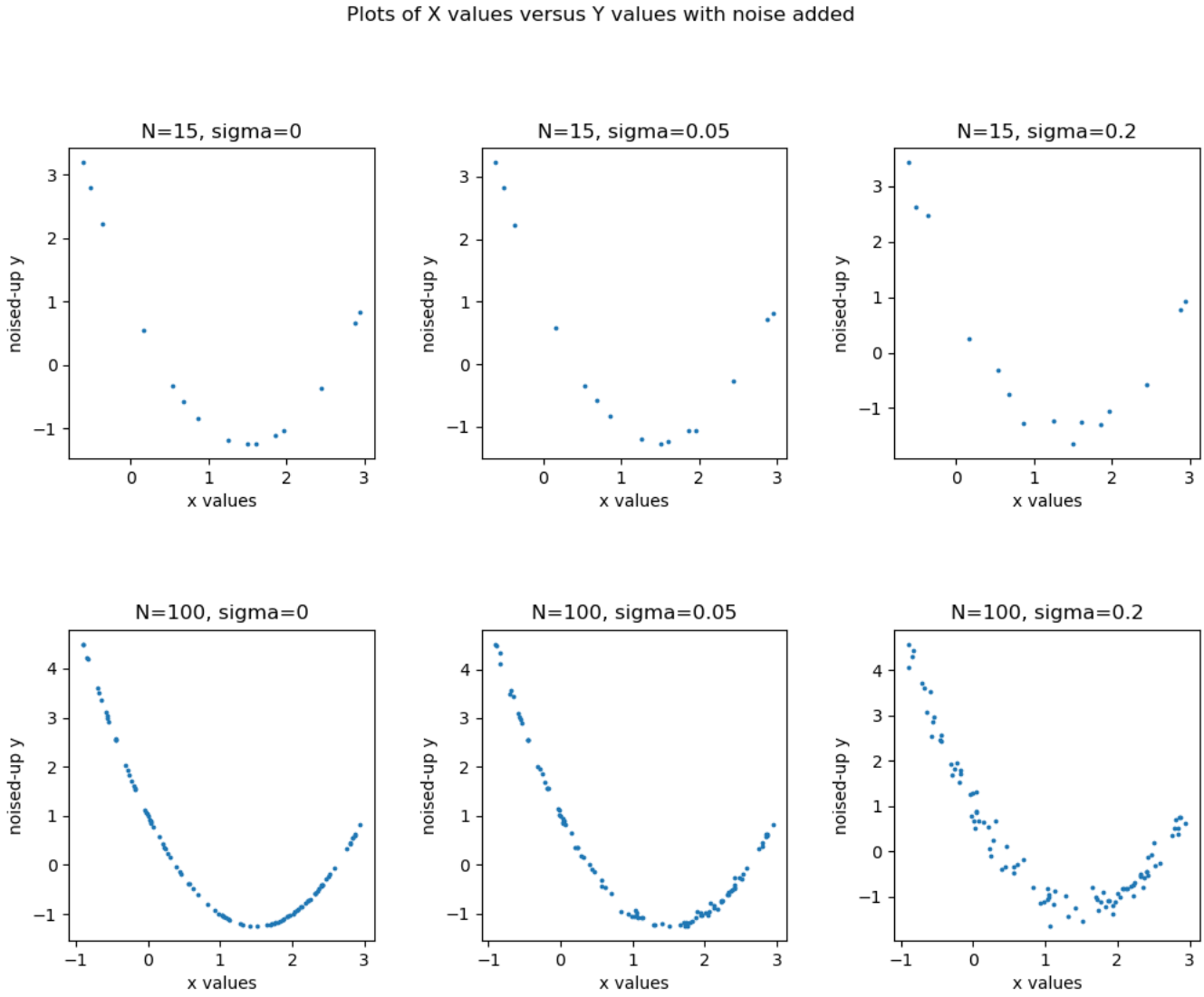

Figure 1: Here are the six plots of the points generated. The title for each individual plot contains the number of data points (N) and the value of σ (sigma) for the added Gaussian noise. As expected, the $\sigma = 0$ plots follow the curve $f(x) = x^2 - 3x + 1$ perfectly, the $\sigma = 0.05$ follow the curve very closely, and the $\sigma = 0.2$ start to adhere a bit less to the original curve.

3. Find the optimal weights (in terms of MSE) for fitting a polynomial function to the data in all 6 cases generated above using a polynomial of degree 1, 2, and 9. Use the equation given above. Include your code in `prob2_3.py`. Do not use built-in methods for regression. Plot the fitted curves on the same plot as the data points (you can plot all 3 polynomial curves on the same plot). Report the fitted weights and the MSE in tables. Do any of the models overfit or underfit the data?

__NOTE:**__ First I will include the three plots for which N = 15 and will discuss any overfitting or underfitting in the respective captions. I'll then give the table containing their weights and MSEs. Then I will do the same thing (plots + discussion, table) for the N = 100 data.

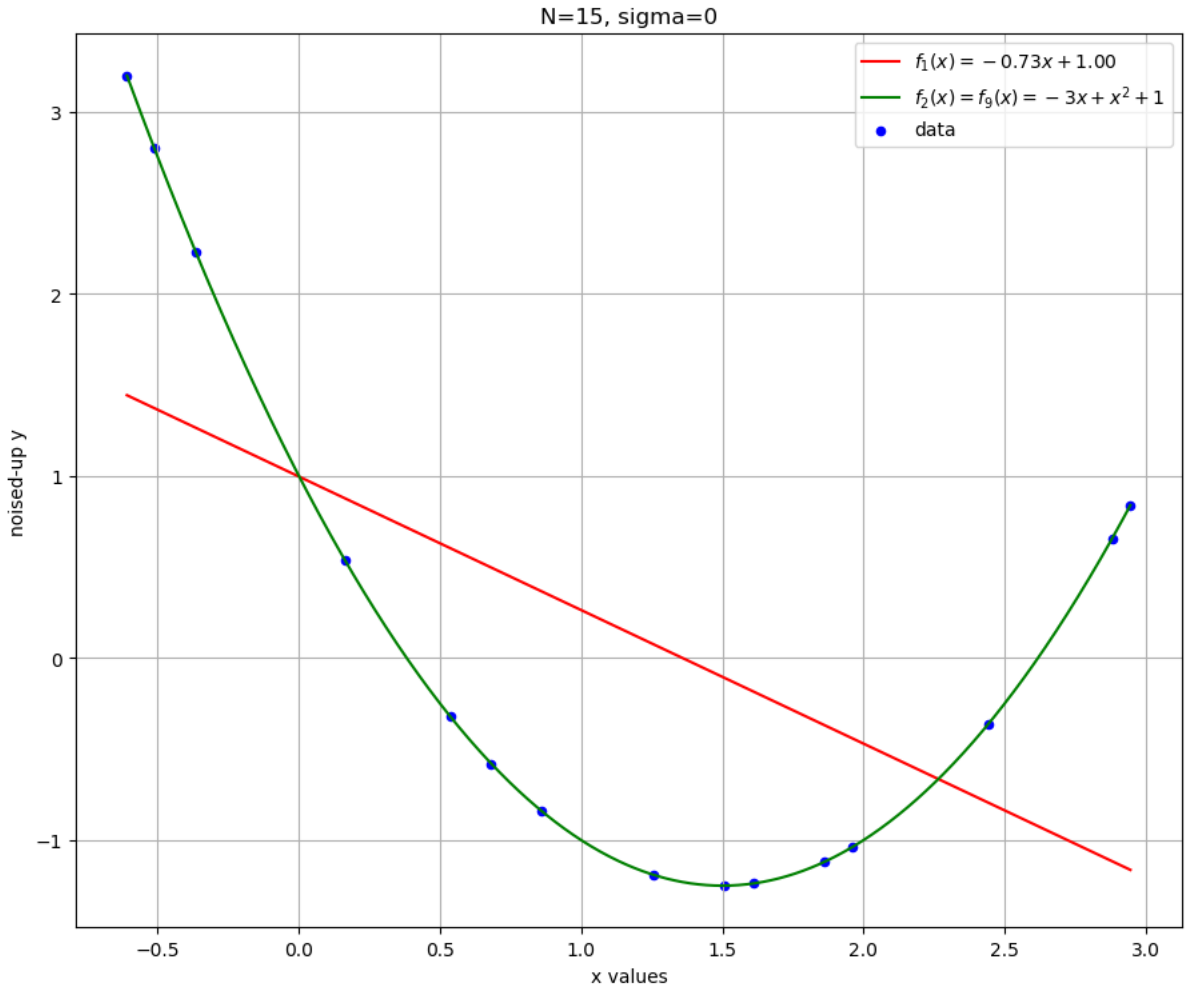

Figure 2: Here is the plot for N = 15 and $\sigma = 0$. Notice in the legend that the 2^{nd} and 9^{th} degree polynomial curves are the same. Both of these curves fit the data perfectly (as we'd expect/hope). Also as we'd expect, the linear function, $f_1(x)$ underfits the data because it doesn't have enough terms to accurately capture the data-generating function.

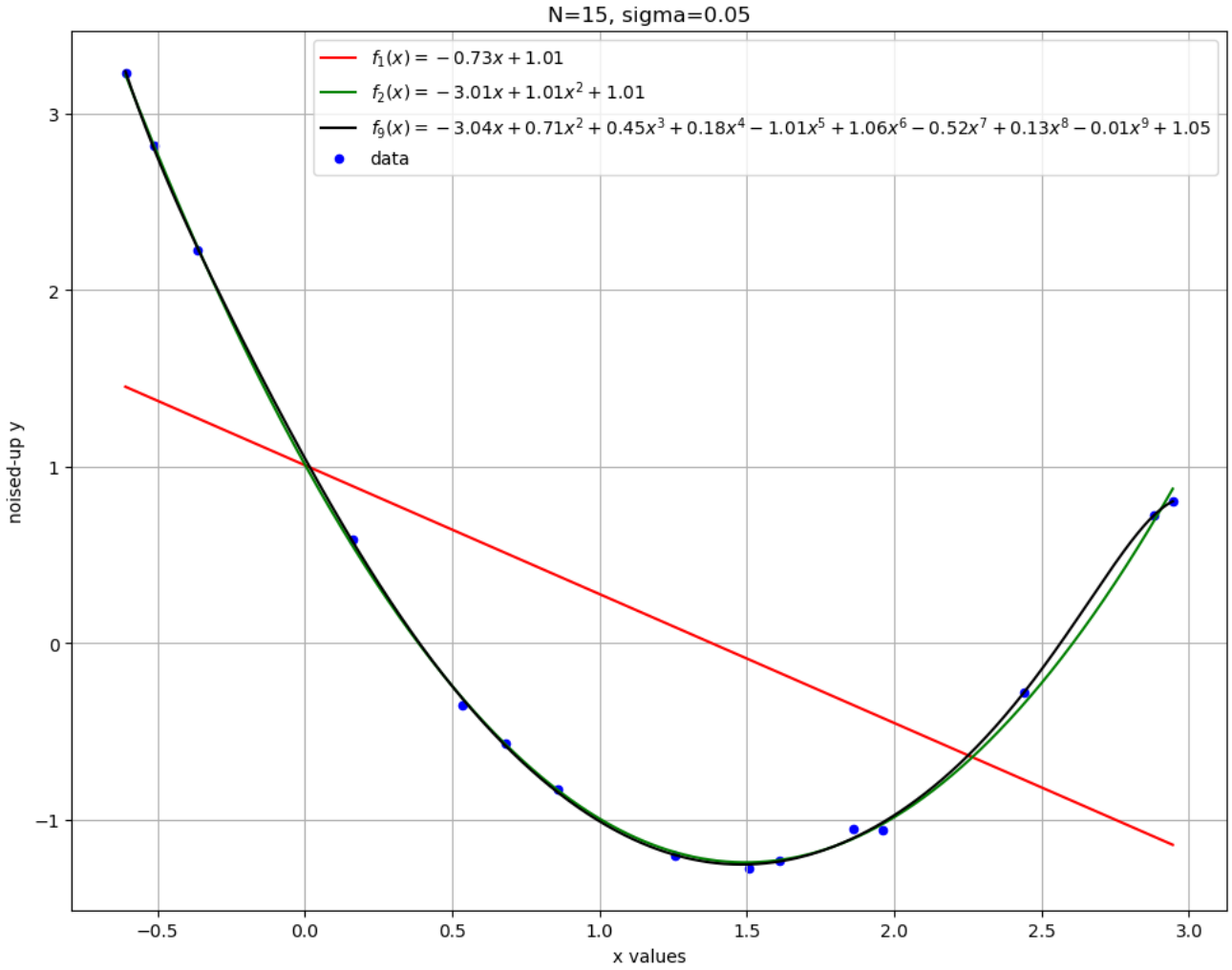

Figure 3: Here is the plot for N = 15 and $\sigma = 0.05$. Although the curves for the 2^{nd} and 9^{th} degree polynomials are similar, we can see a little bit of a difference at the right-hand side of the plot. This is largely due to the fact that the data are still very close to being on the $f(x) = x^2 - 3x + 1$ curve since the Gaussian noise we added had such a lower standard deviation ($\sigma = 0.05$).
Again, the 2^{nd} degree polynomial function gives an excellent fit for the data, and the 9^{th} degree polynomial function does quite well also. We can see a bit of overfitting on the right-hand side, but nothing too egregious. It's also interesting to see how all the non-zero terms for weights that "should" be zero for f_9 essentially cancel each other out, and we get something very close to $f(x) = x^2 - 3x + 1$.
Unsurprisingly, the linear function with only an intercept and slope parameter is nearly identical to the line for the $\sigma = 0$ plot, and again underfits the data.

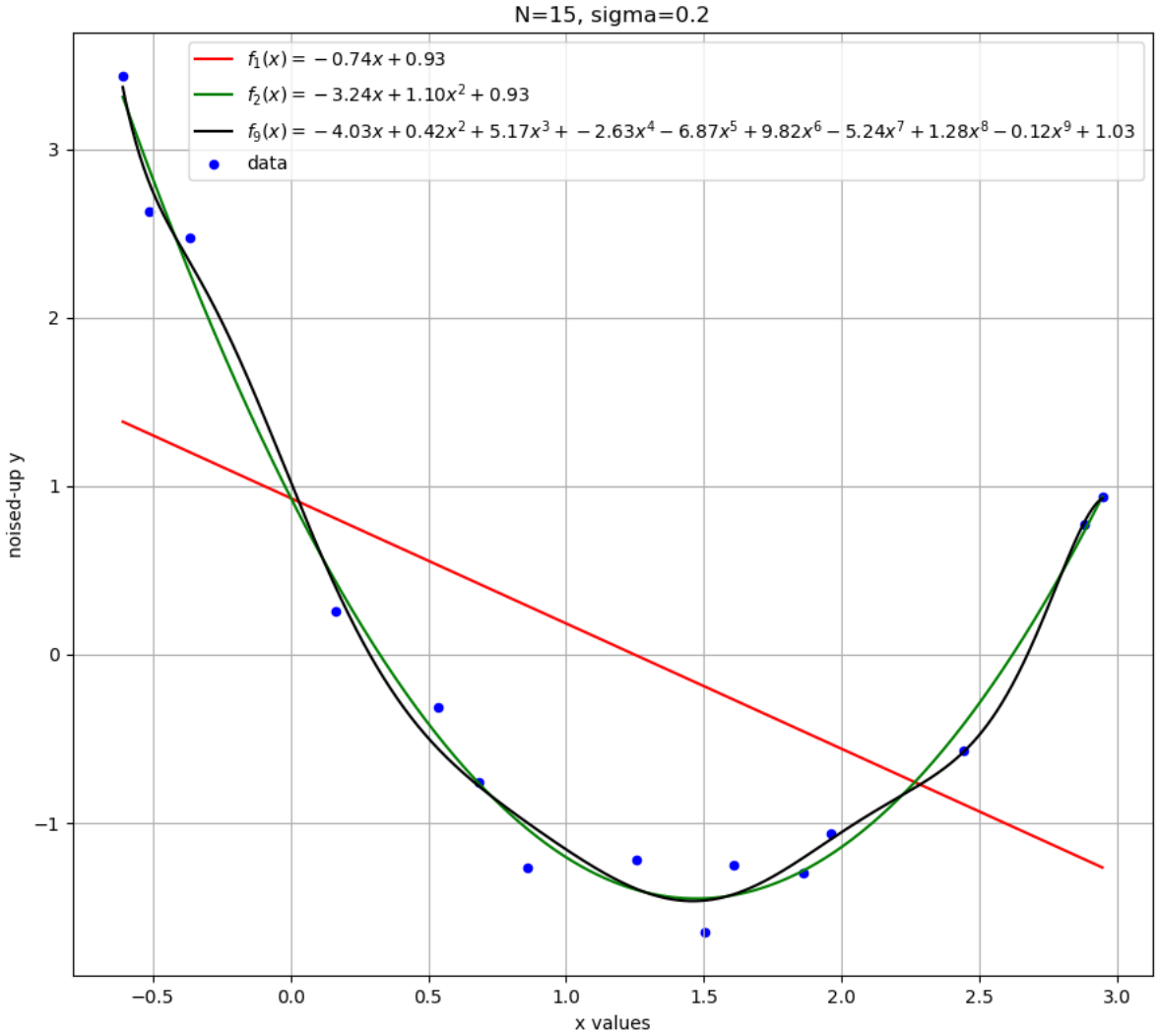

Figure 4: Here is the plot for N = 15 and $\sigma = 0.2$. Here we start to notice a bit of deviation from the $f(x) = x^2 - 3x + 1$ curve for the 2^{nd} degree polynomial function since our Gaussian noise is "noisier" (higher deviation). It's still a fairly good fit though.
However, we do see some overfitting in the 9^{th} degree polynomial function. Since there are only 15 data points, each of them has a fairly significant influence on the curve, leading to some waviness that isn't seen as part of the original function. (NOTE: I ran some of this code again with a different random seed and saw **much** worse overfitting than this. These data points could have been far noisier than the ones seen here.)
Once again the linear model is underfitting the data. This will be the case with all the linear functions.

		N = 15		
		Sigma = 0	Sigma = 0.05	Sigma = 0.2
Plynml of degree 1	MSE	1.459802	1.4600639766325056	1.4663891851418256
	Weights	[-0.73313907] [0.995823]	[-0.73021579] [1.00830778]	[-0.74430206] [0.9284639]
Plynml of degree 2	MSE	8.968760202235325e-22	0.000343759472794841	0.02088313983696508
	Weights	[-3.] [1.] [1.]	[-3.01405715] [1.00749072] [1.01251607]	[-3.23549251] [1.09896042] [0.93305426]
Plynml of degree 9	MSE	1.439428868468577e-14	0.001171158189542939	0.025301363350271323
	Weights	[-2.99999983e+00] [9.99999713e-01] [1.75903551e-07] [-6.59201760e-08] [8.09086487e-09] [3.72529030e-09] [-1.51339918e-09] [3.05590220e-10] [-2.27373675e-11] [1.00000000e+00]	[-3.03656885] [0.71213368] [0.45084999] [0.17577522] [-1.00759782] [1.05554712] [-0.52333814] [0.12838148] [-0.01251608] [1.04827943]	[-4.02590138] [0.4204378] [5.17213917] [-2.62711143] [-6.8679544] [9.82419366] [-5.24297793] [1.28407862] [-0.12041145] [1.02660341]

Figure 5: Here are the weights and MSE values for each of the N = 15 models at each level of added Gaussian noise ($\sigma \in \{0, 0.05, 0.2\}$). Several things to note:

- I took screenshots of the console output because I couldn't find a good way to have Python output a more elegant table.
- Some of the values are very, very close to zero (MSE for the 2^{nd} and 9^{th} degree functions when $\sigma = 0$ but are represented as very, very smalls numbers. They are effectively zero.
- The MSE for each of the linear models, regardless of noisiness in the data, are all roughly the same.

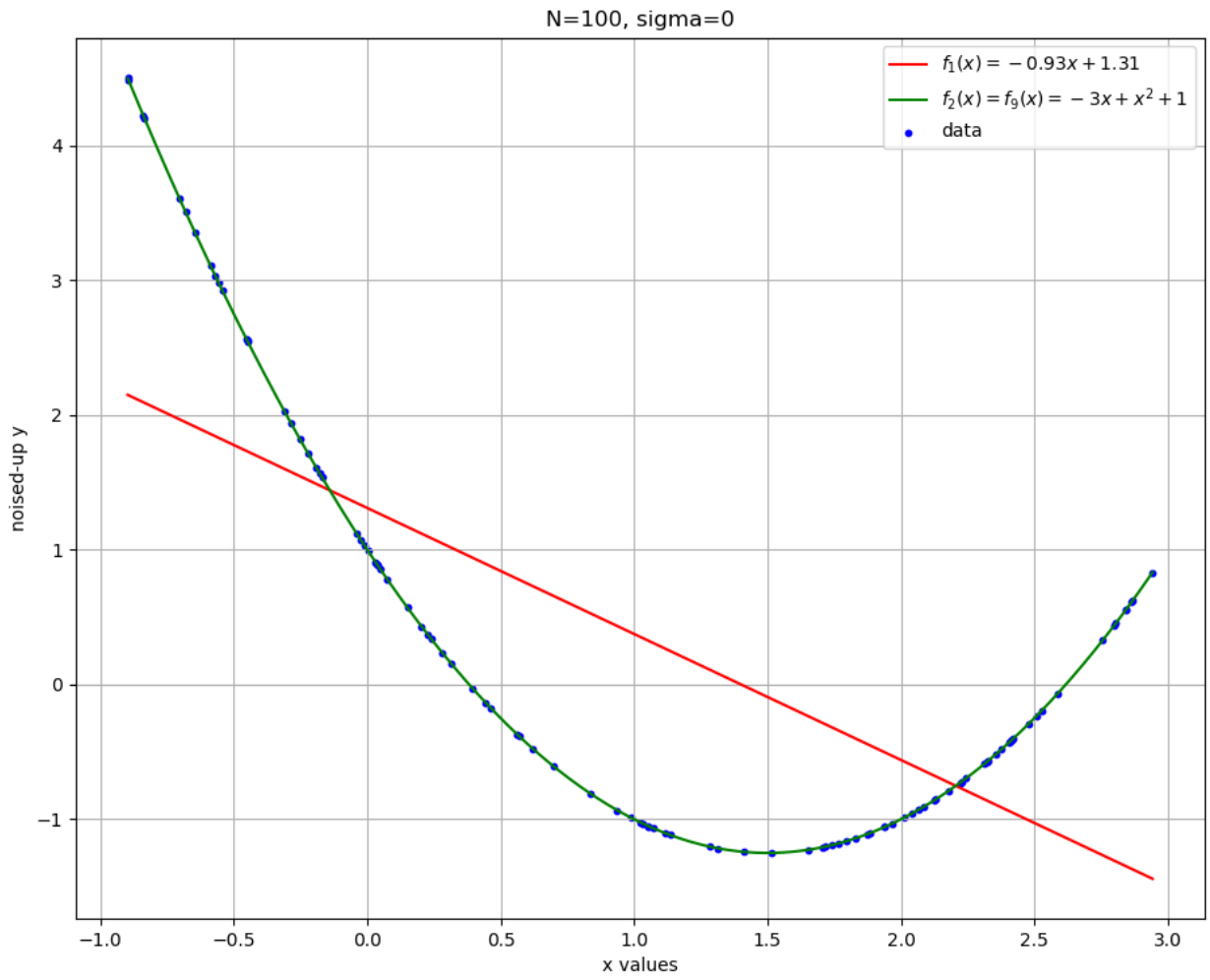

Figure 6: Here is the plot for N = 100 and $\sigma = 0$. Notice in the legend that the 2^{nd} and 9^{th} degree polynomial curves are the same: both are almost exactly the data-generating function of $f(x) = x^2 - 3x + 1$, giving perfect fits. Also as we'd expect, the linear function, $f_1(x)$ again underfits the data because it doesn't have enough terms to accurately capture the data-generating function.

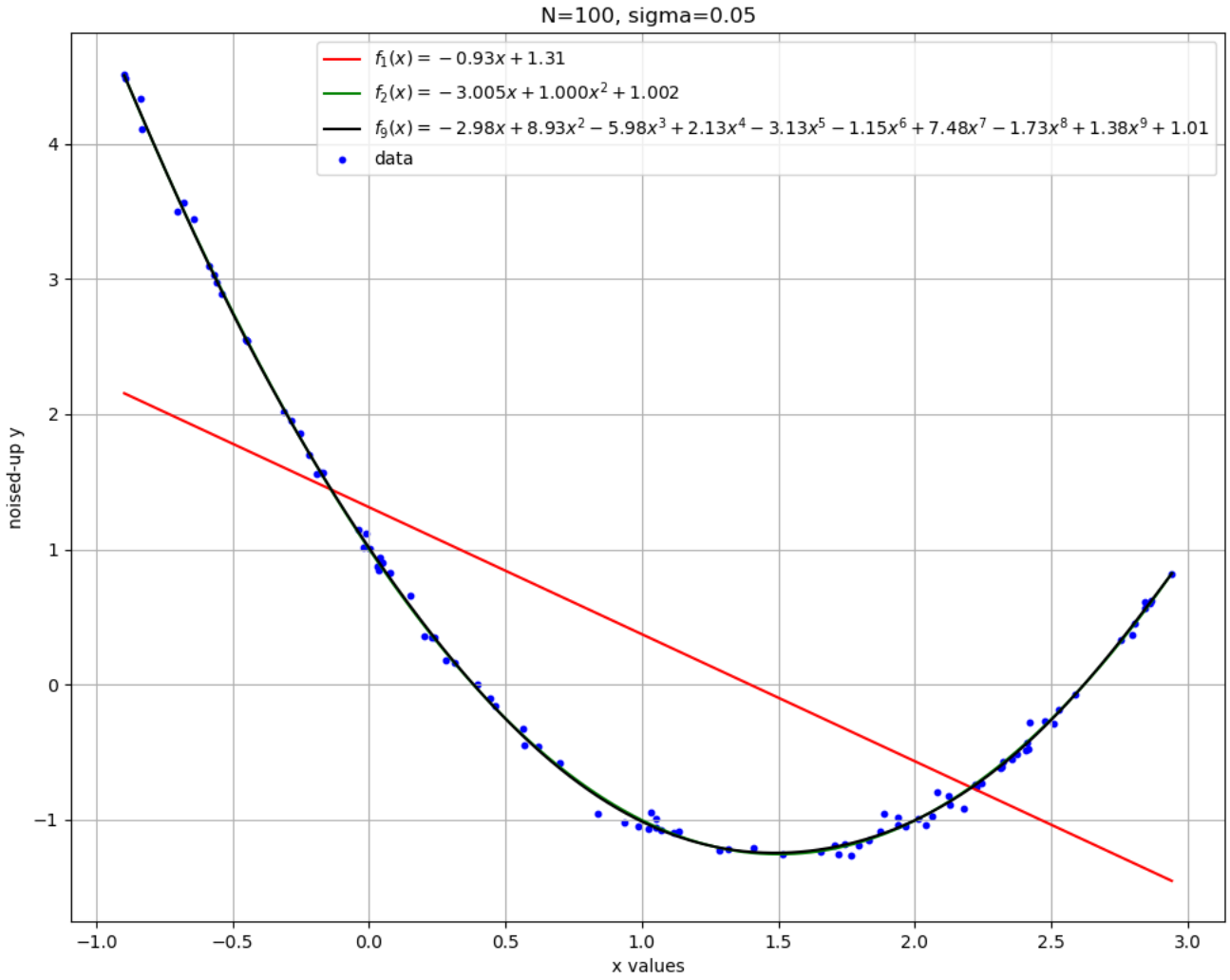

Figure 7: Here is the plot for N = 100 and $\sigma = 0.05$. It is hard to see the color for the f_2 plot (green) since the f_9 curve (black) is very nearly identical. Although the f_9 function has many weights we would've thought would be zero, they again seem to "cancel each other out" in terms of the effect had on the overall curve. In this case, both the f_2 and the f_9 function give an excellent fit to the data.
In this case, the linear function with only an intercept and slope parameter is identical to the line for the $\sigma = 0$ plot, and again underfits the data.

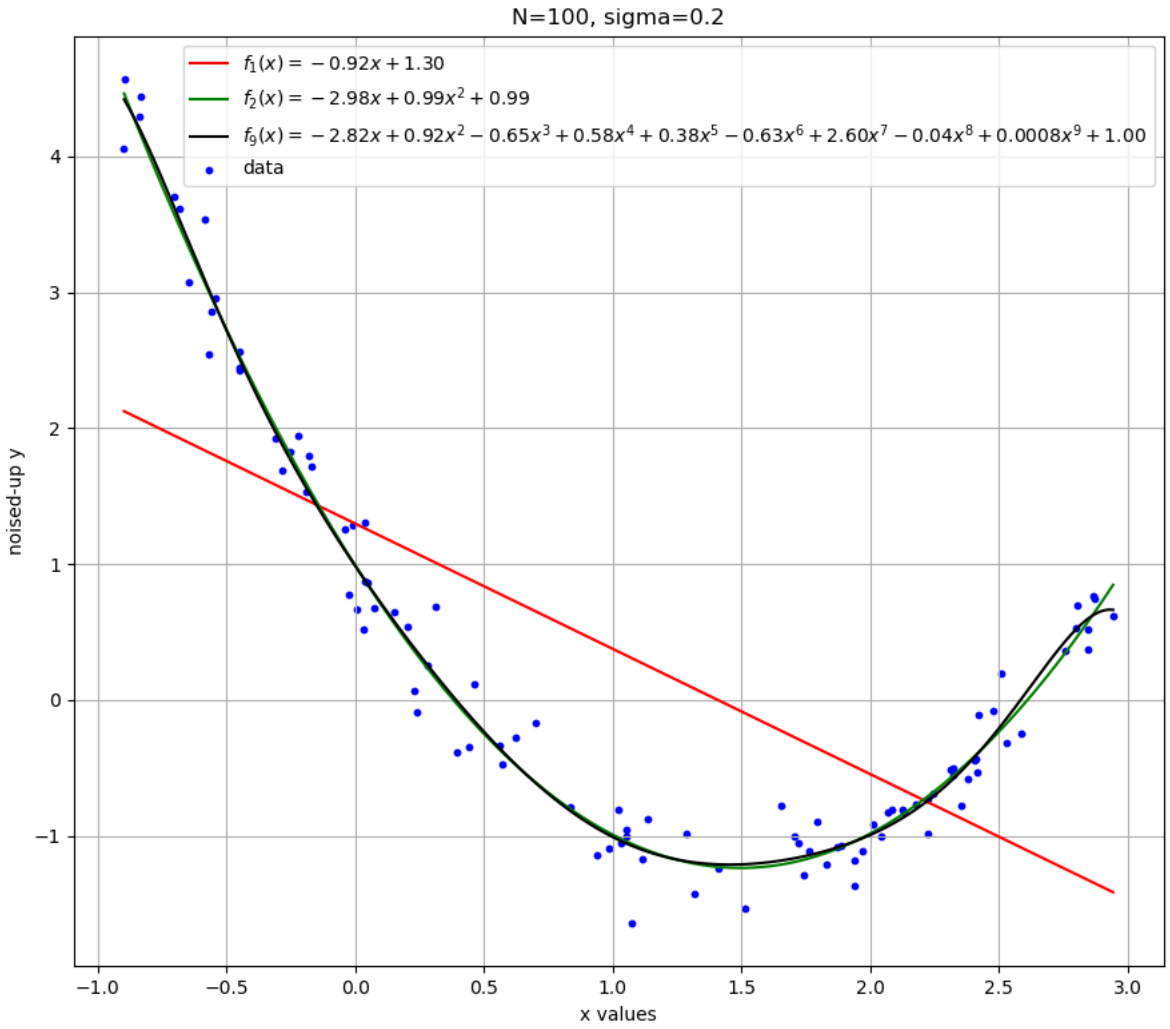

Figure 8: Here is the plot for N = 100 and $\sigma = 0.2$. An interesting finding: the f_2 function for the N = 100 data is better than the f_2 function for the N = 15 data. We can tell this by seeing that the weights are closer to the true values (-3, 1, and 1), and the MSE is two orders of magnitude lower. This is likely because having more noisy data points "balanced each other out" to give a function closer to the true data-generating function.
Here we can see that the 9^{th} order polynomial overfits just a bit on the right side of the plot, but nothing outrageous, and certainly not as bad as for the N = 15 function.
Once again the linear model is underfitting the data. This was the case with all the linear functions.

		N = 100		
		Sigma = 0	Sigma = 0.05	Sigma = 0.2
Plynml of degree 1	MSE	1.147150229703669	1.1471781377270605	1.1473857005466679
	Weights	[-0.93476511] [1.30939514]	[-0.93890759] [1.31177515]	[-0.92177027] [1.29726807]
Plynml of degree 2	MSE	7.84475263738442e-22	2.7980030382750403e-05	0.0002655008947601336
	Weights	[-3.] [1.] [1.]	[-3.00465991] [1.00025054] [1.00230249]	[-2.97643851] [0.99488356] [0.98945593]
Plynml of degree 9	MSE	7.528861745377101e-18	8.73381569788363e-05	0.0012634047382174968
	Weights	[-2.99999999e+00] [1.00000000e+00] [-8.59472493e-10] [-2.80124368e-09] [7.02129910e-10] [2.56659405e-09] [-2.07910489e-09] [6.08451955e-10] [-6.35509423e-11] [1.00000000e+00]	[-2.98295049e+00] [8.93073075e-01] [-5.97726114e-02] [2.12693801e-01] [-3.13052046e-02] [-1.15483292e-01] [7.48234383e-02] [-1.73419036e-02] [1.38168396e-03] [1.01182866e+00]	[-2.81895170e+00] [9.23707487e-01] [-6.53059237e-01] [5.78889338e-01] [3.81292181e-01] [-6.29837495e-01] [2.59900842e-01] [-3.76655166e-02] [7.98810452e-04] [9.85725769e-01]

Figure 9: Here are the weights and MSE values for each of the N = 100 models at each level of added Gaussian noise ($\sigma \in \{0, 0.05, 0.2\}$). Several things to note:

- I again took screenshots of the console output because I couldn't find a good way to have Python output a more elegant table.
- Some of the values are very, very close to zero (MSE for the 2^{nd} and 9^{th} degree functions when $\sigma = 0$ but are represented as very, very smalls numbers. They are effectively zero.
- The MSE for each of the linear models, regardless of noisiness in the data, are all roughly the same.
- As discussed in the caption for the models at N = 100 and $\sigma = 0.2$, the 2^{nd} degree function was much better for the N = 100 data than it was for the N = 15 data. This is likely a result of having more points whose "noisiness" balances each other out.