# CSE4088 Introduction to Machine Learning
# Homework 3

Berk Kırtay - 150118043

## Gradient Descent

For this section, Gradient_Descent.py is implemented. Script can be run to observe results. Plotting is included.
Results after running Gradient_Descent.py:

```
$ py Gradient_Descent.py
Gradient Descent:
Number of iterations: 10
Final(u,v): (0.04473629039778207, 0.023958714099141746)
Error rate after completion: 1.2086833944220747e-15
Coordinate Descent:
Number of iterations: 15
Final(u,v): (6.29707589930517, -2.852306954077811)
Error rate after completion: 0.13981379199615315
```

### 4.

The partial derivative of E(u,v) with respect to $\frac{\partial E}{\partial u}$ is

$$\frac{\partial E}{\partial u} = 2 \cdot (u \cdot e^{v} - 2 \cdot v \cdot e^{-u}) \cdot (e^{v} + 2 \cdot v \cdot e^{-u})$$

### 5. and 6.

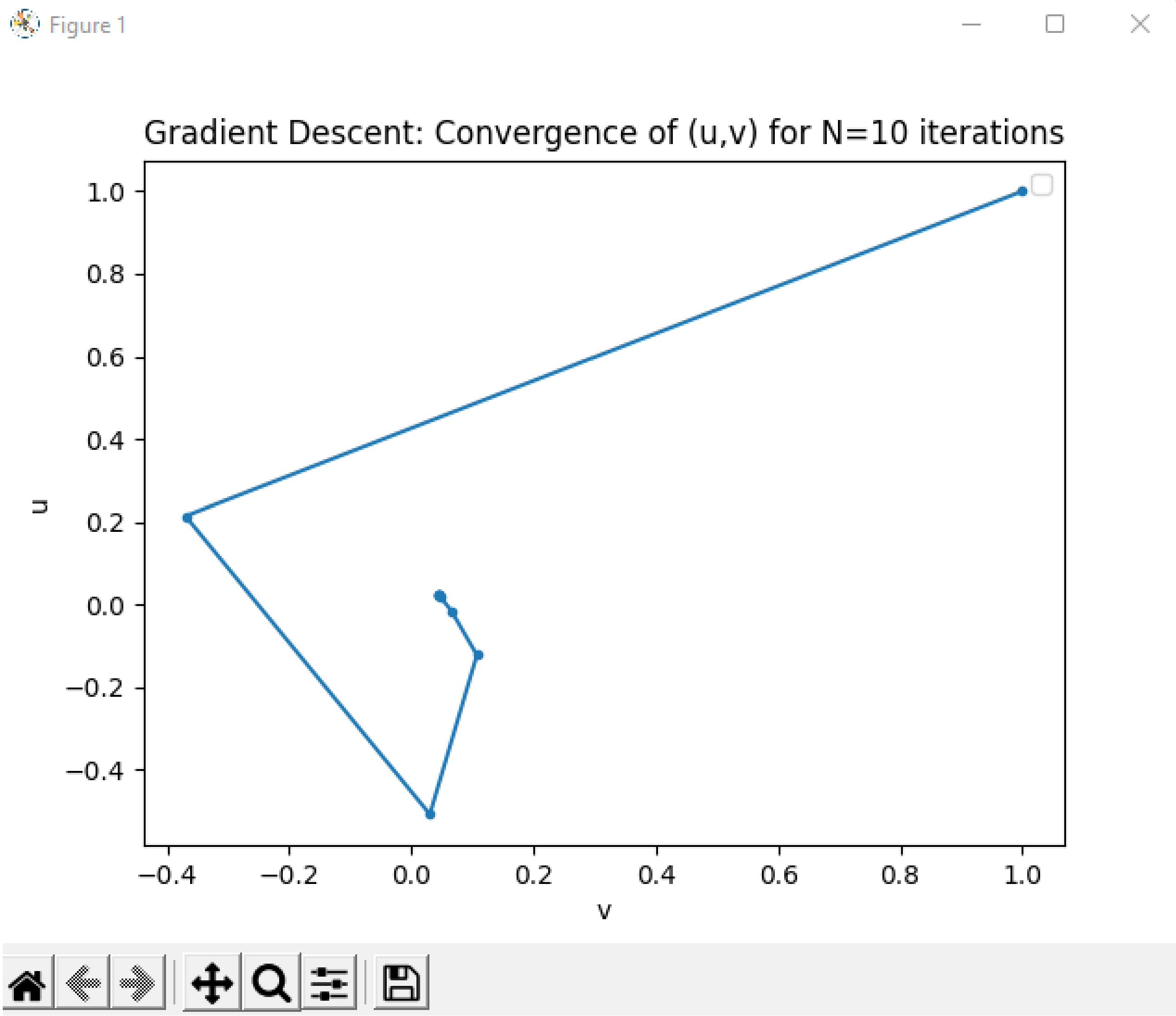
In this section we apply gradient descent algorithm to the nonlinear error space function by calculating the derivatives of squared residuals until error becomes less than $10 - 14$. Derivatives to apply gradient descent:

$$\frac{\partial E}{\partial u} = 2 \cdot (u \cdot e^{v} - 2 \cdot v \cdot e^{-u}) \cdot (e^{v} + 2 \cdot v \cdot e^{-u})$$

$$\frac{\partial E}{\partial v} = 2 \cdot (u \cdot e^{v} - 2 \cdot v \cdot e^{-u}) \cdot (u \cdot e^{v} - 2 \cdot e^{-u})$$

Number of iterations until error becomes 1.2086833944220747e-15 is 10.
The latest (u,v) pair is (0.04473629039778207, 0.023958714099141746)

Plotting of the (u,v) pair on every iteration:



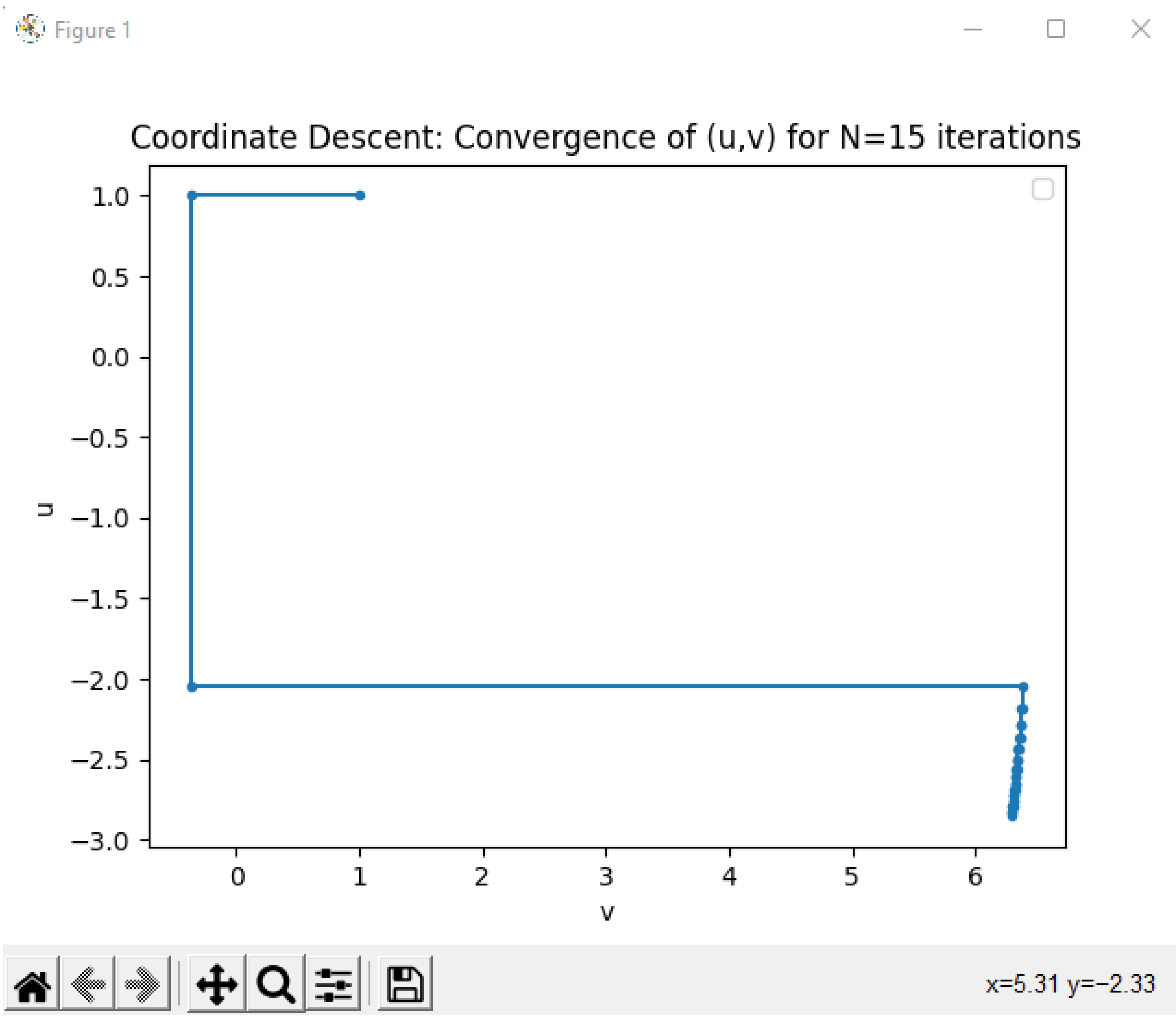

The answers are 5: D, and 6: E

### 7.

The error does not converge and stops at 0.13981379199615315 which is close to 10⁻1

The latest (u,v) pair is (6.29707589930517, -2.852306954077811))

Plotting of the (u,v) pair on every iteration:




The answer is A.

## Logistic Regression

For this section, Logistic_Regression.py is implemented. Script can be run to observe results. Plotting is included.
Results after running Logistic_Regression.py:

```
$ py Logistic_Regression.py
Eout: 0.09444365341119473, epochs: 338.58
```

The used formulas for the logistic regression algorithm are:

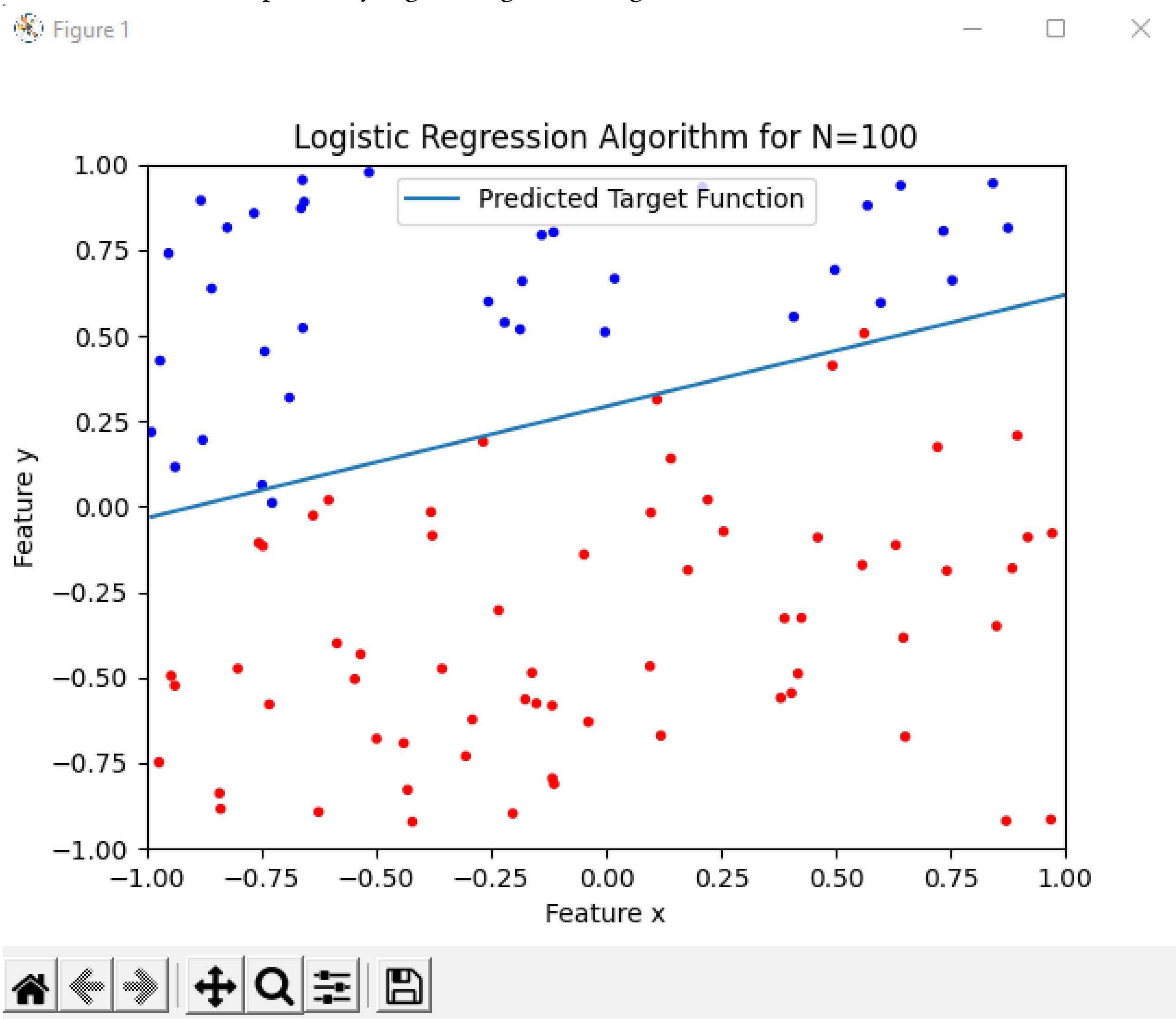
$$\text{Fixed step size: } w(1) = w(0) + n\nabla E_{in}$$

$$\text{v is calculated by stochastic gradient descent: } \nabla E_{in} = -\frac{1}{N}\sum_{n=1}^{N}\frac{y_n \cdot x_n}{ln(1 + e^{y_n \cdot w^T(t) \cdot x_n}})$$

$$\text{cross entropy error: } E_{out}(w) = \frac{1}{N}\sum_{n=1}^{N} ln(1 + e^{-y_n \cdot w^T \cdot x_n})$$

More details can be found in the source-code.

Predicted line on 100 points by logistic regression algorithm:




## 8.

I used the cross entropy error formula to calculate $E_out$.

Average Eout after 100 runs: 0.09444365341119473.
The answer is D.

## 9.

Average epochs after 100 runs: 338.58.
The answer is A.

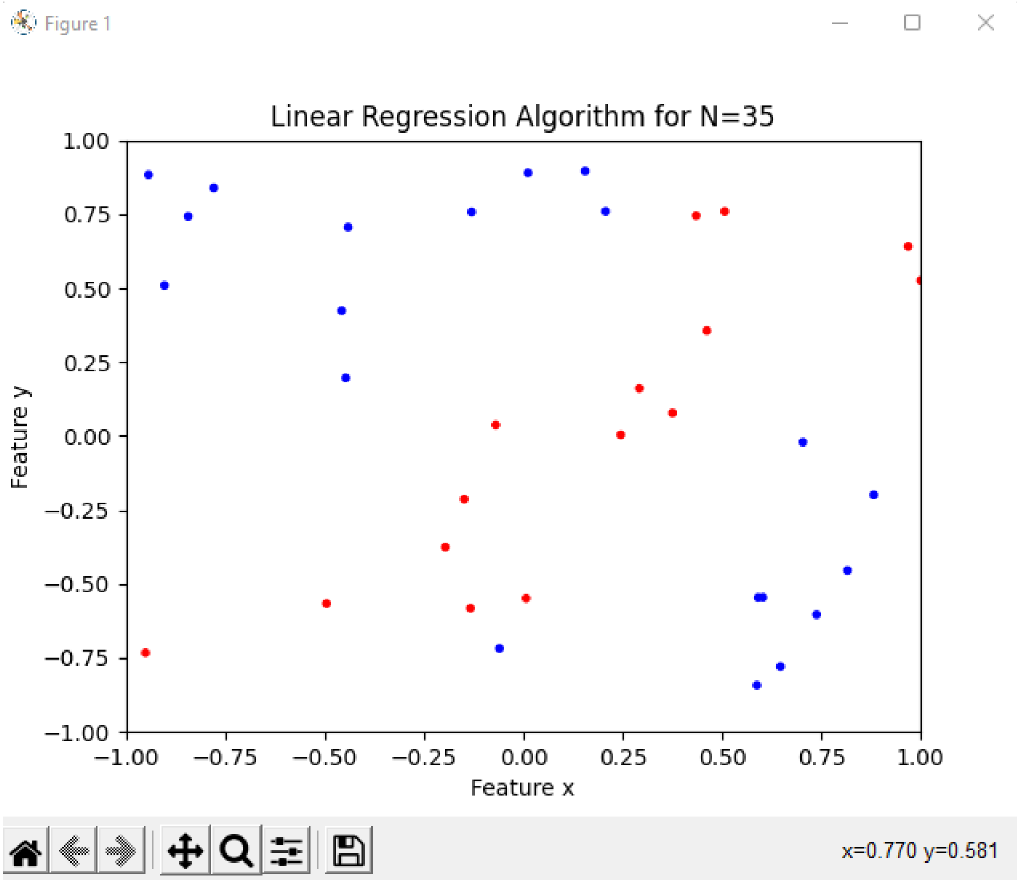
# Regularization with weight decay

For this section, Linear_Regression.py is implemented. Script can be run to observe results. Some data scattering is included.
Results after running Linear_Regression.py:

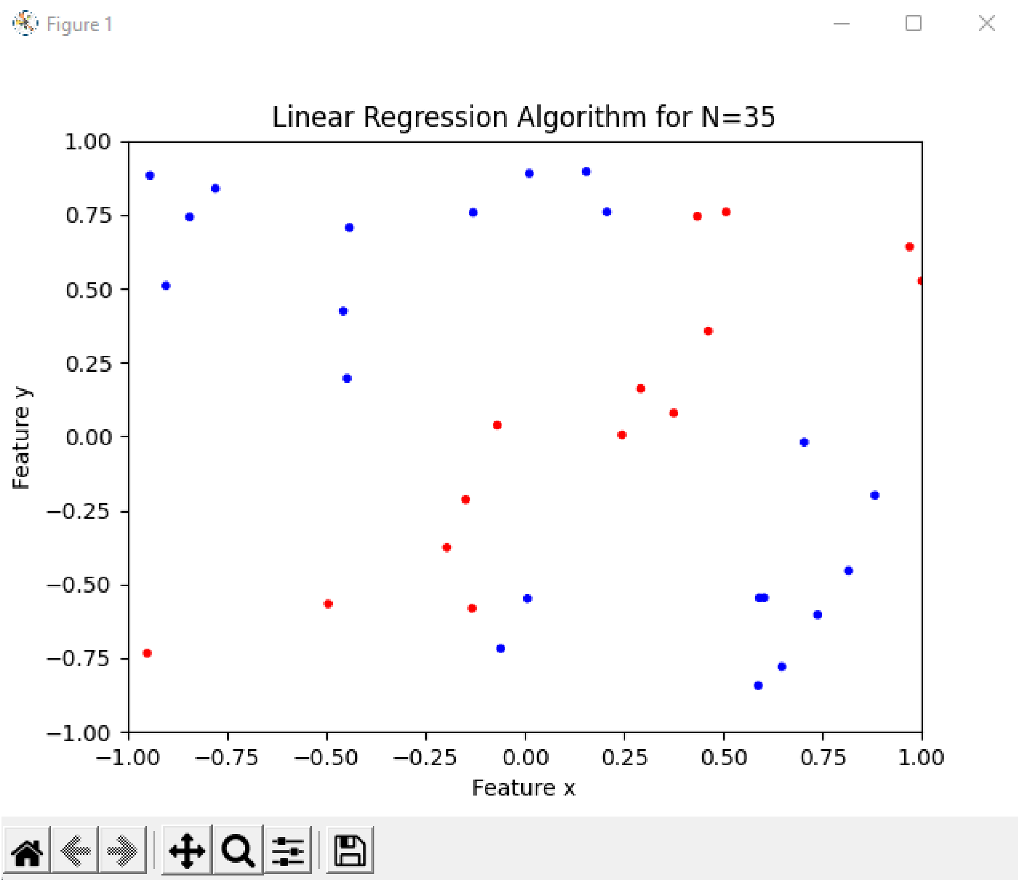
```
$ py Linear_Regression.py
Q2: Ein: 0.02857142857142857, Eout: 0.084
Q3: Ein: 0.02857142857142857, Eout: 0.08
Q4: Ein: 0.37142857142857144, Eout: 0.436
Q5- Q6: k: -1, Eout: 0.056
```

## 2.

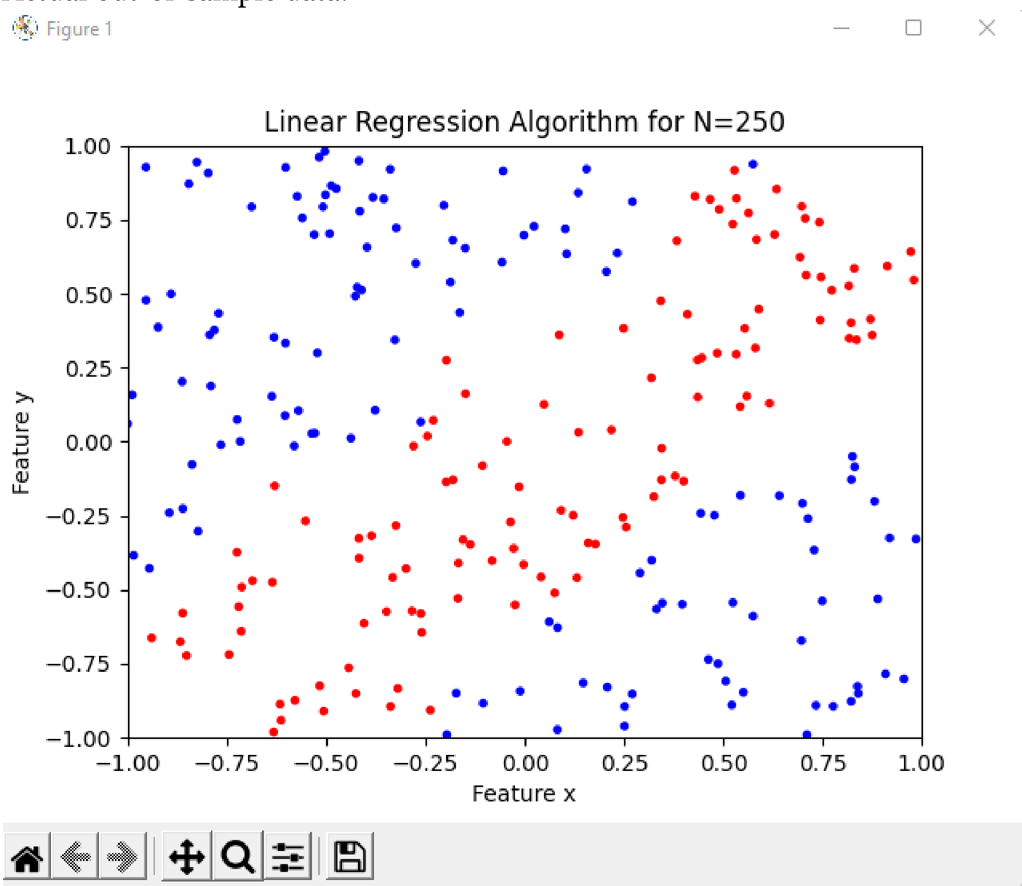
Actual in-sample data:



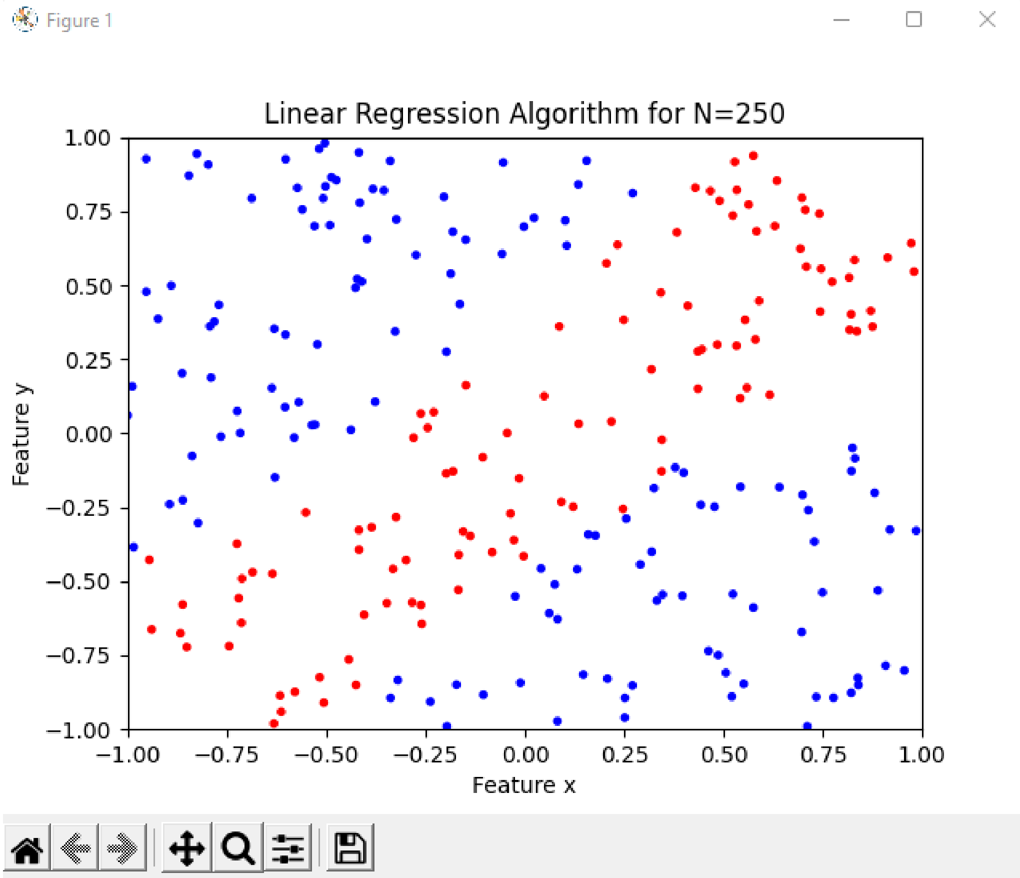

Predicted in-sample data:




Actual out-of-sample data:



Predicted out-of-sample data:




Ein: 0.02857142857142857, Eout: 0.084
The answer is A.

### 3.

0.02857142857142857, Eout: 0.08
The answer is D.

### 4.

Q4: Ein: 0.37142857142857144, Eout: 0.436
The answer is E.

### 5.

k = -1
The answer is D.

### 6.

Eout: 0.056
The answer is B.

# Neural Networks

For this section, Neural_Network.py is implemented using the formulas below. Script can be run to observe results.
**Neural Network operators:**

$$w_{ij}^{(l)} = \left\{ \begin{array}{ll} 1 \leq l \leq L & \text{Layers} \\ 0 \leq i \leq d^{(l-1)} & \text{Inputs} \\ 1 \leq j \leq d^{(l)} & \text{Outputs} \end{array} \right\}$$

**Activation function:**

$$\frac{(e^s - e^{-s})}{(e^s + e^{-s})}$$

**Forward propagation:**

$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} \cdot x_i^{(l-1)})$$

**Backward propagation:**

For final layer: $\delta_1^{(L)} = 2 \cdot (x_1^{(L)} - y_n) \cdot (1 - \theta^2(s_1^{(L)}))$

For other layers: $\delta_i^{(l-1)} = (1 - (x_i^{(l-1)})^2) \cdot \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)} \cdot \delta_j^{(l)}$

Updating perceptron weights: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - n \cdot x_i^{(l-1)} \cdot \delta_j^{(l)}$

Results after running Neural_Network.py:

```
$ py Neural_Network.py
Q8: Total number of operations is: 28
Q9: Minimum number of weights is: 46
Q10: Maximum number of weights is: 522
```

### 8.

Total number of operations is: 28. More details can be observed from run_q8() function in the source-code
The answer is A.

### 9.

By setting 36 hidden layers with 1 weight we count all the weights in the neural network as 46. More details can be observed from run_q9() function in the source-code
The answer is A.

## 10.

By setting 2 hidden layers with weights of 18, 18 we count all the weights in the neural network as 522. More details can be observed from run_q10() function in the source-code.
The answer is E.