

IMPLEMENTATION

In this section I will discuss the implementation of the Gradient Descent (GD) algorithm for the given data. The process used is the same for all three data files.

Files

- main.py*: Contains only `main()`, the driver for the assignment. For each file, for every order prediction equation desired, calculates the final resultant weights from GD, plots the data from the file, plots the prediction equation formed from the list of weights, and prints a summary to the terminal. The code as a whole can be run with the command “`python3 main.py`”.
- gdmath.py*: Contains the functions used to perform the operations of GD. Those are: `h()`, `update_weights()`, `GD()`, and `MSE()`.
- csvread.py*: Contains the functions used to read data in from a .csv file, where the first column represents x and the second column represents y , and output a 2-row 2-D array `[x_list, y_list]`.
- visualize.py*: Contains the functions used to plot and visualize data.
- *.csv*: The data files given to be used in the assignment, where the first and second columns represent x and y , respectively.

Miscellaneous Implementation Notes

- I have chosen to assume that for an n^{th} order polynomial, there are weights $\theta_{0\dots n}$ (e.g. a 4th order polynomial is assumed to have weights for all of x^0, x^1, x^2, x^3, x^4). Thus, no basis expansion was used for my implementation, as the operations which would normally involve it can use the index of the weight (in the list of weights) to determine which power x should be raised to. This is seen in the `h()` function.
- The value I have chosen for α is 0.0001, and the GD function finalizes when weights have gone through 10,000 iterations of updating.
- GD updates using all x each iteration. In other words, full-batch GD is used.
- Arbitrarily, weights are initialized to $[\cdot9, \cdot8, \dots, 1 - \frac{1}{o+1}]$, where o represents the desired order of the prediction function. The order value at each term has 1 added to it to prevent division by 0. For example, for a 4th order function, the initial weights will be `[.9, .8, .7, .6, .5]`. This method seems to produce adequately low values of mean squared error (MSE).

Prediction

The function `h()` takes in the current values of the weights (θ_j), and a value of x , and returns what value would be predicted for that x with this hypothesis. This amounts to:

$$h_{\theta}(x) = \sum_{i=0}^o \theta_i x^i$$

Weight Update Function

The function `get_updated_weight_value()` operates as the ordinary Gradient-Descent weight update process does. That is, according to the equation:

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=0}^{m-1} (h_{\theta}(x_i) - y_i) x_i^j$$

Additionally, all weight values are calculated separately, and then updated at the same time, so that no weight's update value is calculated using weight values previously updated during this iteration. This is the purpose of `update_weights()`, which is effectively the driver for the weight updating process.

Gradient Descent

The function `GD()` is the primary driver of the Gradient Descent algorithm. It relies mostly on other functions, implementing the Gradient Descent algorithm with little deviation from its typical implementation, simply updating the weight values a specified number of times using the given data, alpha value, and prediction equation order.

MSE

The function `MSE()` serves to calculate mean squared error. No special implementation decisions are made here. The returned value is simply:

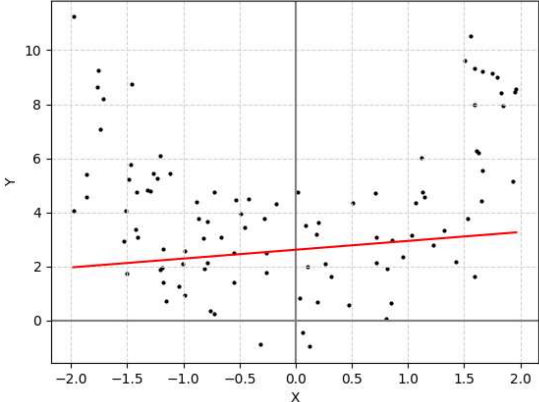
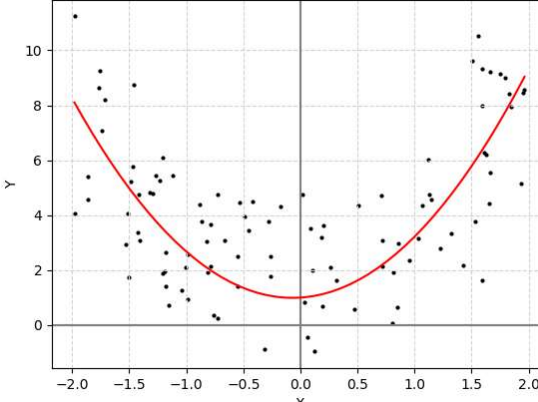
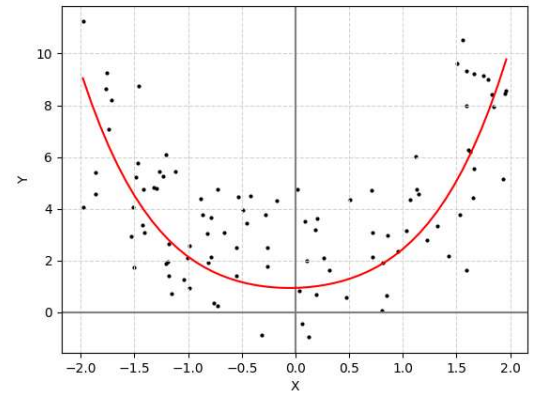
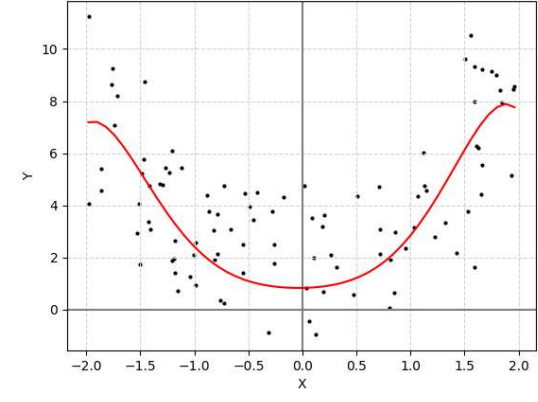
$$MSE = \frac{1}{m} \sum_{i=0}^{m-1} (h(x_i) - y_i)^2$$

Visualization

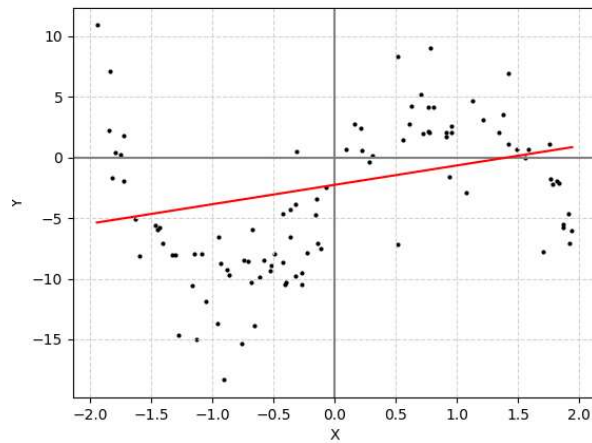
The visualization function `plot_scatter()` serves to visualize the data and, optionally, the trendline produced by the calculated weights. The results of this visualization are at the end of this document.

RESULTS

Note that all of the values presented here are rounded to four decimal places. While the code does use rounded values for calculations, I have rounded them here and in the summaries of `main()` for readability.

SYNTHETIC 1	
<p>1st Order Polynomial</p>  <p>Weights [2.6142, 0.327] Trendline $y = 2.6142 + 0.327x$ MSE: 9.4955 Benchmark MSE: 30</p>	<p>2nd Order Polynomial</p>  <p>Weights [0.9933, 0.2703, 1.9481] Trendline $y = 0.9933 + 0.2703x + 1.9481x^2$ MSE: 4.046 Benchmark MSE: 30</p>
<p>4th Order Polynomial</p>  <p>Weights [0.9375, 0.1174, 1.0691, 0.0315, 0.2855] Trendline $y = 0.9375 + 0.1174x + 1.0691x^2 + 0.0315x^3 + 0.2855x^4$ MSE: 4.4938 Benchmark MSE: 20</p>	<p>7th Order Polynomial</p>  <p>Weights [0.8267, 0.0741, 1.0605, 0.175, 0.936, -0.0313, -0.1979, -0.0025] Trendline $y = 0.8267 + 0.0741x + 1.0605x^2 + 0.175x^3 + 0.936x^4 - 0.0313x^5 - 0.1979x^6 - 0.0025x^7$ MSE: 4.1879 Benchmark MSE: 15</p>

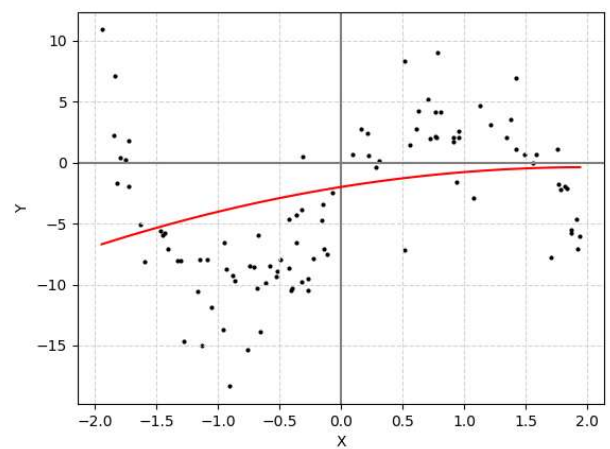
SYNTHETIC 2

1st Order Polynomial**Weights**

[-2.2535, 1.5993]

Trendline

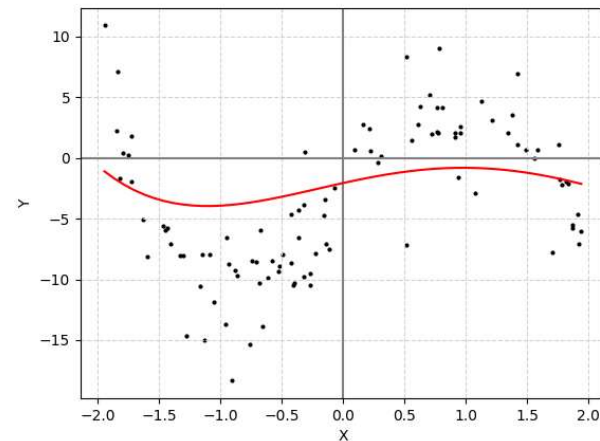
$$y = -2.2535 + 1.5993x$$

MSE: 32.9554**Benchmark MSE:** 602nd Order Polynomial**Weights**

[-1.9919, 1.6266, -0.4071]

Trendline

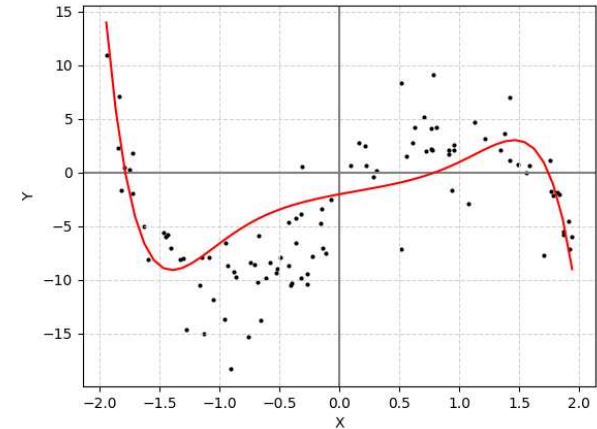
$$y = -1.9919 + 1.6266x - 0.4071x^2$$

MSE: 33.3938**Benchmark MSE:** 604th Order Polynomial**Weights**

[-2.0643, 2.2171, -0.4545, -0.6563, 0.1512]

Trendline

$$y = -2.0643 + 2.2171x - 0.4545x^2 - 0.6563x^3 + 0.1512x^4$$

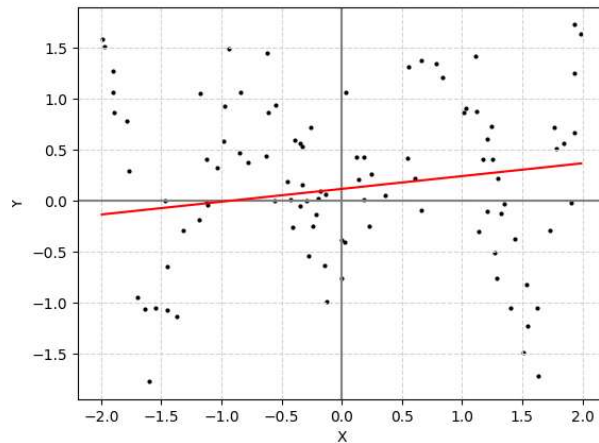
MSE: 28.1123**Benchmark MSE:** 607th Order Polynomial**Weights**

[-2.044, 2.0992, -0.6054, 1.4298, -0.4303, 0.6844, 0.2383, -0.4289]

Trendline

$$y = -2.044 + 2.0992x - 0.6054x^2 + 1.4298x^3 - 0.4303x^4 + 0.6844x^5 + 0.2383x^6 - 0.4289x^7$$

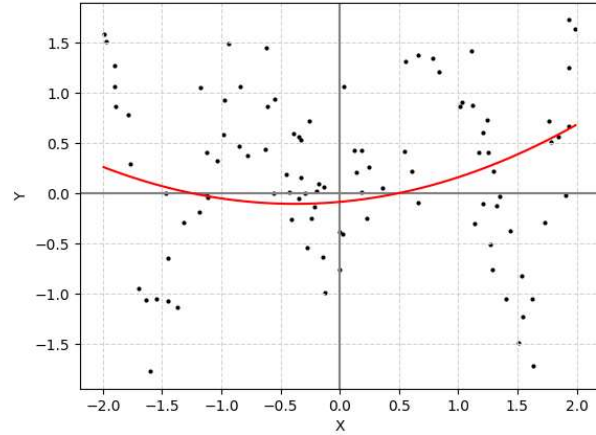
MSE: 18.5084**Benchmark MSE:** 60

SYNTHETIC 3**1st Order Polynomial****Weights**

[0.1183, 0.1255]

Trendline

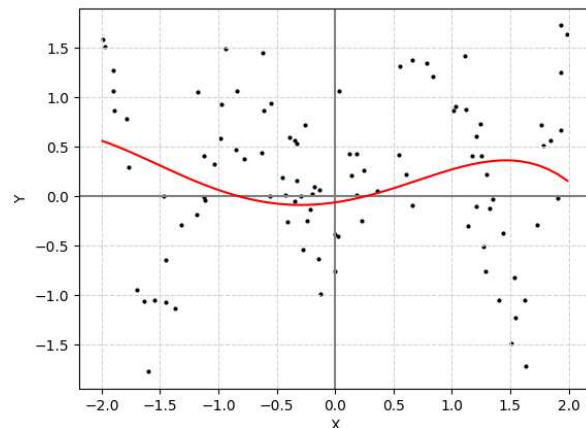
$$y = 0.1183 + 0.1255x$$

MSE: 0.6571**Benchmark MSE: 0.75****2nd Order Polynomial****Weights**

[-0.0833, 0.1052, 0.1397]

Trendline

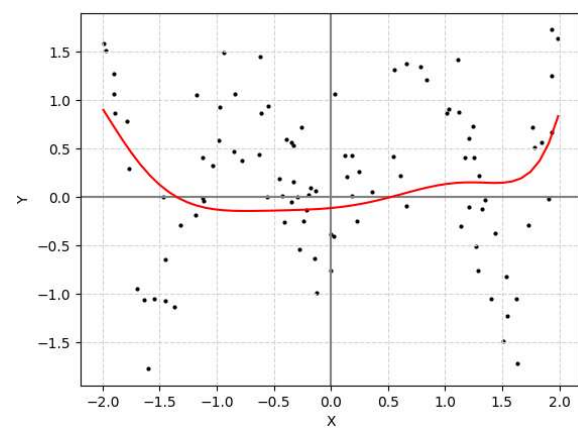
$$y = -0.0833 + 0.1052x + 0.1397x^2$$

MSE: 0.6757**Benchmark MSE: 0.75****4th Order Polynomial****Weights**

[-0.0615, 0.1724, 0.2711, -0.0689, -0.0417]

Trendline

$$y = -0.0615 + 0.1724x + 0.2711x^2 - 0.0689x^3 - 0.0417x^4$$

MSE: 0.6826**Benchmark MSE: 0.7****7th Order Polynomial****Weights**

[-0.1123, 0.1083, 0.1814, 0.1249, -0.0939, -0.1231, 0.0278, 0.0211]

Trendline

$$y = -0.1123 + 0.1083x + 0.1814x^2 + 0.1249x^3 - 0.0939x^4 - 0.1231x^5 + 0.0278x^6 + 0.0211x^7$$

MSE: 0.6109**Benchmark MSE: 0.7**