MI Assimnment-3

The provided csv file was imported after removing the first 2 columns as the id and the date has no paring to the price of the house.

price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
221900	3	1	1180	5650	1	0	0	3	7	1180	0	1955	0	98178	47.5112	-122.257	1340	5650
538000	3	2.25	2570	7242	2	0	0	3	7	2170	400	1951	1991	98125	47.721	-122.319	1690	7639
180000	2	1	770	10000	1	0	0	3	6	770	0	1933	0	98028	47.7379	-122.233	2720	8062
604000	4	3	1960	5000	1	0	0	5	7	1050	910	1965	0	98136	47.5208	-122.393	1360	5000
510000	3	2	1680	8080	1	0	0	3	8	1680	0	1987	0	98074	47.6168	-122.045	1800	7503
1.23E+06	4	4.5	5420	101930	1	0	0	3	11	3890	1530	2001	0	98053	47.6561	-122.005	4760	101930
257500	3	2.25	1715	6819	2	0	0	3	7	1715	0	1995	0	98003	47.3097	-122.327	2238	6819
291850	3	1.5	1060	9711	1	0	0	3	7	1060	0	1963	0	98198	47.4095	-122.315	1650	9711
229500	3	1	1780	7470	1	0	0	3	7	1050	730	1960	0	98146	47.5123	-122.337	1780	8113
323000	3	2.5	1890	6560	2	0	0	3	7	1890	0	2003	0	98038	47.3684	-122.031	2390	7570
662500	3	2.5	3560	9796	1	0	0	3	8	1860	1700	1965	0	98007	47.6007	-122.145	2210	8925
468000	2	1	1160	6000	1	0	0	4	7	860	300	1942	0	98115	47.69	-122.292	1330	6000
310000	3	1	1430	19901	1.5	0	0	4	7	1430	0	1927	0	98028	47.7558	-122.229	1780	12697
400000	3	1.75	1370	9680	1	0	0	4	7	1370	0	1977	0	98074	47.6127	-122.045	1370	10208
530000	5	2	1810	4850	1.5	0	0	3	7	1810	0	1900	0	98107	47.67	-122.394	1360	4850
650000	4	3	2950	5000	2	0	3	3	9	1980	970	1979	0	98126	47.5714	-122.375	2140	4000
395000	3	2	1890	14040	2	0	0	3	7	1890	0	1994	0	98019	47.7277	-121.962	1890	14018
485000	4	1	1600	4300	1.5	0	0	4	7	1600	0	1916	0	98103	47.6648	-122.343	1610	4300
189000	2	1	1200	9850	1	0	0	4	7	1200	0	1921	0	98002	47.3089	-122.21	1060	5095
230000	3	1	1250	9774	1	0	0	4	7	1250	0	1969	0	98003	47.3343	-122.306	1280	8850
385000	4	1.75	1620	4980	1	0	0	4	7	860	760	1947	0	98133	47.7025	-122.341	1400	4980
2.00E+06	3	2.75	3050	44867	1	0	4	3	9	2330	720	1968	0	98040	47.5316	-122.233	4110	20336
285000	5	2.5	2270	6300	2	0	0	3	8	2270	0	1995	0	98092	47.3266	-122.169	2240	7005
252700	2	1.5	1070	9643	1	0	0	3	7	1070	0	1985	0	98030	47.3533	-122.166	1220	8386
329000	3	2.25	2450	6500	2	0	0	4	8	2450	0	1985	0	98030	47.3739	-122.172	2200	6865
233000	3	2	1710	4697	1.5	0	0	5	6	1710	0	1941	0	98002	47.3048	-122.218	1030	4705
937000	3	1.75	2450	2691	2	0	0	3	8	1750	700	1915	0	98119	47.6386	-122.36	1760	3573
667000	3	1	1400	1581	1.5	0	0	5	8	1400	0	1909	0	98112	47.6221	-122.314	1860	3861

- Importing code

```
import os
import numpy as np
from numpy import genfromtxt
from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D
data = genfromtxt('/Users/tarekashraf/Downloads/house_price.csv',delimiter=',')
```

The next step is to divided the acquired data into train, cross validat and testing and normalaizing the data.

- Data diving code

```
X, y = data[1:10000, 1:], data[1:10000, 0]
Xcv, ycv = data[10001:14000, 1:], data[10001:14000, 0]
Xt, yt = data[14001:18000, 1:], data[14001:18000, 0]
```

- Normalaizing code

```
def  featureNormalize(X):
  X_norm = X.copy()
  mu = np.zeros(X.shape[1])
  sigma = np.zeros(X.shape[1])
  for i in range(18):
    mu = np.mean(X[:,i], axis = 0)
    sigma = np.std(X[:,i], axis = 0)
    X[:,i] = (X[:,i]-mu)/sigma
    Xcv[:,i] = (Xcv[:,i]-mu)/sigma
    Xt[:,i] = (Xt[:,i]-mu)/sigma
  return
```

Adding the bais term to the train, test and cross validat dataset.

```
b = y.size
g = np.ones(b)
X = np.column_stack([X,g])
X[:,[0, 18]] = X[:,[18, 0]]
b = ycv.size
g = np.ones(b)
Xcv = np.column_stack([Xcv,g])
Xcv[:,[0, 18]] = Xcv[:,[18, 0]]
b = yt.size
g = np.ones(b)
Xt = np.column_stack([Xt,g])
Xt[:,[0, 18]] = Xt[:,[18, 0]]
```

The next step is to calculate the cost function at different polynomials to get the best degree to fit the model the testing was done using the cross validat set.

- cost function code

```
def computeCostMulti(X, y, theta):
    m = y.shape[0]
    J = 0
    E = X.dot(theta)
    J = 1/(2*m)*np.sum(np.square(E-y))
    return J
def gradientDescentMulti(X, y, theta, alpha, num_iters):
    m = y.shape[0]
```

```
theta = theta.copy()
J_history = []
for i in range(num_iters):
    theta = theta - (alpha / m) * (np.dot(X, theta) - y).dot(X)
    J_history.append(computeCostMulti(Xcv, ycv, theta))
return theta, J_history
```

The error function was the lowest at the first degree function.

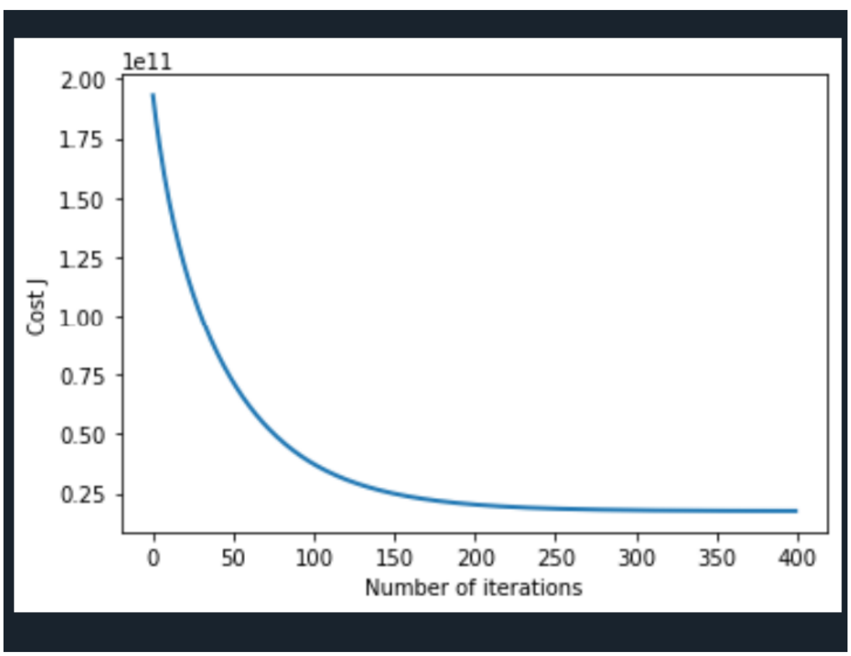
- Error function code (error calculator was done with the testing set)

```
def error(X, y, theta):
    m = y.shape[0]
    J = 0
    E = X.dot(theta)
    J = 1/(2*m)*np.sum(np.square(E-y))
    return J
```

- error at 2nd degree with alpha = 0.01 and num_iters = 400 : 713858722581
- error at 1nd degree with alpha = 0.01 and num_iters = 400 : 159803527143

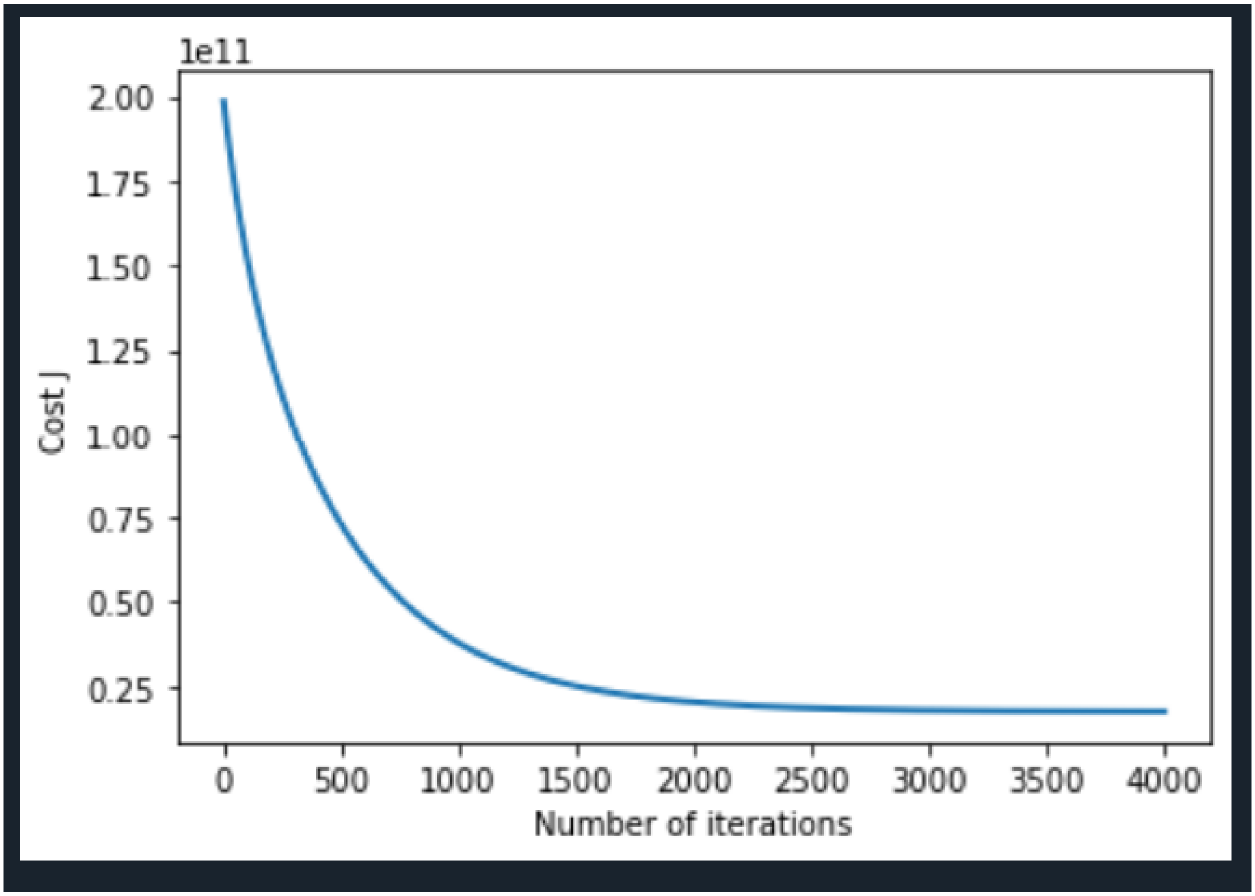
So there is no need to check a higher degree as the cost increased dramatically with the 2nd degree.

Tuning the learning rate and the iteration number.
Using the alpha = 0.01 and num_iters = 400.
The cost seteles very quickly at num_iters = 200 so the alpha maybe to large.

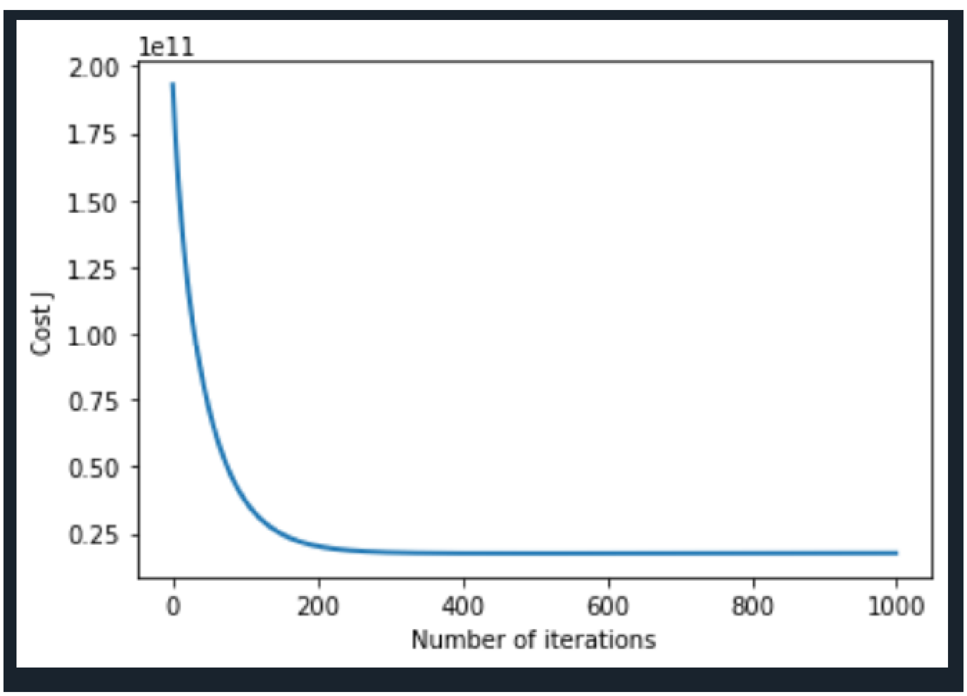


Using the alpha = 0.001 and num_iters = 4000.

The cost seteles very quickly at num_iters = 2000 but there is an increase in the error test so this maybe over fitting.



The best combination of alpha and num_iters is 0.007 and 1000.



The final set is adding the Regularization term and tunning the Regularization factor.

- Regularization code(new theta calculation)
  theta = theta - (alpha / m) * (np.dot(X, theta) - y).dot(X) + lam*theta

Staring with Regularization factor = 0.01

There was an increase in the error function from 153803527143 to 21646999997.

Then moving to a Regularization factor = 0.02 there was a decrease from the original cost function of 153803527143 to 119574722077.

Moving to a Regularization factor = 0.04 there was an incresese in th error function to 47802945792.