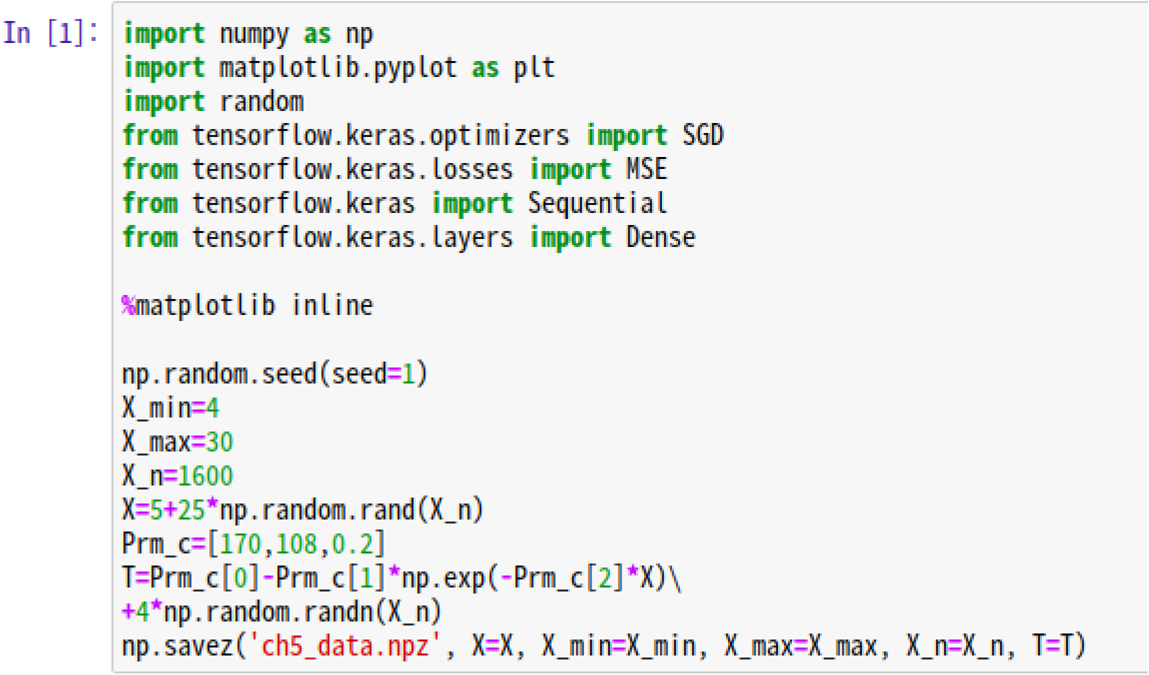
수치 해석 최적화 알고리즘 보고서

20171672 이원형

1. 선택한 최적화 알고리즘의 개요 및 동작원리

저는 두 가지 방법 중 모멘텀 방법을 사용하였습니다. 모멘텀은 이전 방향과 크기에 따라 관성을 주어 가중치에 변화를 갖게 하여 다음 반복시에 바로 전 진행 방향으로 조금 더 이동하게 하는 학습 방법입니다. 그렇기 때문에 진동할 경우에 sgd와 비교하여 높은 성능을 보이는 장점이 있습니다.

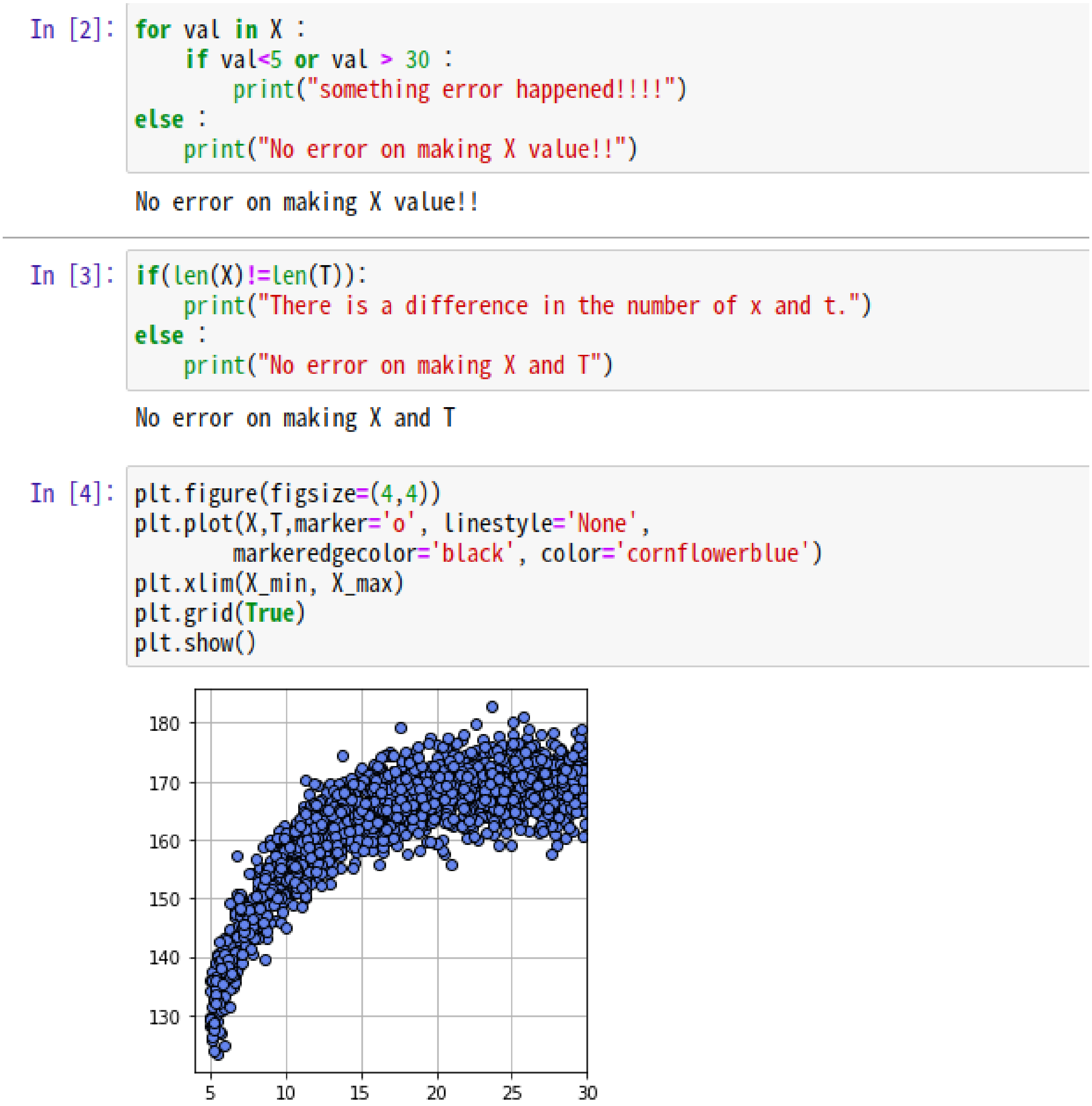
2. 선택한 최적화 알고리즘의 동작 코드와 단위 테스트



In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import random
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.losses import MSE
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

%matplotlib inline

np.random.seed(seed=1)
X_min=4
X_max=30
X_n=1600
X=5+25*np.random.rand(X_n)
Prm_c=[170,108,0.2]
T=Prm_c[0]-Prm_c[1]*np.exp(-Prm_c[2]*X)\
+4*np.random.randn(X_n)
np.savez('ch5_data.npz', X=X, X_min=X_min, X_max=X_max, X_n=X_n, T=T)
```

우선 저는 주교재에 있는 경사 하강법의 데이터를 그대로 차용했습니다. 하지만 표본의 개수가 16개로는 모멘텀을 실행하는데 부족하다고 느껴서 1600개로 바꿔서 사용했습니다.



```
In [2]: for val in X :
            if val<5 or val > 30 :
                print("something error happened!!!!")
        else :
            print("No error on making X value!!")
```

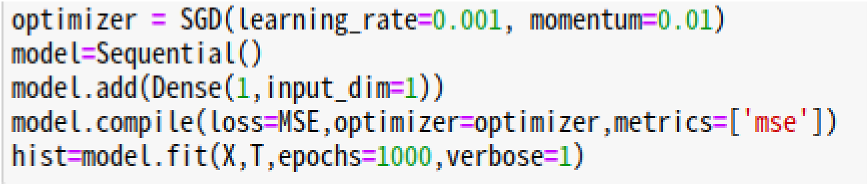
```
No error on making X value!!
```

```
In [3]: if(len(X)!=len(T)):
            print("There is a difference in the number of x and t.")
        else :
            print("No error on making X and T")
```

```
No error on making X and T
```

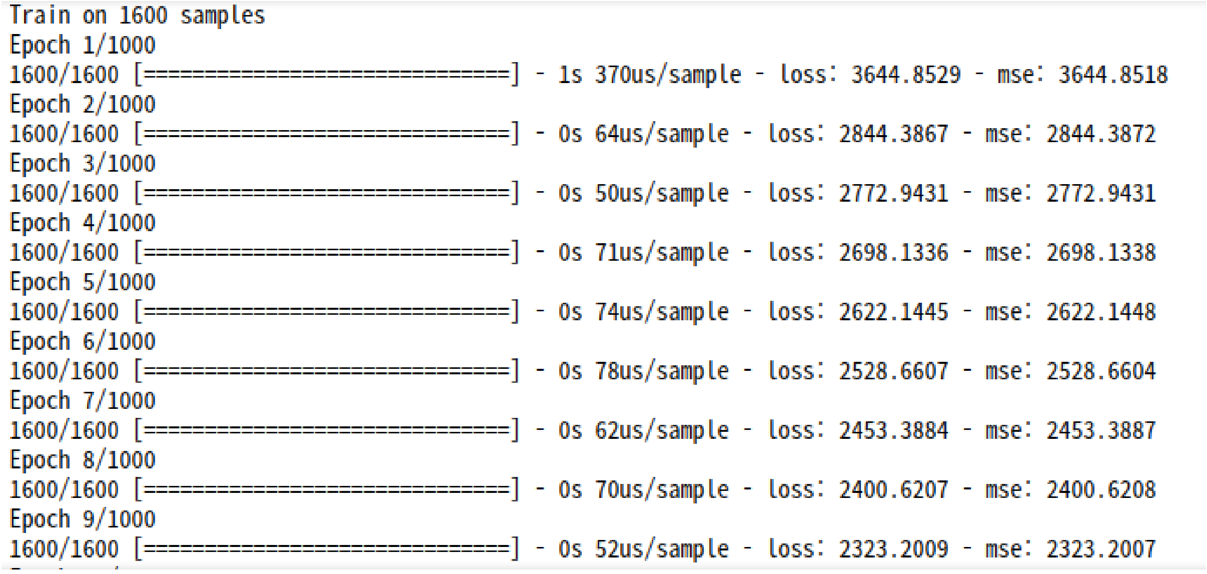
```
In [4]: plt.figure(figsize=(4,4))
        plt.plot(X,T,marker='o', linestyle='None',
                markeredgecolor='black', color='cornflowerblue')
        plt.xlim(X_min, X_max)
        plt.grid(True)
        plt.show()
```

그리고, X의 값이 제대로 들어가고 있는지, 표본 X와 T 값의 개수가 같게 생성되고 있는지 단위 테스트를 진행하였습니다. 그 후 대략적인 모양을 보기 위해 표본들의 집합을 그래프로 표현하였습니다.



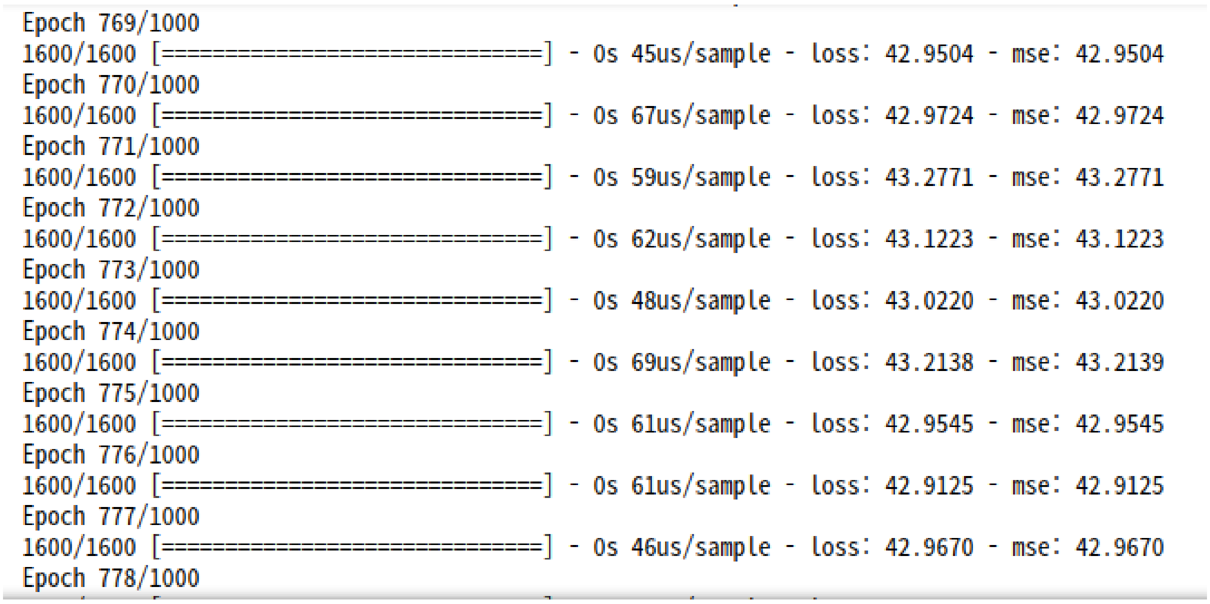
```
optimizer = SGD(learning_rate=0.001, momentum=0.01)
model=Sequential()
model.add(Dense(1,input_dim=1))
model.compile(loss=MSE,optimizer=optimizer,metrics=['mse'])
hist=model.fit(X,T,epochs=1000,verbose=1)
```

그 후, keras의 momentum을 포함하는 SGD optimizer와 Sequential 모델을 생성하였고, Dense를 이용하여 layer를 생성하였습니다. 그 후 keras에 포함되어 있는 MSE를 손실함수로 사용하여 momentum의 결과 값을 확인했습니다.



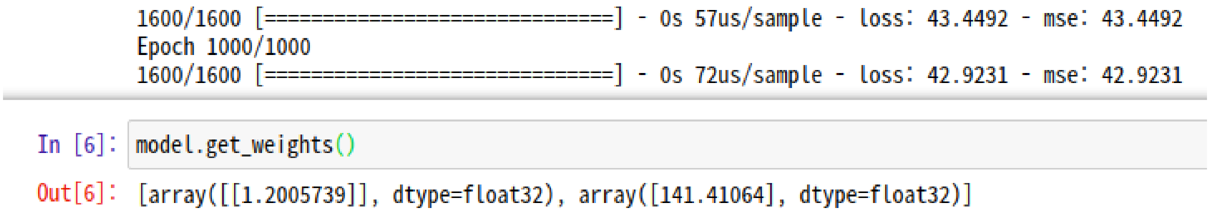
```
Train on 1600 samples
Epoch 1/1000
1600/1600 [==============================] - 1s 370us/sample - loss: 3644.8529 - mse: 3644.8518
Epoch 2/1000
1600/1600 [==============================] - 0s 64us/sample - loss: 2844.3867 - mse: 2844.3872
Epoch 3/1000
1600/1600 [==============================] - 0s 50us/sample - loss: 2772.9431 - mse: 2772.9431
Epoch 4/1000
1600/1600 [==============================] - 0s 71us/sample - loss: 2698.1336 - mse: 2698.1338
Epoch 5/1000
1600/1600 [==============================] - 0s 74us/sample - loss: 2622.1445 - mse: 2622.1448
Epoch 6/1000
1600/1600 [==============================] - 0s 78us/sample - loss: 2528.6607 - mse: 2528.6604
Epoch 7/1000
1600/1600 [==============================] - 0s 62us/sample - loss: 2453.3884 - mse: 2453.3887
Epoch 8/1000
1600/1600 [==============================] - 0s 70us/sample - loss: 2400.6207 - mse: 2400.6208
Epoch 9/1000
1600/1600 [==============================] - 0s 52us/sample - loss: 2323.2009 - mse: 2323.2007
```

(중략)



```
Epoch 769/1000
1600/1600 [==============================] - 0s 45us/sample - loss: 42.9504 - mse: 42.9504
Epoch 770/1000
1600/1600 [==============================] - 0s 67us/sample - loss: 42.9724 - mse: 42.9724
Epoch 771/1000
1600/1600 [==============================] - 0s 59us/sample - loss: 43.2771 - mse: 43.2771
Epoch 772/1000
1600/1600 [==============================] - 0s 62us/sample - loss: 43.1223 - mse: 43.1223
Epoch 773/1000
1600/1600 [==============================] - 0s 48us/sample - loss: 43.0220 - mse: 43.0220
Epoch 774/1000
1600/1600 [==============================] - 0s 69us/sample - loss: 43.2138 - mse: 43.2139
Epoch 775/1000
1600/1600 [==============================] - 0s 61us/sample - loss: 42.9545 - mse: 42.9545
Epoch 776/1000
1600/1600 [==============================] - 0s 61us/sample - loss: 42.9125 - mse: 42.9125
Epoch 777/1000
1600/1600 [==============================] - 0s 46us/sample - loss: 42.9670 - mse: 42.9670
Epoch 778/1000
```

(중략)

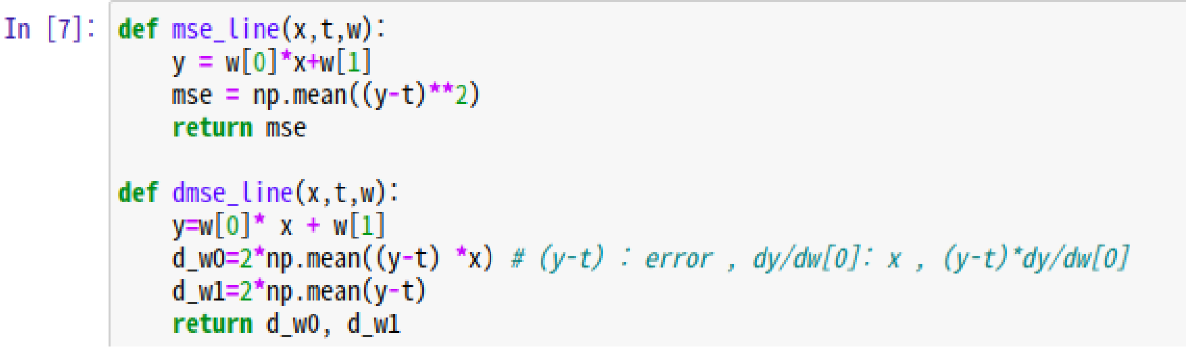


```
1600/1600 [==============================] - 0s 57us/sample - loss: 43.4492 - mse: 43.4492
Epoch 1000/1000
1600/1600 [==============================] - 0s 72us/sample - loss: 42.9231 - mse: 42.9231
```

```
In [6]: model.get_weights()

Out[6]: [array([[1.2005739]], dtype=float32), array([141.41064], dtype=float32)]
```

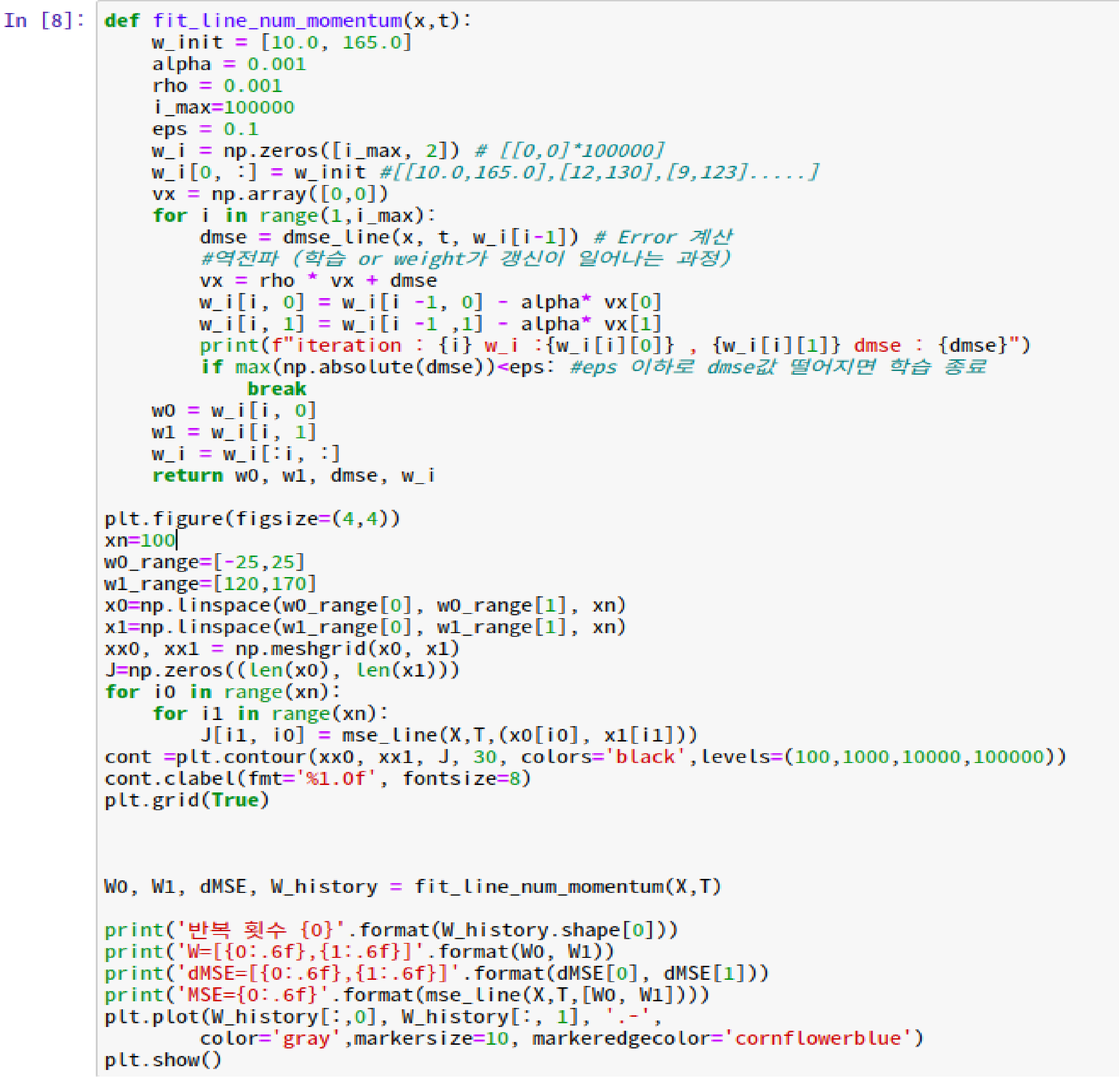
그 결과, 교재와 값은 조금 차이가 있지만 학습이 정상적으로 됨을 확인했고, 마지막 값을 get_weights() 함수를 통해서 확인했습니다.

3. 선택한 최적화 알고리즘의 구체화



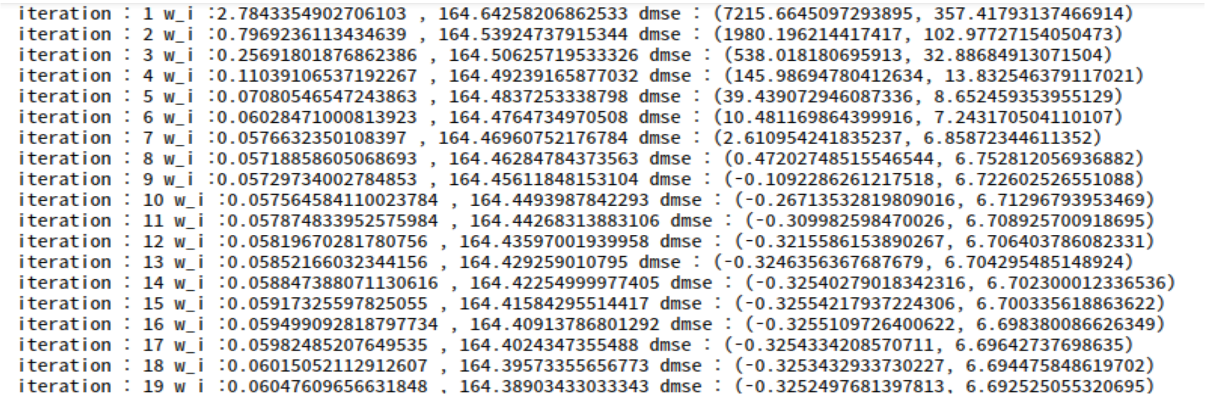
```
In [7]: def mse_line(x,t,w):
            y = w[0]*x+w[1]
            mse = np.mean((y-t)**2)
            return mse

        def dmse_line(x,t,w):
            y=w[0]* x + w[1]
            d_w0=2*np.mean((y-t) *x) # (y-t) : error , dy/dw[0]: x , (y-t)*dy/dw[0]
            d_w1=2*np.mean(y-t)
            return d_w0, d_w1
```

이제 numpy 레벨에서 momentum을 구현하기 위해 먼저 mse 함수를 직접 만들었고, 그 미분값을 return하는 함수를 구현했습니다



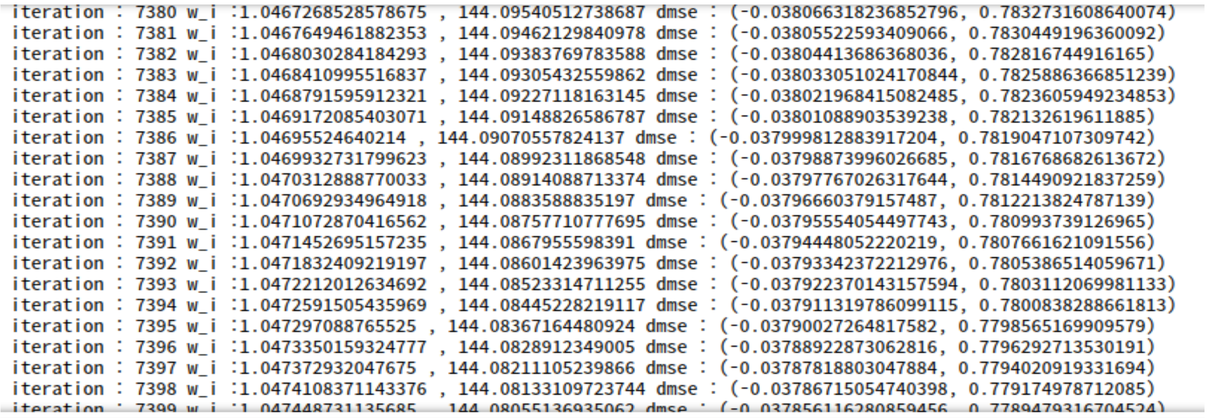
```
In [8]: def fit_line_num_momentum(x,t):
            w_init = [10.0, 165.0]
            alpha = 0.001
            rho = 0.001
            i_max=100000
            eps = 0.1
            w_i = np.zeros([i_max, 2]) # [[0,0]*100000]
            w_i[0, :] = w_init #[[10.0,165.0],[12,130],[9,123].....]
            vx = np.array([0,0])
            for i in range(1,i_max):
                dmse = dmse_line(x, t, w_i[i-1]) # Error 계산
                #역전파 (학습 or weight가 갱신이 일어나는 과정)
                vx = rho * vx + dmse
                w_i[i, 0] = w_i[i -1, 0] - alpha* vx[0]
                w_i[i, 1] = w_i[i -1 ,1] - alpha* vx[1]
                print(f"iteration : {i} w_i :{w_i[i][0]} , {w_i[i][1]} dmse : {dmse}")
                if max(np.absolute(dmse))<eps: #eps 이하로 dmse값 떨어지면 학습 종료
                    break
            w0 = w_i[i, 0]
            w1 = w_i[i, 1]
            w_i = w_i[:i, :]
            return w0, w1, dmse, w_i

        plt.figure(figsize=(4,4))
        xn=100
        w0_range=[-25,25]
        w1_range=[120,170]
        x0=np.linspace(w0_range[0], w0_range[1], xn)
        x1=np.linspace(w1_range[0], w1_range[1], xn)
        xx0, xx1 = np.meshgrid(x0, x1)
        J=np.zeros((len(x0), len(x1)))
        for i0 in range(xn):
            for i1 in range(xn):
                J[i1, i0] = mse_line(X,T,(x0[i0], x1[i1]))
        cont =plt.contour(xx0, xx1, J, 30, colors='black',levels=(100,1000,10000,100000))
        cont.clabel(fmt='%1.0f', fontsize=8)
        plt.grid(True)



        W0, W1, dMSE, W_history = fit_line_num_momentum(X,T)

        print('반복 횟수 {0}'.format(W_history.shape[0]))
        print('W=[{0:.6f},{1:.6f}]'.format(W0, W1))
        print('dMSE=[{0:.6f},{1:.6f}]'.format(dMSE[0], dMSE[1]))
        print('MSE={0:.6f}'.format(mse_line(X,T,[W0, W1])))
        plt.plot(W_history[:,0], W_history[:, 1], '.-',
                 color='gray',markersize=10, markeredgecolor='cornflowerblue')
        plt.show()
```

그 후 다음과 같이 모멘텀을 구현하였습니다. 초기 값을 [10.0, 165.0]으로 주었고, 반복 횟수는 10만 번, 학습률은 0.1로 설정하였습니다. 그 후 dmse의 오차 값이 eps 값보다 낮아지면 종료하도록 했으며, 매번 학습할 때마다 전 학습의 진행 방향과 크기에 따라 가중치를 달리 하도록 하였습니다. 실행 결과는 다음과 같습니다.



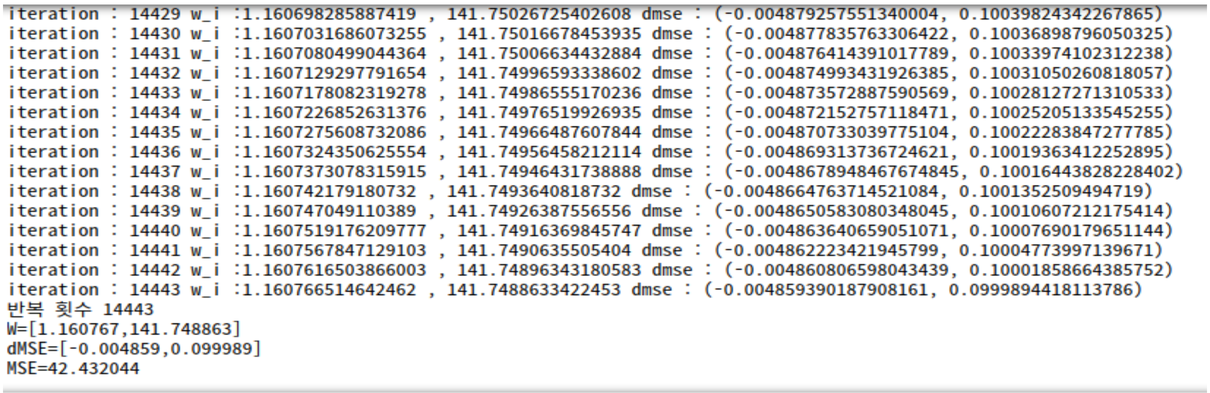
```
 iteration : 1 w_i :2.7843354902706103 , 164.64258206862533 dmse : (7215.6645097293895, 357.41793137466914)
 iteration : 2 w_i :0.7969236113434639 , 164.53924737915344 dmse : (1980.196214417417, 102.97727154050473)
 iteration : 3 w_i :0.25691801876862386 , 164.50625719533326 dmse : (538.018180695913, 32.88684913071504)
 iteration : 4 w_i :0.11039106537192267 , 164.49239165877032 dmse : (145.98694780412634, 13.832546379117021)
 iteration : 5 w_i :0.07080546547243863 , 164.4837253338798 dmse : (39.439072946087336, 8.652459353955129)
 iteration : 6 w_i :0.06028471000813923 , 164.4764734970508 dmse : (10.481169864399916, 7.243170504110107)
 iteration : 7 w_i :0.0576632350108397 , 164.46960752176784 dmse : (2.610954241835237, 6.85872344611352)
 iteration : 8 w_i :0.05718858605068693 , 164.46284784373563 dmse : (0.47202748515546544, 6.752812056936882)
 iteration : 9 w_i :0.05729734002784853 , 164.45611848153104 dmse : (-0.1092286261217518, 6.722602526551088)
 iteration : 10 w_i :0.057564584110023784 , 164.4493987842293 dmse : (-0.26713532819809016, 6.71296793953469)
 iteration : 11 w_i :0.057874833952575984 , 164.44268313883106 dmse : (-0.309982598470026, 6.708925700918695)
 iteration : 12 w_i :0.05819670281780756 , 164.43597001939958 dmse : (-0.3215586153890267, 6.706403786082331)
 iteration : 13 w_i :0.05852166032344156 , 164.429259010795 dmse : (-0.3246356367687679, 6.704295485148924)
 iteration : 14 w_i :0.058847388071130616 , 164.42254999977405 dmse : (-0.32540279018342316, 6.702300012336536)
 iteration : 15 w_i :0.05917325597825055 , 164.41584295514417 dmse : (-0.32554217937224306, 6.700335618863622)
 iteration : 16 w_i :0.059499092818797734 , 164.40913786801292 dmse : (-0.3255109726400622, 6.698380086626349)
 iteration : 17 w_i :0.05982485207649535 , 164.4024347355488 dmse : (-0.325433420857071 1, 6.69642737698635)
 iteration : 18 w_i :0.06015052112912607 , 164.39573355656773 dmse : (-0.3253432933730227, 6.694475848619702)
 iteration : 19 w i :0.06047609656631848 . 164.38903433033343 dmse : (-0.3252497681397813. 6.692525055320695)
```

(중략)



```
iteration : 7380 w_i :1.0467268528578675 , 144.09540512738687 dmse : (-0.038066318236852796, 0.7832731608640074)
iteration : 7381 w_i :1.0467649461882353 , 144.09462129840978 dmse : (-0.03805522593409066, 0.7830449196360092)
iteration : 7382 w_i :1.0468030284184293 , 144.09383769783588 dmse : (-0.03804413686368036, 0.782816744916165)
iteration : 7383 w_i :1.0468410995516837 , 144.09305432559862 dmse : (-0.038033051024170844, 0.7825886366851239)
iteration : 7384 w_i :1.0468791595912321 , 144.09227118163145 dmse : (-0.038021968415082485, 0.7823605949234853)
iteration : 7385 w_i :1.0469172085403071 , 144.09148826586787 dmse : (-0.0380108890353923 8, 0.782132619611885)
iteration : 7386 w_i :1.04695524640214 , 144.09070557824137 dmse : (-0.037999812883917204, 0.7819047107309742)
iteration : 7387 w_i :1.0469932731799623 , 144.08992311868548 dmse : (-0.03798873996026685, 0.7816768682613672)
iteration : 7388 w_i :1.0470312888770033 , 144.08914088713374 dmse : (-0.03797767026317644, 0.7814490921837259)
iteration : 7389 w_i :1.0470692934964918 , 144.0883588835197 dmse : (-0.03796660379157487, 0.7812213824787139)
iteration : 7390 w_i :1.0471072870416562 , 144.08757710777695 dmse : (-0.03795554054497743, 0.780993739126965)
iteration : 7391 w_i :1.0471452695157235 , 144.0867955598391 dmse : (-0.03794448052220219, 0.7807661621091556)
iteration : 7392 w_i :1.0471832409219197 , 144.08601423963975 dmse : (-0.03793342372212976, 0.7805386514059671)
iteration : 7393 w_i :1.0472212012634692 , 144.08523314711255 dmse : (-0.037922370143157594, 0.7803112069981133)
iteration : 7394 w_i :1.0472591505435969 , 144.08445228219117 dmse : (-0.037911319786099115, 0.7800838288661813)
iteration : 7395 w_i :1.047297088765525 , 144.08367164480924 dmse : (-0.03790027264817582, 0.7798565169909579)
iteration : 7396 w_i :1.0473350159324777 , 144.0828912349005 dmse : (-0.03788922873062816, 0.7796292713530191)
iteration : 7397 w_i :1.047372932047675 , 144.08211105239866 dmse : (-0.03787818803047884, 0.7794020919331694)
iteration : 7398 w_i :1.0474108371143376 , 144.08133109723744 dmse : (-0.03786715054740398, 0.779174978712085)
iteration : 7399 w_i :1.047448731135685 , 144.08055136935062 dmse : (-0.037856116280859456, 0.7789479316704524)
```

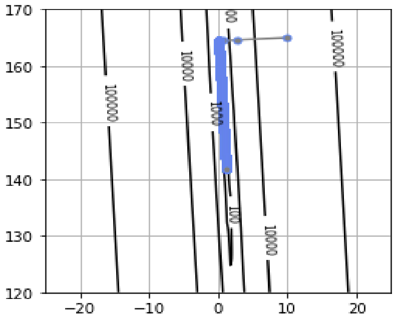
(중략)

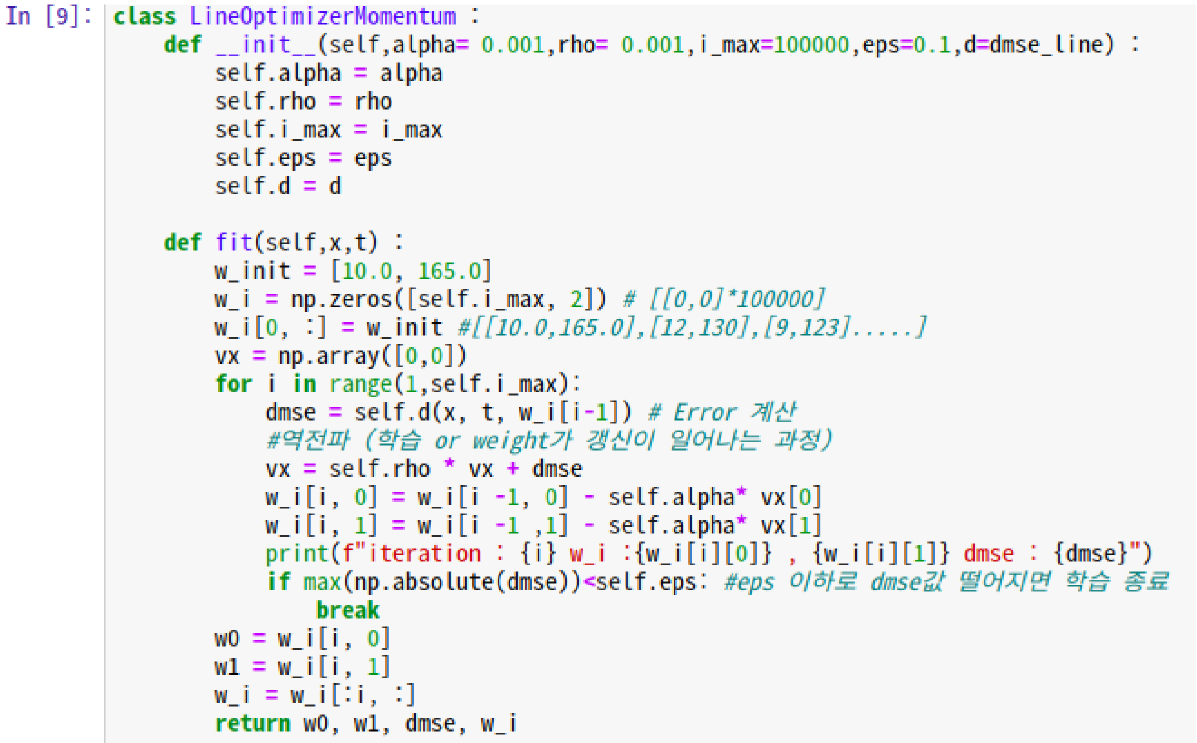


```
iteration : 14429 w_i :1.160698285887419 , 141.75026725402608 dmse : (-0.004879257551340004, 0.10039824342267865)
iteration : 14430 w_i :1.1607031686073255 , 141.75016678453935 dmse : (-0.004877835763306422, 0.10036898796050325)
iteration : 14431 w_i :1.1607080499044364 , 141.75006634432884 dmse : (-0.004876414391017789, 0.10033974102312238)
iteration : 14432 w_i :1.1607129297791654 , 141.74996593338602 dmse : (-0.004874993431926385, 0.10031050260818057)
iteration : 14433 w_i :1.1607178082319278 , 141.74986555170236 dmse : (-0.004873572887590569, 0.10028127271310533)
iteration : 14434 w_i :1.1607226852631376 , 141.74976519926935 dmse : (-0.004872152757118471, 0.10025205133545255)
iteration : 14435 w_i :1.1607275608732086 , 141.74966487607844 dmse : (-0.004870733039775104, 0.10022283847277785)
iteration : 14436 w_i :1.1607324350625554 , 141.74956458212114 dmse : (-0.004869313736724621, 0.10019363412252895)
iteration : 14437 w_i :1.1607373078315915 , 141.74946431738888 dmse : (-0.004867894846767484 5, 0.10016443828228402)
iteration : 14438 w_i :1.160742179180732 , 141.7493640818732 dmse : (-0.004866476371452108 4, 0.1001352509494719)
iteration : 14439 w_i :1.160747049110389 , 141.74926387556556 dmse : (-0.004865058308034804 5, 0.10010607212175414)
iteration : 14440 w_i :1.1607519176209777 , 141.74916369845747 dmse : (-0.004863640659051071, 0.10007690179651144)
iteration : 14441 w_i :1.1607567847129103 , 141.7490635505404 dmse : (-0.004862223421945799, 0.10004773997139671)
iteration : 14442 w_i :1.1607616503866003 , 141.74896343180583 dmse : (-0.004860806598043439, 0.10001858664385752)
iteration : 14443 w_i :1.160766514642462 , 141.7488633422453 dmse : (-0.004859390187908161, 0.0999894418113786)
반복 횟수 14443
W=[1.160767,141.748863]
dMSE=[-0.004859,0.099989]
MSE=42.432044
```

위와 같은 과정을 거쳐 14443 반복을 통해 학습했습니다. 위에서 keras를 활용한 momentum과 거의 같은 값을 가짐을 알 수 있었습니다. 또, 이 결과 값을 교재와 같은 그래프로도 나타냈습니다.

그 결과는 오른쪽과 같습니다.

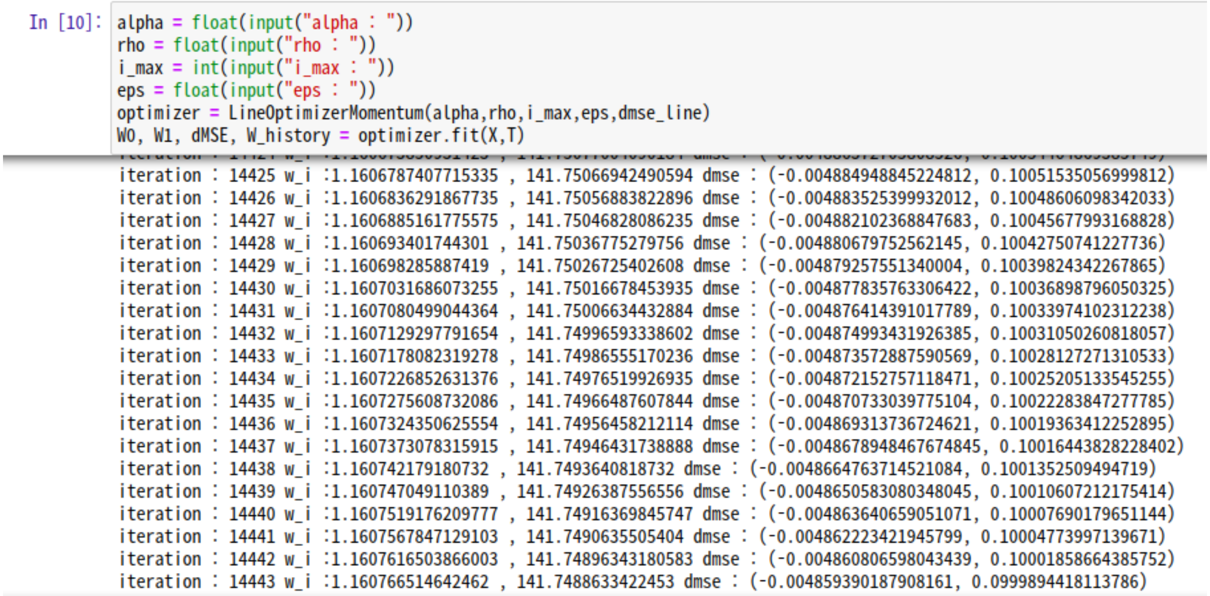




```
In [9]: class LineOptimizerMomentum :
            def __init__(self,alpha= 0.001,rho= 0.001,i_max=100000,eps=0.1,d=dmse_line) :
                self.alpha = alpha
                self.rho = rho
                self.i_max = i_max
                self.eps = eps
                self.d = d

            def fit(self,x,t) :
                w_init = [10.0, 165.0]
                w_i = np.zeros([self.i_max, 2]) # [[0,0]*100000]
                w_i[0, :] = w_init #[[10.0,165.0],[12,130],[9,123].....]
                vx = np.array([0,0])
                for i in range(1,self.i_max):
                    dmse = self.d(x, t, w_i[i-1]) # Error 계산
                    #역전파 (학습 or weight가 갱신이 일어나는 과정)
                    vx = self.rho * vx + dmse
                    w_i[i, 0] = w_i[i -1, 0] - self.alpha* vx[0]
                    w_i[i, 1] = w_i[i -1 ,1] - self.alpha* vx[1]
                    print(f"iteration : {i} w_i :{w_i[i][0]} , {w_i[i][1]} dmse : {dmse}")
                    if max(np.absolute(dmse))<self.eps: #eps 이하로 dmse값 떨어지면 학습 종료
                        break
                w0 = w_i[i, 0]
                w1 = w_i[i, 1]
                w_i = w_i[:i, :]
                return w0, w1, dmse, w_i
```

그 후, 클래스로 만들어 numpy 레벨에서 구현한 momentum을 객체지향화 하였습니다.

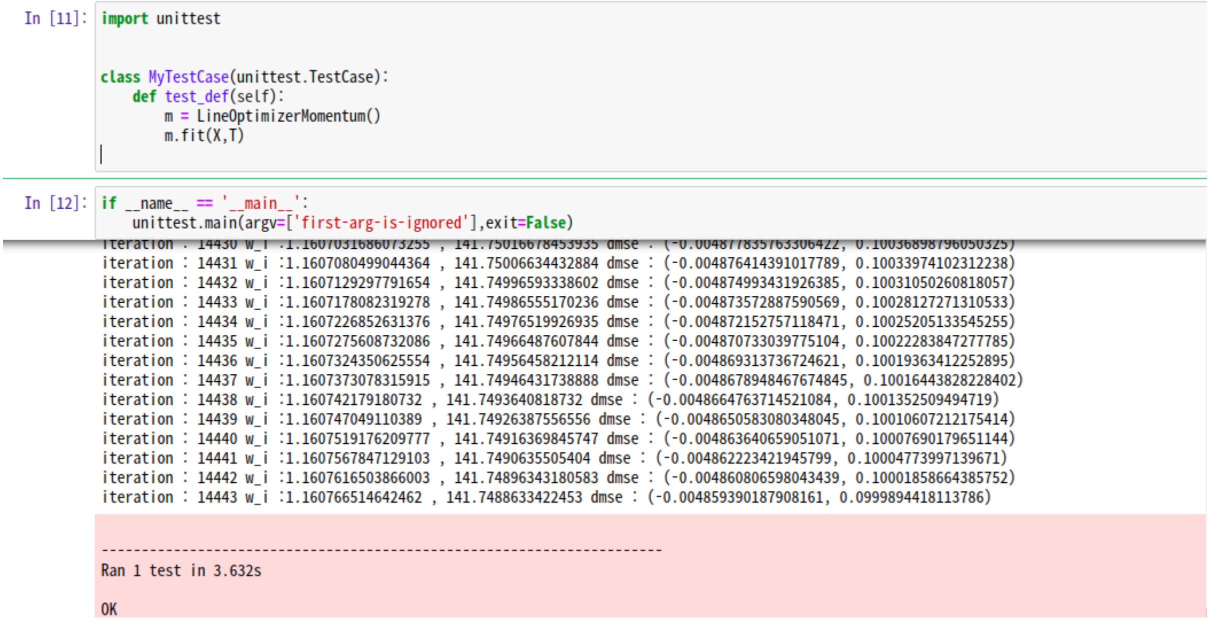
4. 각 구체화 모듈의 단위 테스트



```
In [10]: alpha = float(input("alpha : "))
         rho = float(input("rho : "))
         i_max = int(input("i_max : "))
         eps = float(input("eps : "))
         optimizer = LineOptimizerMomentum(alpha,rho,i_max,eps,dmse_line)
         W0, W1, dMSE, W_history = optimizer.fit(X,T)
```

```
iteration : 14425 w_i :1.1606787407715335 , 141.75066942490594 dmse : (-0.004884948845224812, 0.10051535056999812)
iteration : 14426 w_i :1.1606836291867735 , 141.75056883822896 dmse : (-0.004883525399932012, 0.10048606098342033)
iteration : 14427 w_i :1.1606885161775575 , 141.75046828086235 dmse : (-0.004882102368847683, 0.10045677993168828)
iteration : 14428 w_i :1.160693401744301 , 141.75036775279756 dmse : (-0.004880679752562145, 0.10042750741227736)
iteration : 14429 w_i :1.160698285887419 , 141.75026725402608 dmse : (-0.004879257551340004, 0.10039824342267865)
iteration : 14430 w_i :1.1607031686073255 , 141.75016678453935 dmse : (-0.004877835763306422, 0.10036898796050325)
iteration : 14431 w_i :1.1607080499044364 , 141.75006634432884 dmse : (-0.004876414391017789, 0.10033974102312238)
iteration : 14432 w_i :1.1607129297791654 , 141.74996593338602 dmse : (-0.004874993431926385, 0.10031050260818057)
iteration : 14433 w_i :1.1607178082319278 , 141.74986555170236 dmse : (-0.004873572887590569, 0.10028127271310533)
iteration : 14434 w_i :1.1607226852631376 , 141.74976519926935 dmse : (-0.004872152757118471, 0.10025205133545255)
iteration : 14435 w_i :1.1607275608732086 , 141.74966487607844 dmse : (-0.004870733039775104, 0.10022283847277785)
iteration : 14436 w_i :1.1607324350625554 , 141.74956458212114 dmse : (-0.004869313736724621, 0.10019363412252895)
iteration : 14437 w_i :1.1607373078315915 , 141.74946431738888 dmse : (-0.0048678948467674845, 0.10016443828228402)
iteration : 14438 w_i :1.160742179180732 , 141.7493640818732 dmse : (-0.0048664763714521084, 0.1001352509494719)
iteration : 14439 w_i :1.160747049110389 , 141.74926387556556 dmse : (-0.0048650583080348045, 0.10010607212175414)
iteration : 14440 w_i :1.1607519176209777 , 141.74916369845747 dmse : (-0.004863640659051071, 0.10007690179651144)
iteration : 14441 w_i :1.1607567847129103 , 141.7490635505404 dmse : (-0.004862223421945799, 0.10004773997139671)
iteration : 14442 w_i :1.1607616503866003 , 141.74896343180583 dmse : (-0.004860806598043439, 0.10001858664385752)
iteration : 14443 w_i :1.160766514642462 , 141.7488633422453 dmse : (-0.004859390187908161, 0.0999894418113786)
```

그 후, class화한 LineOptimizerMomentum의 객체를 optimizer로 선언하여, Input 값을 위의 함수와 같게 직접 주고, 출력 값이 같게 나옴을 확인함으로써 단위 테스트를 진행하였습니다.



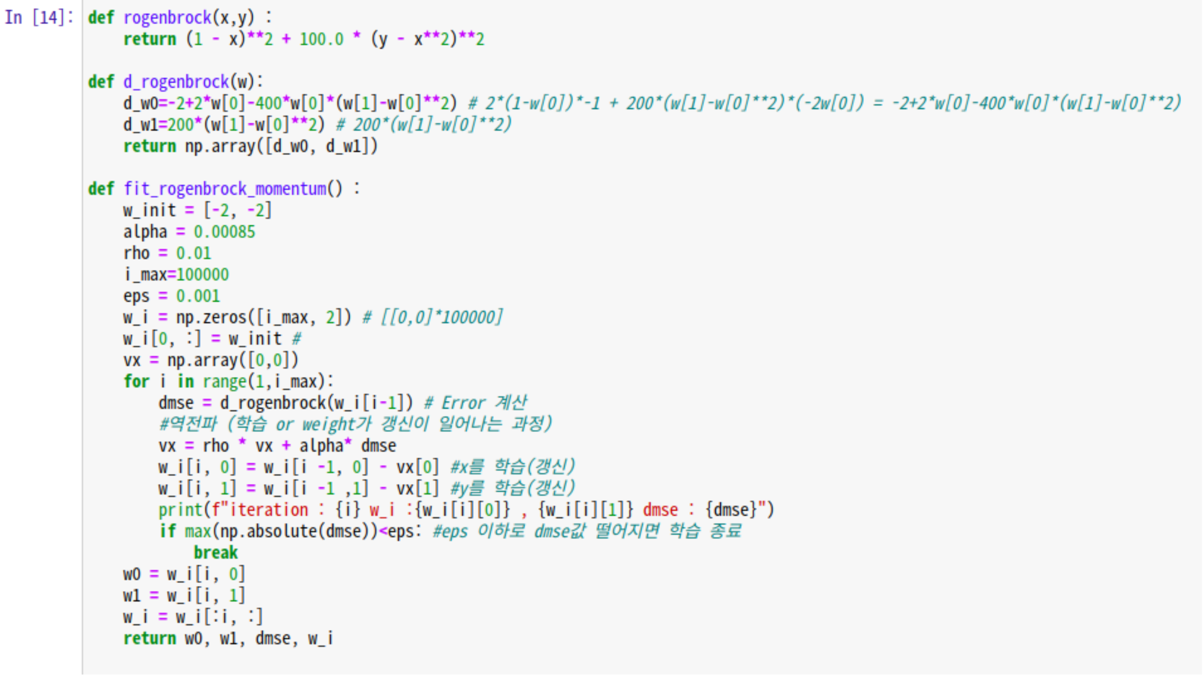
```
In [11]: import unittest


         class MyTestCase(unittest.TestCase):
             def test_def(self):
                 m = LineOptimizerMomentum()
                 m.fit(X,T)
```

```
In [12]: if __name__ == '__main__':
             unittest.main(argv=['first-arg-is-ignored'],exit=False)
```

```
iteration : 14430 w_i :1.1607031686073255 , 141.75016678453935 dmse : (-0.004877835763306422, 0.10036898796050325)
iteration : 14431 w_i :1.1607080499044364 , 141.75006634432884 dmse : (-0.004876414391017789, 0.10033974102312238)
iteration : 14432 w_i :1.1607129297791654 , 141.74996593338602 dmse : (-0.004874993431926385, 0.10031050260818057)
iteration : 14433 w_i :1.1607178082319278 , 141.74986555170236 dmse : (-0.004873572887590569, 0.10028127271310533)
iteration : 14434 w_i :1.1607226852631376 , 141.74976519926935 dmse : (-0.004872152757118471, 0.10025205133545255)
iteration : 14435 w_i :1.1607275608732086 , 141.74966487607844 dmse : (-0.004870733039775104, 0.10022283847277785)
iteration : 14436 w_i :1.1607324350625554 , 141.74956458212114 dmse : (-0.004869313736724621, 0.10019363412252895)
iteration : 14437 w_i :1.1607373078315915 , 141.74946431738888 dmse : (-0.004867894848467674845, 0.10016443828228402)
iteration : 14438 w_i :1.160742179180732 , 141.7493640818732 dmse : (-0.0048664763714521084, 0.1001352509494719)
iteration : 14439 w_i :1.160747049110389 , 141.74926387556556 dmse : (-0.0048650583080348045, 0.10010607212175414)
iteration : 14440 w_i :1.1607519176209777 , 141.74916369845747 dmse : (-0.004863640659051071, 0.10007690179651144)
iteration : 14441 w_i :1.1607567847129103 , 141.7490635505404 dmse : (-0.004862223421945799, 0.10004773997139671)
iteration : 14442 w_i :1.1607616503866003 , 141.74896343180583 dmse : (-0.004860806598043439, 0.10001858664385752)
iteration : 14443 w_i :1.160766514642462 , 141.7488633422453 dmse : (-0.004859390187908161, 0.0999894418113786)

----------------------------------------------------------------------
Ran 1 test in 3.632s

OK
```

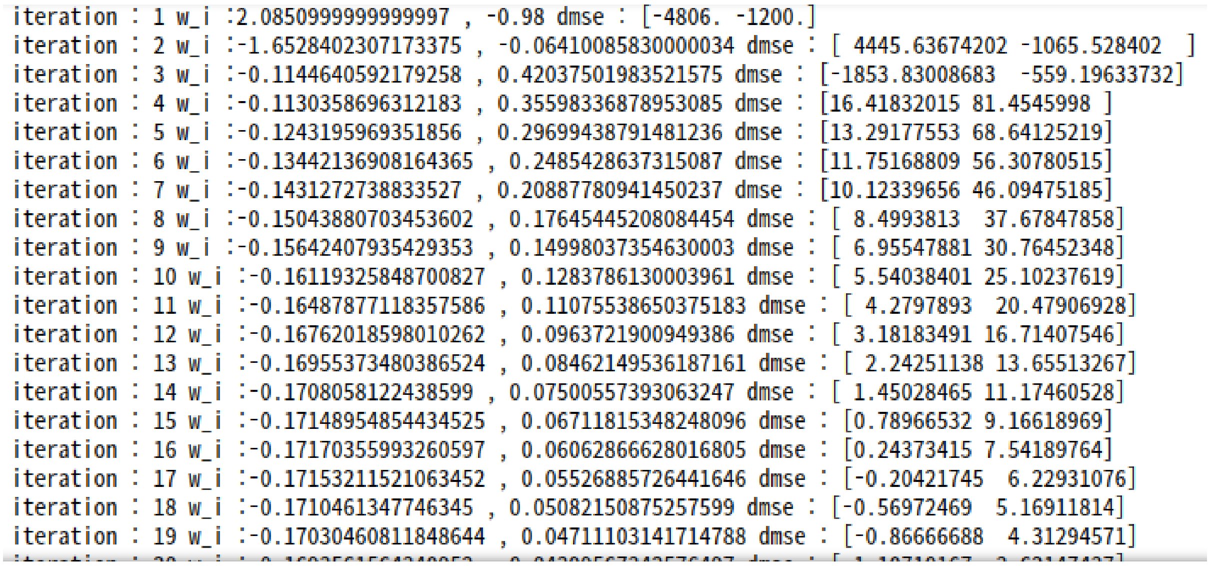
이후, unittest를 import 하여서도 단위 테스트를 한 번 더 진행하였고, 역시나 같은 값이 출력되며 OK가 출력됨을 확인할 수 있었습니다.

5. 선택한 최적화 알고리즘의 검증



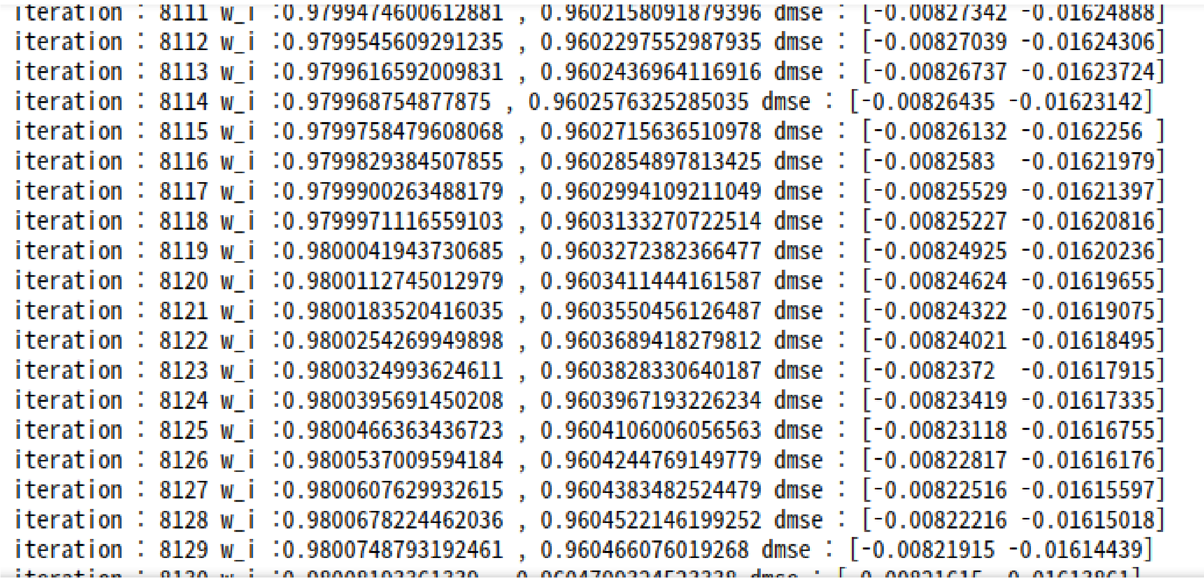
```
In [14]: def rogenbrock(x,y) :
             return (1 - x)**2 + 100.0 * (y - x**2)**2

         def d_rogenbrock(w):
             d_w0=-2+2*w[0]-400*w[0]*(w[1]-w[0]**2) # 2*(1-w[0])*-1 + 200*(w[1]-w[0]**2)*(-2w[0]) = -2+2*w[0]-400*w[0]*(w[1]-w[0]**2)
             d_w1=200*(w[1]-w[0]**2) # 200*(w[1]-w[0]**2)
             return np.array([d_w0, d_w1])

         def fit_rogenbrock_momentum() :
             w_init = [-2, -2]
             alpha = 0.00085
             rho = 0.01
             i_max=100000
             eps = 0.001
             w_i = np.zeros([i_max, 2]) # [[0,0]*100000]
             w_i[0, :] = w_init #
             vx = np.array([0,0])
             for i in range(1,i_max):
                 dmse = d_rogenbrock(w_i[i-1]) # Error 계산
                 #역전파 (학습 or weight가 갱신이 일어나는 과정)
                 vx = rho * vx + alpha* dmse
                 w_i[i, 0] = w_i[i -1, 0] - vx[0] #x를 학습(갱신)
                 w_i[i, 1] = w_i[i -1 ,1] - vx[1] #y를 학습(갱신)
                 print(f"iteration : {i} w_i :{w_i[i][0]} , {w_i[i][1]} dmse : {dmse}")
                 if max(np.absolute(dmse))<eps: #eps 이하로 dmse값 떨어지면 학습 종료
                     break
             w0 = w_i[i, 0]
             w1 = w_i[i, 1]
             w_i = w_i[:i, :]
             return w0, w1, dmse, w_i
```

로젠브록 함수에 a,b 값을 각각 1,100으로 정해 주었습니다. 그 뒤 키와 나이를 데이터로 했을 때와 마찬가지로 미분 값을 구하고, 초기 값과 알파 값을 조금 조정하여 모멘텀 함수를 그대로 가져왔습니다.



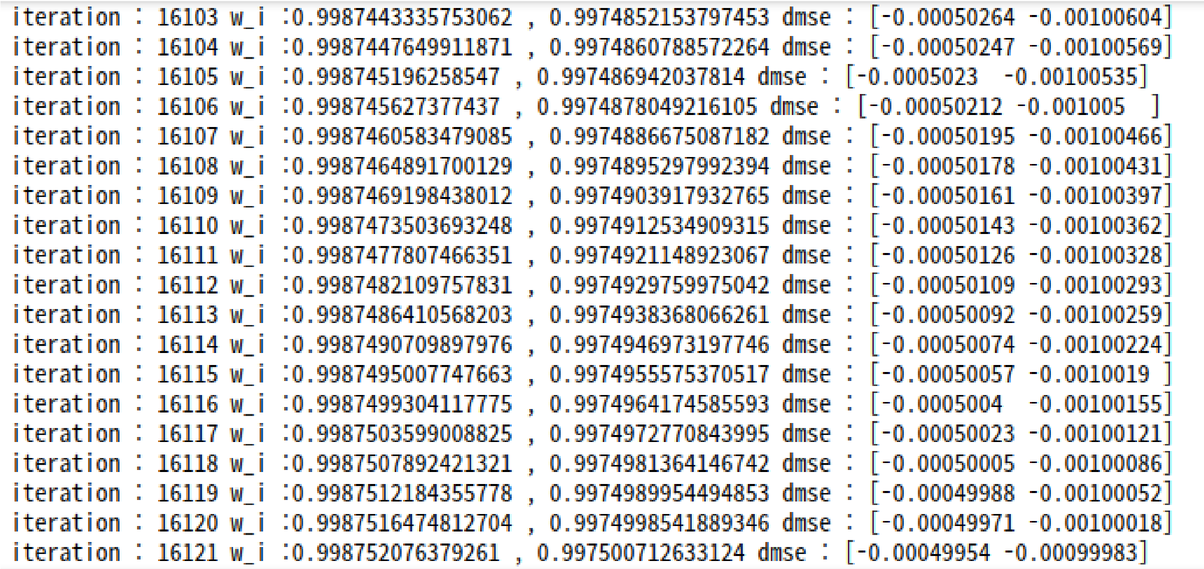
```
iteration : 1 w_i :2.0850999999999997 , -0.98 dmse : [-4806. -1200.]
iteration : 2 w_i :-1.6528402307173375 , -0.06410085830000034 dmse : [ 4445.63674202 -1065.528402  ]
iteration : 3 w_i :-0.1144640592179258 , 0.42037501983521575 dmse : [-1853.83008683  -559.19633732]
iteration : 4 w_i :-0.1130358696312183 , 0.35598336878953085 dmse : [16.41832015 81.4545998 ]
iteration : 5 w_i :-0.1243195969351856 , 0.29699438791481236 dmse : [13.29177553 68.64125219]
iteration : 6 w_i :-0.13442136908164365 , 0.2485428637315087 dmse : [11.75168809 56.30780515]
iteration : 7 w_i :-0.143127273883352 7 , 0.20887780941450237 dmse : [10.12339656 46.09475185]
iteration : 8 w_i :-0.15043880703453602 , 0.17645445208084454 dmse : [ 8.4993813  37.67847858]
iteration : 9 w_i :-0.15642407935429353 , 0.14998037354630003 dmse : [ 6.95547881 30.76452348]
iteration : 10 w_i :-0.16119325848700827 , 0.1283786130003961 dmse : [ 5.54038401 25.10237619]
iteration : 11 w_i :-0.16487877118357586 , 0.11075538650375183 dmse : [ 4.2797893  20.47906928]
iteration : 12 w_i :-0.16762018598010262 , 0.0963721900949386 dmse : [ 3.18183491 16.71407546]
iteration : 13 w_i :-0.16955373480386524 , 0.08462149536187161 dmse : [ 2.24251138 13.65513267]
iteration : 14 w_i :-0.1708058122438599 , 0.07500557393063247 dmse : [ 1.45028465 11.17460528]
iteration : 15 w_i :-0.17148954854434525 , 0.06711815348248096 dmse : [0.78966532 9.16618969]
iteration : 16 w_i :-0.17170355993260597 , 0.06062866628016805 dmse : [0.24373415 7.54189764]
iteration : 17 w_i :-0.17153211521063452 , 0.05526885726441646 dmse : [-0.20421745  6.22931076]
iteration : 18 w_i :-0.1710461347746345 , 0.05082150875257599 dmse : [-0.56972469  5.16911814]
iteration : 19 w_i :-0.17030460811848644 , 0.04711103141714788 dmse : [-0.86666688  4.31294571]
```

(중략)



```
iteration : 8111 w_i :0.9799474600612881 , 0.9602158091879396 dmse : [-0.00827342 -0.01624888]
iteration : 8112 w_i :0.9799545609291235 , 0.9602297552987935 dmse : [-0.00827039 -0.01624306]
iteration : 8113 w_i :0.9799616592009831 , 0.9602436964116916 dmse : [-0.00826737 -0.01623724]
iteration : 8114 w_i :0.979968754877875 , 0.9602576325285035 dmse : [-0.00826435 -0.01623142]
iteration : 8115 w_i :0.9799758479608068 , 0.9602715636510978 dmse : [-0.00826132 -0.0162256 ]
iteration : 8116 w_i :0.9799829384507855 , 0.9602854897813425 dmse : [-0.0082583  -0.01621979]
iteration : 8117 w_i :0.9799900263488179 , 0.9602994109211049 dmse : [-0.00825529 -0.01621397]
iteration : 8118 w_i :0.9799971116559103 , 0.9603133270722514 dmse : [-0.00825227 -0.01620816]
iteration : 8119 w_i :0.9800041943730685 , 0.9603272382366477 dmse : [-0.00824925 -0.01620236]
iteration : 8120 w_i :0.9800112745012979 , 0.9603411444161587 dmse : [-0.00824624 -0.01619655]
iteration : 8121 w_i :0.9800183520416035 , 0.9603550456126487 dmse : [-0.00824322 -0.01619075]
iteration : 8122 w_i :0.9800254269949898 , 0.9603689418279812 dmse : [-0.00824021 -0.01618495]
iteration : 8123 w_i :0.9800324993624611 , 0.9603828330640187 dmse : [-0.0082372  -0.01617915]
iteration : 8124 w_i :0.9800395691450208 , 0.9603967193226234 dmse : [-0.00823419 -0.01617335]
iteration : 8125 w_i :0.9800466363436723 , 0.9604106006056563 dmse : [-0.00823118 -0.01616755]
iteration : 8126 w_i :0.9800537009594184 , 0.9604244769149779 dmse : [-0.00822817 -0.01616176]
iteration : 8127 w_i :0.9800607629932615 , 0.9604383482524479 dmse : [-0.00822516 -0.01615597]
iteration : 8128 w_i :0.9800678224462036 , 0.9604522146199252 dmse : [-0.00822216 -0.01615018]
iteration : 8129 w_i :0.9800748793192461 , 0.960466076019268 dmse : [-0.00821915 -0.01614439]
iteration : 8130 w_i :0.98008193361339 , 0.9604799324523338 dmse : [-0.00821615 -0.01613861]
```

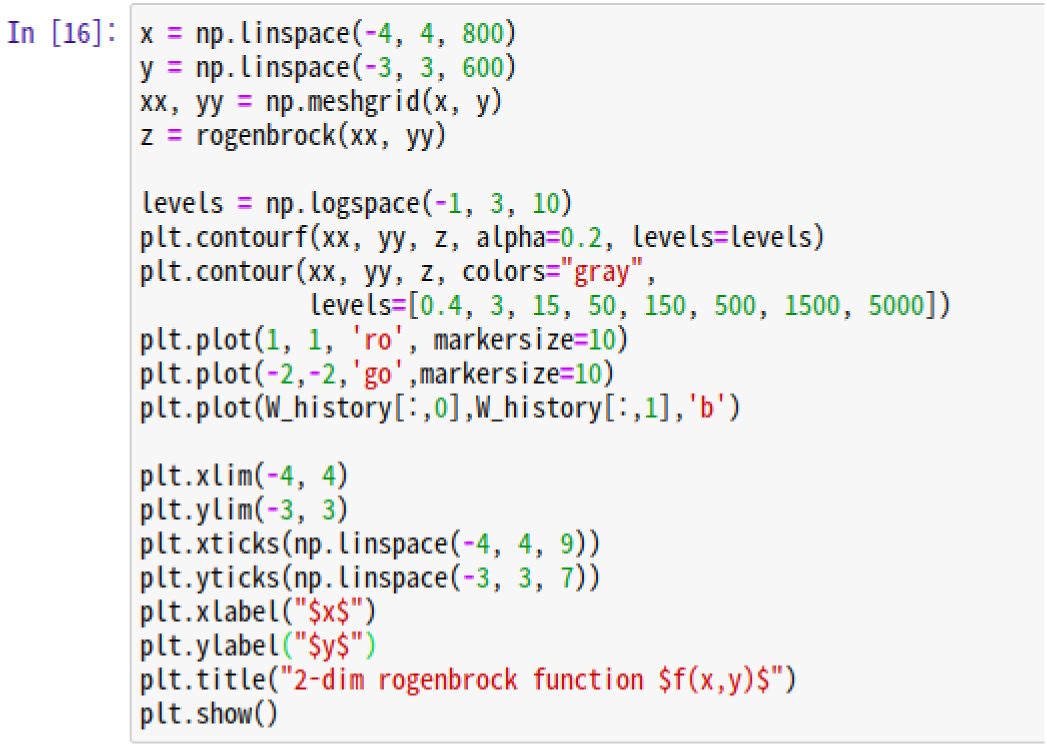
(중략)



```
iteration : 16103 w_i :0.9987443335753062 , 0.9974852153797453 dmse : [-0.00050264 -0.00100604]
iteration : 16104 w_i :0.9987447649911871 , 0.9974860788572264 dmse : [-0.00050247 -0.00100569]
iteration : 16105 w_i :0.998745196258547 , 0.997486942037814 dmse : [-0.0005023  -0.00100535]
iteration : 16106 w_i :0.998745627377437 , 0.9974878049216105 dmse : [-0.00050212 -0.001005  ]
iteration : 16107 w_i :0.9987460583479085 , 0.9974886675087182 dmse : [-0.00050195 -0.00100466]
iteration : 16108 w_i :0.9987464891700129 , 0.9974895297992394 dmse : [-0.00050178 -0.00100431]
iteration : 16109 w_i :0.9987469198438012 , 0.9974903917932765 dmse : [-0.00050161 -0.00100397]
iteration : 16110 w_i :0.9987473503693248 , 0.9974912534909315 dmse : [-0.00050143 -0.00100362]
iteration : 16111 w_i :0.9987477807466351 , 0.9974921148923067 dmse : [-0.00050126 -0.00100328]
iteration : 16112 w_i :0.9987482109757831 , 0.9974929759975042 dmse : [-0.00050109 -0.00100293]
iteration : 16113 w_i :0.9987486410568203 , 0.9974938368066261 dmse : [-0.00050092 -0.00100259]
iteration : 16114 w_i :0.9987490709897976 , 0.9974946973197746 dmse : [-0.00050074 -0.00100224]
iteration : 16115 w_i :0.9987495007747663 , 0.9974955575370517 dmse : [-0.00050057 -0.0010019 ]
iteration : 16116 w_i :0.9987499304117775 , 0.9974964174585593 dmse : [-0.0005004  -0.00100155]
iteration : 16117 w_i :0.9987503599008825 , 0.9974972770843995 dmse : [-0.00050023 -0.00100121]
iteration : 16118 w_i :0.9987507892421321 , 0.9974981364146742 dmse : [-0.00050005 -0.00100086]
iteration : 16119 w_i :0.9987512184355778 , 0.9974989954494853 dmse : [-0.00049988 -0.00100052]
iteration : 16120 w_i :0.9987516474812704 , 0.9974998541889346 dmse : [-0.00049971 -0.00100018]
iteration : 16121 w_i :0.998752076379261 , 0.997500712633124 dmse : [-0.00049954 -0.00099983]
```

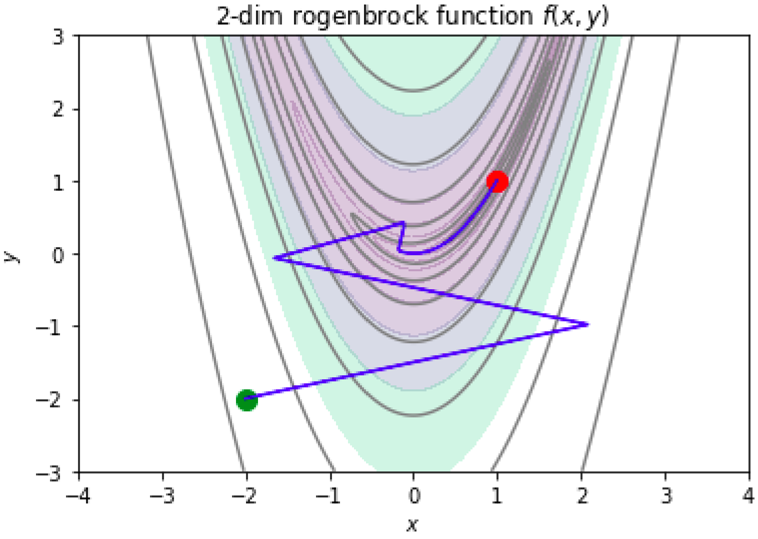
위와 같은 과정을 거쳐 1,1에 근사함을 알 수 있었습니다. 또, 한 눈에 보기 쉽게 결과 값을 2d, 3d로도 나타내 보았습니다.

2D 그래프입니다.

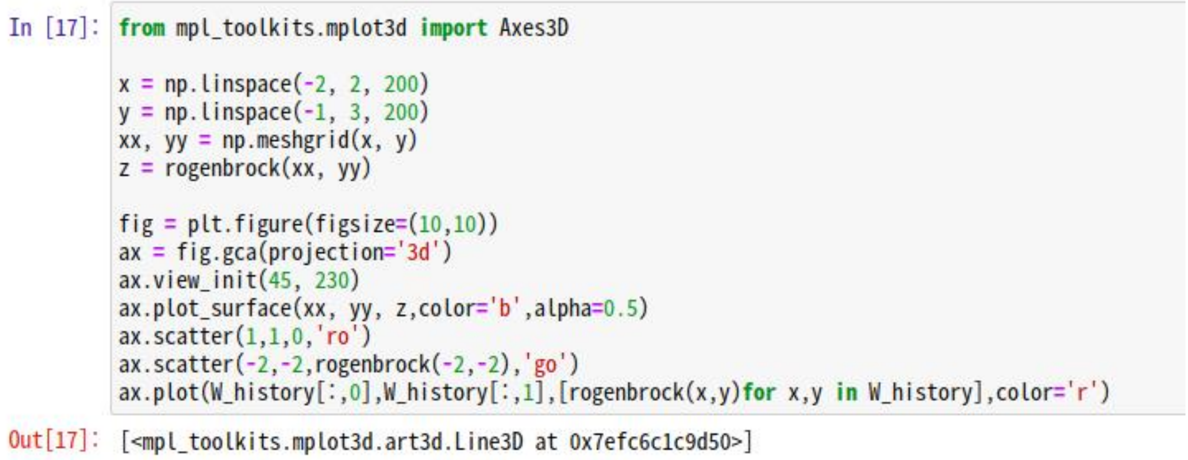


In [16]:

```
x = np.linspace(-4, 4, 800)
y = np.linspace(-3, 3, 600)
xx, yy = np.meshgrid(x, y)
z = rogenbrock(xx, yy)

levels = np.logspace(-1, 3, 10)
plt.contourf(xx, yy, z, alpha=0.2, levels=levels)
plt.contour(xx, yy, z, colors="gray",
            levels=[0.4, 3, 15, 50, 150, 500, 1500, 5000])
plt.plot(1, 1, 'ro', markersize=10)
plt.plot(-2,-2,'go',markersize=10)
plt.plot(W_history[:,0],W_history[:,1],'b')

plt.xlim(-4, 4)
plt.ylim(-3, 3)
plt.xticks(np.linspace(-4, 4, 9))
plt.yticks(np.linspace(-3, 3, 7))
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.title("2-dim rogenbrock function $f(x,y)$")
plt.show()
```

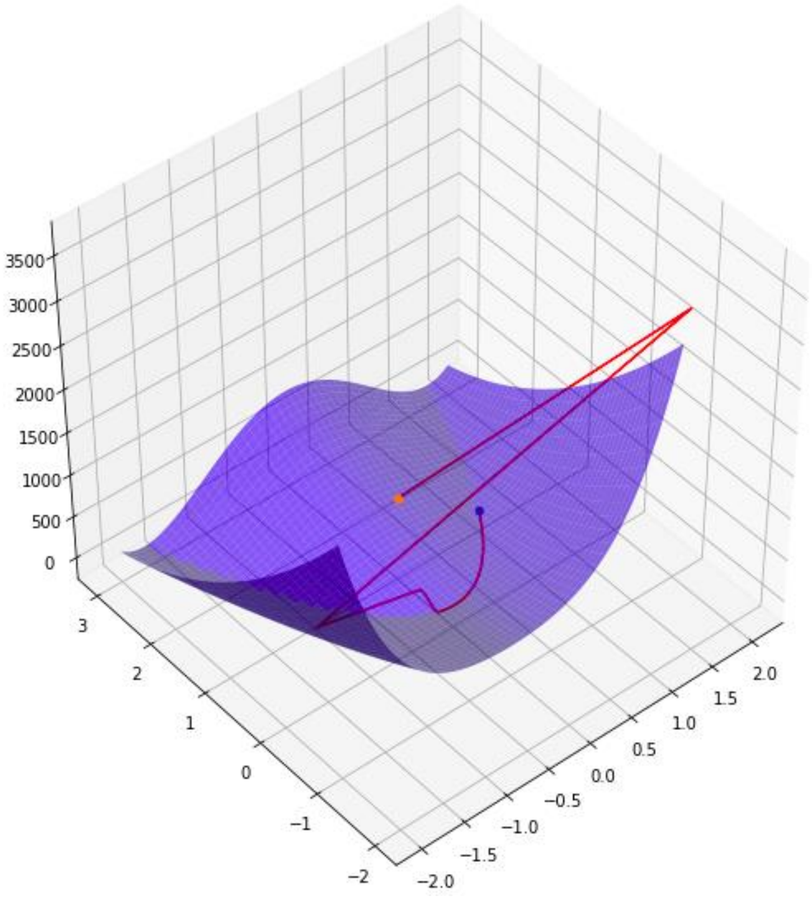


3D그래프입니다.



In [17]:
```
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(-2, 2, 200)
y = np.linspace(-1, 3, 200)
xx, yy = np.meshgrid(x, y)
z = rogenbrock(xx, yy)

fig = plt.figure(figsize=(10,10))
ax = fig.gca(projection='3d')
ax.view_init(45, 230)
ax.plot_surface(xx, yy, z,color='b',alpha=0.5)
ax.scatter(1,1,0,'ro')
ax.scatter(-2,-2,rogenbrock(-2,-2),'go')
ax.plot(W_history[:,0],W_history[:,1],[rogenbrock(x,y)for x,y in W_history],color='r')
```

Out[17]:
```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x7efc6c1c9d50>]
```



이상입니다.