Programming Assignment : Regression

Course Advisor: Prof. S.R.M. Prasanna

Course TA's: Jagabandhu Mishra (183081002@iitdh.ac.in) and Seema K. (173021001@iitdh.ac.in)

Regression:

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

1) Fitting of line (one variable learning)

2) Fitting of line (two variable learning)

3) Fitting of a plane (two variable)

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

5) Polynomial regression

6) Pratical example of regression task (salary prediction)

1) Fitting of line

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

i) Generate x, 1000 points from 0-1.

ii) Take $w_0 = 10$ and $w_1 = 1$ and generate y

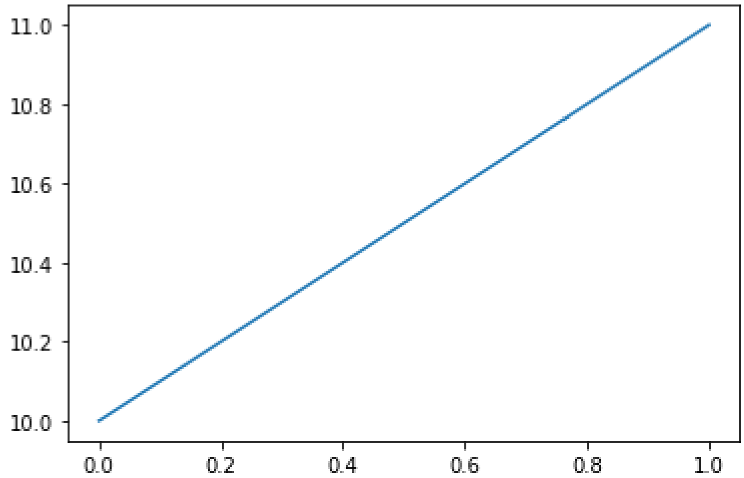
iii) Plot (x,y)

In [ ]:

```
# write your code here
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f177fec77f0>]
```



b) Corrupt the data using uniformly sampled random noise.

i) Generate random numbers uniformly from (0-1) with same size as y.

ii) Corrupt y and generate y_{cor} by adding the generated randomsamples with a weight of 0.1.

iii) Plot (x,y_{cor}) (use scatter plot)

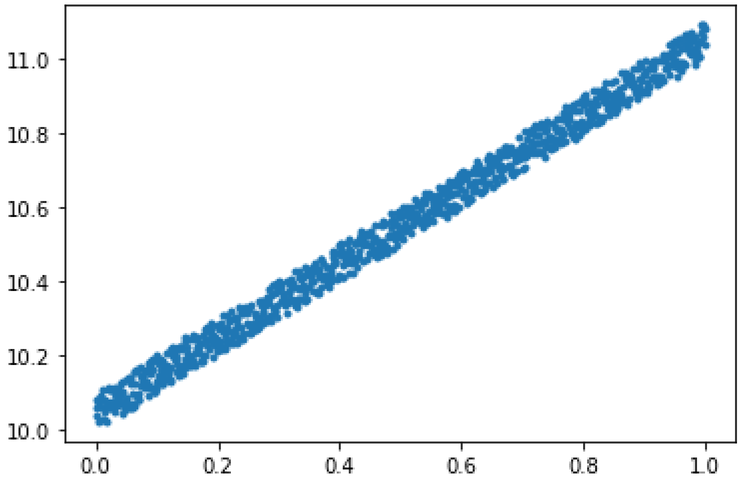
In [ ]:

```
# write your code here
```

```
(1000,)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f177fe5bfd0>]
```



c) Curve prediction using hurestic way.

i) Keep $w_0 = 10$ as constant and find w_1 ?

ii) Create a search space from -5 to 7 for w_1, by generating 1000 numbers between that.

iii) Find y_{pred} using each value of w_1.

iv) The y_{pred} that provide least norm error with y, will be decided as best y_{pred}.

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

v) Plot error vs srch_$w1$

vi) First plot the scatter plot (x,y_{cor}) , over that plot (x,$y_{bestpred}$).

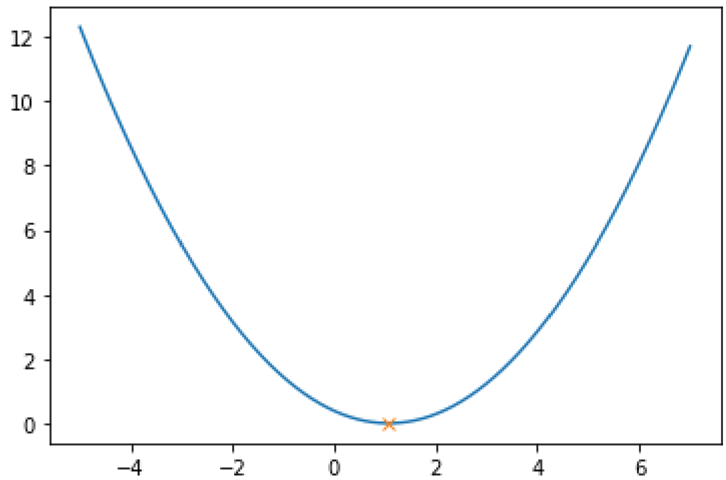
In [ ]:

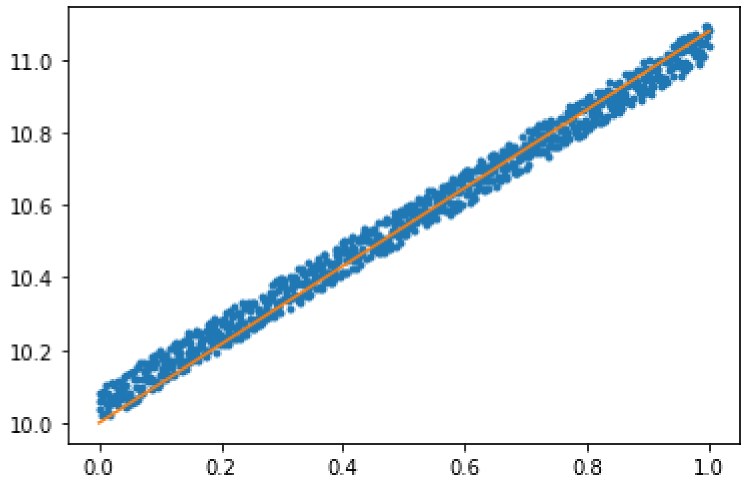
```
# implementation of heurastic search for 1 variable case
# write your code here
```

```
[[1.07807808]]
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f177f985128>]
```





d) Gradient descent

i) $Error = \frac{1}{m}\sum_{i=1}^{M}(y_{cori} - y_{pred_i})^2 = \frac{1}{m}\sum_{i=1}^{M}(y_{cori} - (w_0 + w_1 x_i))^2$

ii) $\nabla Error|_{w1} = \frac{-2}{M}\sum_{i=1}^{M}(y_{cori} - y_{pred_i}) \times x_i$

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

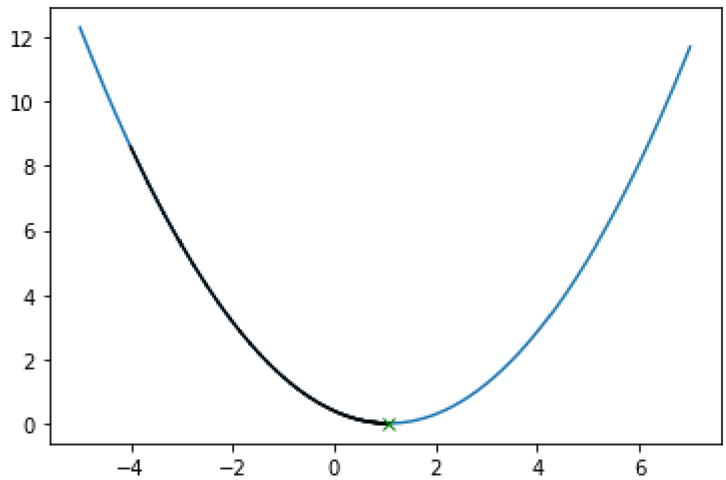
In [ ]:

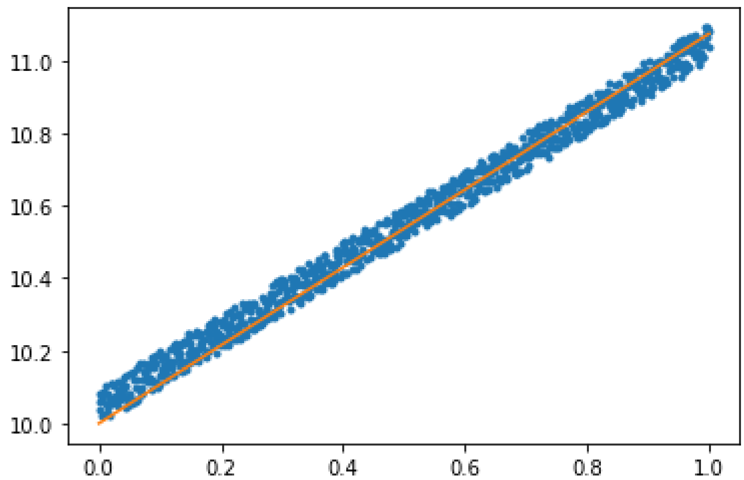
```
# write your code here
```

1.0740457320045673

Out[ ]:

[<matplotlib.lines.Line2D at 0x7f177f8d62b0>]





2) Fitting line with two unknown variables

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

i) Generate x, 1000 points from 0-1.

ii) Take $w_0 = 5$ and $w_1 = 1.5$ and generate y

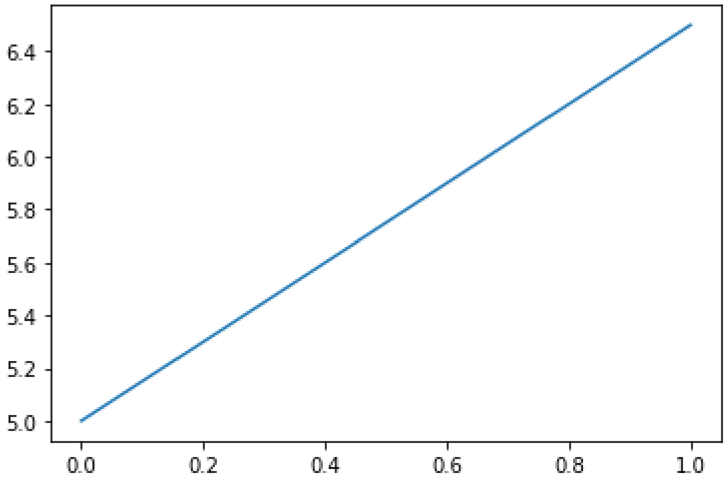
iii) Plot (x,y)

In [ ]:

```
# write your code here
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f177f673908>]
```



b) Corrupt the data using uniformly sampled random noise.

i) Generate random numbers uniformly from (0-1) with same size as y.

ii) Corrupt y and generate y_{cor} by adding the generated randomsamples with a weight of 0.1.

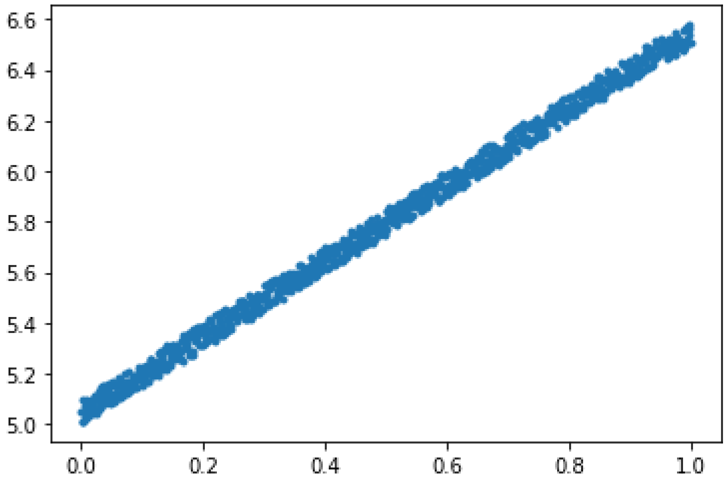
iii) Plot (x,y_{cor}) (use scatter plot)

In [ ]:

```
# write your code here
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f177f655668>]
```



c) Plot the error surface

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)

i) take w_1 and w_0 from -10 to 10, to get the error surface.

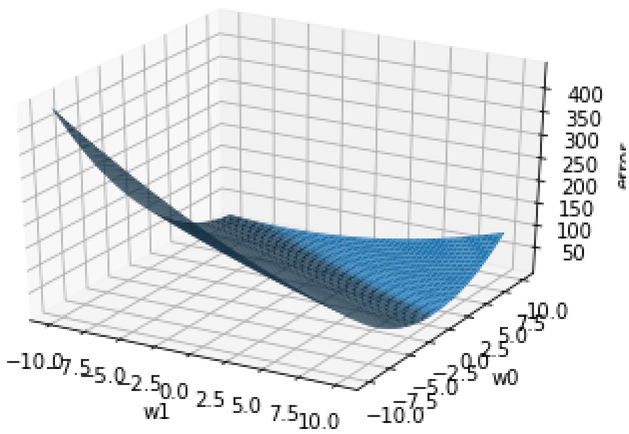
In [ ]:

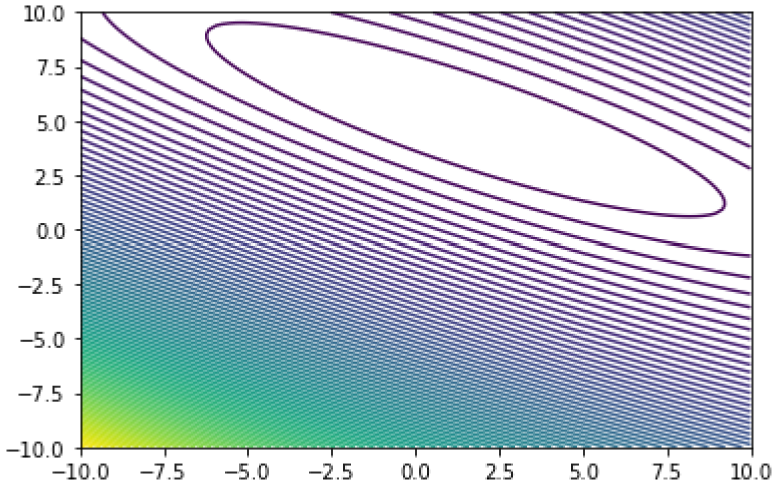
```
# write your code here
```

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

Out[ ]:

```
<matplotlib.contour.QuadContourSet at 0x7f177cd3c358>
```





d) Gradient descent:

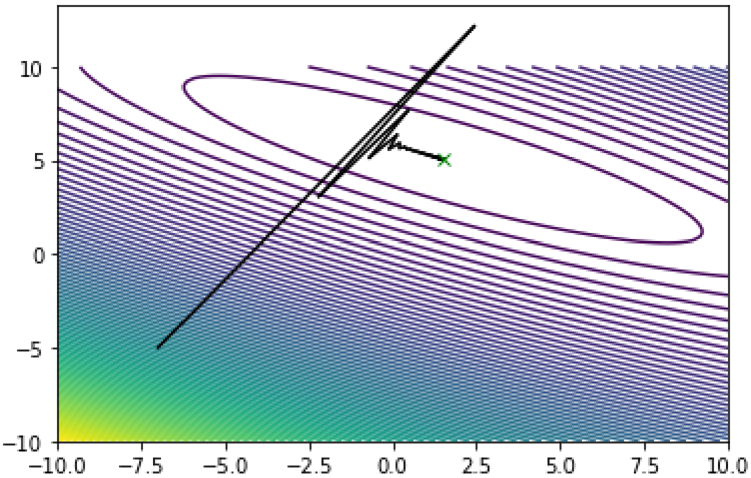
In [ ]:

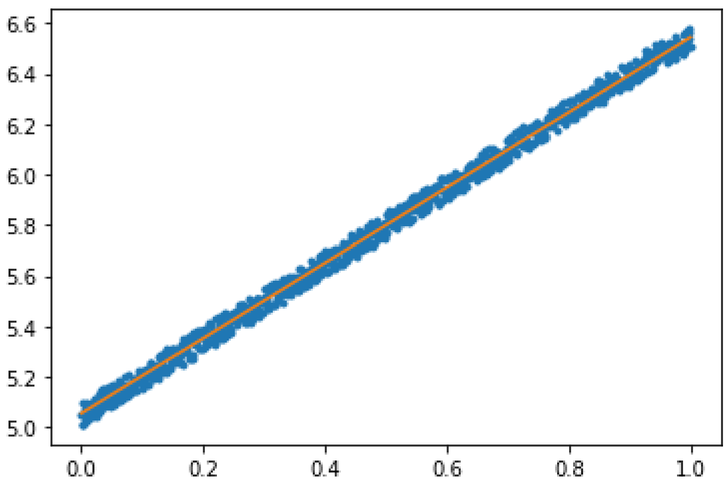
```
# Gradient descent
w1_init = -7 # initialization
w0_init = -5
lr = 0.6  # learning rate (0.9 diverges, 0.6 quite interesting)
eps = 0.000001
# write your code here
```

```
[5.0547047] [1.48986653]
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f177c679048>]
```





3. Fitting of a plane (two variables)

Here, we will try to fit plane using multiveriate regression

i) Generate x1 and x2 from range -1 to 1, (30 samples)

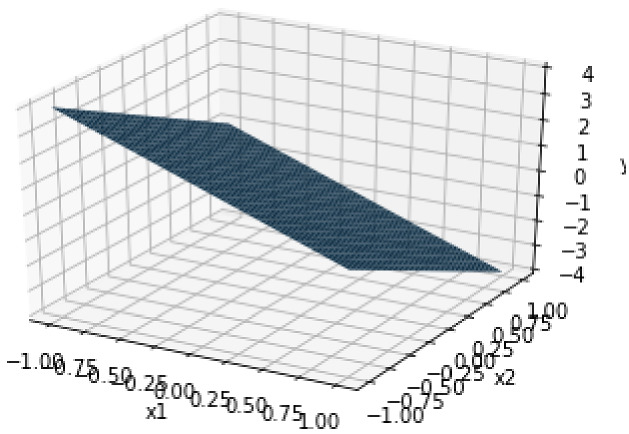
ii) Equation of plane $y = w0 + w1x1 + w2x2$

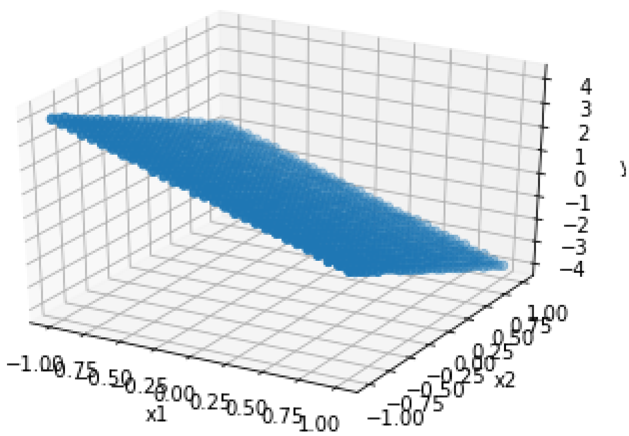
iii) Here we will fix w0 and will learn w1 and w2

In [ ]:

```
# write your code here
```

```
(900,)
```





b) Generate Error surface

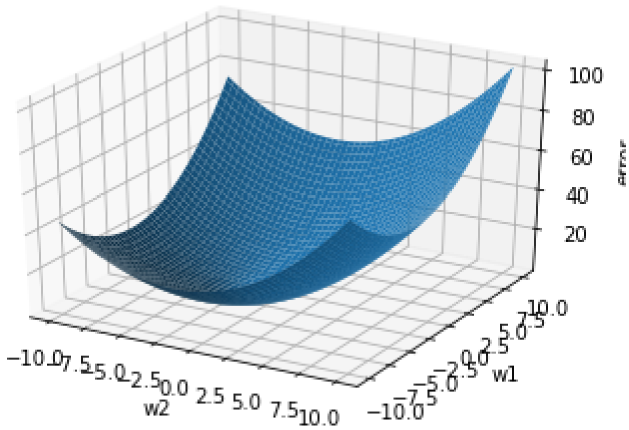
In [ ]:

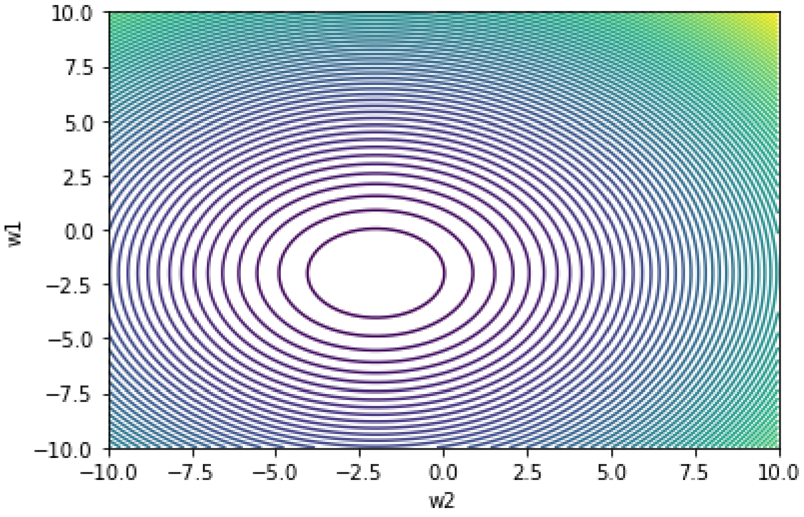
```
# write your code here
```

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

Out[ ]:

```
Text(0, 0.5, 'w1')
```





c) Gradient descent:

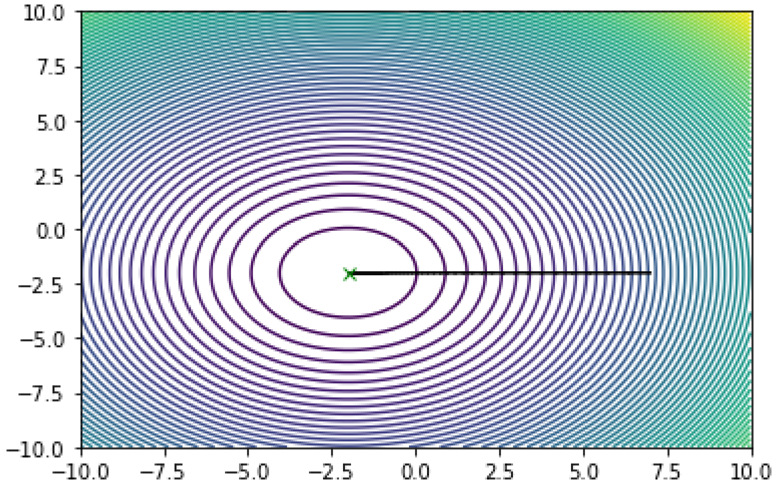
In [ ]:

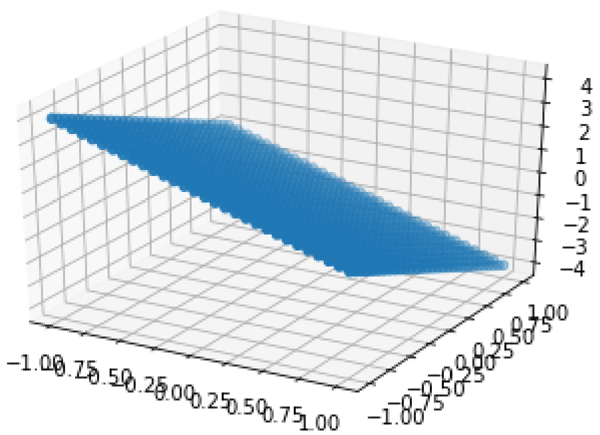
```
# write your code here
```

[-2.00083583] [-1.99902039]

Out[ ]:

<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7f177c993dd8>





4. 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 $x1, x2, x3, \ldots, x_M$. in vector form we can write $[x1, x2, \ldots, x_M]^T$, and similarly the weights are $w1, w2, \ldots w_M$ can be written as a vector $[w1, w2, \ldots w_M]^T$, Then the equation of the plane can be written as:

$$y = w1x1 + w2x2 + \ldots + w_M x_M$$

$w1, w2, \ldots., wM$ 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 $[1, x1, x2, \ldots, x_M]^T$ and the weight matrix is $[w0, w1, w2, \ldots w_M]^T$, now the equation of the plane can be written as:

$$y = w0 + w1x1 + w2x2 + \ldots + w_M x_M$$

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:

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

1. **By gradient descent:**

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

In [ ]:

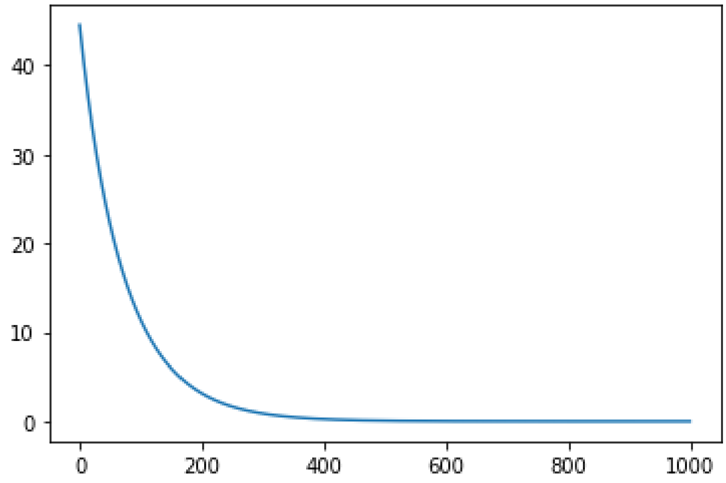
```
import numpy as np
import matplotlib.pyplot as plt

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 your code here
    return w

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

  def mat_inv(self,y,x_aug):
    return # write your code here
    # by Gradien descent
  def Regression_grad_des(self,x,y,lr):

    # write your code here

    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=np.array([[1],[2],[3],[5],[9],[3]]) # W=[w0,w1,.....,wM]'
print(w.shape)

# # augment feat

x_aug=np.concatenate((np.ones((1,x.shape[1])), x),axis=0)
print(x_aug.shape)


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


## corrupted by noise
nois=np.random.uniform(0,1,y.shape)
y=y+0.1*nois

### 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)
```

```
(5, 1000)
(6, 1)
(6, 1000)
(1000, 1)
[[1.0490242 ]
 [1.9998412 ]
 [2.99827382]
 [5.00088607]
 [9.0012772 ]
 [2.99881619]]
[[1.0489947 ]
 [1.98690233]
 [2.99364713]
 [4.99021553]
 [8.98736351]
 [2.99235975]]
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f177ca3ba58>]
```



5. Polynomial regression:

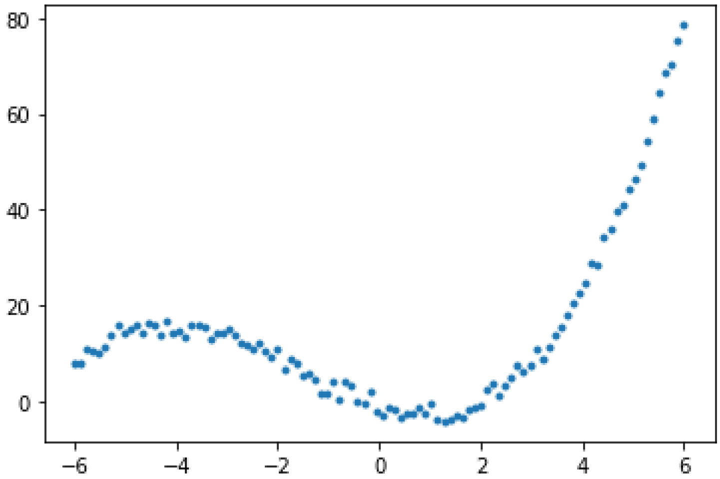
1. Generate data using relation $y = 0.25x^3 + 1.25x^2 - 3x - 3$
2. Corrupt y by adding random noise (uniformly sampled)
3. fit the generated curve using different polynomial order. (Using matrix inversion, and Home work using gradient descent)

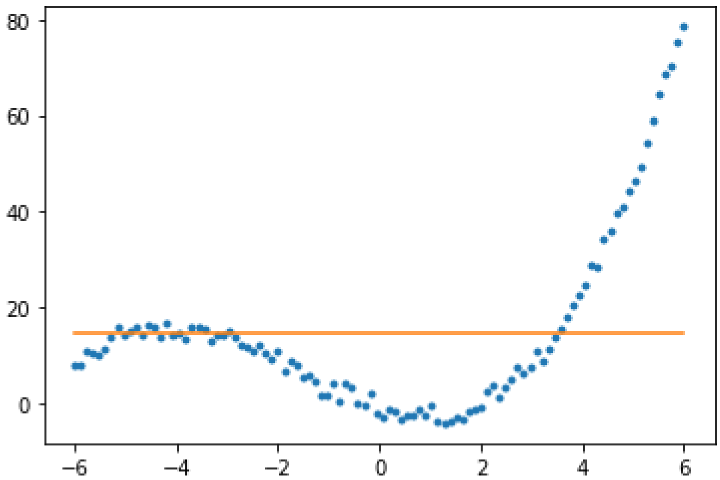
In [ ]:

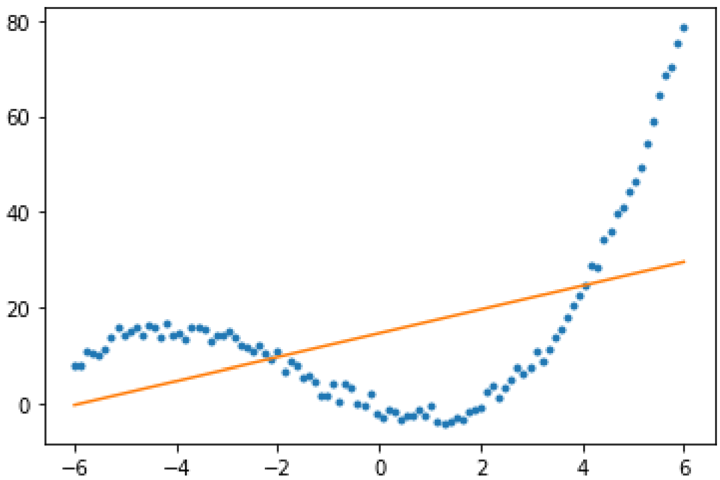
```
## data generation

# write your code here

def data_transform(X,degree):
 # write your code here
  return X_new


X=data_transform(x,3)

y=X.T @ w

y=y+5*np.random.uniform(0,1,y.shape)

plt.plot(x.T,y,'.')

reg=regression()

# by computation

# for degree 0 polynomial fitting
degree=0
X_1=data_transform(x,degree)
# print(X_1.shape)
w_mat=reg.mat_inv(y,X_1)
# print(y.shape)
# print(w_mat.shape)
y_pred=X_1.T @ w_mat
# print(y_pred.shape)
plt.figure()
plt.plot(x.T,y,'.')
plt.plot(x.T,y_pred)


# for degree 1 polynomial fitting
degree=1
# write your code here (like degree 0)

# for degree 2 polynomial fitting
degree=2
# write your code here


# for degree 3 polynomial fitting
degree=3
# write your code here

# for degree 4 polynomial fitting
degree=4
# write your code here

# xx=np.linalg.pinv((X_1 @ X_1.T)) @ X_1 @ y
# print(xx.shape)
```

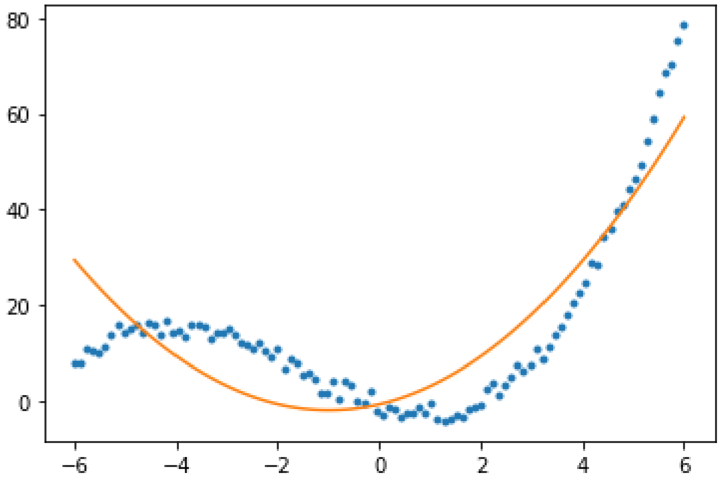
Out[ ]:

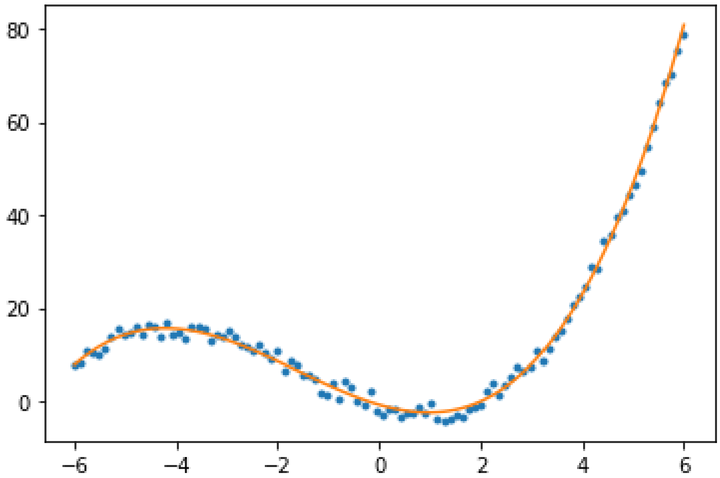
```
[<matplotlib.lines.Line2D at 0x7f177ad4c3c8>]
```

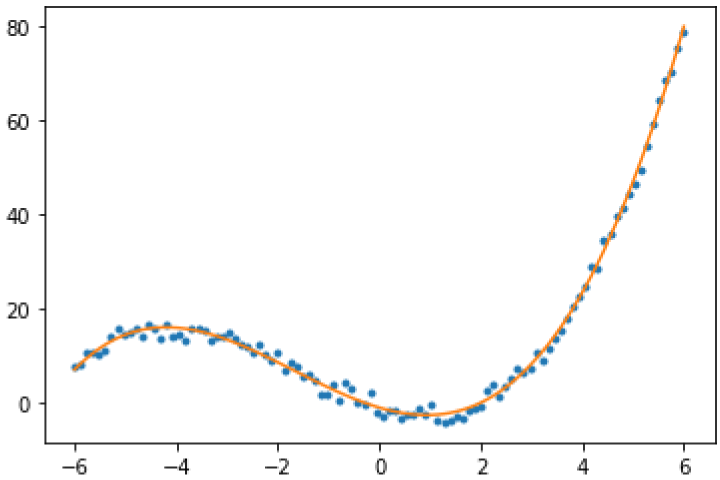







80
60
40
20
0
−6
−4
−2
0
2
4
6


80
60
40
20
0
−6
−4
−2
0
2
4
6


80
60
40
20
0
−6
−4
−2
0
2
4
6

6: Practical example (salary prediction)

1. Read data from csv file
2. Do train test split (90% and 10%)
3. Perform using matrix inversion and using Gradiant descent method
4. find the mean square error in test. (as performance measure)

In [ ]:

```
import numpy as np

# write your code here

# mean square error (testing) (normalized) #############

error=reg.error(w_pred,y_test,aug(x_test))/((np.max(y_test)-np.mean(y_test))**2)

print('Normalized testing error=',error,'\n')

print('predicted salary=',y_pred[0:3],'\n')
print('actual salary=',y_test[0:3])
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/au
th?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.goog
leusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&s
cope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20h
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.g
oogleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.goo
gleapis.com%2fauth%2fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /gdrive
Normalized training error= 0.02827224237168212

Normalized testing error= 0.05534340421775587

predicted salary= [[33469.35497582]
 [52694.83918006]
 [58642.13537189]]

actual salary= [[28084.]
 [48940.]
 [62952.]]
```