Leonardo Gomez

CS 596

February 25, 2018

**Homework 2**

The following code is an iterative algorithm for predicting the prices of meal items. In the following code uses Gradient decent to optimize the data. Gradient Decent is a First order iterative optimization algorithm for finding the minimum of a function. In this case, by finding the local minimum of the function we can find the optimal solution for the problem. Furthermore, the only things changed in this program is the place holders. In PLACEHOLDER1, which include the number of iterations, the minimum delta, and the alpha will be change to see how it affects the prediction capabilities of the algorithm. In PLACEHOLDER2, nothing will be change between runs. Nevertheless, it will show how for every iteration it will calculate an error by subtracting the expected total price minus the cashier price. Once this is calculated it the absolute value taken, it be use to compare to the minimum delta to see if we are getting closer to the true answer. In addition, the estimated unit price will be updated in every iteration using the equation  *(Est Price – True Price) \** , *(Est Price – True Price) \* (Est Price – True Price) \**  for each respective ingredient.

**Code:**

import numpy as np

import matplotlib.pyplot as plt

# Starting codes for the HA2 of CS596

# Ground-truth Cashier

groundUnitPrice = np.array([20, 25, 8]) # for fish, chip, and ketchup, respectively

# step 1: initialize your guess on the unit prices of fish, chip and ketchup.

estimatedUnitPrice = np.array([10,10,10]) # initial unit prices.

#PLACEHOLDER1

#My own stopping conditions and learning rate

#condition 1: maximal iterations, stop.

MAX\_ITERATION = 100000

#condition 2: if the difference between your prediction and the cashier's price is smaller than a threshold, stop.

MIN\_DELTA = 0.001

# learning rate

ALPHA = .001#1e-3

#PLACEHOLDER\_1#end

# Y coordinates for plotting

deltaHistory = []

# step 2: iterative method

for i in range(0, MAX\_ITERATION):

    # order a meal (simulating training data)

    randomMealPortions = np.random.randint(10, size=3)

    # calculate the estimated price

    expectedTotalPrice = np.sum(estimatedUnitPrice \* randomMealPortions )

    # calculate cashier/true price;

    cashierPrice = np.sum(groundUnitPrice \* randomMealPortions)

    #PLACEHOLDER\_2

    # calculate current error

    iterError = expectedTotalPrice - cashierPrice

    #Declaring Delta

    delta = abs(iterError)

    # append iterError to the history array

    deltaHistory.append(delta)

    #update unit prices

    Theta\_1 = estimatedUnitPrice[0] - ALPHA\*(estimatedUnitPrice[0] - groundUnitPrice[0])\*randomMealPortions[0]

    Theta\_2 = estimatedUnitPrice[1] - ALPHA\*(estimatedUnitPrice[1] - groundUnitPrice[1])\*randomMealPortions[1]

    Theta\_3 = estimatedUnitPrice[2] - ALPHA\*(estimatedUnitPrice[2] - groundUnitPrice[2])\*randomMealPortions[2]

    estimatedUnitPrice = np.array([Theta\_1,Theta\_2,Theta\_3])

    #%%%%PLACEHOLDER\_2#end

    #check stop conditions

    if abs(delta) < MIN\_DELTA:

        break

    print('iteration:{}, delta:{}'.format(i, abs(delta)))

# step 3: evaluation

error = np.mean(abs(estimatedUnitPrice - groundUnitPrice))

print('estimation error:{}'.format(error))

# visualize convergence curve: error v.s. iterations

plt.plot(range(0, len(deltaHistory)), deltaHistory)

plt.xlabel('iteration (cnt:{})'.format(len(deltaHistory)))

plt.ylabel('abs(delta)')

plt.title('Final:{}  est err:{}  actl Δ:{}'.format([ '%.4f' % elem for elem in estimatedUnitPrice ], round(error, 4), round(delta, 4)))

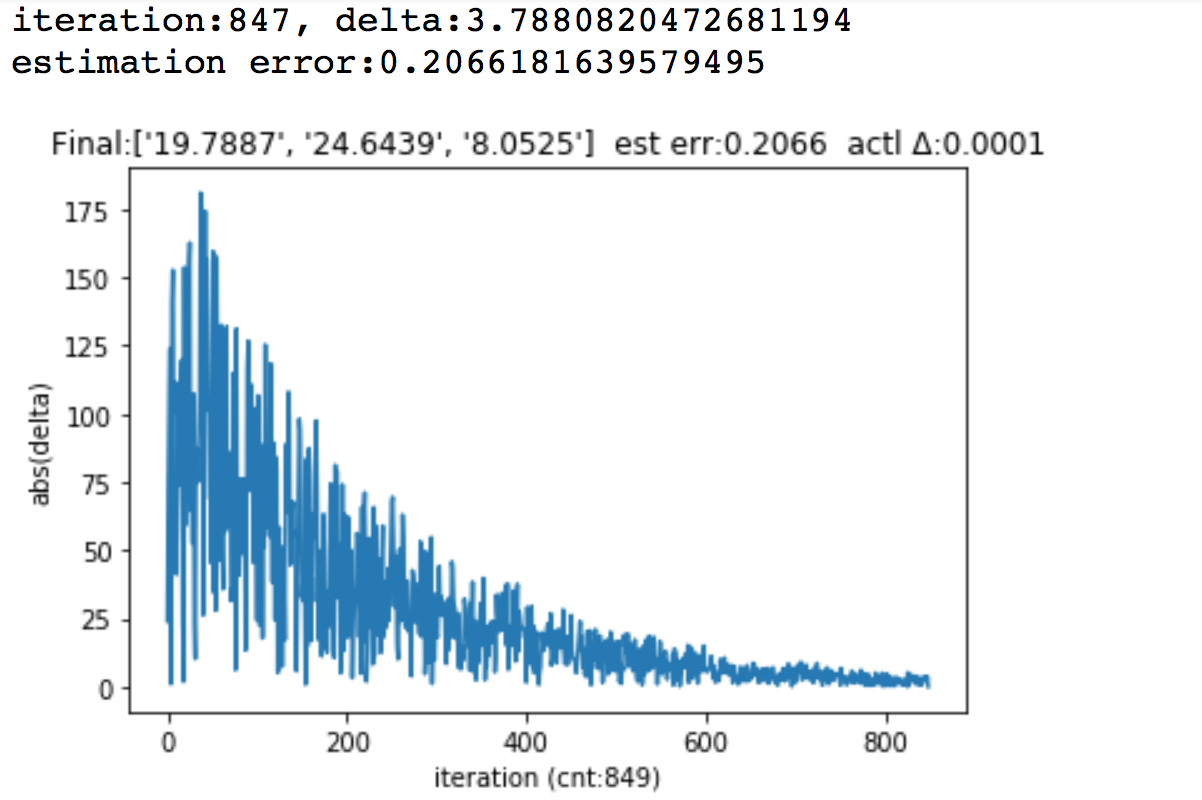
plt.show()

**Results:**

MAX\_ITERATION = 100000

MIN\_DELTA = 0.001

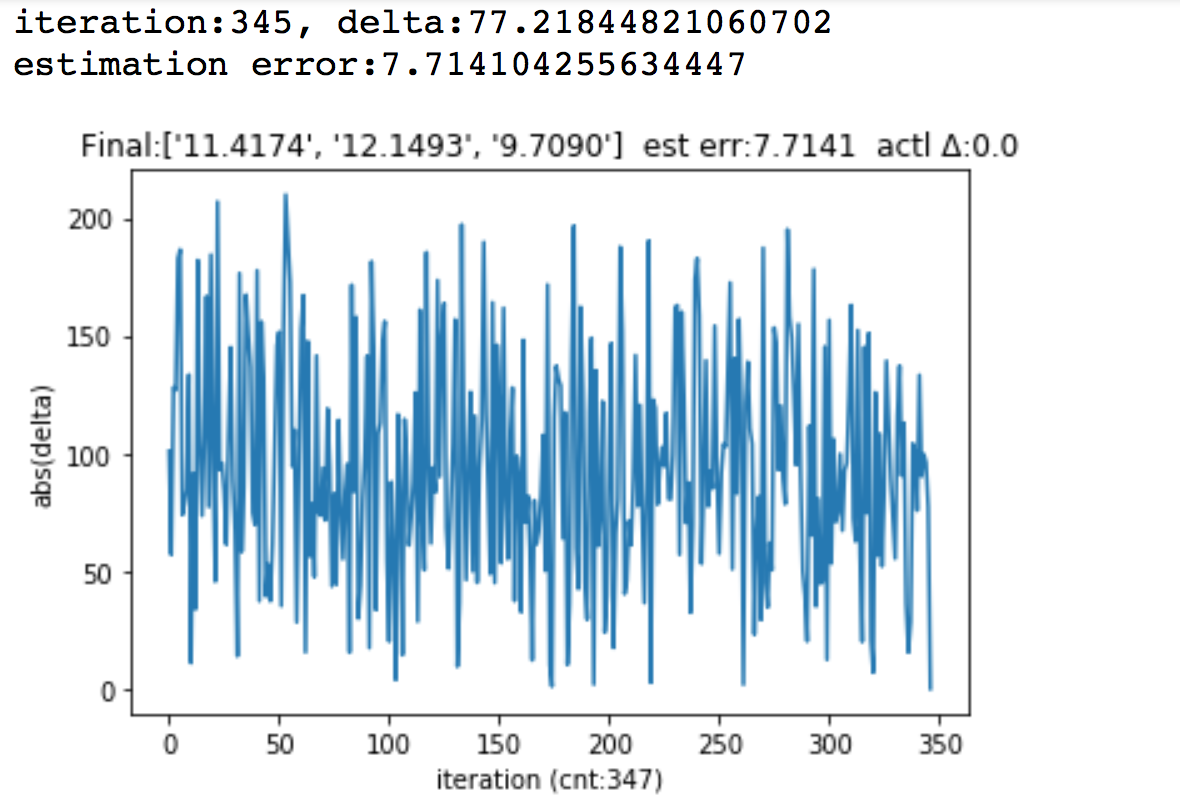
ALPHA = .001



MAX\_ITERATION = 1000000

MIN\_DELTA = 0.0001

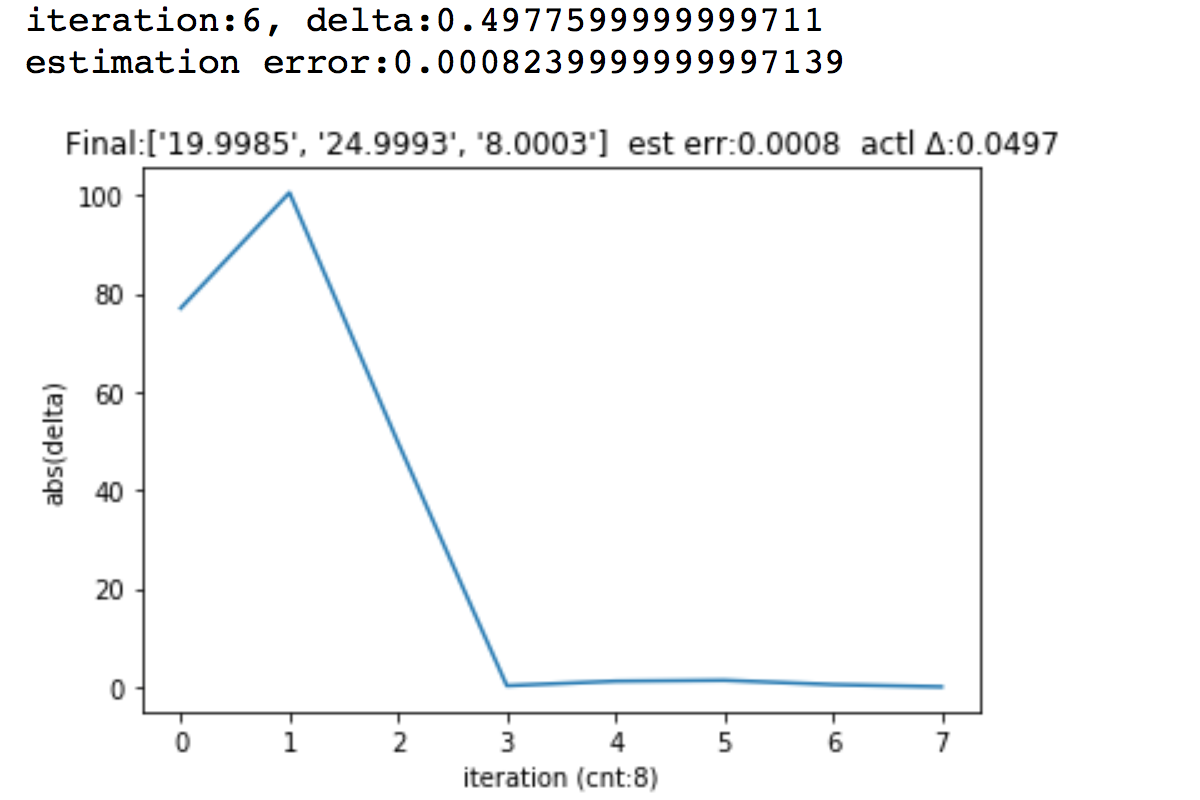
ALPHA = .0001



MAX\_ITERATION = 1000000

MIN\_DELTA = 0.1

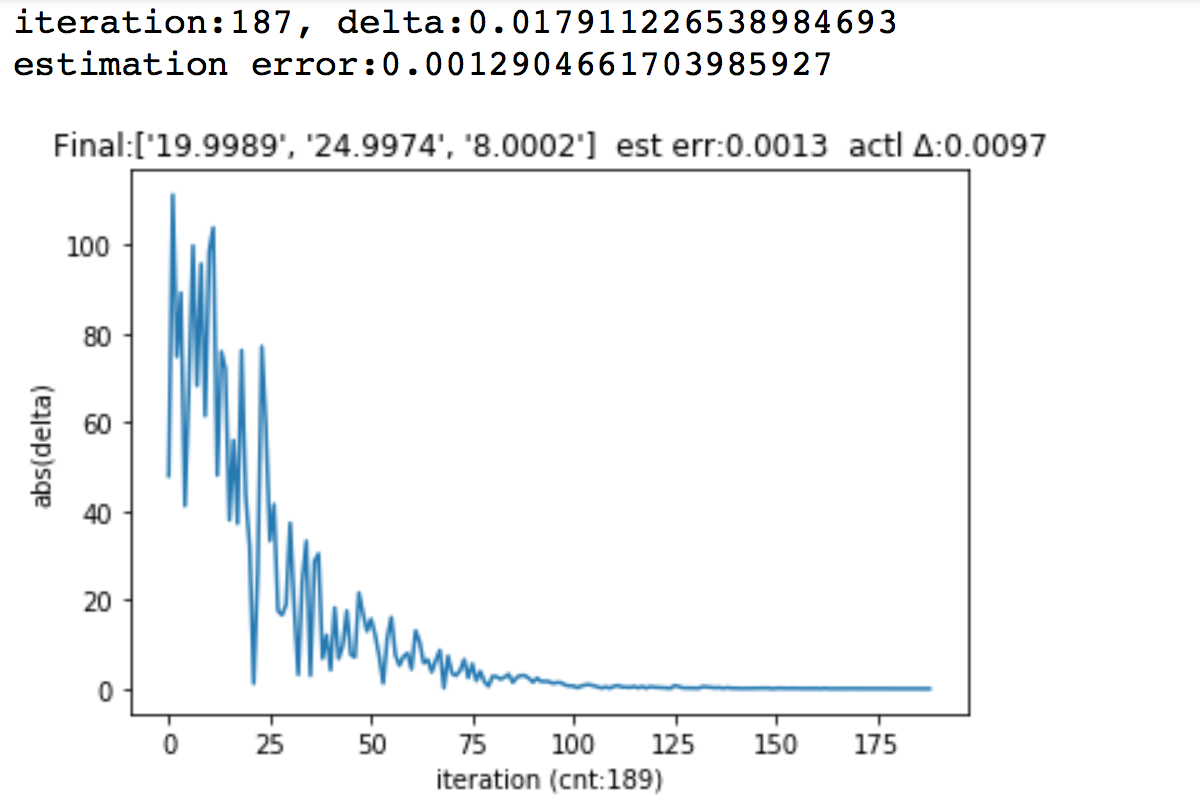
ALPHA = .1



MAX\_ITERATION = 1000000

MIN\_DELTA = 0.01

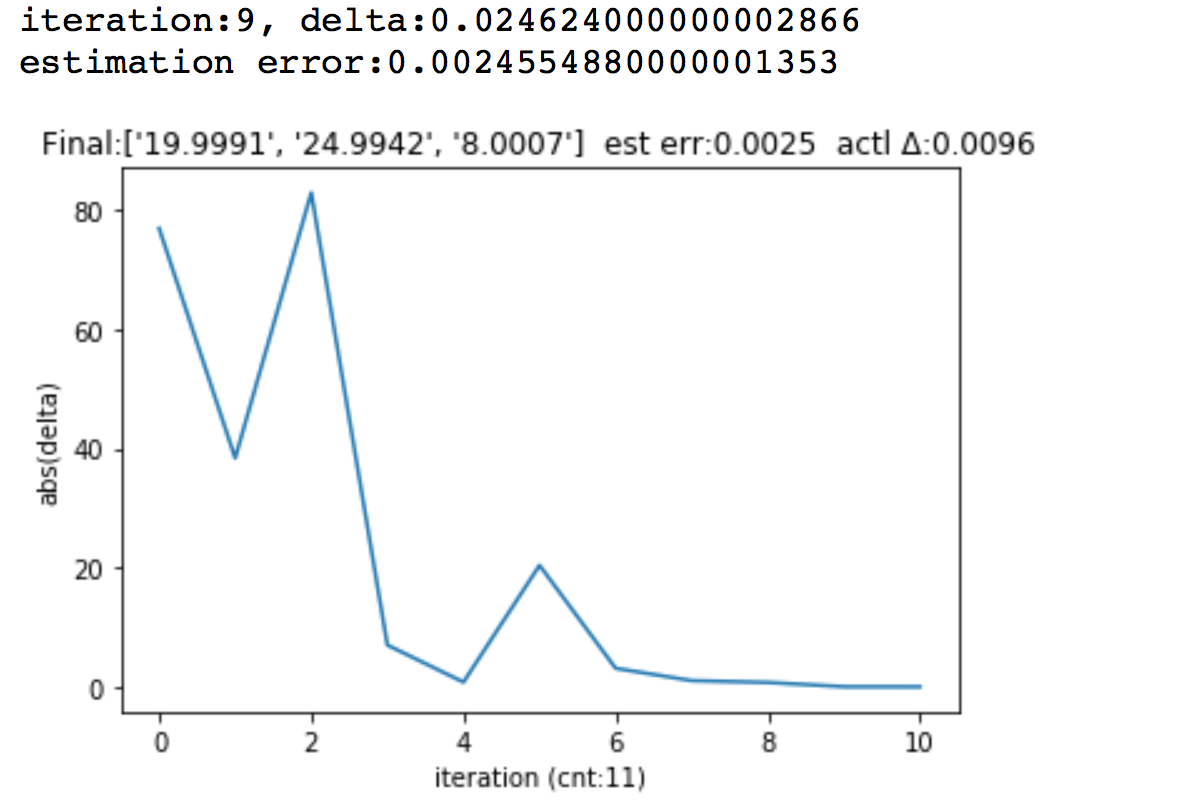
ALPHA = .01



MAX\_ITERATION = 1000000

MIN\_DELTA = 0.01

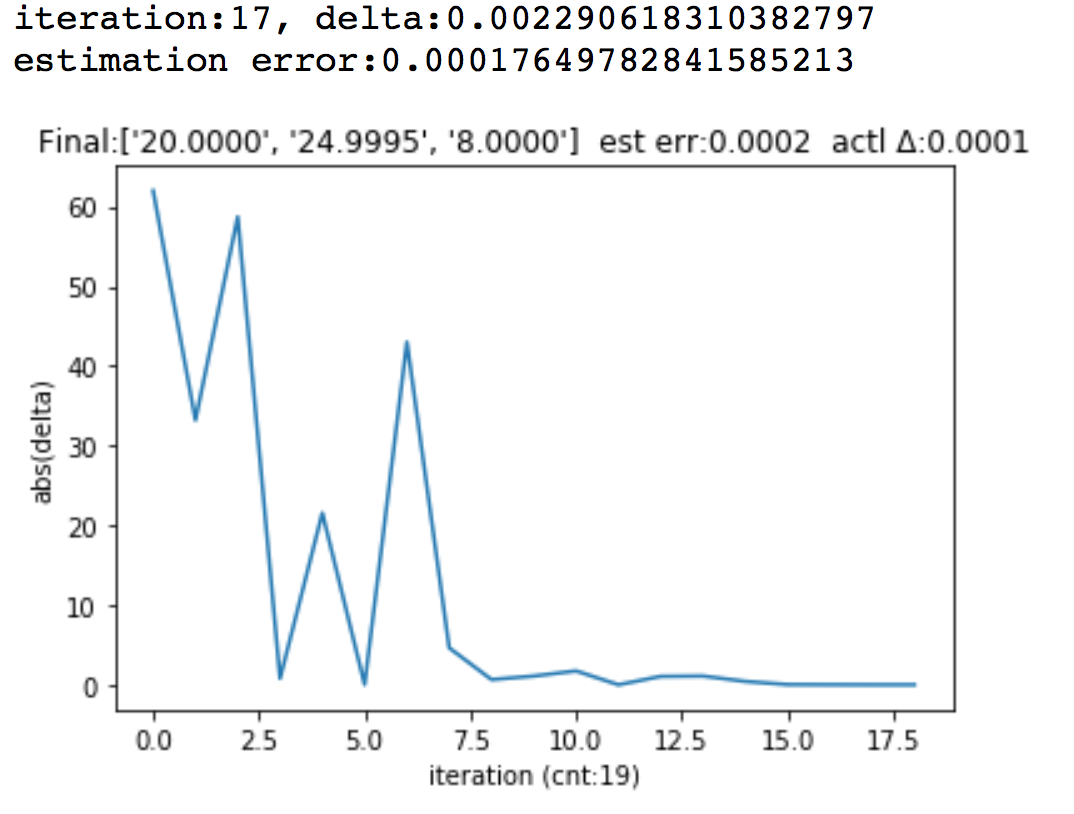
ALPHA = .1



MAX\_ITERATION = 1000000

MIN\_DELTA = 0.001

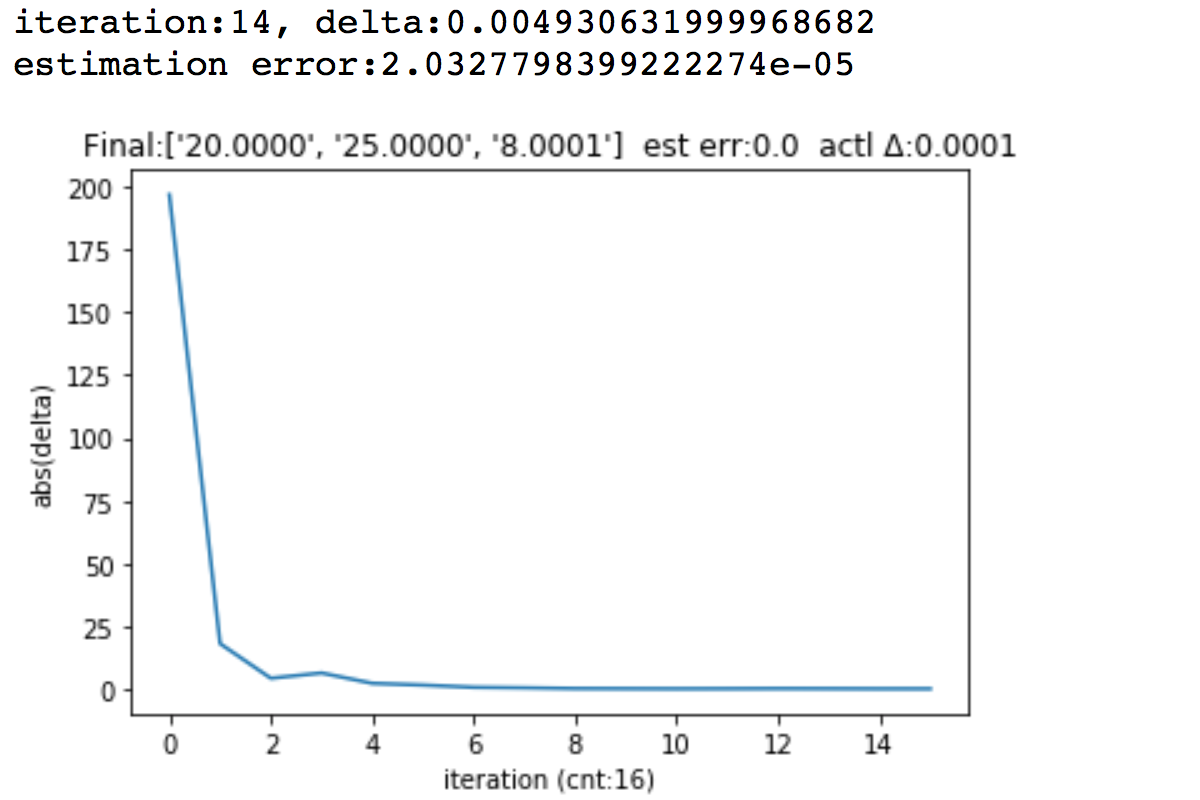
ALPHA = .1



MAX\_ITERATION = 1000000

MIN\_DELTA = 0.0001

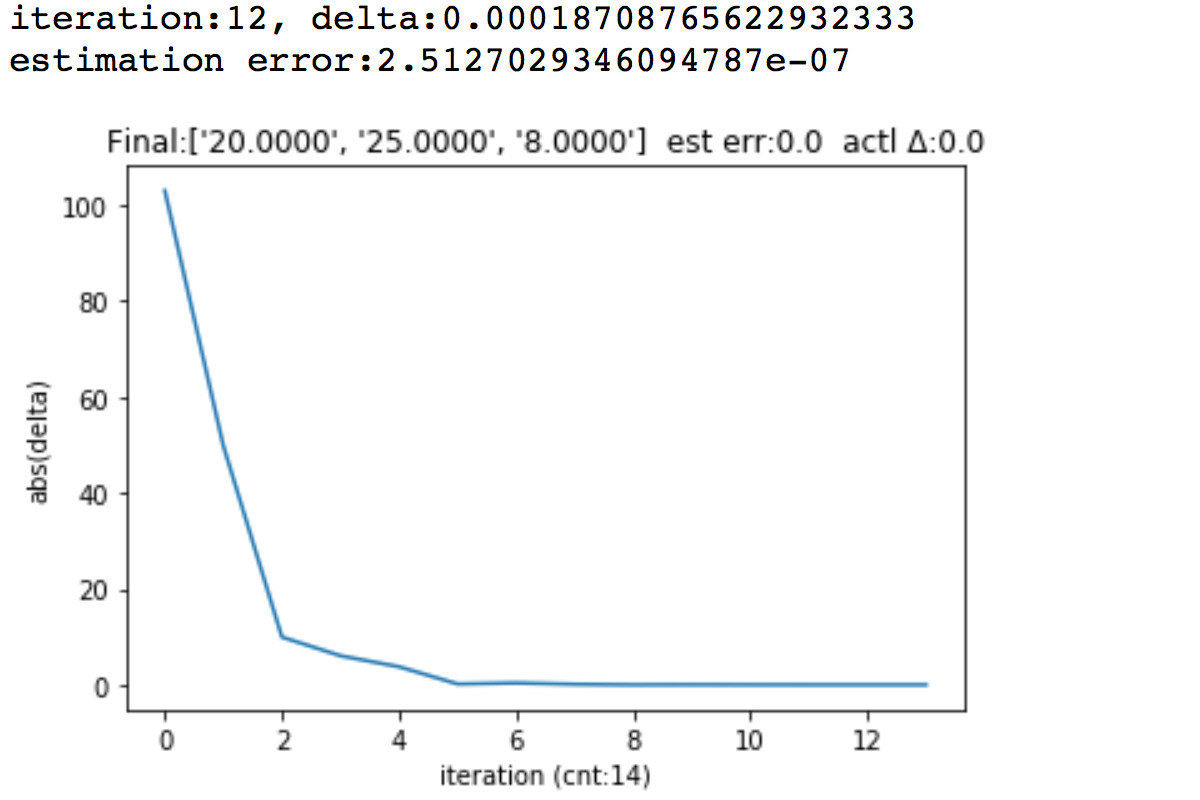
ALPHA = .1



MAX\_ITERATION = 1000000

MIN\_DELTA = 0.0001

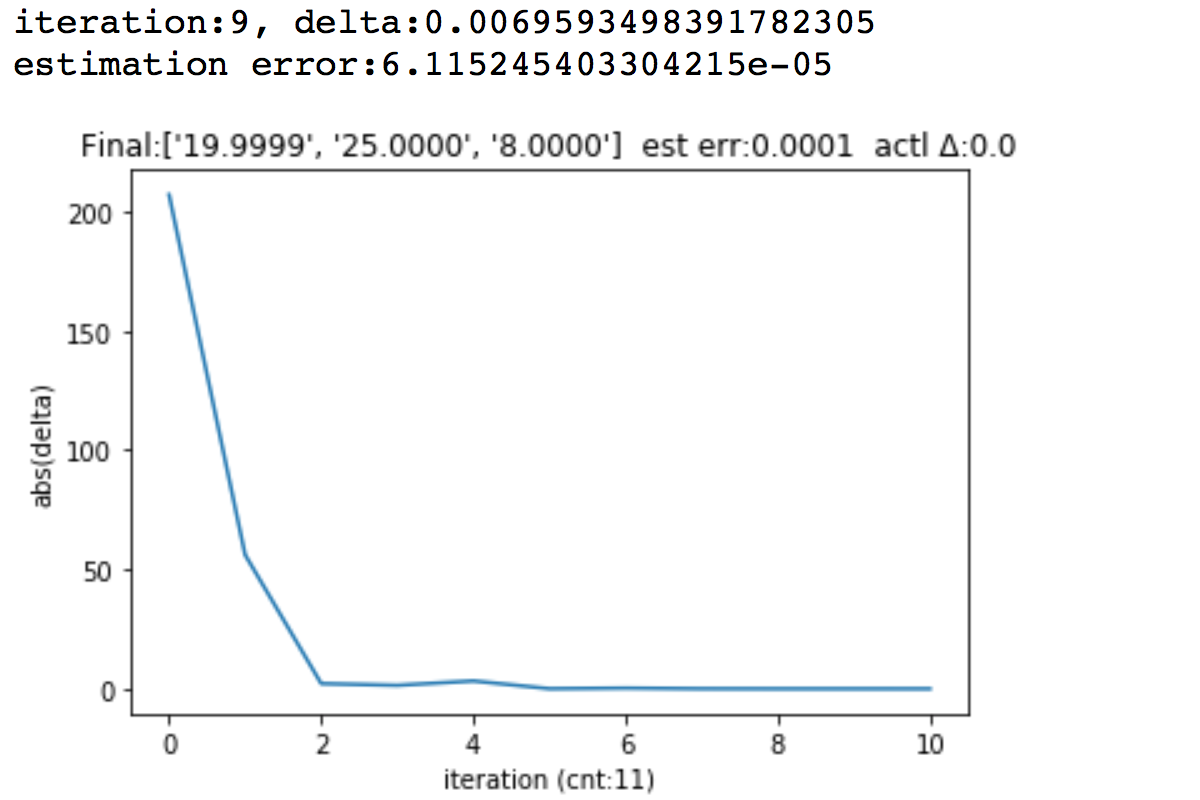
ALPHA = .15



MAX\_ITERATION = 1000000

MIN\_DELTA = 0.0001

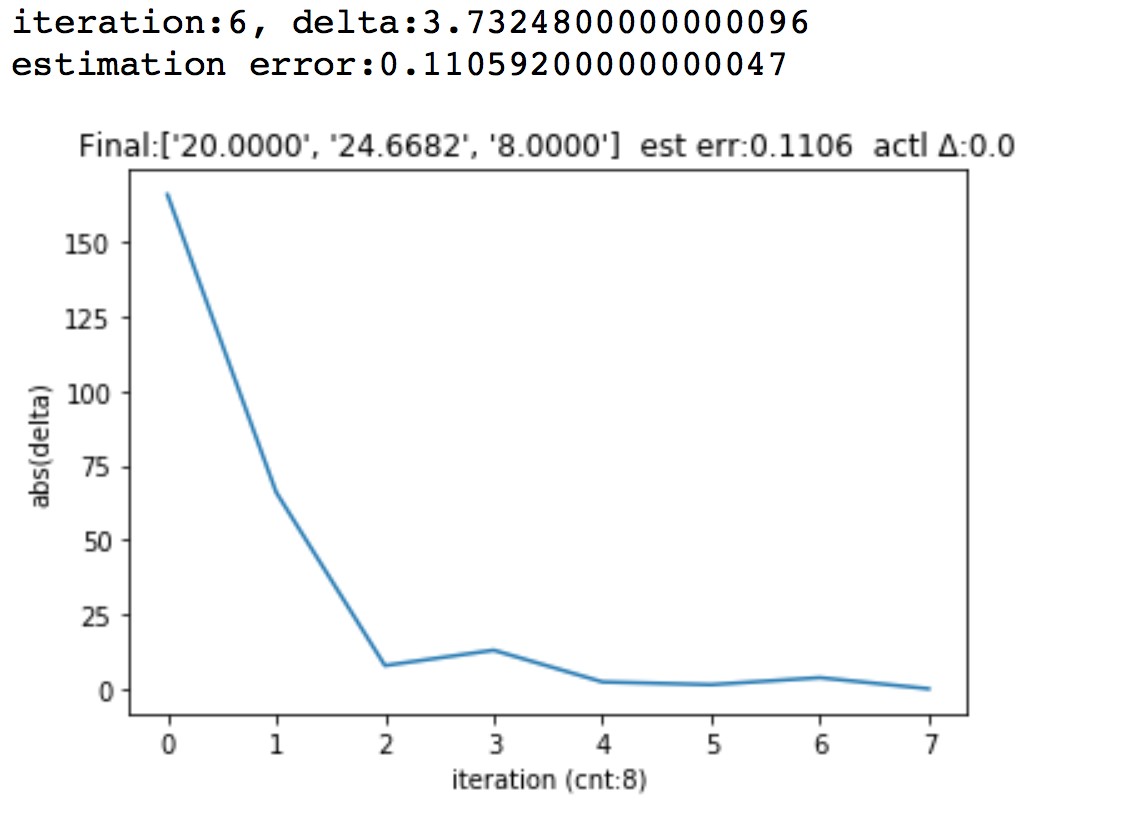
ALPHA = .16



MAX\_ITERATION = 1000000

MIN\_DELTA = 0.0001

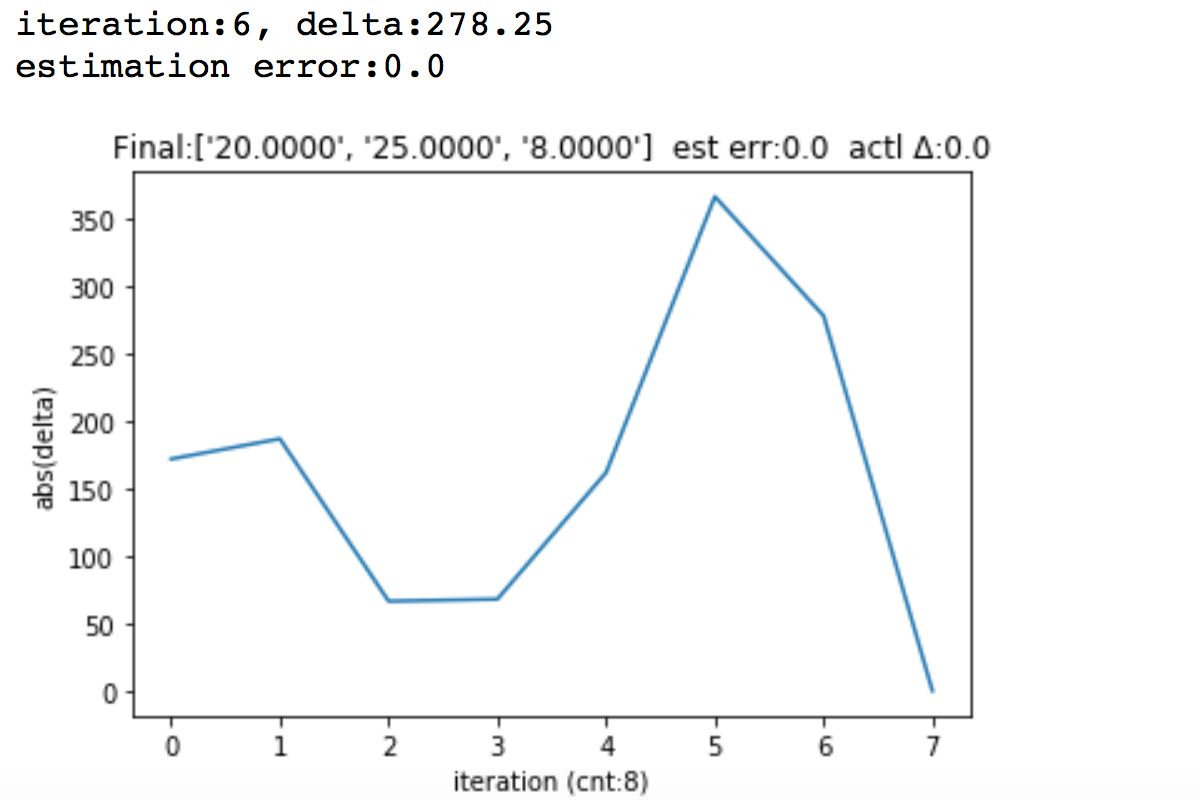
ALPHA = .2



MAX\_ITERATION = 1000000

MIN\_DELTA = 0.000001

ALPHA = .5



**Conclusion:**

From the multiple trials that were run, we see that changing the number of iterations does not matter too much. With our current settings, most trials stop before the reaching the maximum iteration number. When changing the minimum delta, we are changing focusing on minimizing the error gap. By making min\_delta small we can make sure that our results will not deviate too much from the actual answer. Alpha oversees how far apart we jump in between points. In other words, a small Alpha will check more numbers in between two points while a big Alpha will skip more points. In the trials we ran, We notice that by making a tiny delta we minimize error and by making alpha jump half of point we got the best results. This is to say that we set alpha to 0.5 and min\_delta to 0.000001 we got the least amount of iterations with the least amount of error.