Lab_6_Regression

September 13, 2021

1 LAB 6 : Regression

Regression is generally used for curve fitting task. Here we will demonstrate regression task for the following :

1. Fitting of a Line (One Variable and Two Variables)
2. Fitting of a Plane
3. Fitting of M-dimensional hyperplane
4. Practical Example of Regression task

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

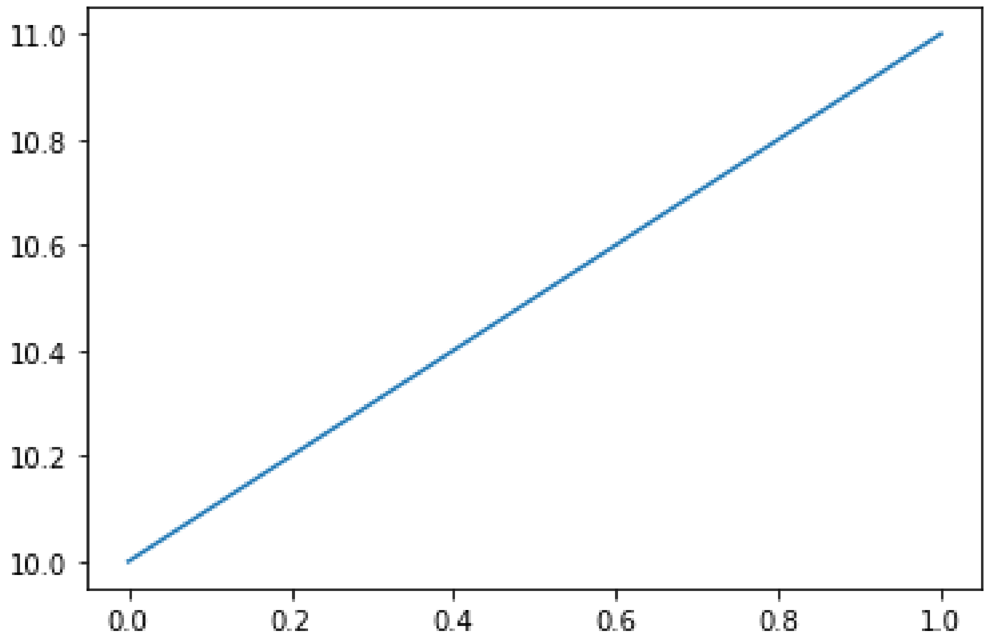
2 Fitting of a Line (One Variable)

Generation of line data ($y = w_1 x + w_0$)

1. Generate x, 1000 points from 0-1
2. Take $w_0 = 10$ and $w_1 = 1$ and generate y
3. Plot (x,y)

[ ]:
```
## Write your code here
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f9a00729350>]
```



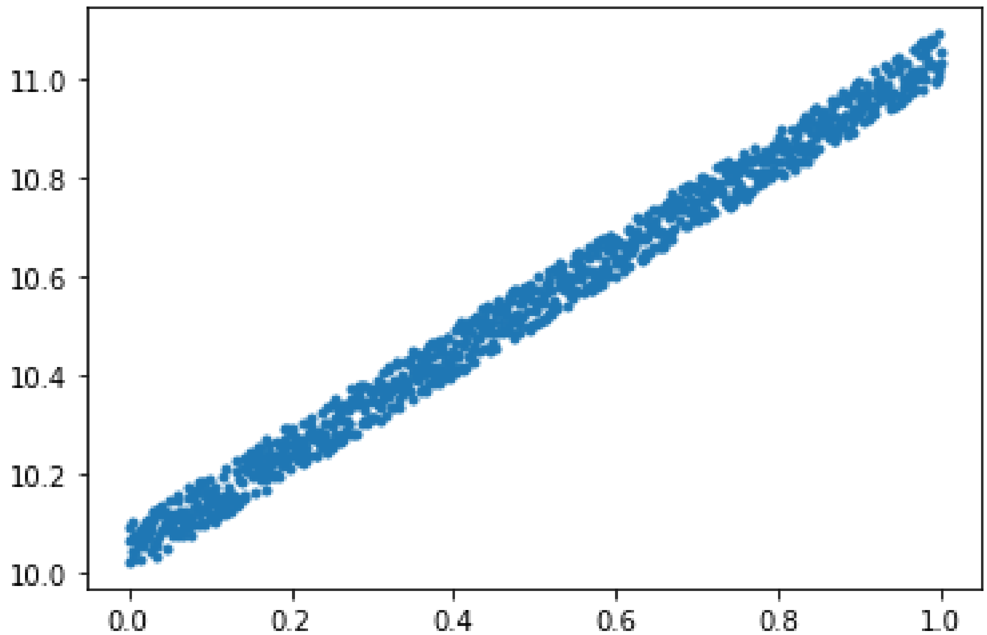
Corruption of data using uniformly sampled random noise

1. Generate random numbers uniformly from (0-1) with same size as y
2. Corrupt y and generate y_{cor} by adding the generated random samples with a weight of 0.1.
3. Plot (x,y_{cor}) (use scatter plot)

[ ]:
```
## Write your code here
```

```
(1000,)
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f9a00218190>]
```



Heuristically predicting the curve (Generating the Error Curve)

1. Keep $w_0 = 10$ as constant and find w_1
2. Create a search space from -5 to 7 for w_1, by generating 1000 numbers between that
3. Find y_{pred} using each value of w_1
4. The y_{pred} that provide least norm error with y, will be decided as best y_{pred}

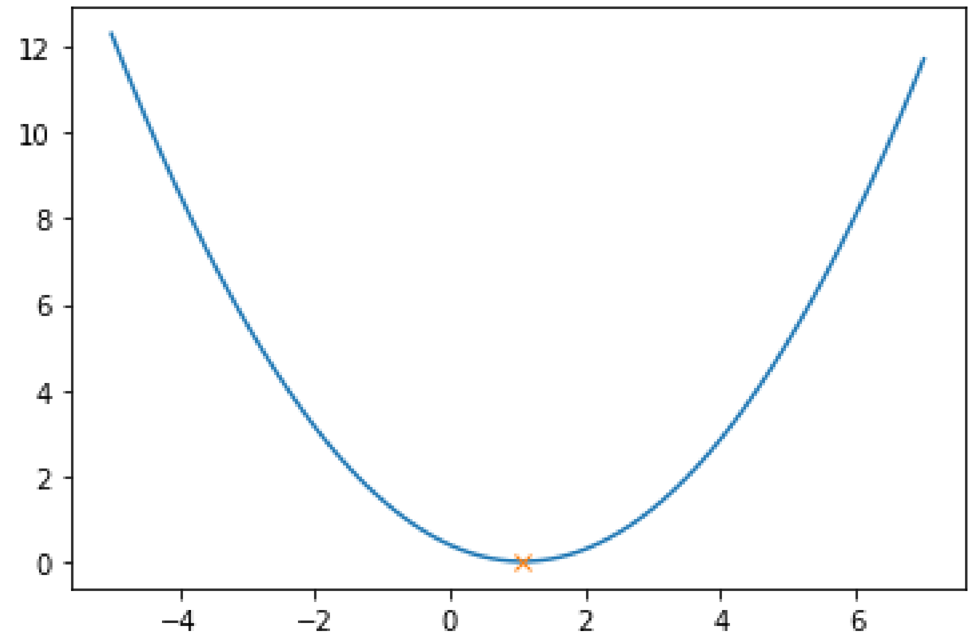
$$error = \frac{1}{m}\sum_{i=1}^{M}(y_i - y_{pred_i})^2$$

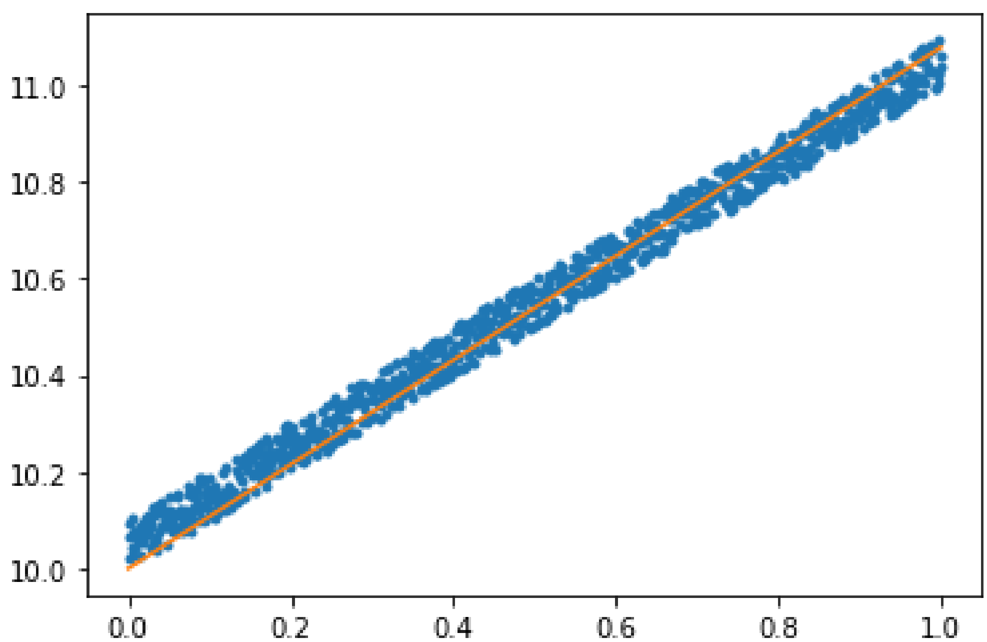
5. Plot error vs search_$w1$
6. First plot the scatter plot (x,y_{cor}) , over that plot $(x,y_{bestpred})$

[ ]:
```
## Write your code here
```

```
Optimal Value of w1 is :  1.0780780780780779
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f99fbf57710>]
```





Using Gradient Descent to predict the curve

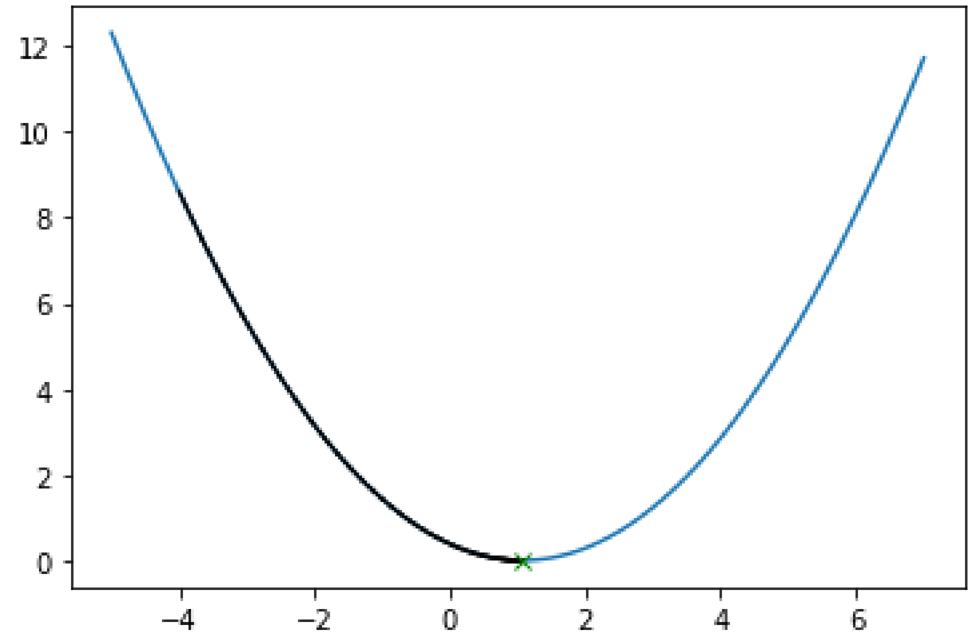
1. $Error = \frac{1}{m}\sum_{i=1}^{M}(y_i - y_{pred_i})^2 = \frac{1}{m}\sum_{i=1}^{M}(y_i - (w_0 + w_1 x_i))^2$
2. $\nabla Error|_{w1} = \frac{-2}{M}\sum_{i=1}^{M}(y_i - y_{pred_i}) \times x_i$

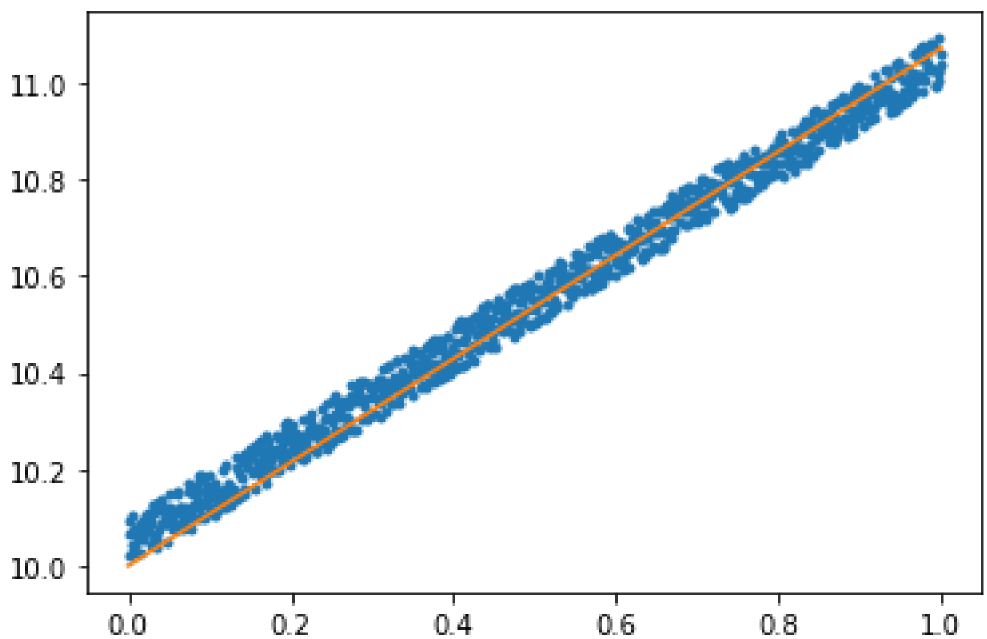
3. $w_1|_{new} = w_1|_{old} - \lambda \nabla Error|_{w1} = w_1|_{old} + \frac{2\lambda}{M} \sum_{i=1}^{M} (y_i - y_{pred_i}) \times x_i$

[ ]:
```
## Write your code here
```

```
Optimal Value of w1 is : 1.072589537989739
```

[ ]: [<matplotlib.lines.Line2D at 0x7f99fbda5d90>]





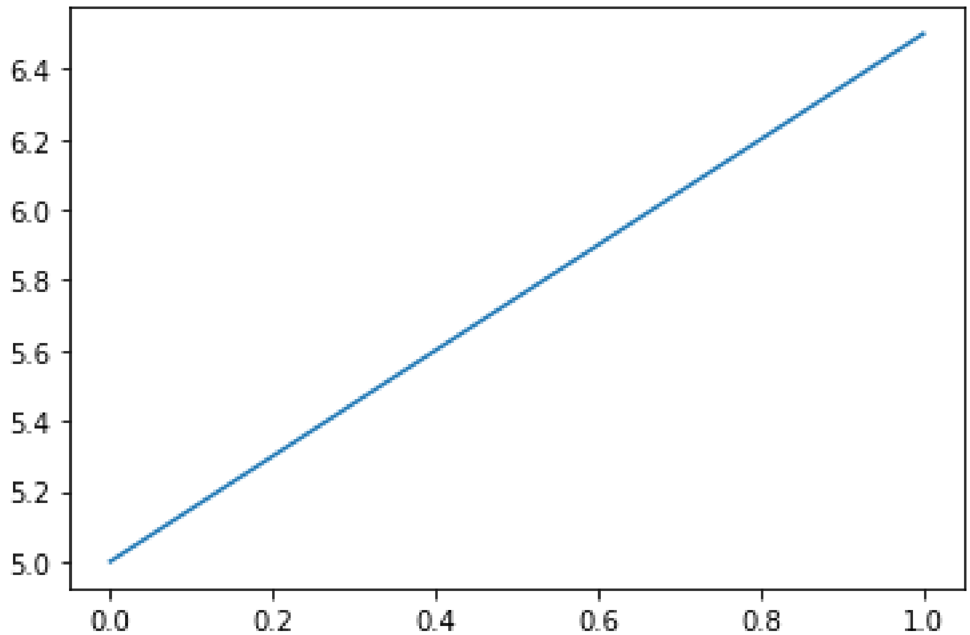
3 Fitting of a Line (Two Variables)

Generation of Line Data ($y = w_1 x + w_0$)

1. Generate x, 1000 points from 0-1
2. Take $w_0 = 5$ and $w_1 = 1.5$ and generate y
3. Plot (x,y)

[2]:
```
## Write your code here
```

[2]: `[<matplotlib.lines.Line2D at 0x7ffb45cc9610>]`

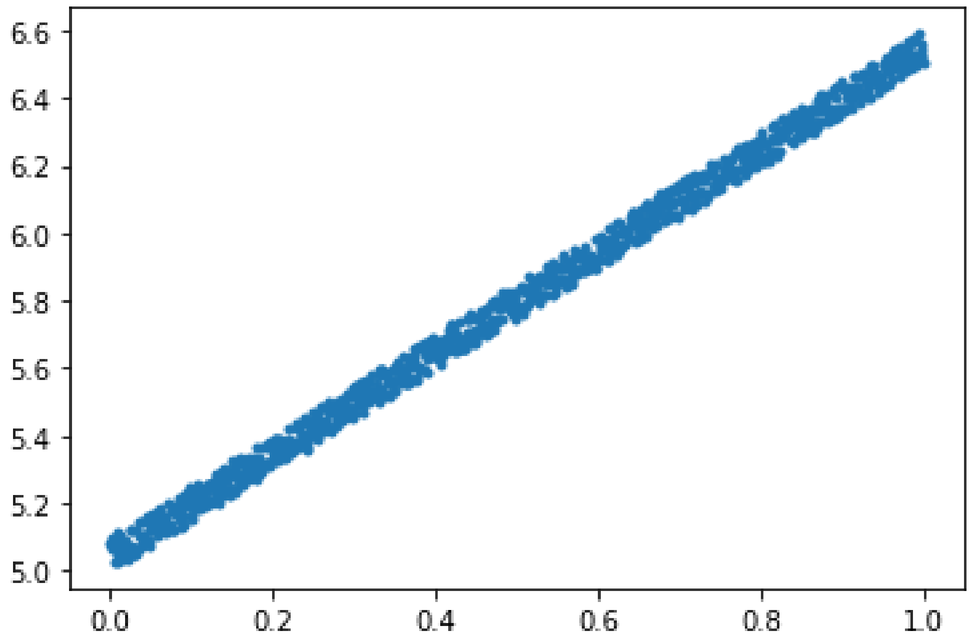


Corrupt the data using uniformly sampled random noise

1. Generate random numbers uniformly from (0-1) with same size as y
2. Corrupt y and generate y_{cor} by adding the generated random samples with a weight of 0.1
3. Plot (x,y_{cor}) (use scatter plot)

[3]:
```
## Write your code here
```

[3]: `[<matplotlib.lines.Line2D at 0x7ffb457b2250>]`



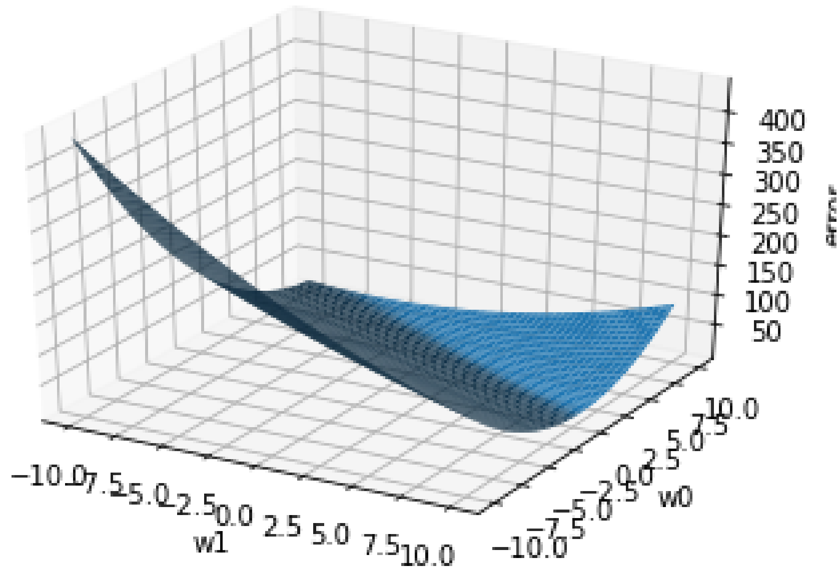
Plot the Error Surface

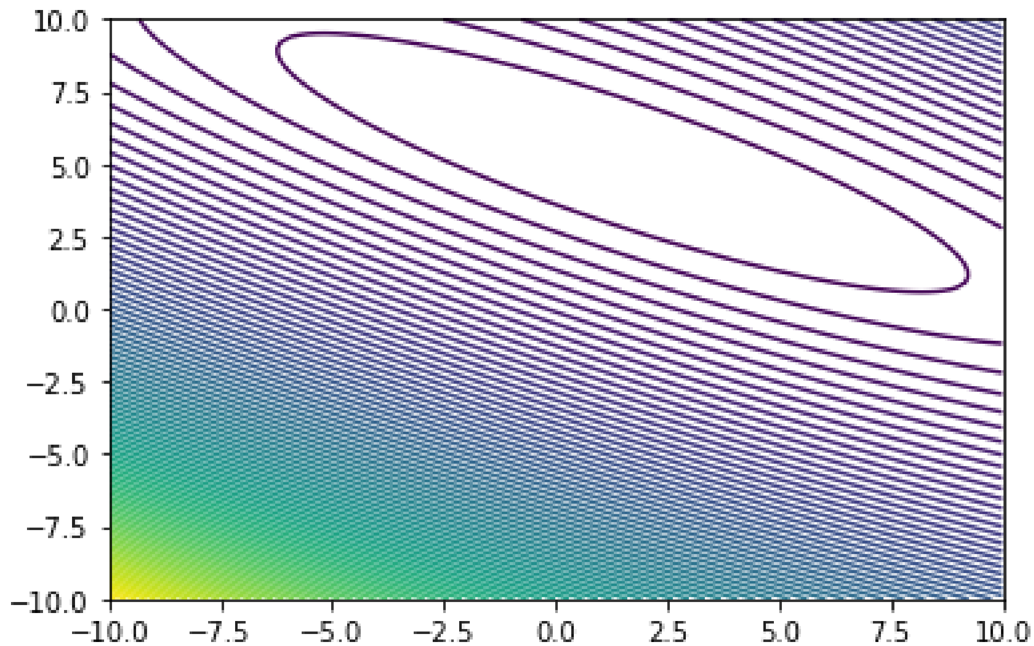
1. we have all the data points available in y_{cor}, now we have to fit a line with it. (i.e from y_{cor} we have to predict the true value of w_1 and w_0)
2. Take w_1 and w_0 from -10 to 10, to get the error surface

[4]:
```
## Write your code here
```

```
(100, 100)
(100, 100)
```

[4]: `<matplotlib.contour.QuadContourSet at 0x7ffb45731b90>`



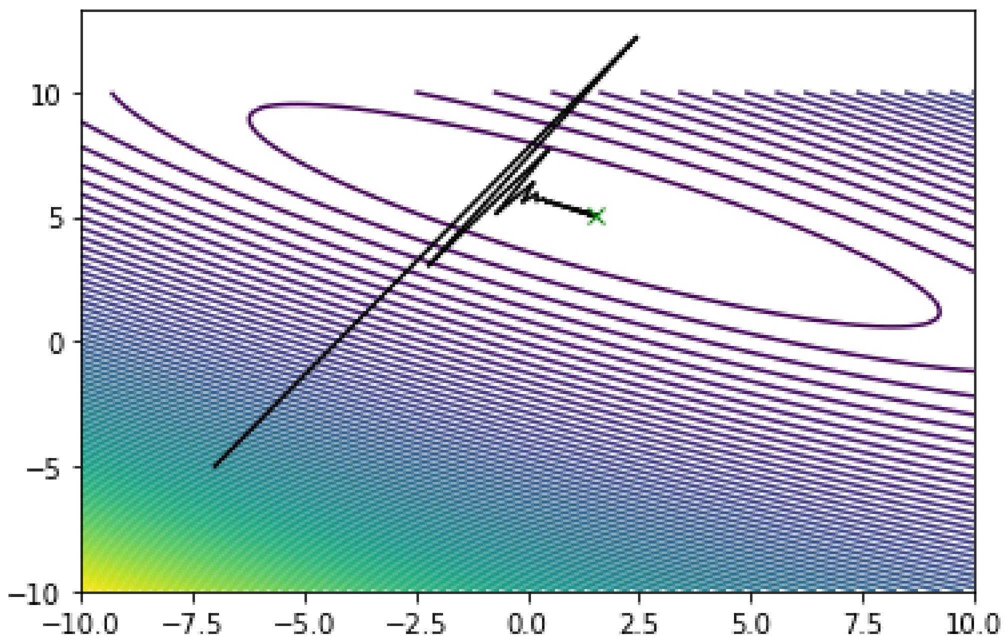


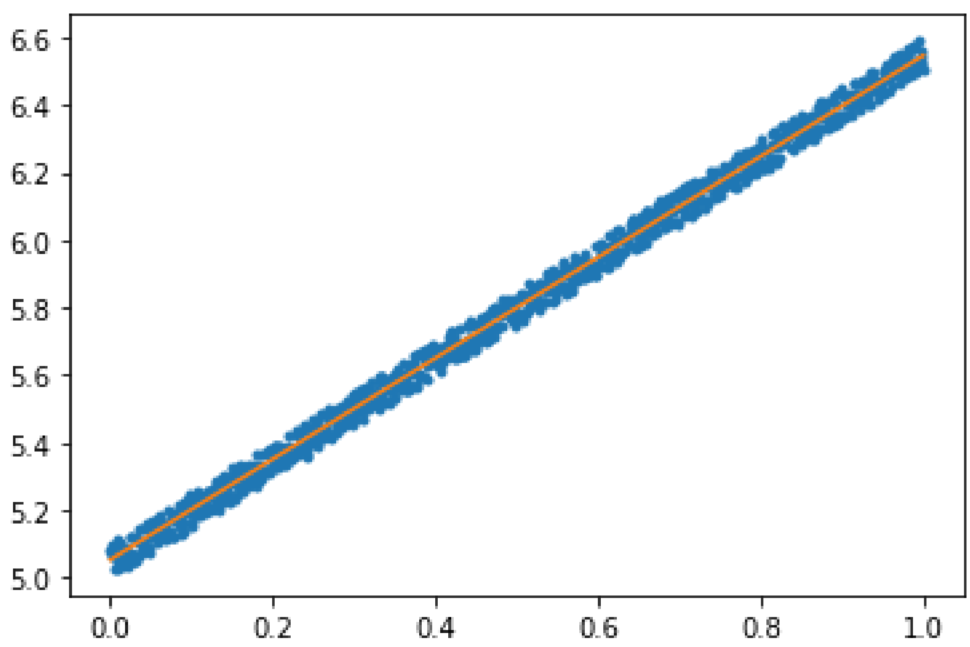
Gradient Descent to find optimal Values

[7]:
```
## Write your code here
```

```
Optimal value of w0 is : 5.053546903100848
Optimal value of w1 is : 1.4931645949404873
```

[7]: [<matplotlib.lines.Line2D at 0x7ffb37514590>]





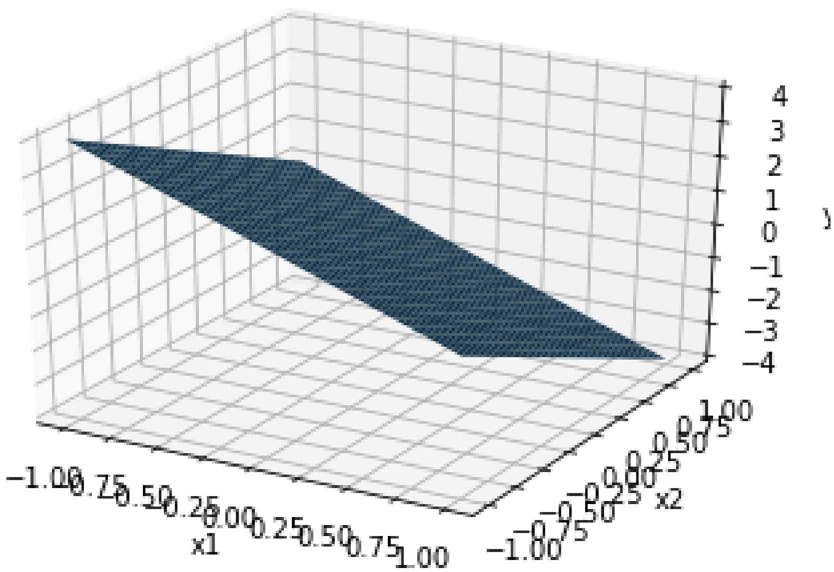
4 Fitting of a Plane

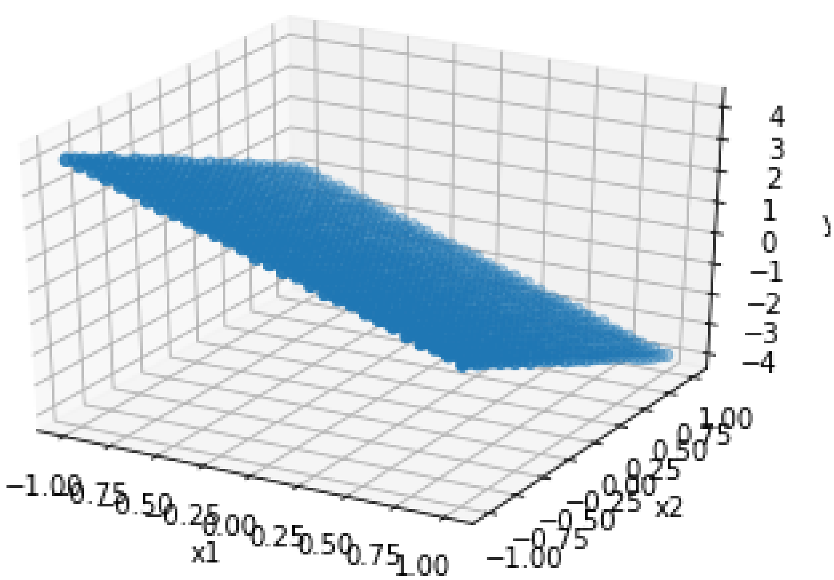
Generation of plane data

1. Generate x_1 and x_2 from range -1 to 1, (30 samples)
2. Equation of plane $y = w_0 + w_1 x_1 + w_2 x_2$
3. Here we will fix w_0 and will learn w_1 and w_2

[ ]:
```
## Write your code here
```

```
(900,)
```





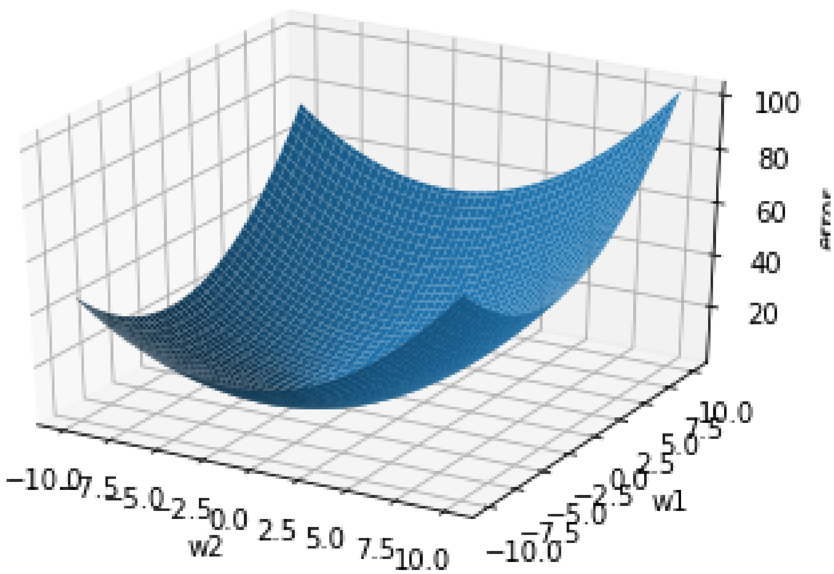
Generate the Error Surface

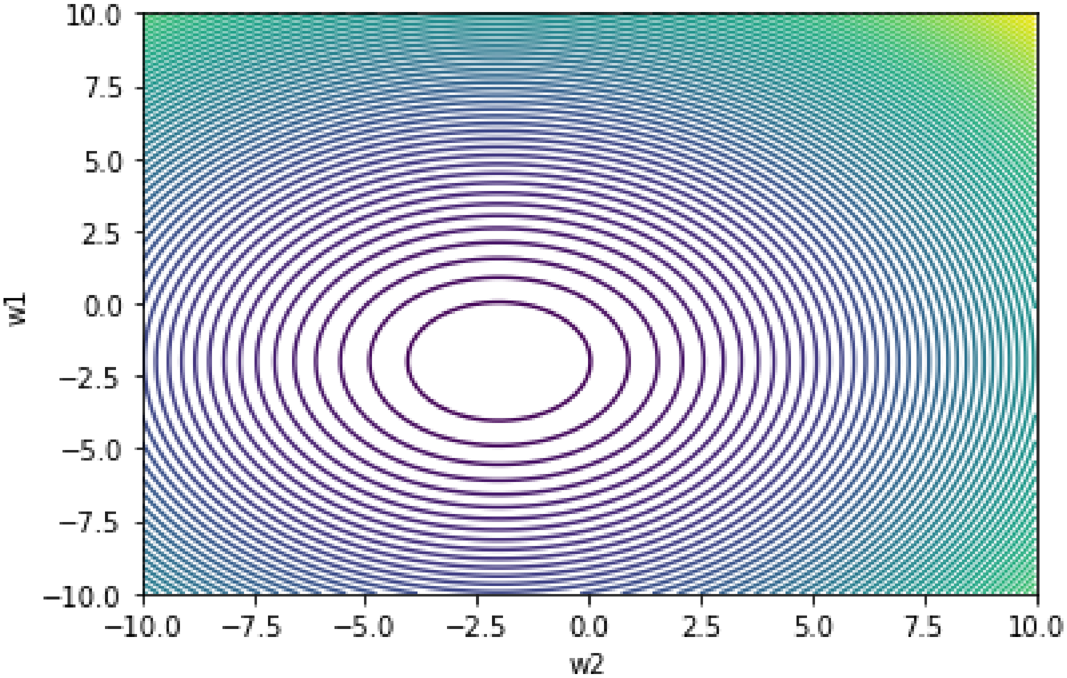
1. Vary w_1 and w_2 and generate the error surface and find their optimal value
2. Also plot the Contour

[ ]:
```
## Write your code here
```

```
(100, 100)
(100, 100)
```

[ ]:
```
Text(0, 0.5, 'w1')
```



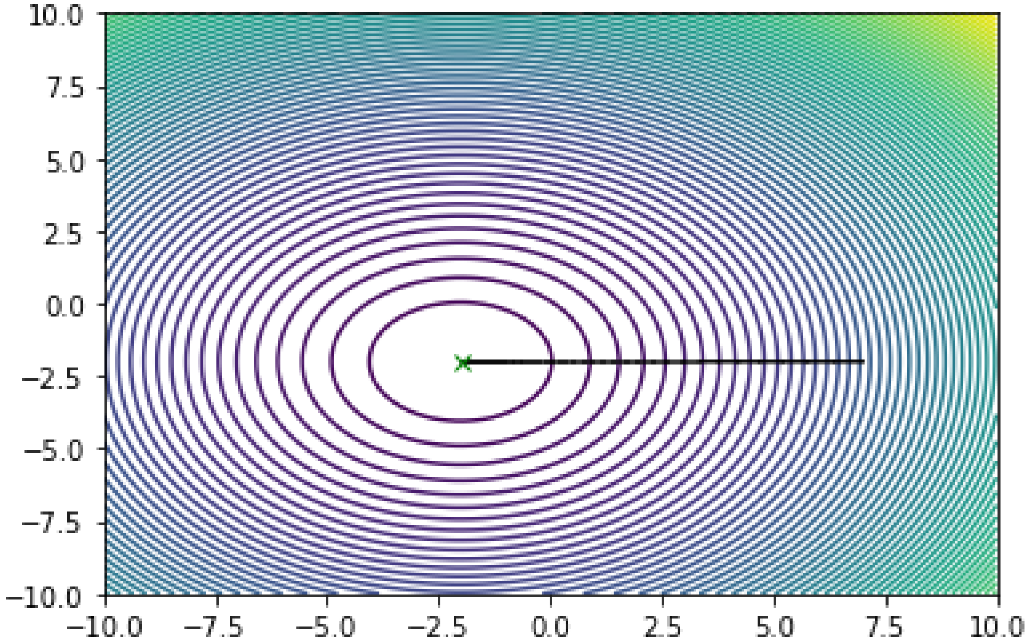


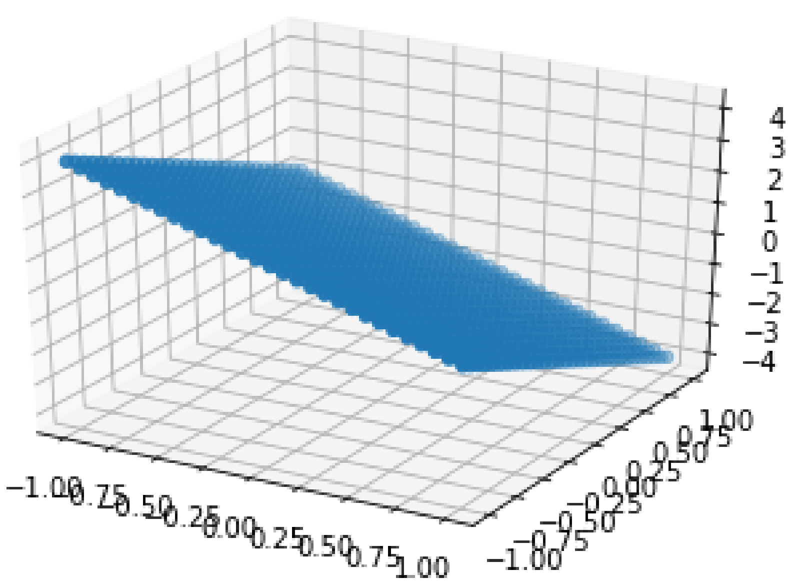
Prediction using Gradient Descent

[ ]:
```
## Write your code here
```

```
Optimal Value of w1 is : -2.000211641046013
Optimal Value of w2 is : -1.9992951536638703
```

[ ]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7f99f22b7610>
```





5 Fitting of M-dimentional hyperplane (M-dimention, both in matrix inversion and gradient descent)

Here we will vectorize the input and will use matrix method to solve the regression problem.

let we have M- dimensional hyperplane we have to fit using regression, the inputs are $x1, x2, x3, ..., x_M$. in vector form we can write $[x1, x2, ..., x_M]^T$, and similarly the weights are $w1, w2, ...w_M$ can be written as a vector $[w1, w2, ...w_M]^T$, Then the equation of the plane can be written as:

$$y = w1x1 + w2x2 + ... + w_M x_M$$

$w1, w2, ..., wM$ are the scalling parameters in M different direction, and we also need a offset parameter w0, to capture the offset variation while fitting.

The final input vector (generally known as augmented feature vector) is represented as $[1, x1, x2, ..., x_M]^T$ and the weight matrix is $[w0, w1, w2, ...w_M]^T$, now the equation of the plane can be written as:

$$y = w0 + w1x1 + w2x2 + ... + w_M x_M$$

In matrix notation: $y = x^T w$ (for a single data point), but in general we are dealing with N-data points, so in matrix notation

$$Y = X^T W$$

where Y is a $N \times 1$ vector, X is a $M \times N$ matrix and W is a $M \times 1$ vector.

$$Error = \frac{1}{N} ||Y - X^T W||^2$$

it looks like a optimization problem, where we have to find W, which will give minimum error.

1. **By computation:**

$\nabla Error = 0$ will give us W_{opt}, then W_{opt} can be written as:

$$W_{opt} = (XX^T)^{-1} XY$$

2. **By gradient descent:**

$$W_{new} = W_{old} + \frac{2\lambda}{N} X(Y - X^T W_{old})$$

1. Create a class named Regression
2. Inside the class, include constructor, and the following functions:

   a. grad_update: Takes input as previous weight, learning rate, x, y and returns the updated weight.

   b. error: Takes input as weight, learning rate, x, y and returns the mean squared error.

   c. mat_inv: This returns the pseudo inverse of train data which is multiplied by labels.

   d. Regression_grad_des: Here, inside the for loop, write a code to update the weights. Also calulate error after each update of weights and store them in a list. Next, calculate the deviation in error with new_weights and old_weights and break the loop, if it's below a threshold value mentioned the code.

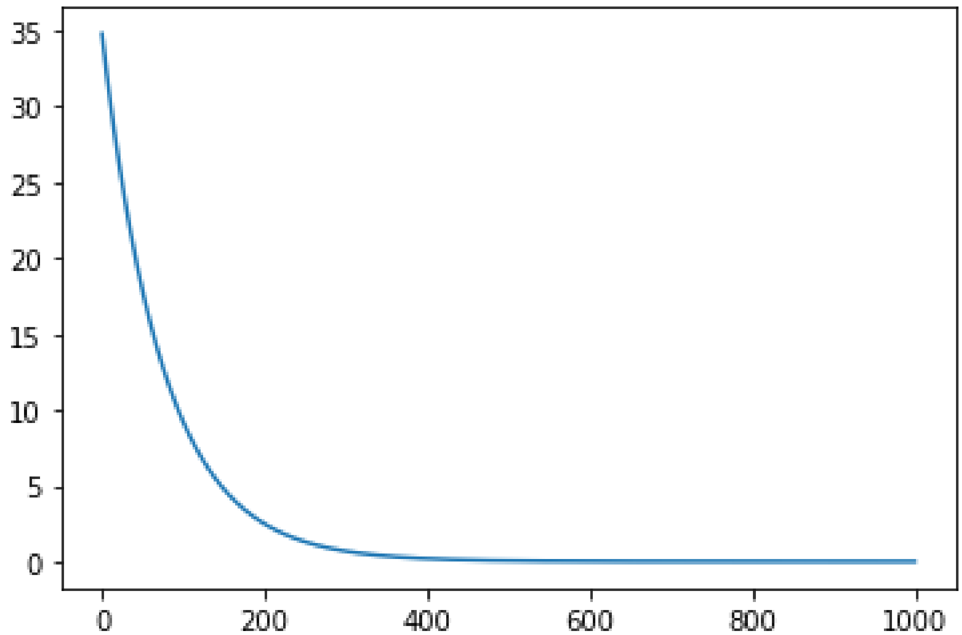
[8]:
```
class regression:
  # Constructor
  def __init__(self, name='reg'):
```

```
    self.name = name  # Create an instance variable

  def grad_update(self,w_old,lr,y,x):
    #write code here
    return w

  def error(self,w,y,x):
    return # write code here

  def mat_inv(self,y,x_aug):
    return # write code here

  # By Gradien descent

  def Regression_grad_des(self,x,y,lr):

    for i in range(1000):
      # write code here

      dev=np.abs(# write code here)
          # print(i)
      if dev<=0.000001:
        break

    return w_pred,err


##############################################################################
# Generation of data

sim_dim=5
sim_no_data=1000
x=np.random.uniform(-1,1,(sim_dim,sim_no_data))
print(x.shape)

w = ## Write your code here (Initialise the weight matrix) (W=[w0,w1,.....
 ↪,wM]')
print(w.shape)

## Augment the Input
x_aug = ## Write your code here (Augment the data so as to include x0 also␣
 ↪which is a vector of ones)
print(x_aug.shape)

y=x_aug.T @ w  # vector multiplication
print(y.shape)
```

```
## Corrupt the input by adding noise
noise=np.random.uniform(0,1,y.shape)
y=y+0.1*noise

### The data (x_aug and y) is generated ###

##############################################################################

# By Computation (Normal Equation)
reg = regression()
w_opt=reg.mat_inv(y,x_aug)
print(w_opt)

# By Gradien descent
lr=0.01
w_pred,err=reg.Regression_grad_des(x_aug,y,lr)
print(w_pred)

plt.plot(err)
```

```
Initial Data shape : (5, 1000)
Dimension of Weight matrix :  (6, 1)
Data shape after augmenting : (6, 1000)
Shape of Output : (1000, 1)
Optimal weights obatained by computation :  [[1.0493477 ]
 [2.00058824]
 [2.99789885]
 [5.00259039]
 [9.00250786]
 [3.00058042]]
Optimal weights obatained by Gradient descent :  [[1.04871051]
 [2.00031284]
 [3.00062405]
 [4.99439216]
 [8.98796357]
 [2.99669071]]
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7ffb37043910>]
```



6 Practical Example (Salary Prediction)

1. Read data from csv file
2. Do train test split (90% and 10%)
3. Compute optimal weight values and predict the salary using the regression class created above (Use both the methods)
4. Find the mean square error in test.
5. Also find the optimal weight values using regression class from the Sci-kit learn library

[11]:
```
## Write your code here
```