COM761 Machine Learning Coursework 1

2021/22

Deadline: 12:00 noon, Wednesday 27th October 2021

This is an individual assignment: University regulations on plagiarism will apply.

This piece of CW is worth 40\% of your overall mark for COM761.

The total number of marks available in this assignment is 40.

Enter your solutions directly into this Jupyter notebook.

For each question, you should provide your answer **in the cell immediately below the question**.

If a question requires a **numerical answer**, your cell should include python code carrying out the relevant calculations, and include a `print` statement to make clear what your answer to the question is.

If a question requires a **graphical plot**, your cell should include python code that results in the required plot being printed to screen.

You need to submit BOTH of the following via Blackboard before the deadline:

1. **This Jupyter notebook (containing your solutions), AND**
2. **A pdf of this Jupyter notebook (containing your solutions).**

   You can convert your Jupyter notebook to pdf by:

   - printing directly from your browser to pdf
   - if the above does not work in your browser, go to File-> Download As... -> HTML (.html). Then open the html file in Goggle chrome, and print from chrome to pdf.

Data set for use in Q1 and Q2

At the Tokyo olympics, the decathlon took place over 4-5 August 2021.

The events on day one (4th August 2021) took place in this order:

- 100m
- long jump
- shot put
- high jump
- 400m

The events on day two (5th August 2021) took place in this order:

- 110m hurdles
- discus throw
- pole vault
- javelin throw
- 1500m

There are two data-files provided:

1. `decathlon_Tokyo.csv` contains the names of 19 athletes who completed the decathlon at the Tokyo olympics, along with their results in each event. The columns represent the events in the order they took place: 100m, long jump, shot put, high jump, 400m, 110m hurdles, discus throw, pole vault, javelin throw and 1500m.
2. `decathlon_pb.csv` contains the the names of 19 athletes who completed the decathlon at the Tokyo olympics, along with their height (cm), weight (kg), age (months) and personal best (PB) performances in each event going into the Tokyo olympics.

Question 1

Total marks for question: 20

(a) Create a single dataframe representing the two data-sets `decathlon_Tokyo.csv` and `decathlon_pb.csv`. Drop the `name` column from the single dataframe and display the first five rows. **[3 marks]**

In [2]:

```
## Enter code for (a) here
#import libraries for reading and merge data Frame
from pandas import DataFrame
import pandas as pd
import numpy as np
tokyo = pd.read_csv("C:\\Users\\KALYAN\\Downloads\\decathlon_Tokyo.csv")
pb = pd.read_csv("C:\\Users\\KALYAN\\Downloads\\decathlon_pb.csv")
#merge two data frame into single using merge function.
ave_merge = (tokyo.merge(pb))
#drop the name column from data frame
ave_merge.drop(['Name'], axis=1, inplace = True)
ave_merge.head()
```

Out[2]:

	TOK_100m	TOK_LJ	TOK_SP	TOK_HJ	TOK_400m	TOK_110m_H	TOK_DT	TOK_PV	TOK_JT	TOK_1500m	...
0	10.12	8.24	14.80	2.02	47.48	13.46	48.67	4.9	63.44	271.1	...
1	10.68	7.50	15.07	2.08	50.31	13.90	48.08	5.2	73.09	283.2	...
2	10.34	7.64	14.49	2.11	46.29	14.08	44.38	5.0	57.12	279.2	...
3	10.67	7.30	15.59	1.99	48.25	14.03	45.46	5.1	69.10	275.5	...
4	10.43	7.65	15.31	1.99	46.92	14.39	47.14	5.0	57.24	271.8	...

5 rows × 23 columns

(b) There are several missing values for weights and heights of athletes. Imput values for these into your dataframe, adding a brief comment explaining your method. *All of this data will be used as training data, so you do not need to worry about data leakage.* **[4 marks]**

In [4]:

```
## Enter code for (b) here
#replace missing values with of the remaning values.
ave_merge['Height'].fillna(ave_merge['Height'].mean(), inplace = True)
ave_merge['Weight'].fillna(ave_merge['Weight'].mean(), inplace = True)
print(ave_merge.isnull().sum())
```

```
TOK_100m      0
TOK_LJ        0
TOK_SP        0
TOK_HJ        0
TOK_400m      0
TOK_110m_H    0
TOK_DT        0
TOK_PV        0
TOK_JT        0
TOK_1500m     0
Height        0
Weight        0
Age           0
PB_100m       0
PB_LJ         0
PB_SP         0
PB_HJ         0
PB_400m       0
PB_110m_H     0
PB_DT         0
PB_PV         0
PB_JT         0
PB_1500m      0
dtype: int64
```

(c) One competitor (not in the data set), Thomas Van der Plaetsen, injured himself in the second event (the long jump) and had to withdraw from the remainder of the competition. His performance in the first event (the 100m) was 11.05 s [1].

Thomas is 186 cm tall, 81 kg and was 30 years and 7 months old at the time of the Tokyo Olympics. His personal bests at the time of the Olympics were [2]:

- 100 metres – 11.04 s
- 400 metres – 48.64 s
- 1500 metres – 4 minutes 32.52 s
- 110 metres hurdles – 14.36 s
- High jump – 2.17 m
- Pole vault – 5.45 m
- Long jump – 7.90 m
- Shot put – 14.28 m
- Discus Throw – 48.81 m
- Javelin Throw – 65.31 m

Use sklearn to construct multiple linear regression models to predict how Thomas Van der Plaetsen would have performed in the remaining nine events at Tokyo, had he been able to compete. You should print your predictions for each event to screen. **[13 marks]**

[1] https://www.eurosport.com/athletics/tokyo-2020/2021/tokyo-2020-injury-sees-desperately-unfortunate-thomas-van-der-plaetsen-plummet-head-first-into-long-_sto8471315/story.shtml

[2] https://en.wikipedia.org/wiki/Thomas_Van_der_Plaetsen

In [5]:

```
## Enter code for (c) here
##in this scenario i am taking thomas numerical features of personal best as my indepe
##in order to avoid data leakage and i dont want these numerical features of thomas d
ave_merge2 = {'TOK_100m': [np.nan],
             'TOK_LJ': [np.nan],
             'TOK_SP': [np.nan],
             'TOK_HJ': [np.nan],
             'TOK_400m': [np.nan],
             'TOK_110m_H': [np.nan],
             'TOK_DT': [np.nan],
             'TOK_PV': [np.nan],
             'TOK_JT': [np.nan],
             'TOK_1500m': [np.nan],
             'P_100m': [11.04], 'P_LJ': [7.9], 'P_SP': [14.28], 'P_HJ': [2.17], 'P_400n
dff = DataFrame(ave_merge2,columns=['TOK_100m','TOK_LJ','TOK_SP','TOK_HJ','TOK_400m',
dff.head()
```

Out[5]:

	TOK_100m	TOK_LJ	TOK_SP	TOK_HJ	TOK_400m	TOK_110m_H	TOK_DT	TOK_PV	TOK_JT	TOK_1500m	P_
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

In [25]:

```
##selecting my independent and target features for each event column11 is my target fe
column11 = ['TOK_100m']
##column12 is my independent feature of thomas which is used to find tokyo 100m of the
column12 = ['P_100m']
##assigning a variable 'yt' as my finding variable for thomas tokyo 100m
yt = dff[column11]
print(yt)
##assigning a variable 'xtt' to my training dataset
xtt = ave_merge
##selecting training features from remaining atheletes for each event.
column1 = ['TOK_100m']
column2 = ['PB_100m']
##assigning a variable 'zz' is having only personal best of thomas using as my indeper
zz = dff[column12]
print(zz)
##assigning 'xx' and 'yy' variables to training data
xx = ave_merge[column1]
yy = ave_merge[column2]
##scaling my training data using minmaxscaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
xx = scaler.fit_transform(xx)
yy = scaler.fit_transform(yy)
##fit training data 'xx' and 'yy' into my linearregression model
from sklearn.linear_model import LinearRegression
LRR = LinearRegression()
LRR.fit(xx,yy)
##print weights and functions
print('w0 = ',LRR.intercept_)
print('coefficients = ',LRR.coef_)
## now using only personal best feature of thomas which is nothing but my independent
y1_p = LRR.predict(zz)
print('thomas in tokyo Tok_100m',y1_p)


##the process repeats the same for each and every event of thomas nine events
```

```
   TOK_100m
0       NaN
   P_100m
0   11.04
w0 =  [0.06921805]
coefficients =  [[0.94490791]]
thomas in tokyo Tok_100m [[10.50100139]]
```

In [7]:

```
column21 = ['TOK_LJ']
column22 = ['P_LJ']
yt1 = dff[column21]
print(yt1)
xtt1 = ave_merge
column3 = ['TOK_LJ']
column4 = ['PB_LJ']
zz1 = dff[column22]
print(zz1)
xx1 = ave_merge[column3]
yy1 = ave_merge[column4]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
xx1 = scaler.fit_transform(xx1)
yy1 = scaler.fit_transform(yy1)
from sklearn.linear_model import LinearRegression
LRR1 = LinearRegression()
LRR1.fit(xx1,yy1)
print('w1 = ',LRR1.intercept_)
print('coefficients = ',LRR1.coef_)
y2_p = LRR.predict(zz1)
print('thomas in tokyo Tok_LJ',y2_p)
```

```
   TOK_LJ
0     NaN
   P_LJ
0   7.9
w1 =  [0.23349325]
coefficients =  [[0.74191631]]
thomas in tokyo Tok_LJ [[7.53399055]]
```

In [8]:

```
column31 = ['TOK_SP']
column32 = ['P_SP']
yt2 = dff[column31]
print(yt2)
xtt2 = ave_merge
column5 = ['TOK_SP']
column6 = ['PB_SP']
zz2 = dff[column32]
print(zz2)
xx2 = ave_merge[column5]
yy2 = ave_merge[column6]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
xx2 = scaler.fit_transform(xx2)
yy2 = scaler.fit_transform(yy2)
from sklearn.linear_model import LinearRegression
LRR2 = LinearRegression()
LRR2.fit(xx2,yy2)
print('w2 = ',LRR2.intercept_)
print('coefficients = ',LRR2.coef_)
y3_p = LRR.predict(zz2)
print('thomas in tokyo Tok_SP',y3_p)
```

```
   TOK_SP
0     NaN
    P_SP
0  14.28
w2 =  [0.14683437]
coefficients =  [[0.53050969]]
thomas in tokyo Tok_SP [[13.56250302]]
```

In [9]:

```
column41 = ['TOK_HJ']
column42 = ['P_HJ']
yt3 = dff[column41]
print(yt3)
xtt3 = ave_merge
column7 = ['TOK_HJ']
column8 = ['PB_HJ']
zz3 = dff[column42]
print(zz3)
xx3 = ave_merge[column7]
yy3 = ave_merge[column8]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
xx3 = scaler.fit_transform(xx3)
yy3 = scaler.fit_transform(yy3)
from sklearn.linear_model import LinearRegression
LRR3 = LinearRegression()
LRR3.fit(xx3,yy3)
print('w3 = ',LRR3.intercept_)
print('coefficients = ',LRR3.coef_)
y4_p = LRR.predict(zz3)
print('thomas in tokyo Tok_HJ',y4_p)
```

```
   TOK_HJ
0     NaN
   P_HJ
0  2.17
w3 =  [0.25020756]
coefficients =  [[0.58369655]]
thomas in tokyo Tok_HJ [[2.11966822]]
```

In [10]:

```
column51 = ['TOK_400m']
column52 = ['P_400m']
yt4 = dff[column51]
print(yt4)
xtt4 = ave_merge
column9 = ['TOK_400m']
column10 = ['PB_400m']
zz4 = dff[column52]
print(zz4)
xx4 = ave_merge[column9]
yy4 = ave_merge[column10]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
xx4 = scaler.fit_transform(xx4)
yy4 = scaler.fit_transform(yy4)
from sklearn.linear_model import LinearRegression
LRR4 = LinearRegression()
LRR4.fit(xx4,yy4)
print('w4 = ',LRR4.intercept_)
print('coefficients = ',LRR4.coef_)
y5_p = LRR.predict(zz4)
print('thomas in tokyo Tok_400m',y5_p)
```

```
   TOK_400m
0       NaN
   P_400m
0   48.64
w4 =  [0.16271258]
coefficients =  [[0.66913139]]
thomas in tokyo Tok_400m [[46.02953885]]
```

In [11]:

```
column61 = ['TOK_110m_H']
column62 = ['P_110m_H']
yt5 = dff[column61]
print(yt5)
xtt5 = ave_merge
column68 = ['TOK_110m_H']
column78 = ['PB_110m_H']
zz5 = dff[column62]
print(zz5)
xx5 = ave_merge[column68]
yy5 = ave_merge[column78]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
xx5 = scaler.fit_transform(xx5)
yy5 = scaler.fit_transform(yy5)
from sklearn.linear_model import LinearRegression
LRR5 = LinearRegression()
LRR5.fit(xx5,yy5)
print('w5 = ',LRR5.intercept_)
print('coefficients = ',LRR5.coef_)
y5_p = LRR.predict(zz5)
print('thomas in tokyo Tok_110m_H',y5_p)
```

```
   TOK_110m_H
0         NaN
   P_110m_H
0     14.36
w5 =  [0.03406825]
coefficients =  [[0.83037872]]
thomas in tokyo Tok_110m_H [[13.63809566]]
```

In [12]:

```
column71 = ['TOK_DT']
column72 = ['P_DT']
yt6 = dff[column71]
print(yt6)
xtt6 = ave_merge
column67 = ['TOK_DT']
column77 = ['PB_DT']
zz6 = dff[column72]
print(zz6)
xx6 = ave_merge[column67]
yy6 = ave_merge[column77]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
xx6 = scaler.fit_transform(xx6)
yy6 = scaler.fit_transform(yy6)
from sklearn.linear_model import LinearRegression
LRR6 = LinearRegression()
LRR6.fit(xx6,yy6)
print('w6 = ',LRR6.intercept_)
print('coefficients = ',LRR6.coef_)
y6_p = LRR.predict(zz6)
print('thomas in tokyo Tok_DT',y6_p)
```

```
   TOK_DT
0     NaN
    P_DT
0  48.81
w6 =  [0.13330235]
coefficients =  [[0.63431068]]
thomas in tokyo Tok_DT [[46.1901732]]
```

In [13]:

```
column81 = ['TOK_PV']
column82 = ['P_PV']
yt7 = dff[column81]
print(yt7)
xtt7 = ave_merge
column66 = ['TOK_PV']
column76 = ['PB_PV']
zz7 = dff[column82]
print(zz7)
xx7 = ave_merge[column66]
yy7 = ave_merge[column76]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
xx7 = scaler.fit_transform(xx7)
yy7 = scaler.fit_transform(yy7)
from sklearn.linear_model import LinearRegression
LRR7 = LinearRegression()
LRR7.fit(xx7,yy7)
print('w7 = ',LRR7.intercept_)
print('coefficients = ',LRR7.coef_)
y7_p = LRR.predict(zz7)
print('thomas in tokyo Tok_PV',y7_p)
```

```
   TOK_PV
0     NaN
   P_PV
0  5.45
w7 =  [0.26549708]
coefficients =  [[0.68421053]]
thomas in tokyo Tok_PV [[5.21896617]]
```

In [14]:

```
column91 = ['TOK_JT']
column92 = ['P_JT']
yt8 = dff[column91]
print(yt8)
xtt8 = ave_merge
column65 = ['TOK_JT']
column75 = ['PB_JT']
zz8 = dff[column92]
print(zz8)
xx8 = ave_merge[column65]
yy8 = ave_merge[column75]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
xx8 = scaler.fit_transform(xx8)
yy8 = scaler.fit_transform(yy8)
from sklearn.linear_model import LinearRegression
LRR8 = LinearRegression()
LRR8.fit(xx8,yy8)
print('w8 = ',LRR8.intercept_)
print('coefficients = ',LRR8.coef_)
y8_p = LRR.predict(zz8)
print('thomas in tokyo Tok_PV',y8_p)
```

```
   TOK_JT
0     NaN
    P_JT
0  65.31
w8 =  [0.09190466]
coefficients =  [[0.7879057]]
thomas in tokyo Tok_PV [[61.78115373]]
```

In [15]:

```
column101 = ['TOK_1500m']
column102 = ['P_1500m']
yt9 = dff[column101]
print(yt9)
xtt9 = ave_merge
column64 = ['TOK_1500m']
column74 = ['PB_1500m']
zz9 = dff[column102]
print(zz9)
xx9 = ave_merge[column64]
yy9 = ave_merge[column74]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
xx9 = scaler.fit_transform(xx9)
yy9 = scaler.fit_transform(yy9)
from sklearn.linear_model import LinearRegression
LRR9 = LinearRegression()
LRR9.fit(xx9,yy9)
print('w9 = ',LRR9.intercept_)
print('coefficients = ',LRR9.coef_)
y9_p = LRR.predict(zz9)
print('thomas in tokyo Tok_1500m',y9_p)
```

```
   TOK_1500m
0        NaN
   P_1500m
0    432.5
w9 =  [0.31142776]
coefficients =  [[0.36638188]]
thomas in tokyo Tok_1500m [[408.74188966]]
```

Question 2

Total marks for question: 10

The optimal coefficients for a least squares linear regression model $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ can be derived analytically, by differentiating the mean squares cost function with respect to the weight and setting the derivative equal to zero [3].

Doing this, the vector $\hat{\mathbf{w}}$ of weights that minimizes the mean squares cost function is given by the *normal equation*:

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{1}$$

where $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} & \dots & x_n^{(N)} \end{bmatrix}$

where $x_i^{(j)}$ is the i-th feature of example j.

Choose one of your multiple linear regression models from Q1 (c), and print out the value of its weights.

Then set up the appropriate matrices $\mathbf{X}$ and $\mathbf{y}$, and write your own code to calculate the optimal weight vector $\hat{\mathbf{w}}$ using the normal equation. Print the value of $\hat{\mathbf{w}}$ to screen, and compare to the weights for the corresponding sklearn model. **[10 marks]**

[3] pp. 9-14, 22-27 of *A First Course in Machine Learning*, 2nd Edition.

In [26]:

```
## Enter code for Q2 here
##in this scenario i am defining my 'a1','a2' are one of the array values of 19 athele
a1 = np.array([[10.12],[8.24],[14.8],[2.02],[47.48],[13.46],[48.67],[4.9],
        [63.44],[271.1]])
a2 = np.array([[10.12],[8.28],[15.34],[2.09],[46.54],[13.27],[50.26],[4.9],
        [64.67],[264.7]])
##assigning 'a4' variables as thomas personal best and 'b4' as thomas tokyo perfomance
a4 = np.array([[11.04],[7.9],[14.28],[2.17],[48.64],[14.36],[48.81],[5.45],[65.31],[4
b4 = np.array([[10.50100139],[7.53399055],[13.56250302],[2.11966822],[46.02953885],[1
```

In [30]:

```
##deriving normalized equation in python code and fitting 'a1' values
##solving the transpose and inverse'a1' is one of the array row vector of training da
x_transpose = np.transpose(a1)
x_transpose_dot_x = x_transpose.dot(a1)
varian1 = np.linalg.inv(x_transpose_dot_x)
varian1
print('varian1',varian1)
##varian1 is the first part of the equation

##varian 2 is the sceond part of the equation
## x transpose dot a2
varian2 = x_transpose.dot(a2)
varian2
print('varian2',varian2)

##solving all the equation to get intercept and coefficients

theta0 = varian1.dot(varian2)
print(theta0)

##now assigning variable 'a4' values which consists only thomas personal best values


predvalues = np.dot(a4,theta0)
predvalues
print('w',predvalues)
```

```
varian1 [[1.20858224e-05]]
varian2 [[81123.2278]]
[[0.98044092]]
w [[ 10.82406776]
 [  7.74548327]
 [ 14.00069634]
 [  2.1275568 ]
 [ 47.68864636]
 [ 14.07913162]
 [ 47.85532132]
 [  5.34340302]
 [ 64.0325965 ]
 [424.04069801]]
```

In [31]:

```
from sklearn.metrics import r2_score
#acheived 99 percent accuracy when compared with  corresponding sklearn model
r2 = r2_score(b4,predvalues)
print('r2',r2*100)
```

```
r2 99.8221214012668
```

Question 3

Total marks for question: 10

In this question, you will use the `random_search` algorithm introduced in the lectures. The code is already provided in the cell below.

(a) Write a function called `himmelblau` that implements the *Himmelblau function*:

$$g(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{2}$$

This function should take in a numpy column vector representing $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and return the value of $g(x_1, x_2)$. **[2 marks]**

(b) Amend the `random_search` function to implement another option for `alpha_choice` called `decay`. In this setting, alpha should be set to $\alpha = e^{-k/10}$, where $e = 2.718281828\ldots$ is the natural base. **[2 marks]**

(c) Perform an experiment to investigate the effect of `alpha_choice` when optimizing the Himmelblau function with the following settings:

- Number of steps $K = 50$
- Number of random directions at each step $P = 1000$
- Initial point $\mathbf{w}^0 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$

You should investigate three settings for α: fixed steplength of $\alpha = 1$, 'diminishing' and 'decay'.

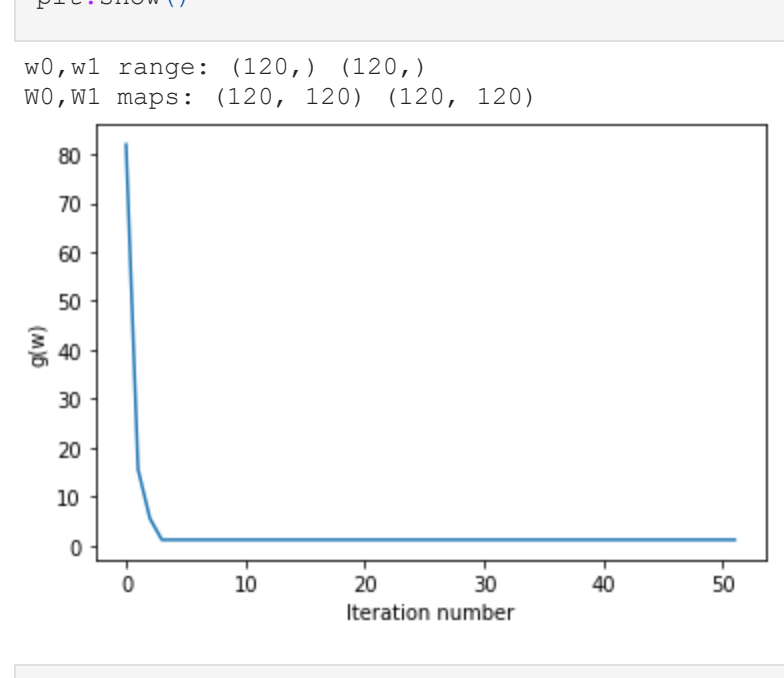
Show the results from your experiments by plotting (on the same figure) the cost history (i.e. history of objective function values) against iteration number for the three settings of the random search. Any differences in the performance of the three settings should be clear from the figure. **[4 marks]**

(d) Comment on the worst performing setting of the random search in **(c)**, giving an explanation of why it does not perform as well as the other two settings. **[2 marks]**

In [21]:

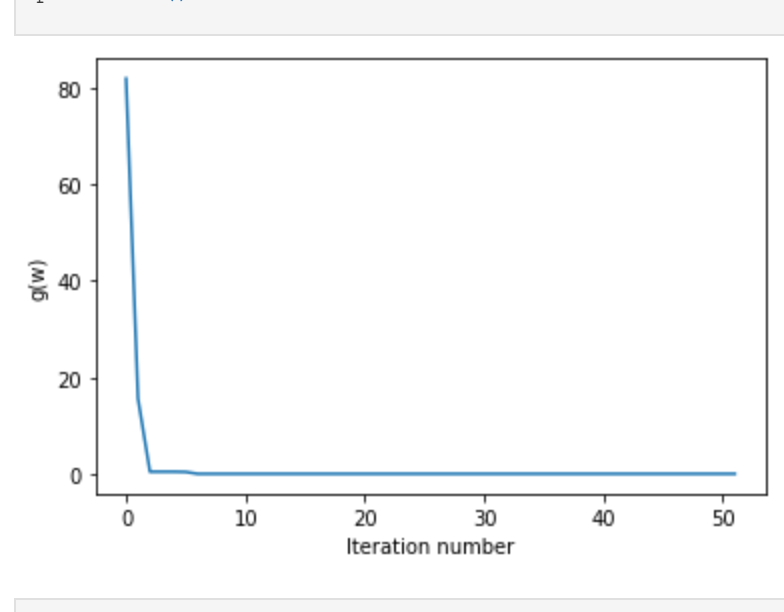
```
## Enter code for part (a) below [2 marks]
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def g(w):
    return (w0[0]**2+w1[1]-11)**2 + (w0[0]+w1[1]**2-7)**2

w0=np.arange(-6,6,0.1)
w1=np.arange(-6,6,0.1)
print("w0,w1 range:",w0.shape,w1.shape)
W0,W1=np.meshgrid(w0,w1)
print("W0,W1 maps:",W0.shape,W1.shape)
g=g([W0,W1])



# random search function - ammend for part (b) [2 marks]
def random_search(g,alpha_choice,max_its,w,num_samples):
    # run random search
    weight_history = []         # container for weight history
    cost_history = []           # container for corresponding cost function history
    alpha = 0
    for k in range(1,max_its+1):
        # check if diminishing steplength rule used
        if alpha_choice == 'diminishing':
            alpha = 1/float(k)
        else:
            alpha = alpha_choice

        # record weights and cost evaluation
        weight_history.append(w.T)
        cost_history.append(g(w.T))

        # construct set of random unit directions
        directions = np.random.randn(num_samples,np.size(w))
        norms = np.sqrt(np.sum(directions*directions,axis = 1))[:,np.newaxis]
        directions = directions/norms

        ### pick best descent direction
        # compute all new candidate points
        w_candidates = w + alpha*directions

        # evaluate all candidates
        evals = np.array([g(w_val.T) for w_val in w_candidates])

        # if we find a real descent direction take the step in its direction
        ind = np.argmin(evals)
        if g(w_candidates[ind]) < g(w.T):
            # pluck out best descent direction
            d = directions[ind,:]

            # take step
            w = w + alpha*d

    # record weights and cost evaluation
    weight_history.append(w.T)
    cost_history.append(g(w.T))
    return weight_history,cost_history

w_init = np.array([[-2,-2]])
def testfunct(w):
 val = (w[0]**2 + w[1] - 11)**2 + (w[0] + w[1]**2 - 7)**2
 return val
w_init = np.array([[2,-2]])
weights,cost = random_search(g=testfunct,alpha_choice=1,max_its=50,w = w_init,num_samp
xiter = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
plt.plot(xiter, cost)
plt.xlabel('Iteration number')
plt.ylabel('g(w)')
plt.show()
```

```
w0,w1 range: (120,) (120,)
W0,W1 maps: (120, 120) (120, 120)
```



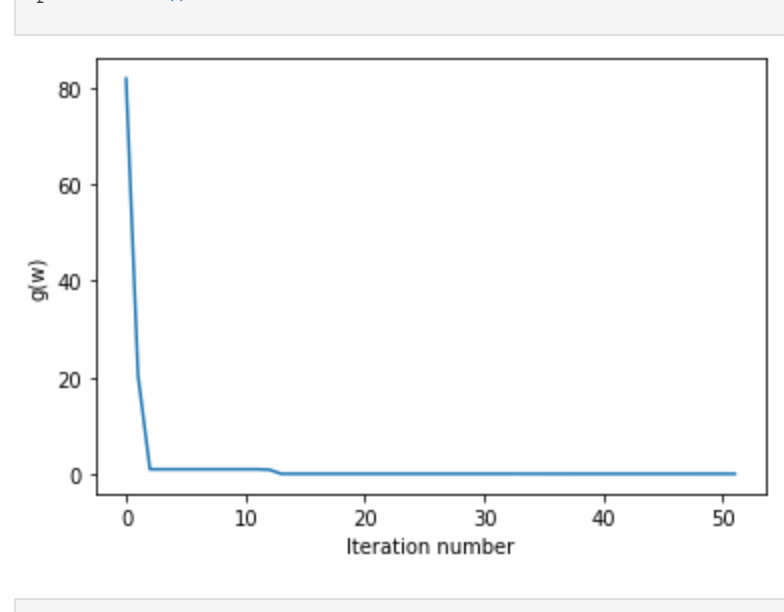
In [22]:

```
# Enter code for part (c) below [4 marks]
def testfunct(w):
 val = (w[0]**2 + w[1] - 11)**2 + (w[0] + w[1]**2 - 7)**2
 return val
w_init = np.array([[2,-2]])
weights,cost = random_search(g=testfunct,alpha_choice='diminishing',max_its=50,w = w_i
xiter = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
plt.plot(xiter, cost)
plt.xlabel('Iteration number')
plt.ylabel('g(w)')
plt.show()
```



In [24]:

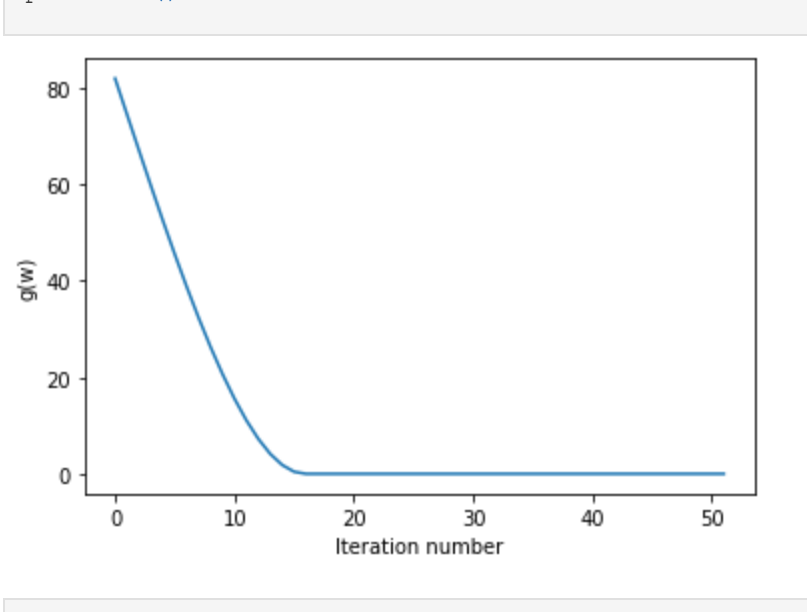
```
def random_search(g,alpha_choice,max_its,w,num_samples):
 # run random search
    weight_history = [] # container for weight history
    cost_history = [] # container for corresponding cost function history
    alpha = 0
    for k in range(1,max_its+1):
 # check if diminishing steplength rule used
        if alpha_choice == 'decay':
            alpha = np.exp(1)**(-k/10)
        else:
            alpha = alpha_choice

 # record weights and cost evaluation
        weight_history.append(w.T)
        cost_history.append(g(w.T))

 # construct set of random unit directions
        directions = np.random.randn(num_samples,np.size(w))
        norms = np.sqrt(np.sum(directions*directions,axis = 1))[:,np.newaxis]
        directions = directions/norms

 ### pick best descent direction
 # compute all new candidate points
        w_candidates = w + alpha*directions

 # evaluate all candidates
        evals = np.array([g(w_val.T) for w_val in w_candidates])
 # if we find a real descent direction take the step in its direction
        ind = np.argmin(evals)
        if g(w_candidates[ind]) < g(w.T):
 # pluck out best descent direction
            d = directions[ind,:]

 # take step
            w = w + alpha*d

 # record weights and cost evaluation
    weight_history.append(w.T)
    cost_history.append(g(w.T))
    return weight_history,cost_history
def testfunct(w):
 val = (w[0]**2 + w[1] - 11)**2 + (w[0] + w[1]**2 - 7)**2
 return val
w_init = np.array([[2,-2]])
weights,cost = random_search(g=testfunct,alpha_choice='decay',max_its=50,w = w_init,nu
xiter = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
plt.plot(xiter, cost)
plt.xlabel('Iteration number')
plt.ylabel('g(w)')
plt.show()
```



In [32]:

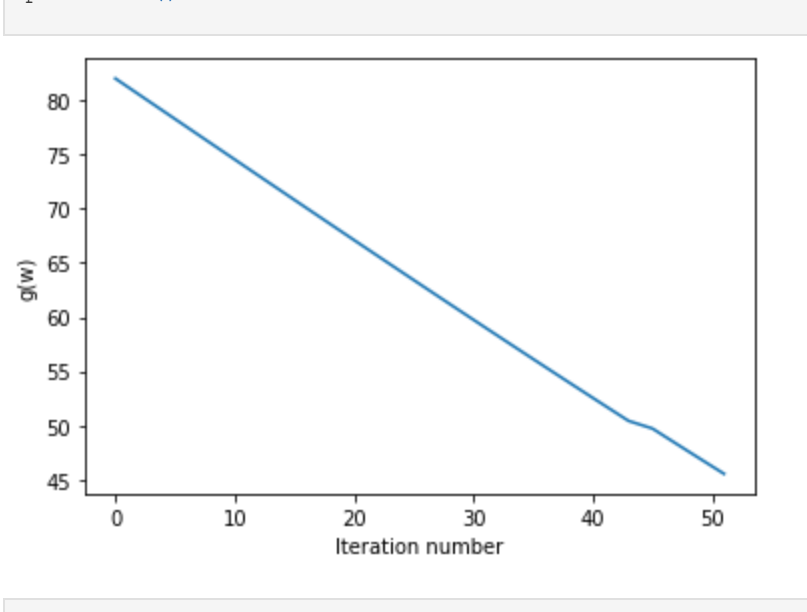
```
def random_search(g,alpha_choice,max_its,w,num_samples):
    # run random search
    weight_history = []         # container for weight history
    cost_history = []           # container for corresponding cost function history
    alpha = 0
    for k in range(1,max_its+1):
        # check if diminishing steplength rule used
        if alpha_choice == 'diminishing':
            alpha = 1/float(k)
        else:
            alpha = alpha_choice

        # record weights and cost evaluation
        weight_history.append(w.T)
        cost_history.append(g(w.T))

        # construct set of random unit directions
        directions = np.random.randn(num_samples,np.size(w))
        norms = np.sqrt(np.sum(directions*directions,axis = 1))[:,np.newaxis]
        directions = directions/norms

        ### pick best descent direction
        # compute all new candidate points
        w_candidates = w + alpha*directions

        # evaluate all candidates
        evals = np.array([g(w_val.T) for w_val in w_candidates])

        # if we find a real descent direction take the step in its direction
```

```
            ind = np.argmin(evals)
            if g(w_candidates[ind]) < g(w.T):
                # pluck out best descent direction
                d = directions[ind,:]

                # take step
                w = w + alpha*d

        # record weights and cost evaluation
        weight_history.append(w.T)
        cost_history.append(g(w.T))
    return weight_history,cost_history

w_init = np.array([[-2,-2]])
def testfunct(w):
 val = (w[0]**2 + w[1] - 11)**2 + (w[0] + w[1]**2 - 7)**2
 return val
w_init = np.array([[2,-2]])
weights,cost = random_search(g=testfunct,alpha_choice=0.1,max_its=50,w = w_init,num_sa
xiter = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
plt.plot(xiter, cost)
plt.xlabel('Iteration number')
plt.ylabel('g(w)')
plt.show()
```



In [33]:

```
def testfunct(w):
 val = (w[0]**2 + w[1] - 11)**2 + (w[0] + w[1]**2 - 7)**2
 return val
w_init = np.array([[2,-2]])
weights,cost = random_search(g=testfunct,alpha_choice=0.01,max_its=50,w = w_init,num_s
xiter = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
plt.plot(xiter, cost)
plt.xlabel('Iteration number')
plt.ylabel('g(w)')
plt.show()
```



In [ ]:

```
#Enter comment below for part (d); or alternatievly type your answer into a new markd
1) when compared all three model settings of alpha choice=1,'diminishing','decay' the
with unsteadily for few steps and started declining steadily towards global minimum.
2)when alpha choice is 'diminishing' cost funtion descending steadily when compared w
3)when alphachoice is '1' cost function started descending after a step further when
4)additionally, when i performed cutting the aplha choice = '0.1' started descending
5)alphachoice = '0.001'is far more steps than alpha choice ='0.001'
```