

**Gradient Descent Method:**

Implementing the gradient descent method with g being the gradient vector whose calculation is shown on paper. The method is implemented with initial weights=0.25 and 0.4, learning rate=0.08 and threshold=0.5 It is seen that the model converges in 3 iterations and begins to diverge thereafter. With the given parameters, it has run 7 iterations as shown in the graph and the global minimum is found to be 3.3098396711116163. The trajectory followed by the points at each iteration as well as the energy trajectory is plotted.

import math

import numpy as np

import matplotlib.pyplot as plt

w=np.array([0.25,0.4])

W=[]

W.append(w)

threshold=0.5

learning\_rate=0.08

gradient=np.array([(1/(1-w[0]-w[1]))-(1/w[0]),(1/(1-w[0]-w[1]))-(1/w[1])])

epoch=0

F=[]

Epoch=[]

while(np.linalg.norm(learning\_rate\*gradient) < threshold):

if(w[0]+w[1]<1 and w[0]>0 and w[1]>0):

f=-(math.log(1-w[0]-w[1]))-(math.log(w[0]))-(math.log(w[1]))

else:

print('The points do not match the boundary conditions.')

#break

F.append(f)

epoch+=1

Epoch.append(epoch)

w=w-(learning\_rate\*gradient)

W.append(w)

gradient=np.array([(1/(1-w[0]-w[1]))-(1/w[0]),(1/(1-w[0]-w[1]))-(1/w[0])])

print(epoch)

print(F)

W1=[]

W2=[]

for i in range(epoch+1):

W1.append(W[i][0])

W2.append(W[i][1])

plt.plot(W1,W2)

plt.xlabel('Weights W0 ')

plt.ylabel('Weights W1')

plt.title('W0 vs W1')

plt.show()

plt.plot(Epoch,F)

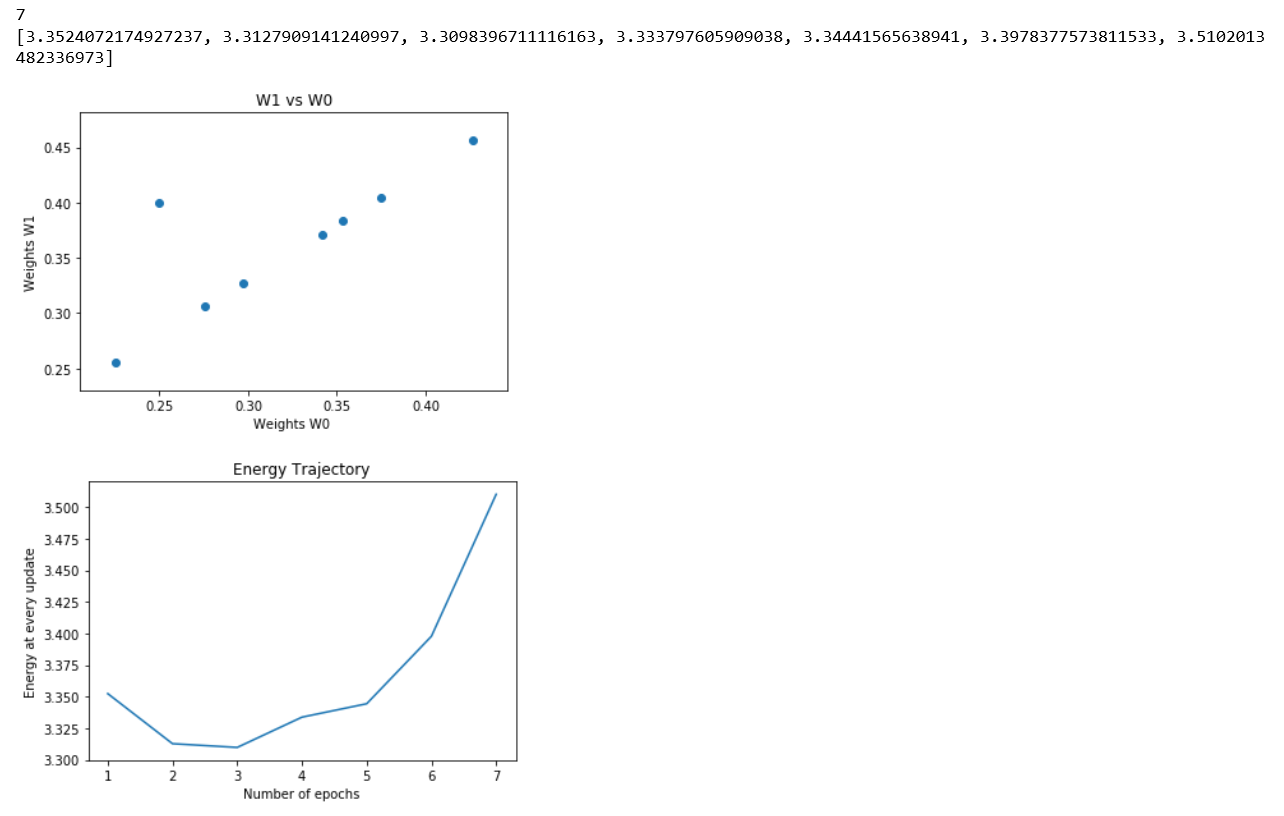
plt.xlabel('Number of epochs')

plt.ylabel('Energy at every update')

plt.title('Energy Trajectory')

plt.show()

Output:



# Newton's Method:

Newton's method has been implemented with the same parameters, learning rate=0.08, threshold=0.5 and initial weights=[0.25,0.4]. The model begins to converge at the 20th epoch and reaches a global minimum at 370th epoch with the minimum energy being 3.3023469020054357.

import math

import numpy as np

import matplotlib.pyplot as plt

w=np.array([0.25,0.4])

w=w.reshape(2,1)

#print(w.size)

threshold=0.5

learning\_rate=0.08

gradient=np.array([(1/(1-w[0][0]-w[1][0]))-(1/w[0][0]),(1/(1-w[0][0]-w[1][0]))-(1/w[1][0])])

gradient=gradient.reshape(2,1)

#print(gradient)

h=np.array([[(1/np.square(1-w[0][0]-w[1][0]))+1/np.square(w[0][0]),1/np.square(1-w[0][0]-w[1][0])],[1/np.square(1-w[0][0]-w[1][0]),(1/np.square(1-w[0][0]-w[1][0]))+1/np.square(w[1][0])]]).reshape(2,2)

H\_inv=np.linalg.inv(h)

epoch=0

F=[]

Epoch=[]

while(np.linalg.norm(learning\_rate\*H\_inv\*gradient) < threshold):

if(w[0][0]+w[1][0]<1 and w[0][0]>0 and w[1][0]>0):

f=-(math.log(1-w[0][0]-w[1][0]))-(math.log(w[0][0]))-(math.log(w[1][0]))

else:

print('The points do not match the boundary conditions.')

break

F.append(f)

epoch+=1

Epoch.append(epoch)

w=w-(learning\_rate\*H\_inv\*gradient)

gradient=np.array([(1/(1-w[0][0]-w[1][0]))-(1/w[0][0]),(1/(1-w[0][0]-w[1][0]))-(1/w[1][0])])

#gradient=Gradient.reshape(2,1)

h=np.array([[(1/np.square(1-w[0][0]-w[1][0]))+1/np.square(w[0][0]),1/np.square(1-w[0][0]-w[1][0])],[1/np.square(1-w[0][0]-w[1][0]),(1/np.square(1-w[0][0]-w[1][0]))+1/np.square(w[1][0])]]).reshape(2,2)

H\_inv=np.linalg.inv(h)

print('The global minumum value and its corresponding index(epoch): ', min(F),',',np.argmin(F))

plt.plot(Epoch[1:400],F[1:400])

plt.xlabel('Number of epochs')

plt.ylabel('Energy at every update')

plt.title('Energy Trajectory')

plt.show()

plt.plot(Epoch[1:50],F[1:50])

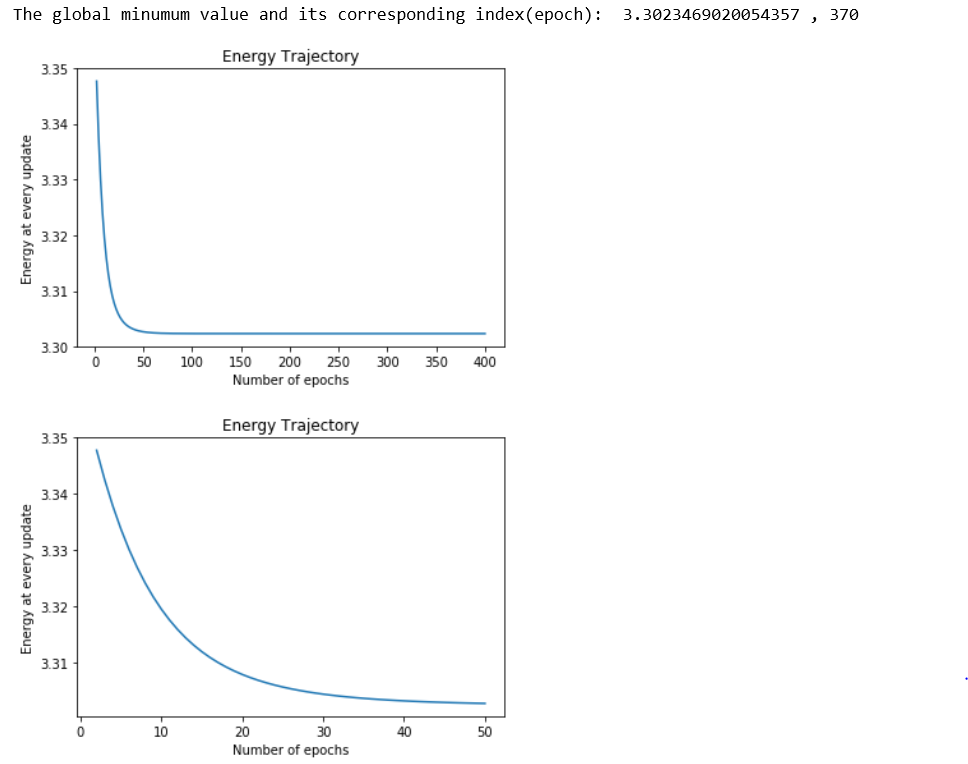
plt.xlabel('Number of epochs')

plt.ylabel('Energy at every update')

plt.title('Energy Trajectory')

plt.show()

Output:



# The speed of convergence between Gradient Descent and Newton's method:

For the given set of parameters, it is seen that the Gradient Descent method reached its estimated global minimum in 3 iterations. The model however diverged beyond that point. Newton's method took alot more iterations to begin convergence. It took around 370 iterations to reach its estimated global minimum. This could be due to the complex computation of the Hessian matrix.