A.I. Linear regression

Our github repository for this assignment(Group11) :https://github.com/BrokenYouth0707/Machine_Learning_Task1

Dataset

x	x
27	442
26	442
20	457
31	431
14	471
16	463
25	434
7	487
6	480
10	480
15	470
31	430
6	489
28	436

26	444
16	464
28	447
24	430
11	475
24	445

Z-score Standardization

We used z-score standardization to normalize our data. Z-score standardization puts all features to the same scale by setting their mean to zero and standard deviation to one. By doing so, it preserves the original shape of the data, as the transformation only shifts and rescales the values without changing their distribution and internal relationships. This standardization is useful because many models—such as k-NN, linear regression and logistic regression—assume that the input data are centered and scaled. Moreover, it is easy to interpret: a z-score of 1 just indicates that a value lies one standard deviation above the mean. Z-score is defined as:

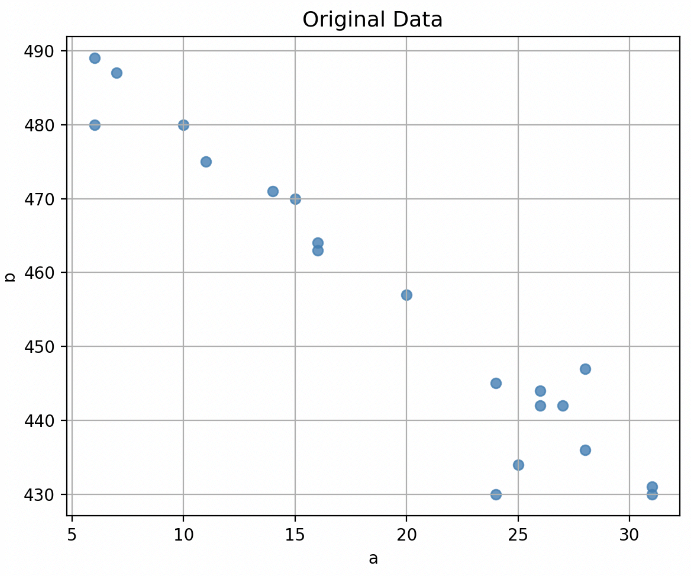
$$z_i = \frac{x_i - \mu}{\sigma}, \tag{1}$$

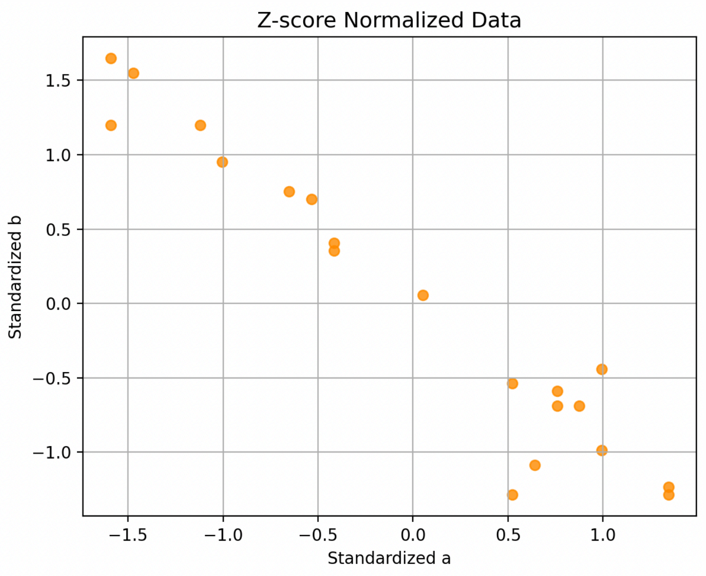
where

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{2}$$

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{3}$$

We comparted the data through pictures:





x	y
0.8744979860368924	-0.6881427304417103
0.7571157060319403	-0.6881427304417103
0.05282202600222832	0.057138205054725624
1.3440271060567004	-1.2346820831390966
-0.6514716540274837	0.7527337448513991
-0.4167070940175797	0.3552505792533
0.6397334260269883	-1.0856258960398095
-1.4731476140621478	1.5477000760475974
-1.5905298940670998	1.1999023061492606
-1.1210007740472918	1.1999023061492606
-0.5340893740225318	0.7030483491516367
1.3440271060567004	-1.284367478838859
-1.5905298940670998	1.6470708674471222
0.9918802660418444	-0.9862551046402847
0.7571157060319403	-0.5887719390421855

-0.4167070940175797	0.4049359749530624
0.9918802660418444	-0.4397157519428983
0.5223511460220364	-1.284367478838859
-1.0036184940423398	0.9514753276504487
0.5223511460220364	-0.5390865433424231

Least Square

For one demintional feature, we can simply use:

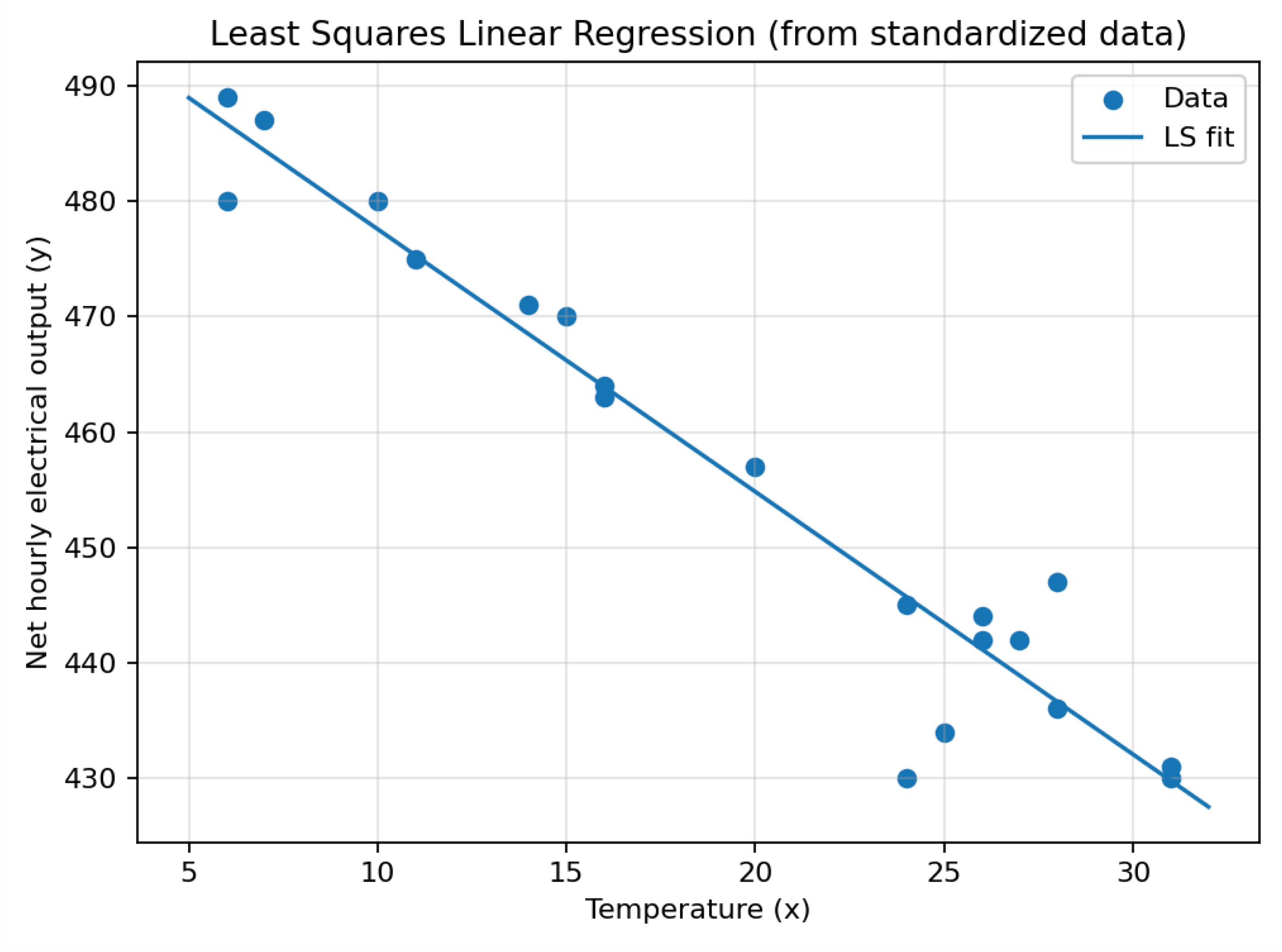
$$m = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4}$$

to caculate its slope, and then using it to calculate the intercept:

$$b = \bar{y} - m\bar{x}. \tag{5}$$

Finially, we can get the equation of the line:

$$y = -2.275898x + 500.343812. \tag{6}$$



Linear regression with Gradient Descent

We define the hypothesis function as

$$h_\theta(x_i) = \theta_0 + \theta_1 x_i, \quad i = 1, \ldots, m \tag{7}$$

where m is the numbers of training samples.

We choose the **Mean Squared Error** as the cost function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \tag{8}$$

Reason
MSE is smooth and differentiable, allowing gradient-based optimization and it measures the average squared difference between predicted and true values, and the square term penalizes large errors more strongly, helping the algorithm converge more stably.

Paritial Derivative

Partial derivative with respect to θ_0:

$$\begin{aligned}\frac{\partial J}{\partial \theta_0} &= \frac{1}{2m}\sum_{i=1}^{m} 2(h_\theta(x_i) - y_i)\frac{\partial h_\theta(x_i)}{\partial \theta_0} \\ &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)\end{aligned} \tag{9}$$

Partial derivative with respect to θ_1:

$$\begin{aligned}\frac{\partial J}{\partial \theta_1} &= \frac{1}{2m}\sum_{i=1}^{m} 2(h_\theta(x_i) - y_i)\frac{\partial h_\theta(x_i)}{\partial \theta_1} \\ &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_i\end{aligned} \tag{10}$$

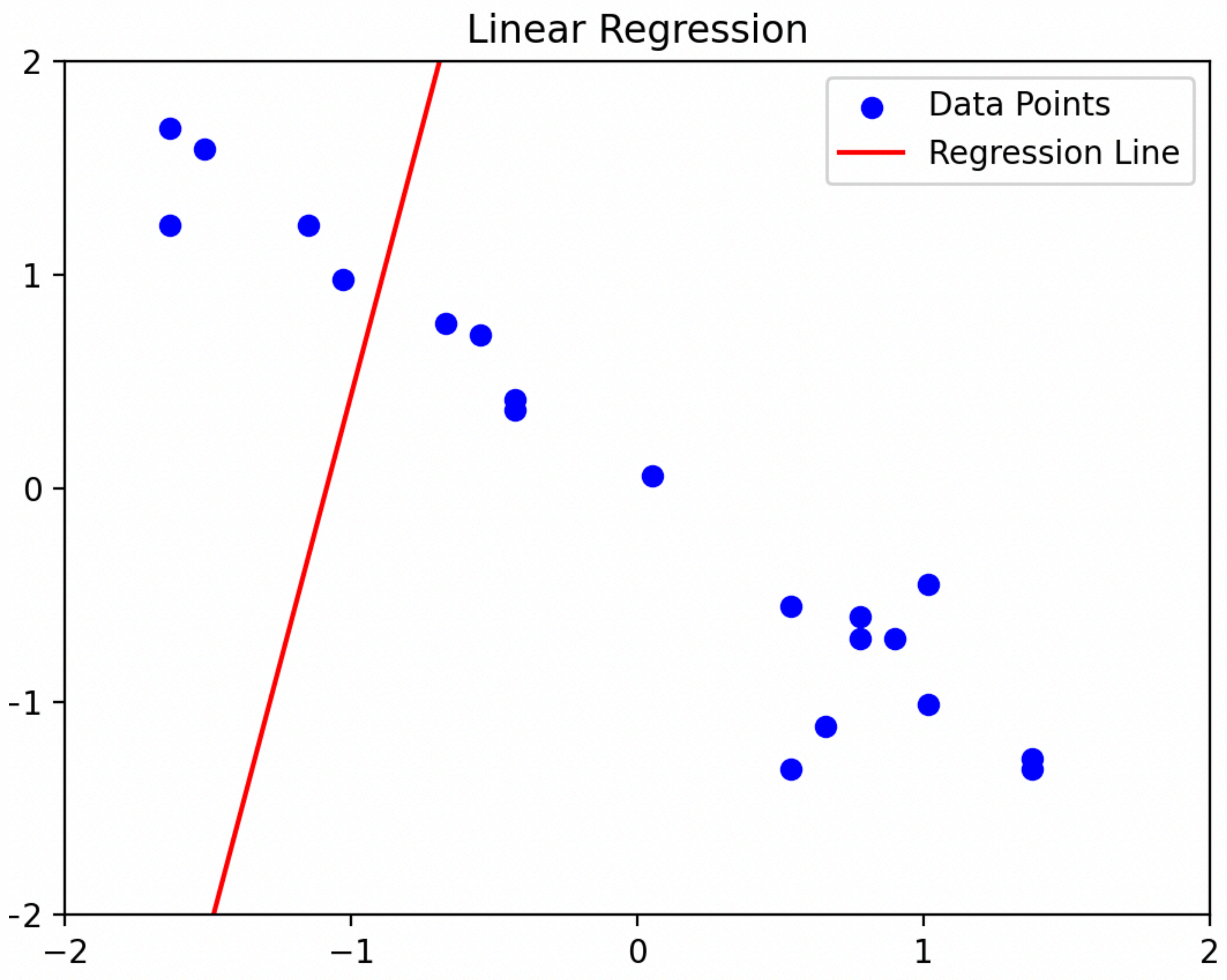
The gradient descent update rules are given by:

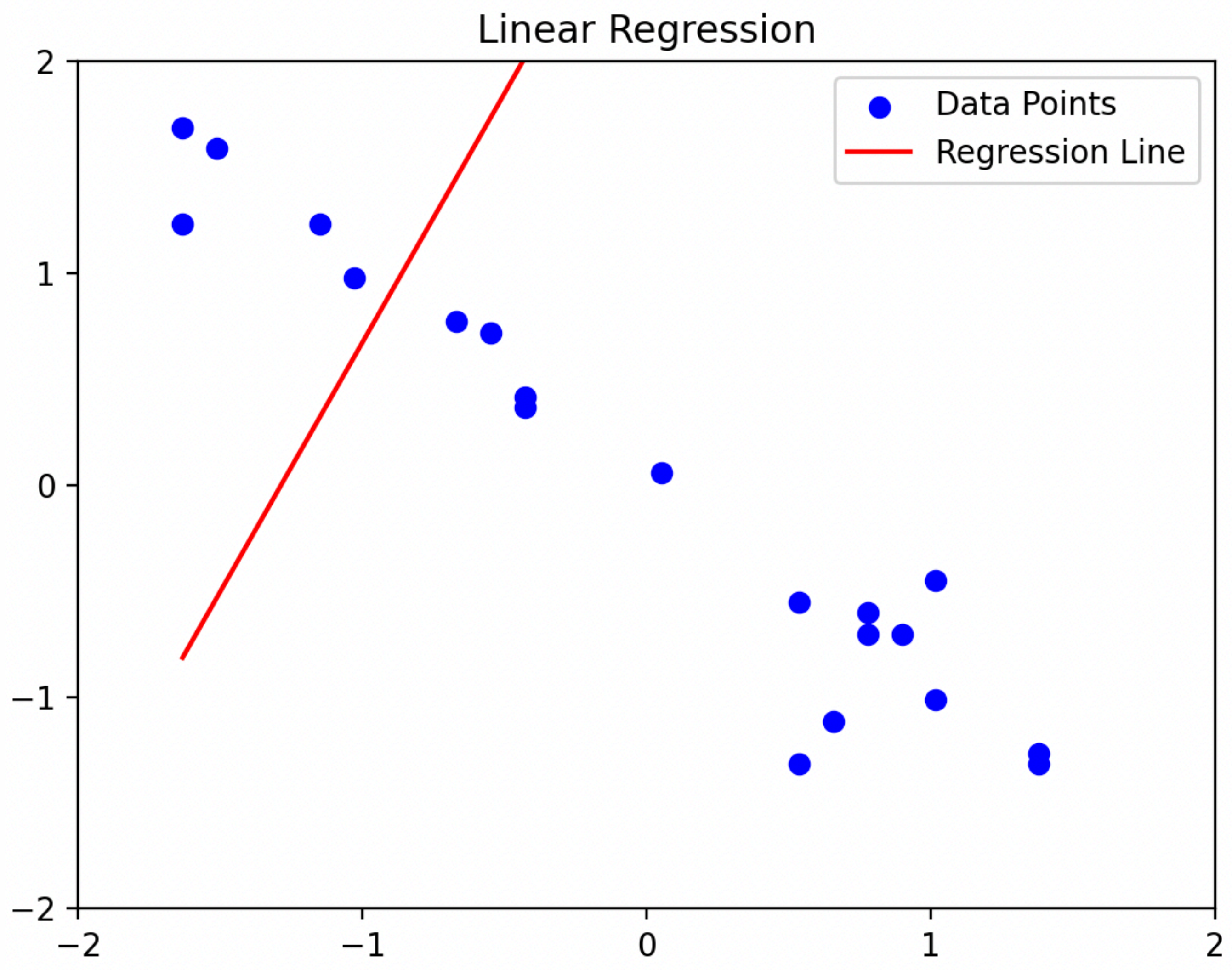
$$\theta_0 \leftarrow \theta_0 - \alpha \cdot \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i) \tag{11}$$

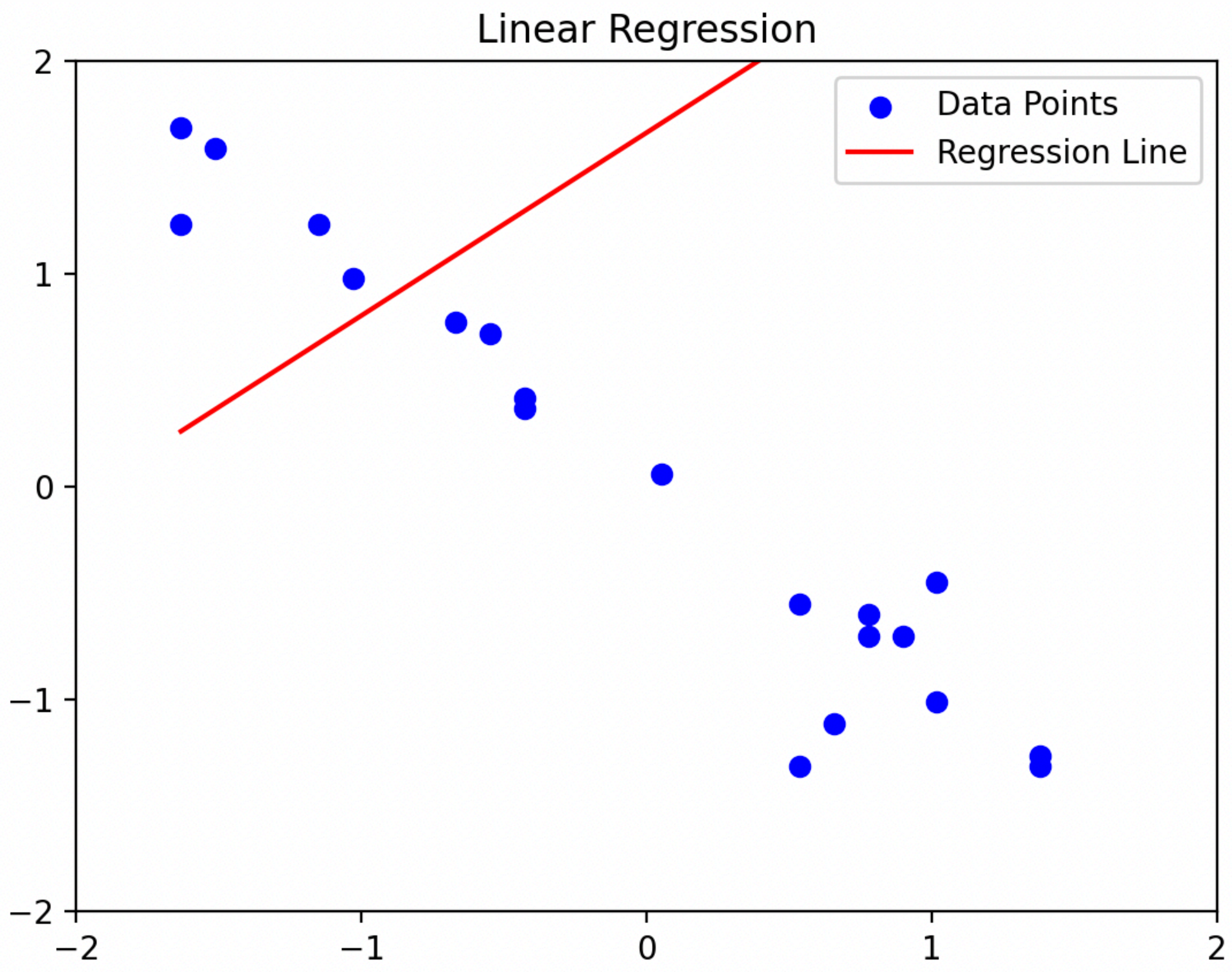
$$\theta_1 \leftarrow \theta_1 - \alpha \cdot \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_i \tag{12}$$

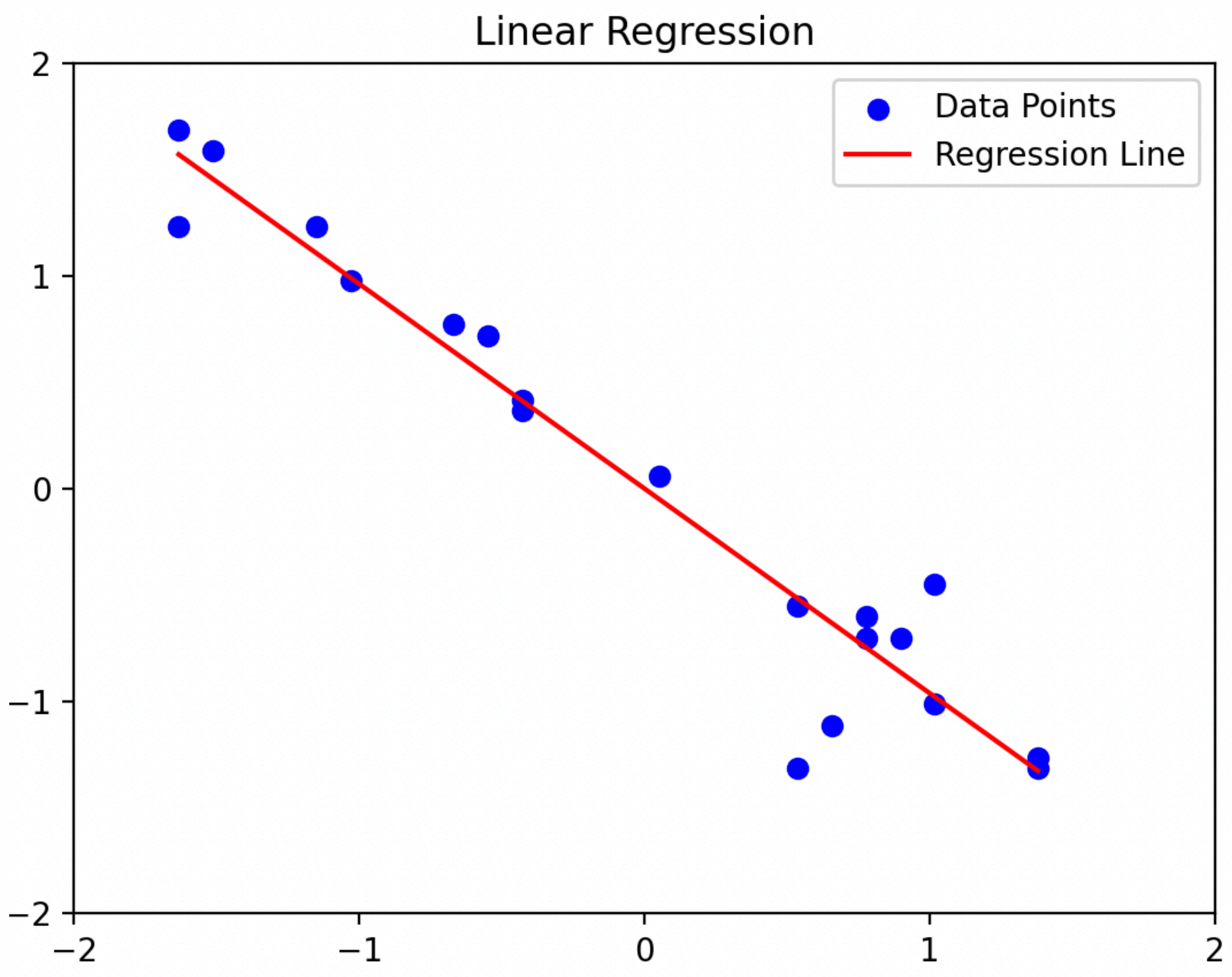
We started with $(\theta_0, \theta_1) = (10, 10)$ and a learning rate of 0.9; after the first, second, and third iterations, the parameters were updated to $(5.5, 5.07)$, $(3.03, 2.35)$, and $(1.66, 0.86)$ respectively, and finally converged to about $(0, -0.96)$.

Here are the regression lines after the first, second, and third iterations, as well as the final converged line:

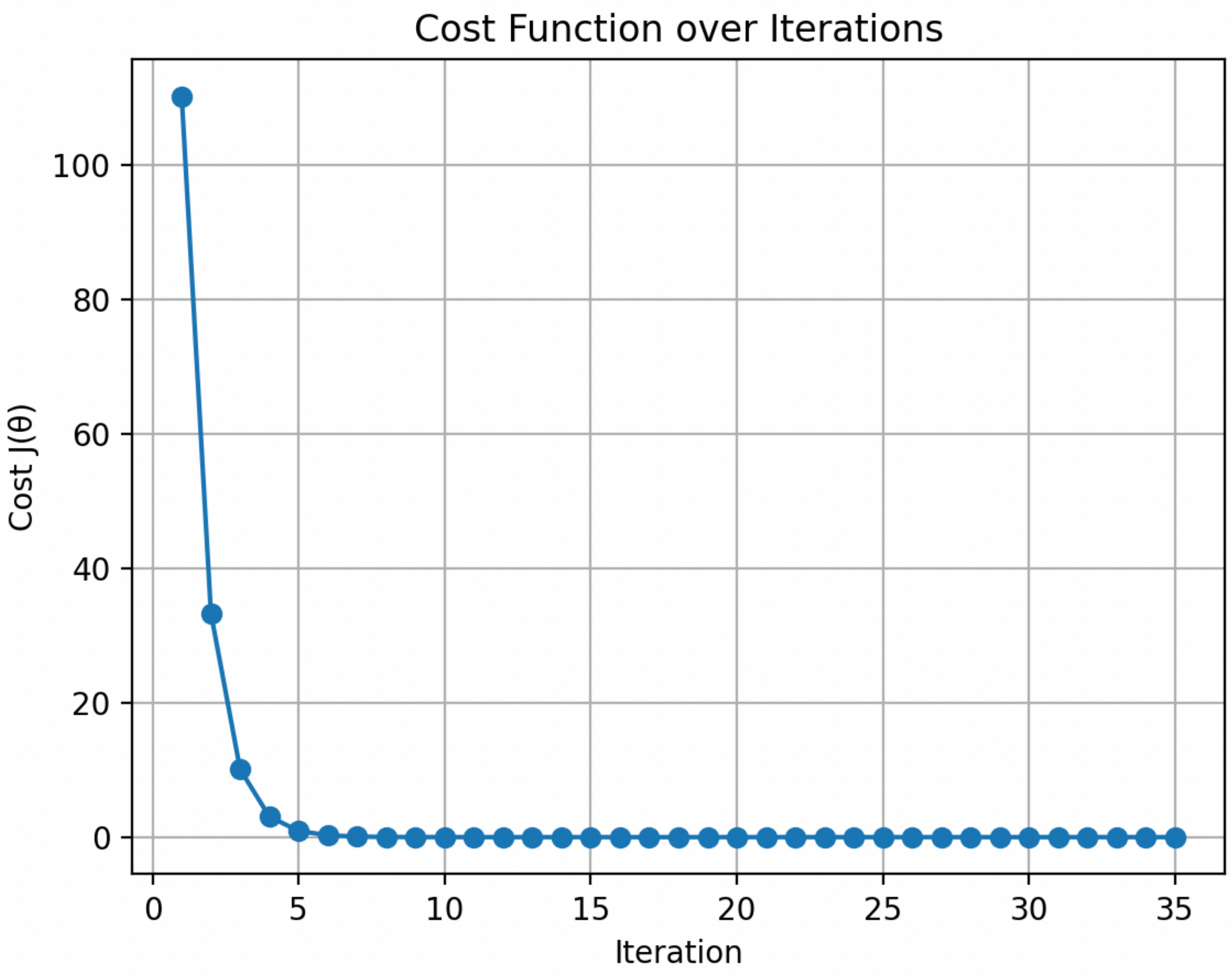

Linear Regression
Data Points
Regression Line
2
1
0
-1
-2
−2
−1
0
1
2


Linear Regression
Data Points
Regression Line
2
1
0
−1
−2
−2
−1
0
1
2


Linear Regression
Data Points
Regression Line
2
1
0
−1
−2
−2
−1
0
1
2



Here is the picture for the cost:



Comparision

Similarities:

Both methods aim to find the optimal parameters for a linear regression model.

Differences:

- Least Squares: Computes the analytical (closed-form) solution in one step without iteration. However, when the feature dimension is large, calculating the inverse matrix can be very slow. It generally has low bias but high variance.
- Gradient Descent: An iterative method that gradually approaches the optimal solution. It is more suitable for large-scale datasets or high-dimensional features. The results are affected by the choice of learning rate and initial values. It typically has low variance but high bias.

Performance:

Least Squares performs well on small to medium-sized datasets with relatively few features. It provide an exact solution efficiently. By contrast, Gradient Descent performs better on large-scale or high-dimensional datasets, in which computing the matrix inverse in Least Squares may be computationally expensive. However, Gradient Descent may require more time to converge and its performance depends on proper hyperparameter tuning.

A.II. Poly regression

(1) Partial Derivatives of the Cost Function

Given the cost function:

$$J(\theta) = \frac{1}{4n} \sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)}) \right)^4 \tag{13}$$

where the hypothesis function is:

$$h_\theta(x^{(i)}) = \theta_2 (x^{(i)})^2 + \theta_1 x^{(i)} + \theta_0 \tag{14}$$

We define the error term as:

$$\varepsilon^{(i)} = y^{(i)} - h_\theta(x^{(i)}) \tag{15}$$

Thus, the cost function can be rewritten as:

$$J(\theta) = \frac{1}{4n} \sum_{i=1}^{n} (\varepsilon^{(i)})^4 \tag{16}$$

Derivatives with respect to each parameter:

With respect to θ_2:

$$\frac{\partial J}{\partial \theta_2} = -\frac{1}{n} \sum_{i=1}^{n} (\varepsilon^{(i)})^3 (x^{(i)})^2 \tag{17}$$

With respect to θ_1:

$$\frac{\partial J}{\partial \theta_1} = -\frac{1}{n} \sum_{i=1}^{n} (\varepsilon^{(i)})^3 x^{(i)} \tag{18}$$

With respect to θ_0:

$$\frac{\partial J}{\partial \theta_0} = -\frac{1}{n}\sum_{i=1}^{n}(\varepsilon^{(i)})^3 \tag{19}$$

(2) Parameter Update Rules

We use gradient descent to update the parameters.
The general update rule is:

$$\theta_j := \theta_j - \alpha\frac{\partial J}{\partial \theta_j} \tag{20}$$

where:

- α is the learning rate,
- $\frac{\partial J}{\partial \theta_j}$ is the gradient for parameter θ_j.

Specifically, in out problem, the update rules for each parameter are:

$$\theta_2 := \theta_2 - \alpha\left(-\frac{1}{n}\sum_{i=1}^{n}(\varepsilon^{(i)})^3(x^{(i)})^2\right) \tag{21}$$

$$\theta_1 := \theta_1 - \alpha\left(-\frac{1}{n}\sum_{i=1}^{n}(\varepsilon^{(i)})^3 x^{(i)}\right) \tag{22}$$

$$\theta_0 := \theta_0 - \alpha\left(-\frac{1}{n}\sum_{i=1}^{n}(\varepsilon^{(i)})^3\right) \tag{23}$$

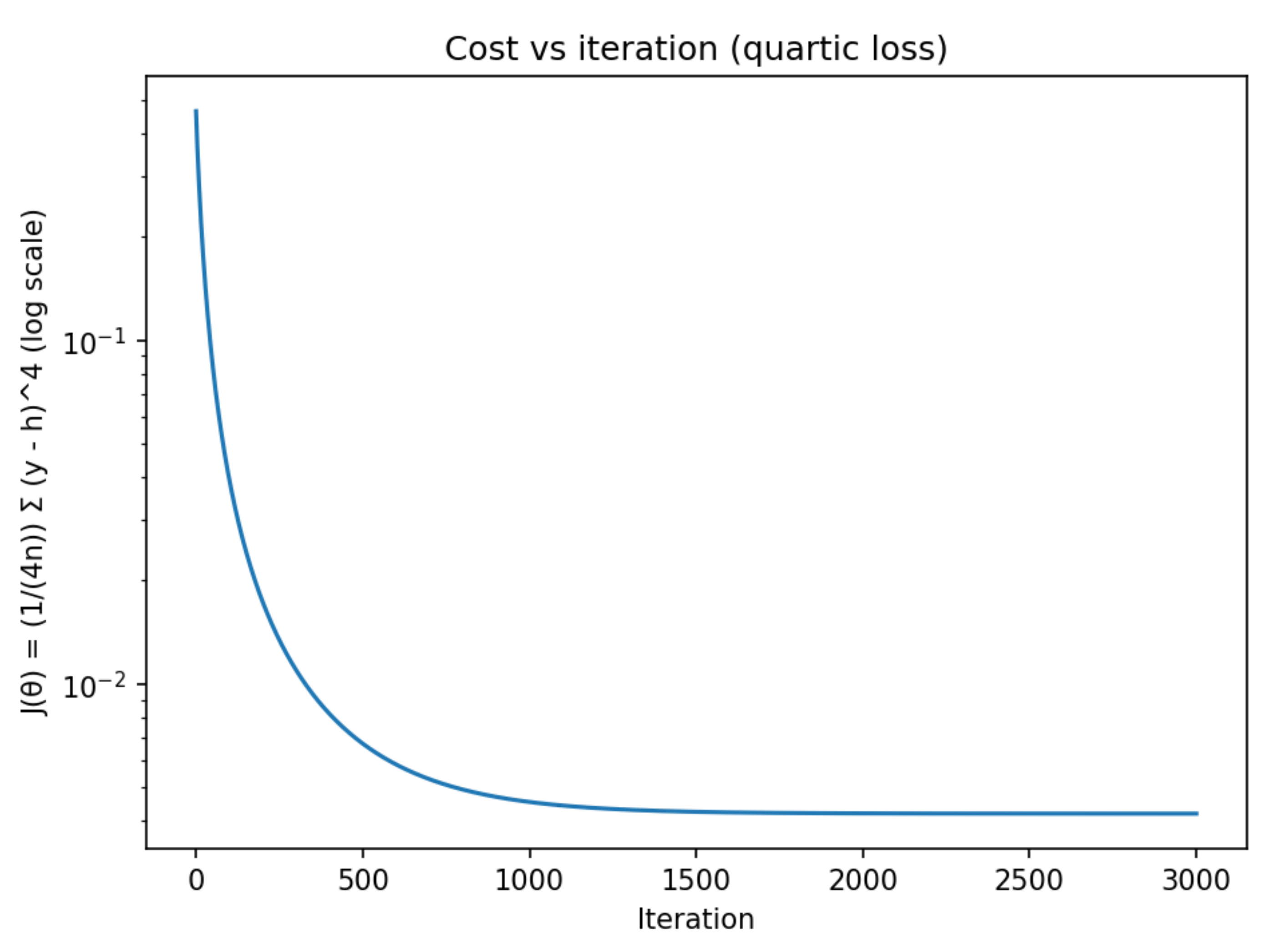
(3) Iterations

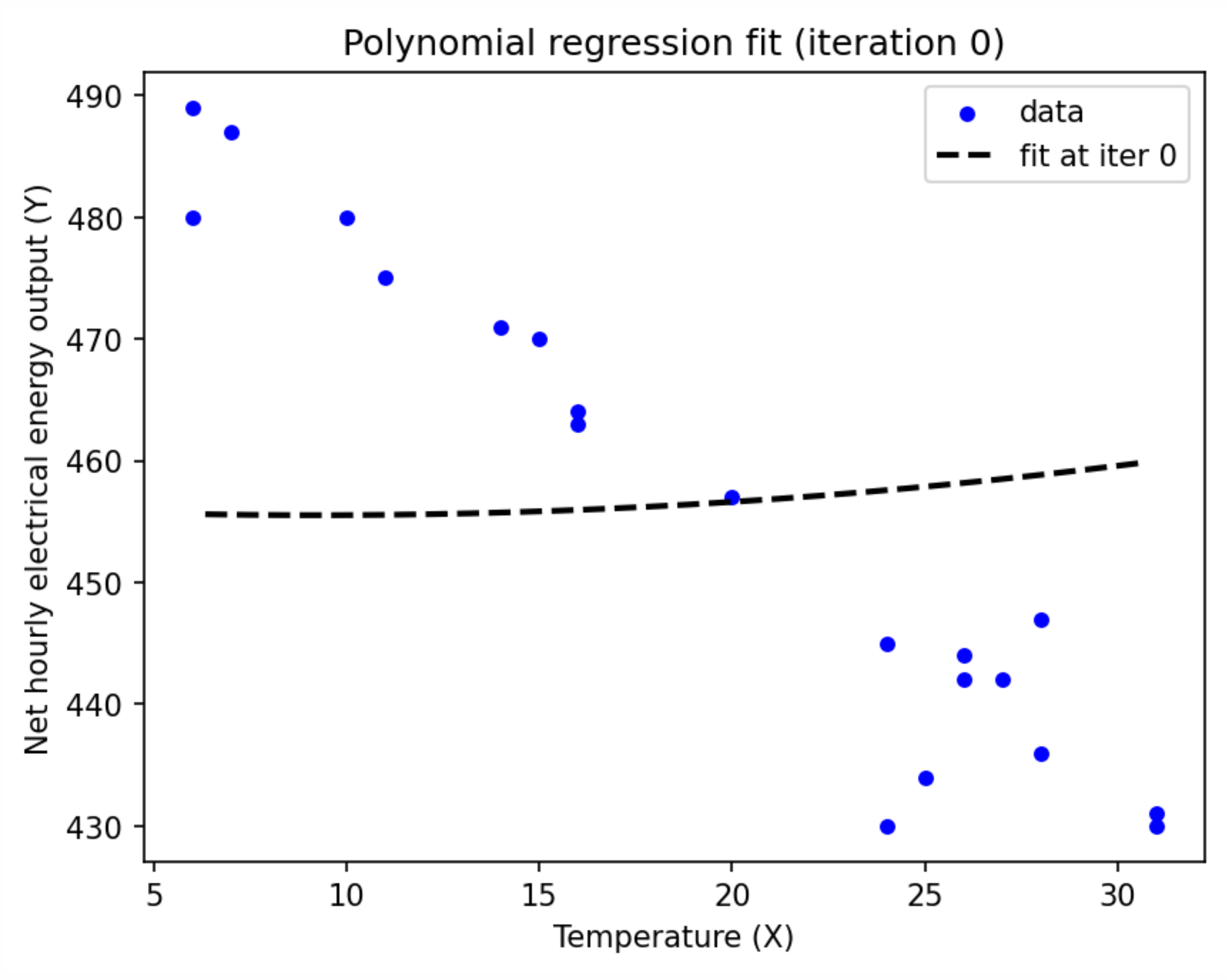
Parameter evolution and Cost during training(learning rate $\alpha = 0.01$):

Iteration	θ0	θ1	θ2	Cost (J)
0	0.03455842	0.08216181	0.03304371	0.4643122
100	-0.06609397	-0.4526044	0.09791056	0.03924249
200	-0.1092821	-0.5906548	0.1053334	0.01732243
500	-0.1683796	-0.7493620	0.1199282	0.006708472
1000	-0.2047500	-0.8408934	0.1345493	0.004537533

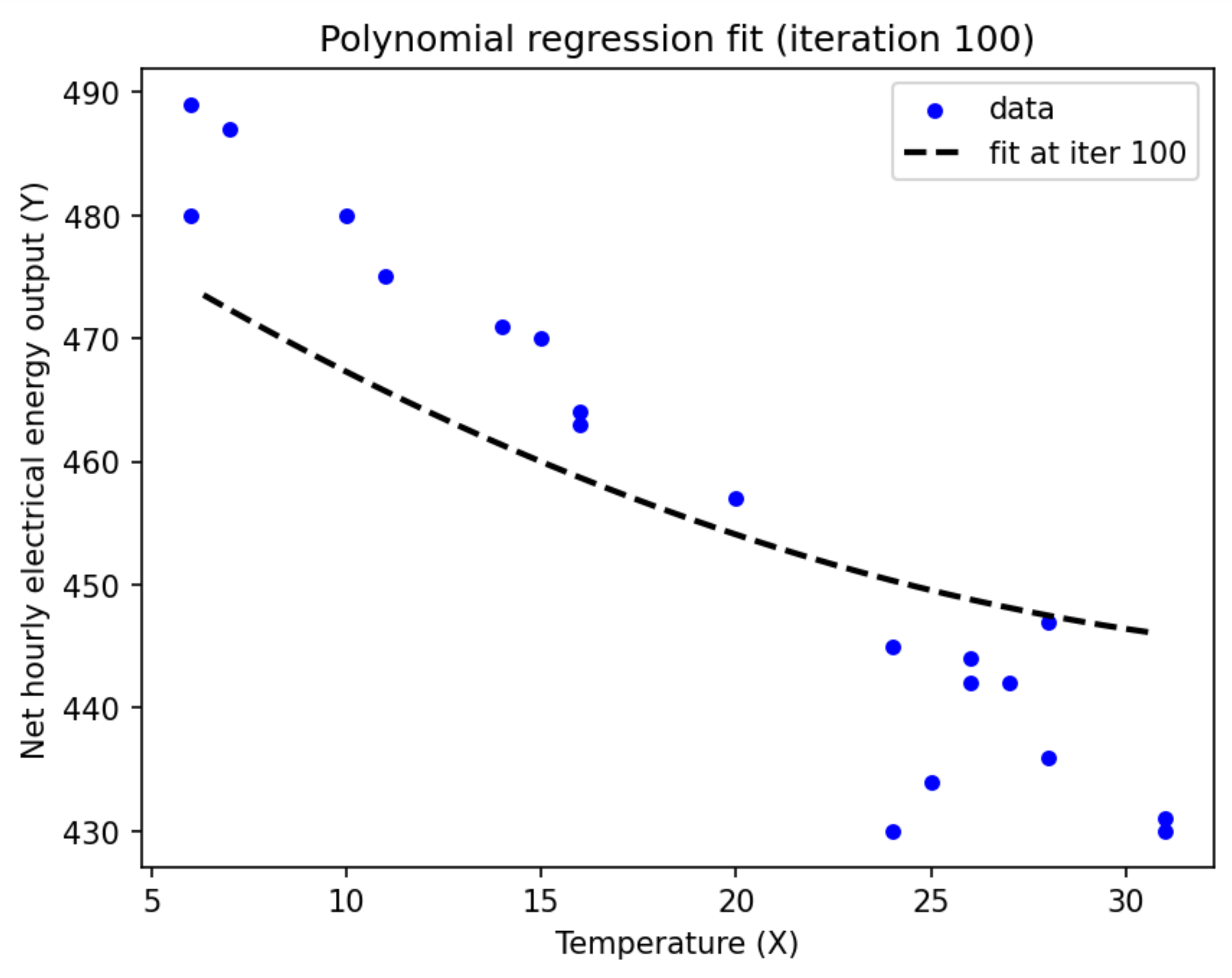
2000	-0.2221575	-0.8875152	0.1457154	0.004204270
3000	-0.2249141	-0.8943075	0.1478593	0.004196765

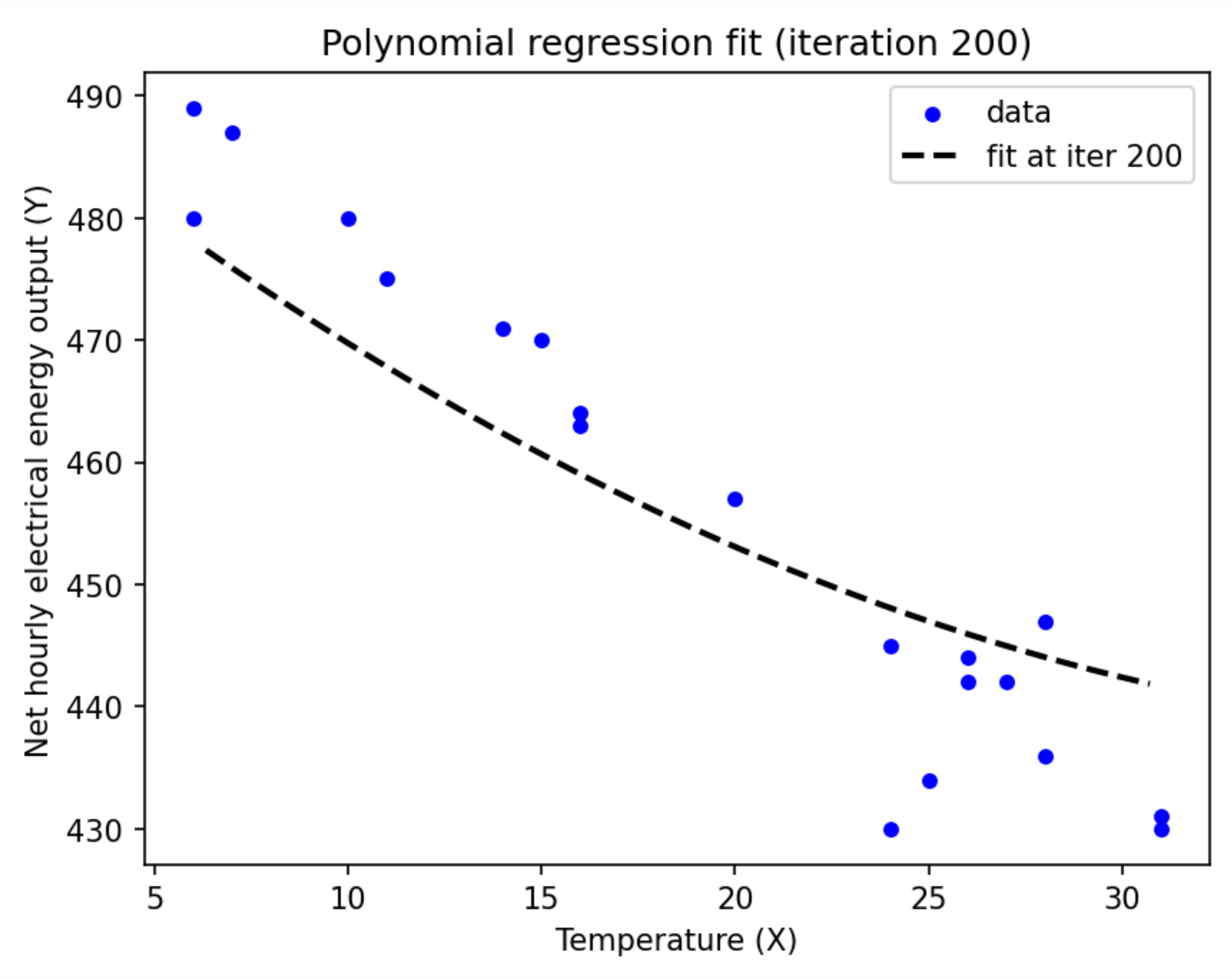
The first image illustrates the decrease of the cost function over iterations, while the others display the fitting results at different stages. It is obvious that the model becomes stable and shows good convergence after about 1,000 iterations.

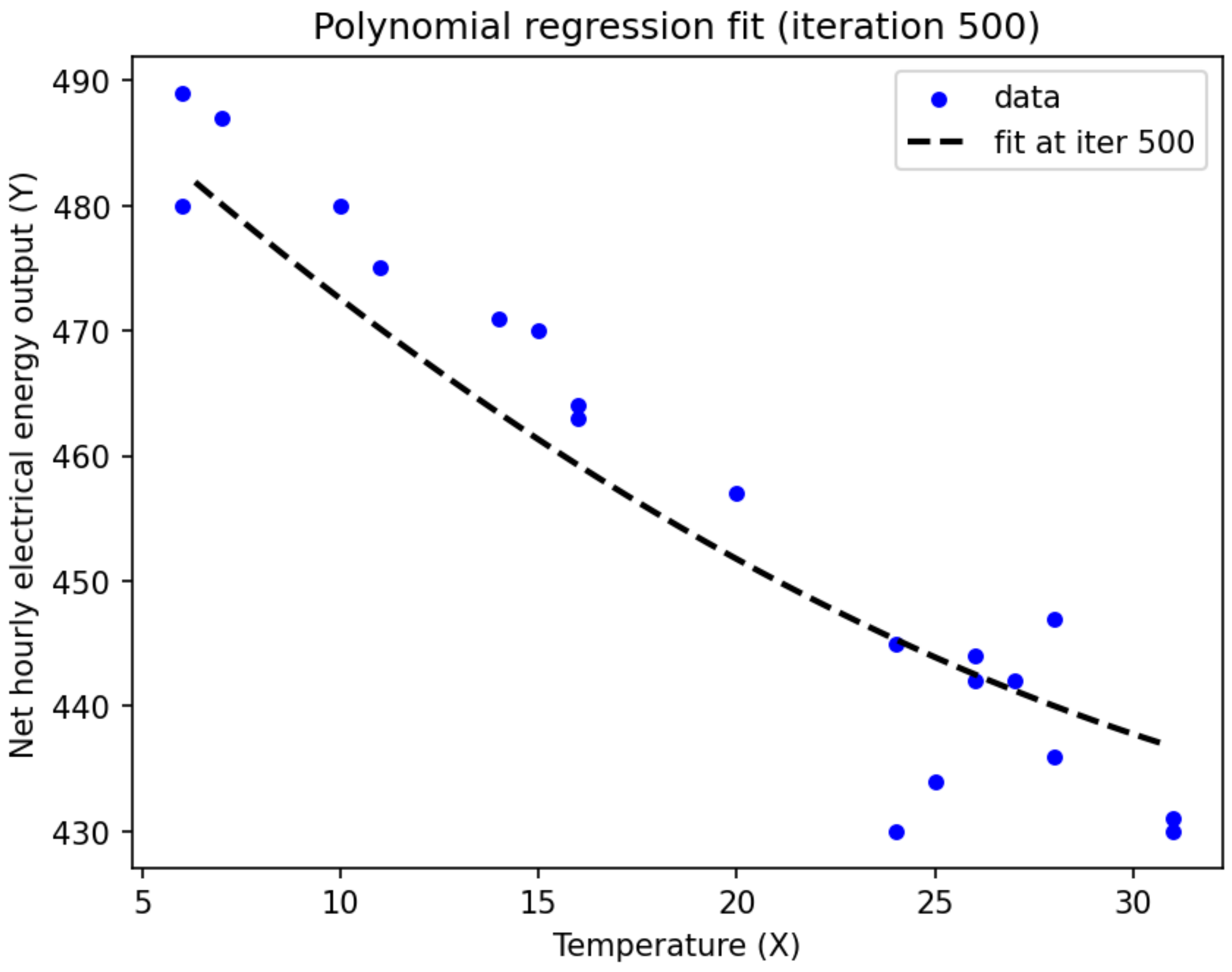


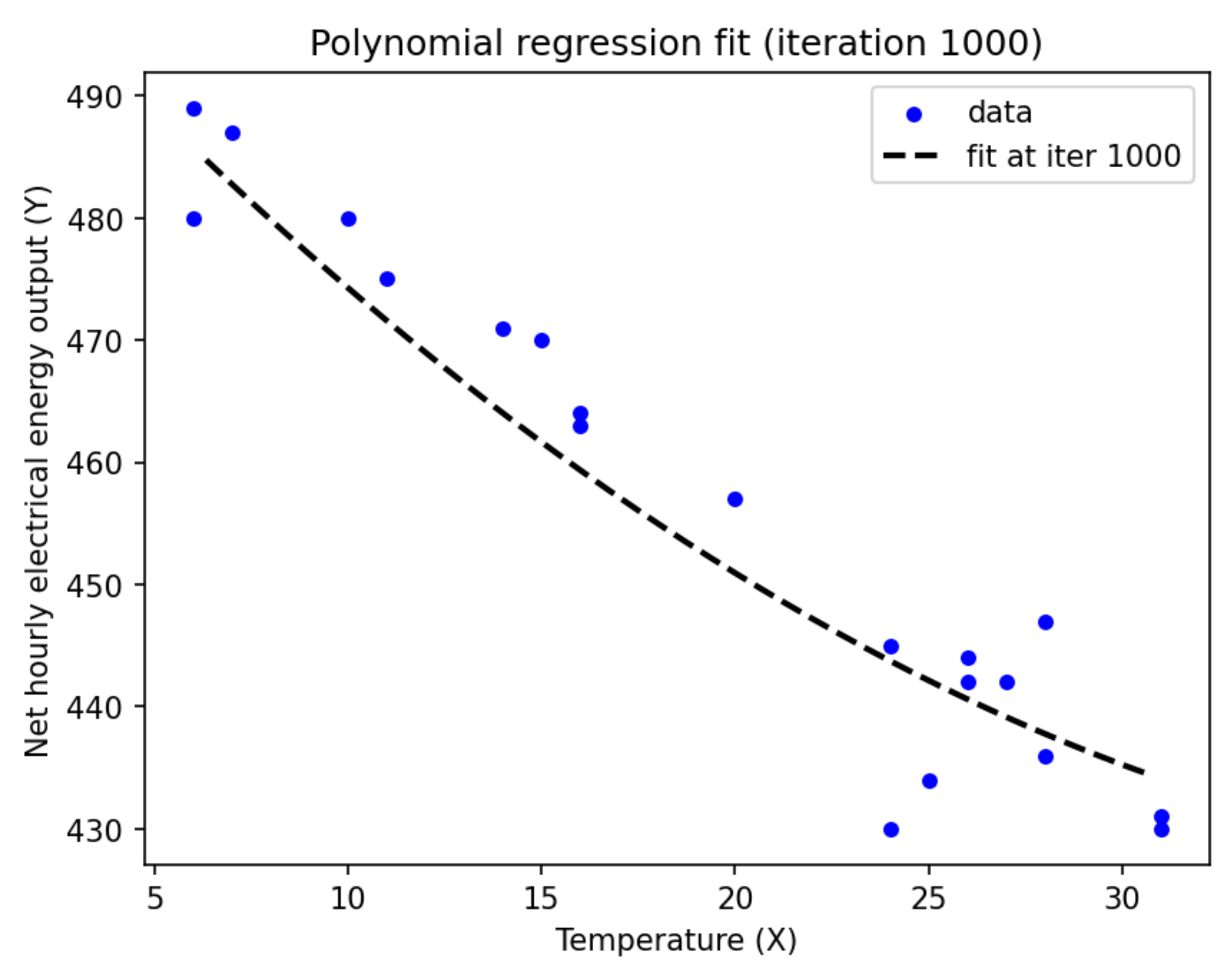


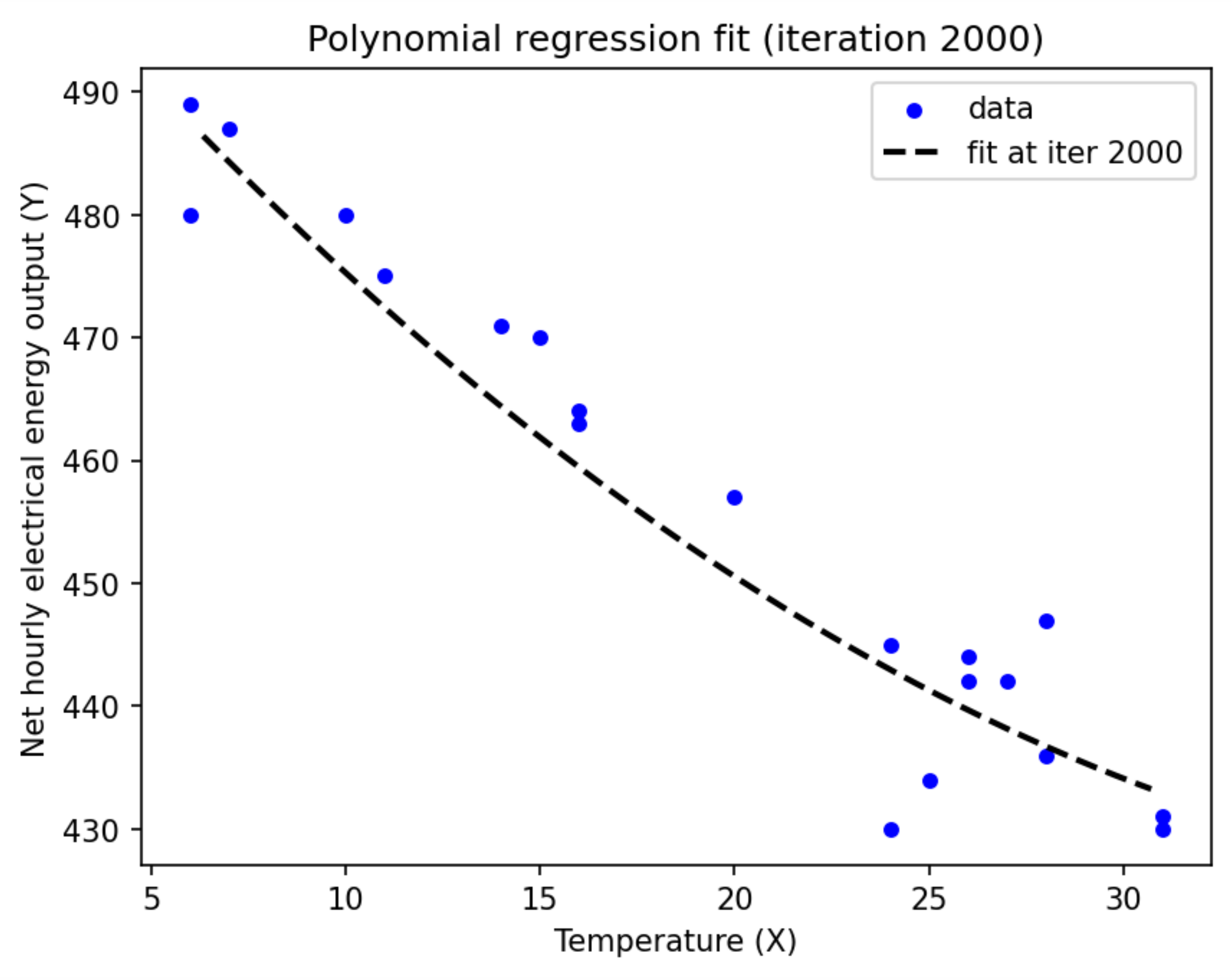
Polynomial regression fit (iteration 0)
data
fit at iter 0
Net hourly electrical energy output (Y)
Temperature (X)
490
480
470
460
450
440
430
5
10
15
20
25
30

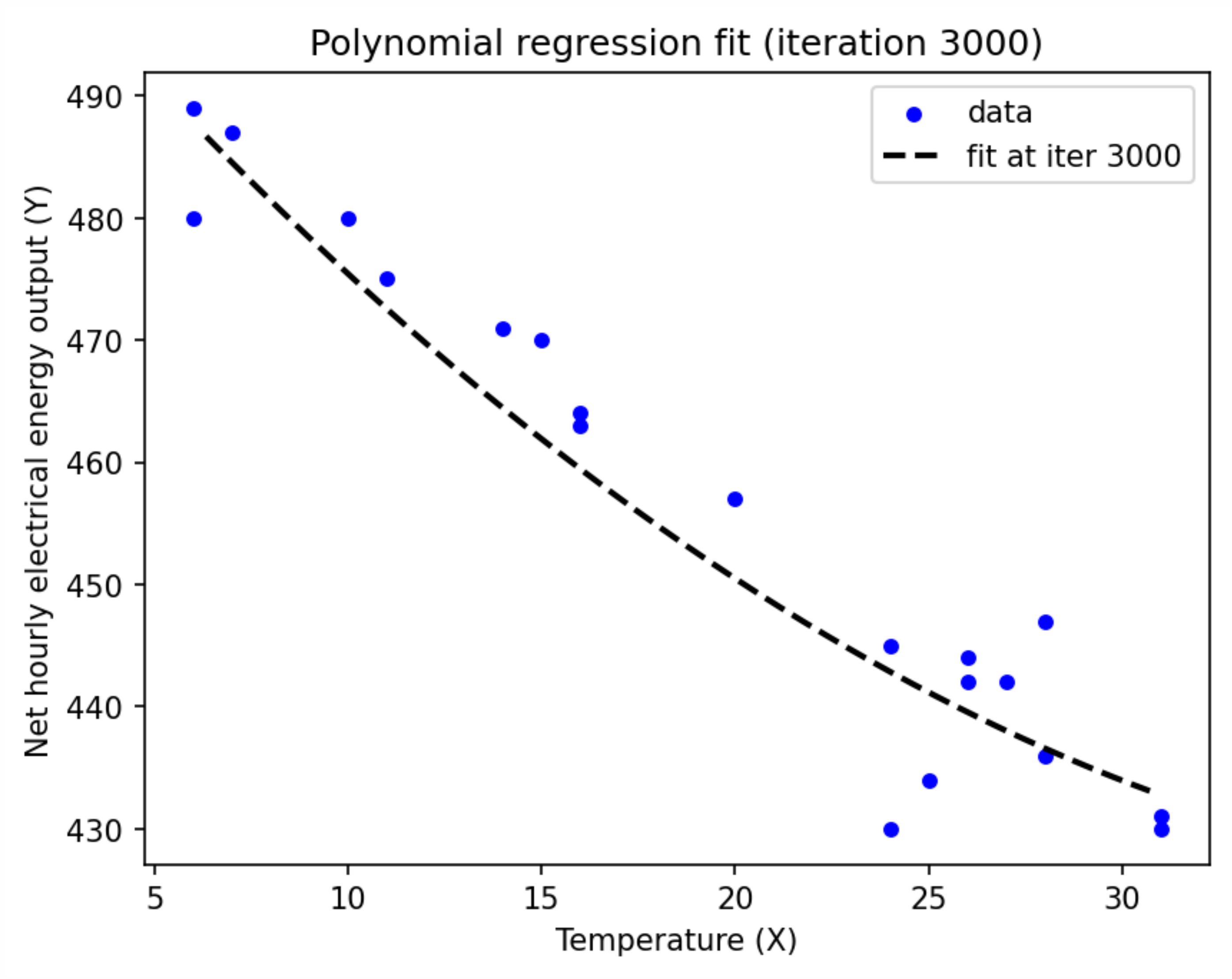

Polynomial regression fit (iteration 100)
data
fit at iter 100
Net hourly electrical energy output (Y)
Temperature (X)
490
480
470
460
450
440
430
5
10
15
20
25
30


Polynomial regression fit (iteration 200)
data
fit at iter 200
Net hourly electrical energy output (Y)
Temperature (X)
490
480
470
460
450
440
430
5
10
15
20
25
30


Polynomial regression fit (iteration 500)
data
fit at iter 500
Net hourly electrical energy output (Y)
Temperature (X)
430
440
450
460
470
480
490
5
10
15
20
25
30


Polynomial regression fit (iteration 1000)
data
fit at iter 1000
Net hourly electrical energy output (Y)
Temperature (X)
490
480
470
460
450
440
430
5
10
15
20
25
30


Polynomial regression fit (iteration 2000)
Net hourly electrical energy output (Y)
Temperature (X)
data
fit at iter 2000
490
480
470
460
450
440
430
5
10
15
20
25
30



Similarities

The two are very consistent in their core principles and objectives: their goal is the same: regression algorithms within supervised learning. They aim to find a model (a line or a curve) that minimizes error on the training data. This error is typically measured by Mean Squared Error or the Sum of Squared Errors .

They have the same mathematical principle : The term “linear” in “linear regression” refers to linearity with respect to the model parameters (coefficients β), not necessarily linearity with respect to the features.

Univariate linear regression model:

$$y = \beta_0 + \beta_1 x \tag{24}$$

This model is linear in the parameters β_0 and β_1.

For Second-order polynomial regression model:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 \tag{25}$$

The model remains linear in the parameters $\beta_0, \beta_1, \beta_2$.

In addition, they have the same approach to get $h(\vec{x})$: Least Square and Gradient Descent.

Differences

Views	**Linear Regression**	**Polynomial Regression**
Assumed Relationship	Assumes a linear relationship between features (X) and target (Y).	Assumes a non-linear relationship between features (X) and target (Y).
Model Form	$y = \beta_0 + \beta_1 x + \ldots \beta_n x_n$	$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n$
Graphical Representation	A straight line.	A curve.
Model Complexity	Low. The model is simple.	High (depends on the degree n).
Bias	High Bias. It will underfit if the relationship is non-linear.	Low Bias. Can fit the training data better.
Variance	Low Variance. Not sensitive to small changes in training data.	High Variance. Prone to overfitting, especially with a high degree n.