Jeremy Grifski

CSE 5521

11/21/19

Homework 10

1. Complete the function calc\_linLSQ\_line(): Use linear least squares to estimate the parameters (a and b) for the following model:

𝑓(𝑥, 𝑎, 𝑏) = 𝑎 𝑥 + 𝑏

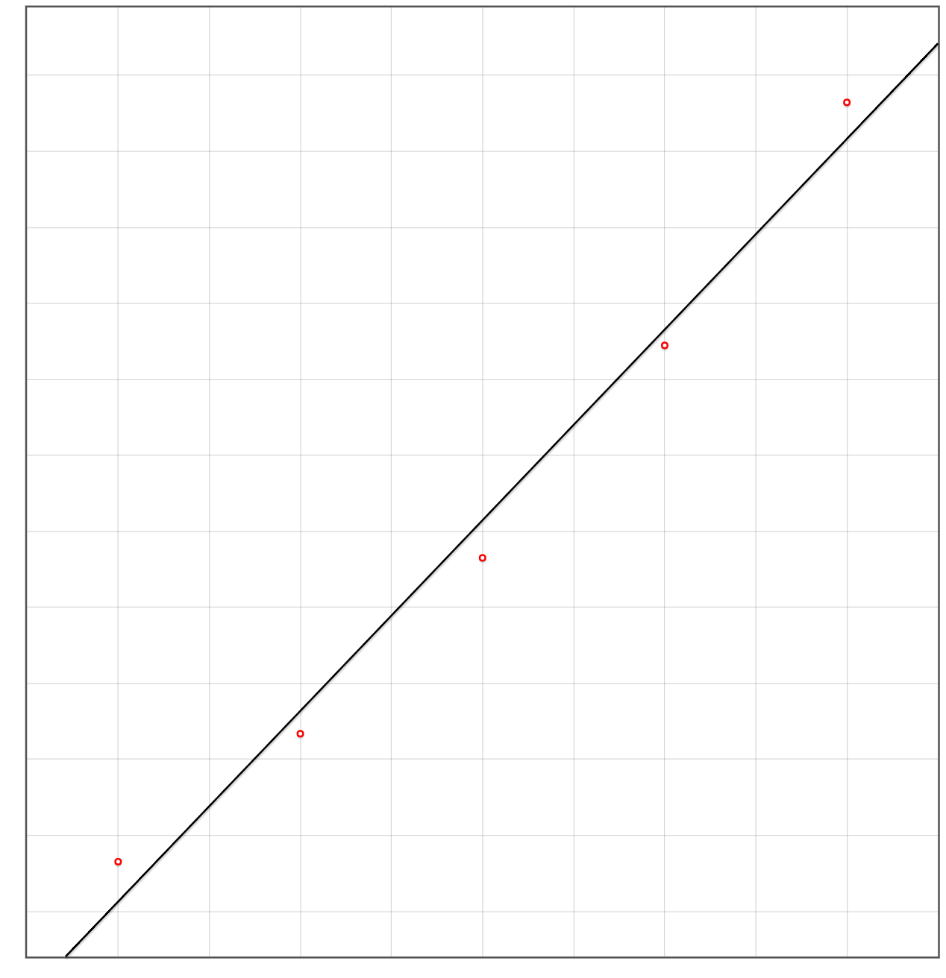
Calculate the sum of squared error using your results. Also, create a plot of the model function using the parameter values you found (the “Save As Img” button can be used for this). Include these in your report. (4 pts)

**First Order**

**a = 1.2547000000000004**

**b = -0.5629500000000007**

**sse = 0.22131790000000037**



1. Similar to problem 1, complete the function calc\_linLSQ\_poly(): Use linear least squares to estimate the parameters for various polynomials, such as:

𝑓(𝑥, 𝑎, 𝑏, 𝑐) = 𝑎 𝑥 ^ 2 + 𝑏 𝑥 + 𝑐

𝑓(𝑥, 𝑎, 𝑏, 𝑐, 𝑑) = 𝑎 𝑥 ^ 3 + 𝑏 𝑥 ^ 2 + 𝑐 𝑥 + 𝑑

𝑓(𝑥, 𝑎, 𝑏, 𝑐, 𝑑,𝑒) = 𝑎 𝑥 ^ 4 + 𝑏 𝑥 ^ 3 + 𝑐 𝑥 ^ 2 + 𝑑 𝑥 + 𝑒

Create a plot and calculate SSE as before, for your report. Compare the plots and errors to your results from problem 1. Which of these 4 models (including problem 1) do you think best fits the data? Why? (7 pts)

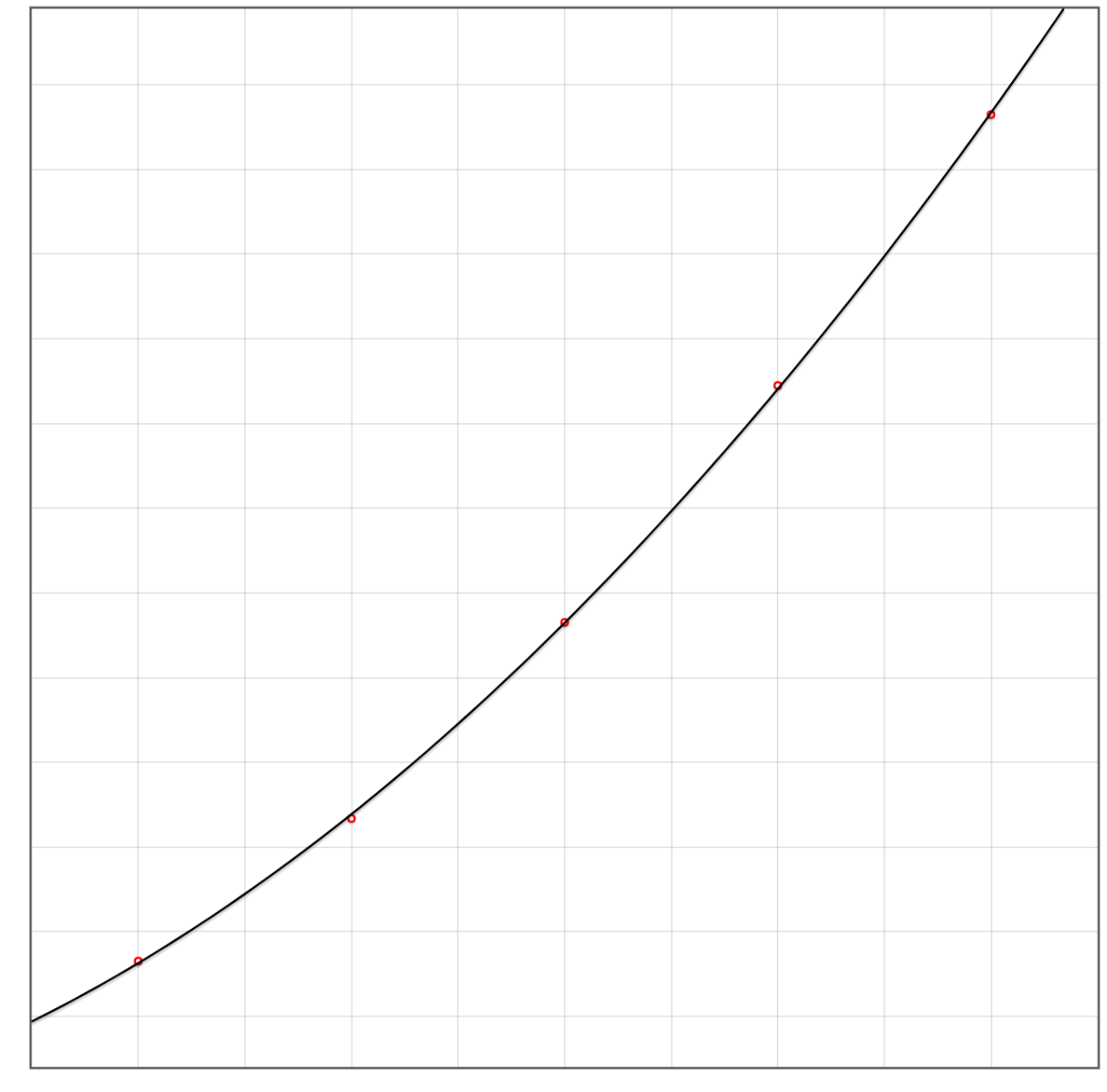
**Second Order**

**a = 0.12535714285714403**

**b = 0.6279142857142741**

**c = -0.03018214285714671**

**sse = 0.0013161142857142784**



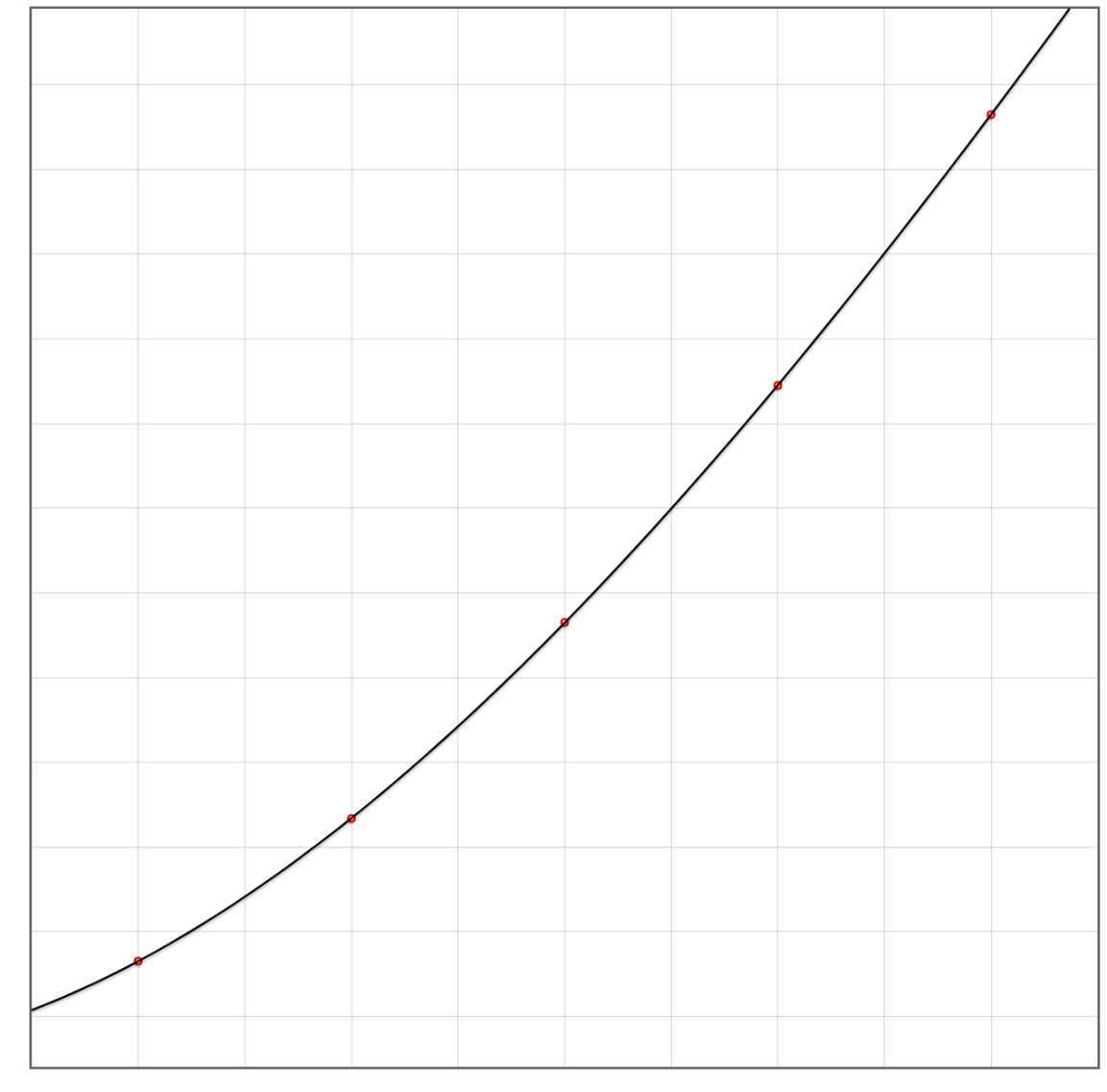
**Third Order**

**a = -0.00950000000001533**

**b = 0.19660714285725422**

**c = 0.4820892857143928**

**d = 0.03750535714299619**

**sse = 0.00001651428571428709**

**Fourth Order**

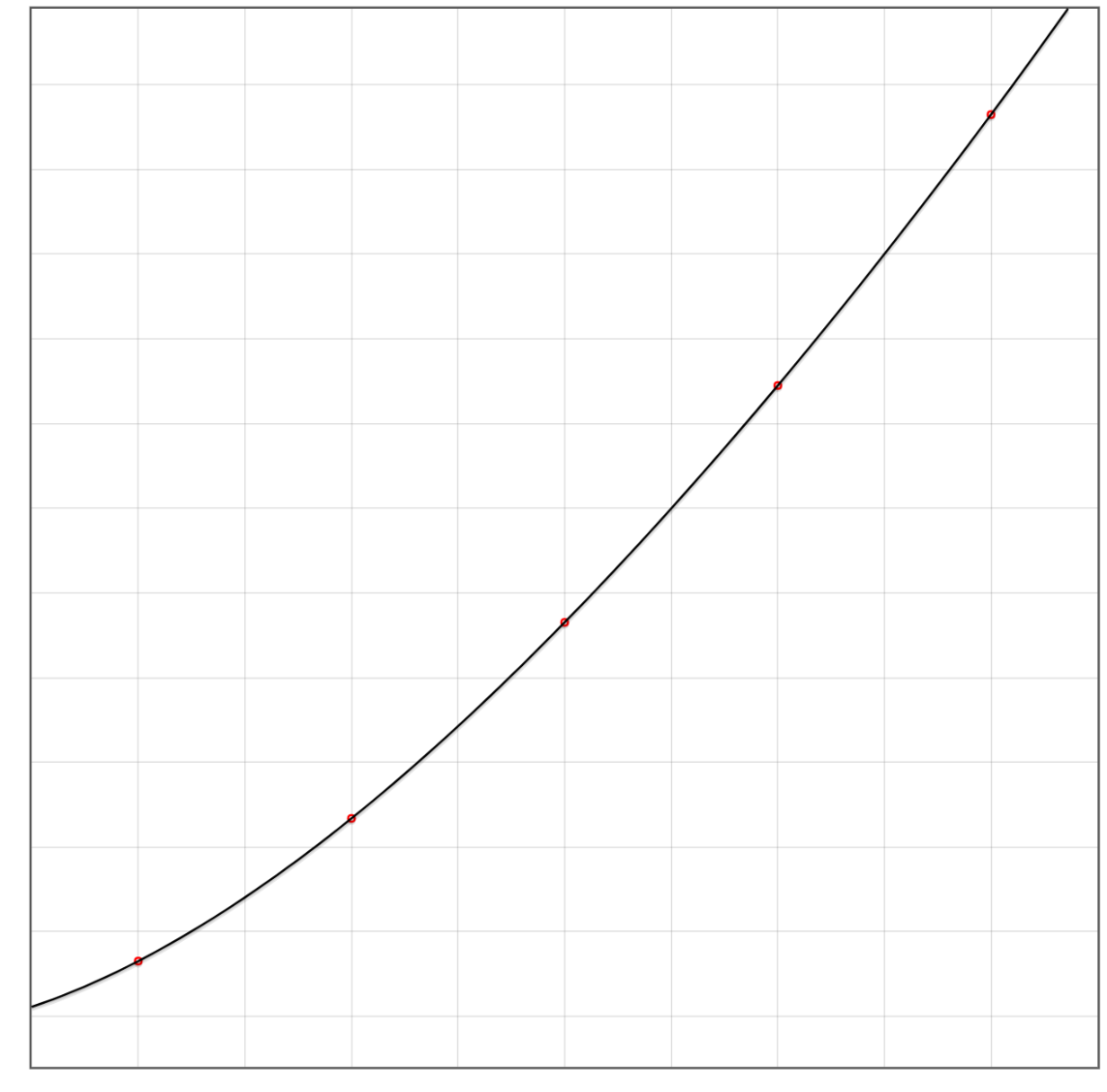
**a = 0.0014166666665633154**

**b = -0.023666666664953873**

**c = 0.24345833333173417**

**d = 0.42491666667529593**

**e = 0.056546874997625896**

**sse = 1.9420303619057346e-20**

**Of the four plots, the fourth order function has the best fit according to sum of squares error (SSE). Of course, this make sense because higher dimensional functions work like taylor series expansion to approximate a curve. In other words, you get more control over the shape of the curve with each additional dimension.**

1. Derive the function for the Jacobian matrix for the following non-linear model:

𝑓(𝑥, 𝒑) = 𝑎 𝑥 ^ 𝑏 + 𝑐 𝑥 + 𝑑

In addition to including this derivation in your report, implement this in the calc\_jacobian() function. (2 pts)

**df(x, p) / da = x^b**

**df(x, p) / db = a \* x^b \* log(x)**

**df(x, p) / dc = x**

**df(x, p) / dd = 1**

1. Complete the function calc\_nonlinLSQ\_gaussnewton(): Use Gauss-Newton non-linear least squares to estimate the parameters for the function from (3).

Use the following for your initial guess:

a=0.5, b=2, c=0.5, d=0.5

Stop after 10 iterations. (Note, these are the default values in the template.)

As with problems 1-2, create a plot of the resulting model function. Also, calculate the sum of squared error after each iteration. Do you think this model fits the data better than previous ones? Why or why not? (5 pts)

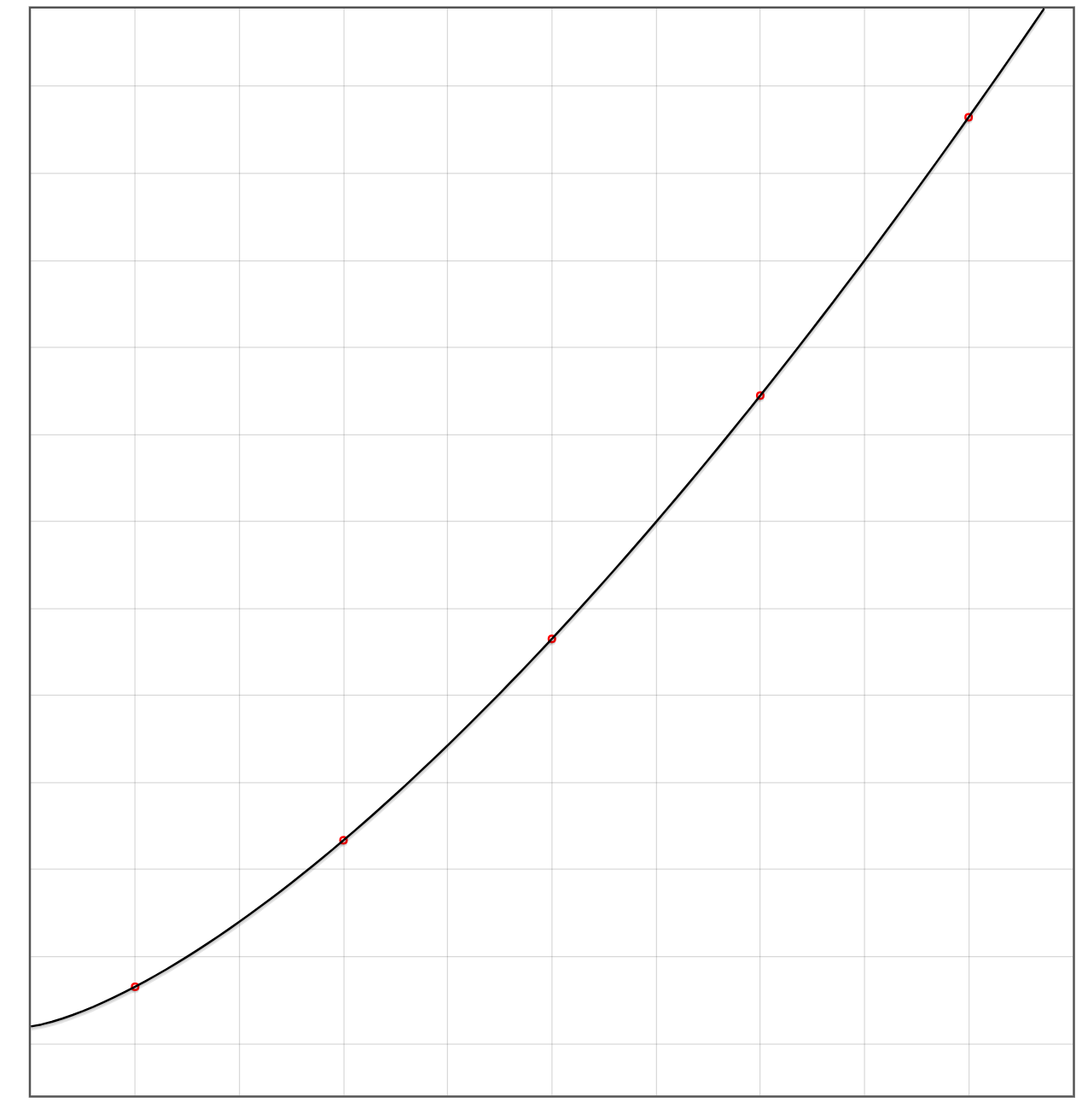
**Nonlinear Solution**

**a = 0.4938455104742545**

**b = 1.5037635080613851**

**c = 0.10730333930639983**

**d = 0.09923420477446322**

**sse= 2.2090783285673085e-7**

**Considering the final SSE for this curve, I’d have to say that the fourth order polynomial does a better job. That said, the error is so small that I can’t really tell a difference between the two curves. However, I do think this curve would be better for mitigating issues related to overfitting. After all, it appears that the SSE converges without driving to zero.**

1. Complete the function calc\_nonlinLSQ\_gradientdescent(): Use Gradient Descent non-linear least squares to estimate the parameters for the function from (3).

Use the same initial guess as (4). Use a learning rate of 0.001 and stop after 5000 iterations.

As with problem 4, create a plot and find the sum of squared error after each iteration. How does this algorithm compare to (4)?

Try different values for learning rate. Can you achieve a better convergence rate (lower error or less iterations for same error)? What are your observations on how the algorithm behaves with different values? (5 pts)