I. Linear regression

*1) Data acquisition*  
 x y   
22,72 453,13

18,14 461,71

11,49 471,08

9,94 473,74

23,54 448,56

*2) Data transformation*

Min-max Standardization was used to transform the data. Formula: (X-min(X)) / (Max(X) – min(X)). This method was chosen as it results in all-positive values on a small interval while keeping the data spacing.

[[0.93970588 0.18149325]

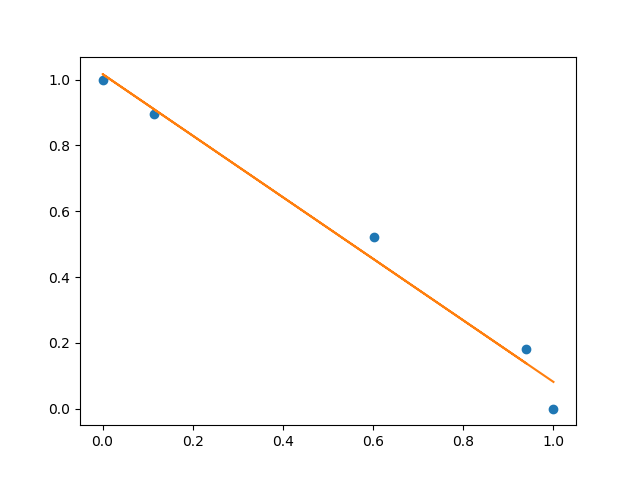
[0.60294118 0.52223987]

[0.11397059 0.8943606 ]

[0. 1. ]

[1. 0. ]]

*3) Linear regression with least squares – closed form*  
Theta equals: [[-0.93516462]

 [ 1.01649371]]s

*4) Linear regression with gradient descent – cost function*  
Cost function:

In our case we only use one variable for the regression line so = 1\*x + 0

Because I am new to machine learning and data mining, approaching SSE from both the closed form and gradient descent method will improve my understanding of the basic concepts. It is also interesting to see how close the gradient descent method will end up to the closed form solution. Hence the choice for SSE.

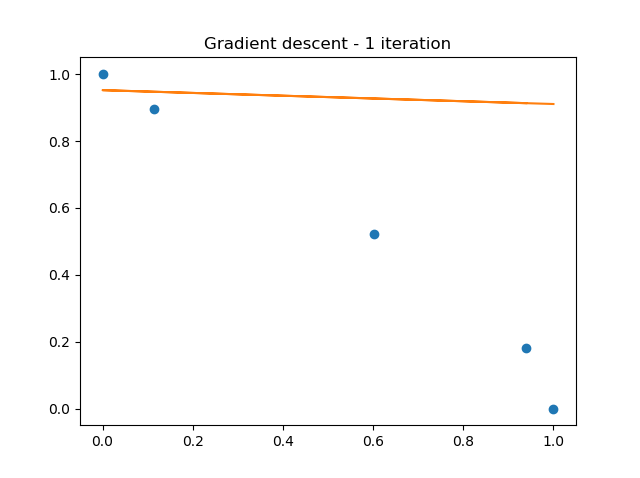
Partial derivative (using the chain rule):

This leads to the update rules:

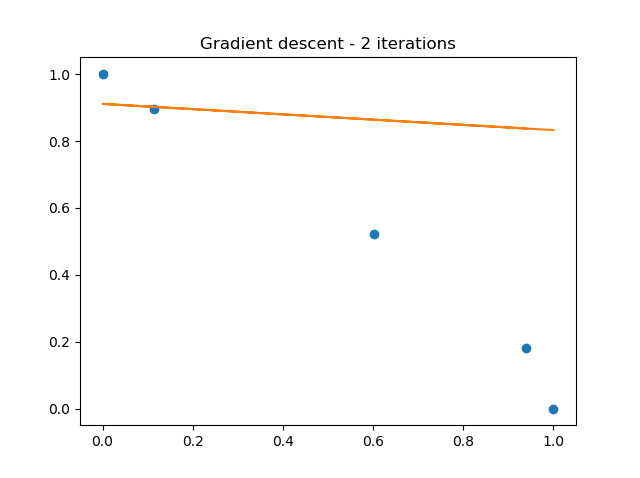
With r the chosen learning rate.

*5) Linear Regression with Gradient Descent - first iteration*

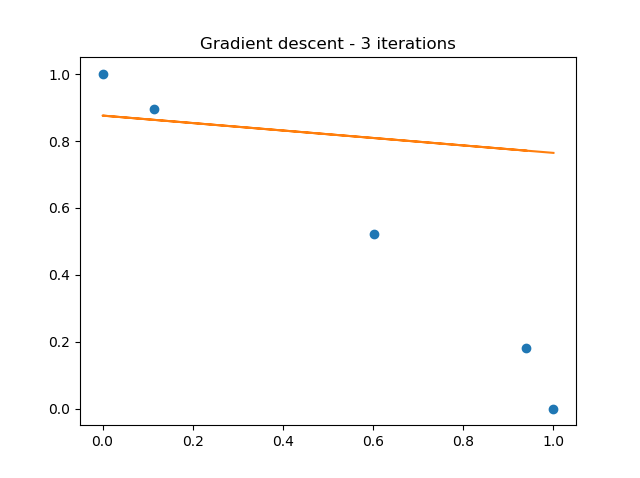
Chosen initial values: , and r = 0.1 (note that theta\_0 and theta\_1 are switched in this exercise compared to the closed form one.)

[0.9519618745035742, -0.04138513292529088]

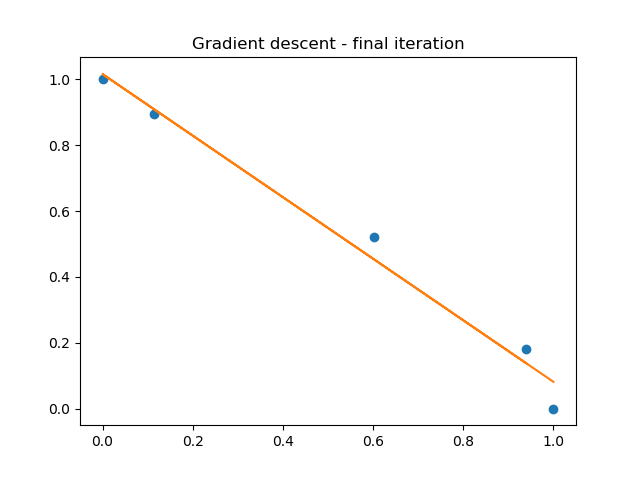
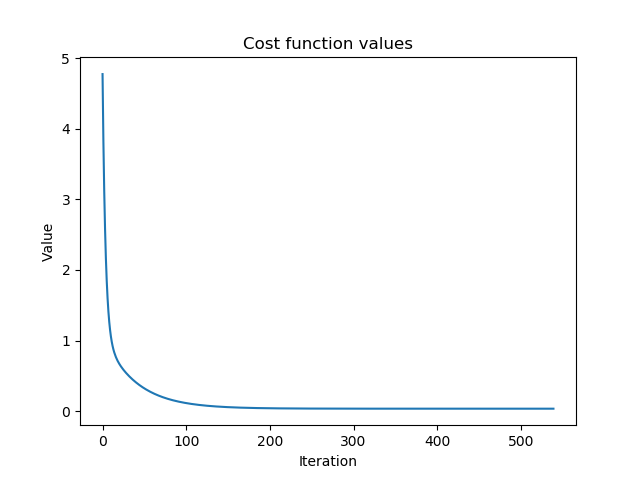
*6) Linear Regression with Gradient Descent - second iteration*

[0.910926451045895, -0.07834763138918316]

7) *Linear Regression with Gradient Descent - third iteration*

[0.8759584744479549, -0.11145943086920693]

*8) Linear Regression with Gradient Descent - last iteration*

[1.016096430954657, -0.9345127025284499]

*9) Discussion*

The final parameter values obtained using the gradient descent method are very close to the closed form solution. This did take over 500 iterations however. It should be noted that the iterations could have been stopped quite a bit sooner and we would still have obtained a very good approximation as can be seen in the cost function graph. This analysis was performed on only 5 data points, it is clear that when the number of points increases, the computational power required is gone increases dramatically with such a high number of iterations. In the end both forms result in accurate regression lines.

One difference between the two methods has already been mentioned in the number of calculations that need to be performed. The closed form is a clear winner over gradient descent here. However, the gradient descent method can be used on a greater number of cases; computing the inverse of a matrix is not always possible and in these cases the gradient descent approach is still able to find a solution where the closed form is not.

# -\*- coding: utf-8 -\*-

"""

Created on Sat Oct 5 20:57:44 2019

@author: Lycolus

"""

import numpy as np

import matplotlib.pyplot as plt

def standerise(vector):

minV = np.amin(vector)

maxV = np.amax(vector)

with np.nditer(vector, op\_flags=['readwrite']) as it:

for value in it:

value[...] = (value - minV) / (maxV - minV)

return vector

def find\_linear\_parameter(x,y):

theta = np.linalg.inv(x.T@x)@(x.T@y)

return theta

data\_x = np.array([[22.72, 18.14, 11.49, 9.94, 23.54]]).T

data\_y = np.array([[453.13, 461.71, 471.08, 473.74, 448.56]]).T

x= standerise(data\_x)

y= standerise(data\_y)

#Closed form solution

ones = np.ones((x.shape[0],1))

x\_ = np.hstack((x,ones))

theta = find\_linear\_parameter(x\_,y)

y2 = x\_@theta

#print('Theta equals: ',theta)

#Gradient descent

r = 0.1

n = x.shape[0]

theta\_GD = [1,0]

change\_theta = 1

cost\_function\_values = []

#for i in range(3):

while change\_theta > 0.00001:

cost\_function = (1/2\*n)\*np.sum((theta\_GD[1]\*x+theta\_GD[0]-y)\*\*2)

cost\_function\_values.append(cost\_function)

new\_theta0 = theta\_GD[0] - r \* (1/n)\*np.sum(theta\_GD[1]\*x+theta\_GD[0]-y)

new\_theta1 = theta\_GD[1] - r \* (1/n)\*np.sum(x\*(theta\_GD[1]\*x+theta\_GD[0]-y))

change\_theta = sum([(theta\_GD[0] - new\_theta0)\*\*2, (theta\_GD[1]-new\_theta1)\*\*2])\*\*0.5

theta\_GD = [new\_theta0,new\_theta1]

print(theta\_GD)

print(len(cost\_function\_values))

y\_GD = x \* theta\_GD[1] + theta\_GD[0]

fig, ax = plt.subplots()

ax.plot(x,y,'o')

ax.plot(x,y\_GD)

ax.set\_title('Gradient descent - final iteration')