1. Write down the hypothesis parameters as you have obtained in completing the assignment using housing price dataset of Windsor city of Canada using:
   a. Normal equation algorithm
   b. Gradient descent algorithm. Also mention approximately how many epochs the algorithm took to converge with different learning rates.
   c. Explain in details how was the performance of the predictor/hypothesis in both the cases.

Answer:

a) Hypothesis parameters obtained using normal equation implementation is as below. Normalization has not been done for normal algorithm.

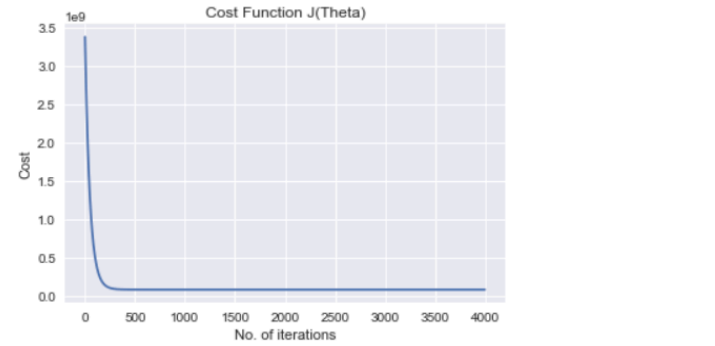
```
In [32]: theta = np.matmul(np.linalg.inv(np.matmul(X.T,X)),np.matmul(X.T,y))
         print(theta)

         [78609.          7350.84253883  2923.45007788 10651.20328512
           5427.00783509  6414.41197345   160.76852306  4855.06325266
           2776.95324847   884.08217151  4619.18067772]
```

b) I have used following code for implementation of gradient descent algorithm.

```
In [33]: def gradient_descent(x,y,theta,num_iterations,alpha):
             past_costs = []
             past_thetas = [theta]
             for i in range(num_iterations):
                 prediction = np.dot(x,theta)
                 error = prediction - y
                 cost = 1 / (2 * m) * np.dot(error.T, error)
                 past_costs.append(cost)
                 theta = theta - (alpha * (1/m) * np.dot(x.T, error))
                 past_thetas.append(theta)
             return past_thetas, past_costs
```

Below is a table showing no. of epochs and learning rate for the convergence of our algorithm and Cost vs. number of iterations.

Alpha	Epochs
0.01	250
0.03	100
0.1	19
0.3	8
1	1
3	Gradient Descent Diverges



c) After increasing the number of iterations for alpha = 0.01, I obtained the same value of theta as obtained using the normal equation.

```
In [74]: alpha = 0.01
         iterations = 4000
         m = y.size
         np.random.seed(123)
         theta1 = np.random.rand(11)
```

```
In [80]: weight,cost = gradient_descent(X,y,theta1,iterations,alpha)
         z = weight[-1]
         print(z)
```

```
[78609.          7350.85432143  2923.47138742 10651.20337073
  5427.00337995  6414.4055667    160.8010363   4855.02671085
  2776.94630657   884.07281603  4619.17734368]
```

The root mean squared error obtained after prediction was 12376.02. So we can use below hypothesis to predict a model.
Predicted_value = new_data['Predicted_val'] ± 12376.02

```
In [96]: squared_error=np.sqrt(np.sum((new_data['price']-new_data['Predicted_val'])**2)/new_data.shape[0]

         print(squared_error)
```

```
12376.023550581374
```

2. Write down the hypothesis parameters as you have obtained in completing the assignment using housing price dataset of Windsor city of Canada using:
   a. Normal equation algorithm with regularization.
   b. Gradient descent algorithm with regularization. Also mention(in tabular form) approximately how many epochs the algorithm took to converge with different learning rates and regularization parameter.
   c. Explain in details how was the performance of the predictor/hypothesis with regularizer and without regularizer.

Answer:

a) Hypothesis parameters obtained using normal equation implementation is as below. Normalization has not been done for normal algorithm.

```
In [186]: lemda = 10
          a = np.identity(X.shape[1])
          a[0][0] = 0
          theta = np.matmul(np.linalg.inv(np.matmul(X.T,X) + lemda * a),np.matmul(X.T,y))
          print(theta)
```

```
[40258.60166749 38571.11193726 12725.66543547 32027.71391458
 22746.16796064 19101.72772245]
```

b) Below implementation of gradient descent algorithm has been done by me for L2 regularization.

```
In [108]: def gradient_descent(x,y,theta,num_iterations,alpha,lemda):
              past_costs = []
              past_thetas = []
              lembda_mat = lemda * np.identity(X.shape[1])
              lembda_mat[0][0] = 0
              for i in range(num_iterations):
                  prediction = np.dot(x,theta)
                  error = prediction - y
                  cost = 1 / (2 * m) * ( np.dot(error.T, error) + lemda * np.dot(theta.T, theta))
                  past_costs.append(cost)
                  theta = theta - (alpha * (1/m) * ( np.dot(x.T, error) + np.matmul(lembda_mat,theta)))
                  past_thetas.append(theta)
              return past_thetas, past_costs
```

```
In [ ]: weight,cost = gradient_descent(X,y,theta1,3000,alpha,lemda)
```

The weight obtained from gradient descent with regularization and normal equation with regularization should be same if our implementation is correct.

In [184]:
```
print(weight[-1])
print(theta)
```

```
[40258.30454663 38570.83798238 12726.71113715 32027.31451667
 22746.04504824 19101.79074926]
[40258.60166749 38571.11193726 12725.66543547 32027.71391458
 22746.16796064 19101.72772245]
```

We can say that weights are almost same as we have obtained using normal equation.
The table below shows the number of iterations my algorithm took to converge for different values of

Alpha	Lembda	Epochs
0.01	10	190
0.01	30	200
0.03	10	60
0.03	30	55
0.1	10	17
0.1	30	19
0.3	10	5
0.3	30	6
1	10	3
1	30	1
3	10	Gradient Descent Diverges
3	30	Gradient Descent Diverges

c) Root mean squared error obtained after calculating the hypothesis is as below.

In [58]:
```
predicted_val = np.matmul(X,weight[-1].T)
df['prediction'] = predicted_val
df['Squared_error'] = (df['price'] - df['prediction']) ** 2
mean_error = np.sum(df['Squared_error']) * (1 / (df.shape[0]))
print(np.sqrt(mean_error))
```

```
18504.369905958927
```

Observation: Root mean squared error obtained using regularization is 18504.3699 while root mean squared error obtained without regularization is 12376.02. With regularization our error is a bit higher and it is intuitive because we use regularization in order to overcome overfitting. And we penalize the higher order polynomials with a regularization term. Which will increase the cost, but it will help in better generalization. We will definitely get good results on test data.

3. Write down the performance of the algorithm using LWR algorithm in comparison to normal equation and Gradient Descent algorithm with regularizer. Explain how have you tuned the parameters with the final value of Γ. Where the symbols has their usual meaning as mentioned.

Answer: Cost function and Gradient descnet for LWR can be given as :

In [66]:
```
def cost(x1, x, y, w,t = 1):
    m = len(y)
    e = (np.dot(x, w) - y) ** 2
    tau = np.array(np.exp(-1 * np.sum(((x - x1) **2), axis = 1)/(2 * t))).reshape(-1,1)
    return np.sum(np.multiply(tau,e))
```

In [68]:
```
def gradient_descend(x, y, alpha, x1, iterations, t = 1):
    cost_list = []
    w = np.zeros((6,1))

    for i in range(iterations):
        tau = np.array(np.exp(-1 * np.sum(((x - x1) **2), axis = 1)/(2 * t)  )).reshape(-1,1)
        grade = (np.dot(x.T, np.multiply(np.dot(x,w) - y, tau)) )
        w = w - alpha * grade
        cost_list.append(cost(x1, x, y, w, t))
    return cost_list, w
```

I have calculated the weights for different values of Γ and below observation has been found.

Observation:

When value of tau is very small, it will overfit the model because width of the curve will be small and very few points will be responsible for the training. And hence, value of actual cost and predicted costs will be same.