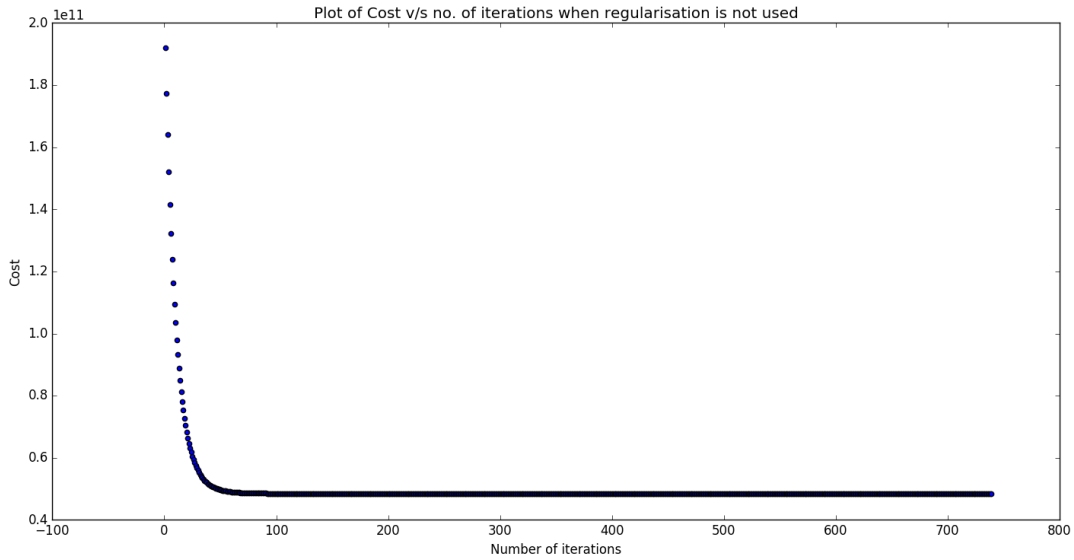
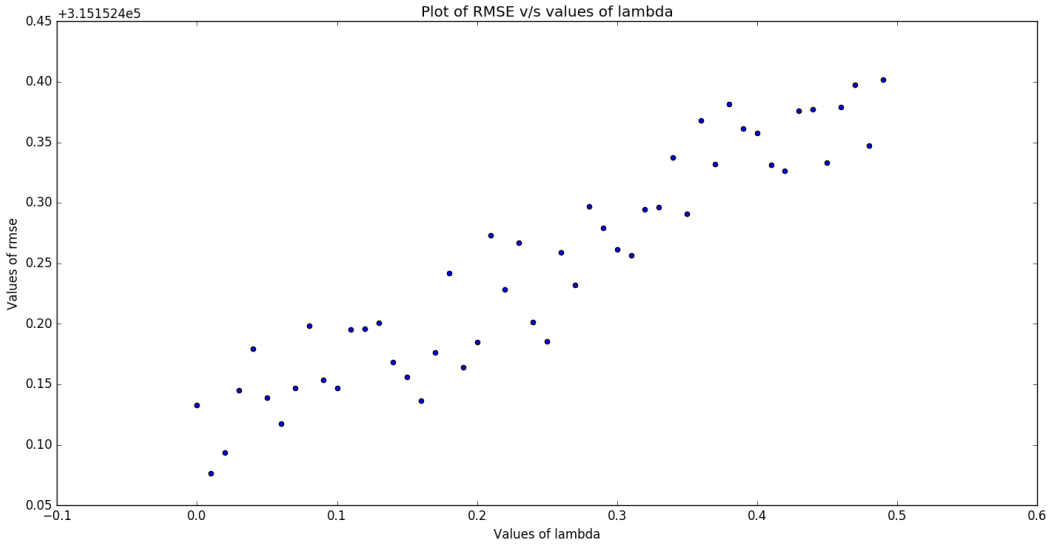
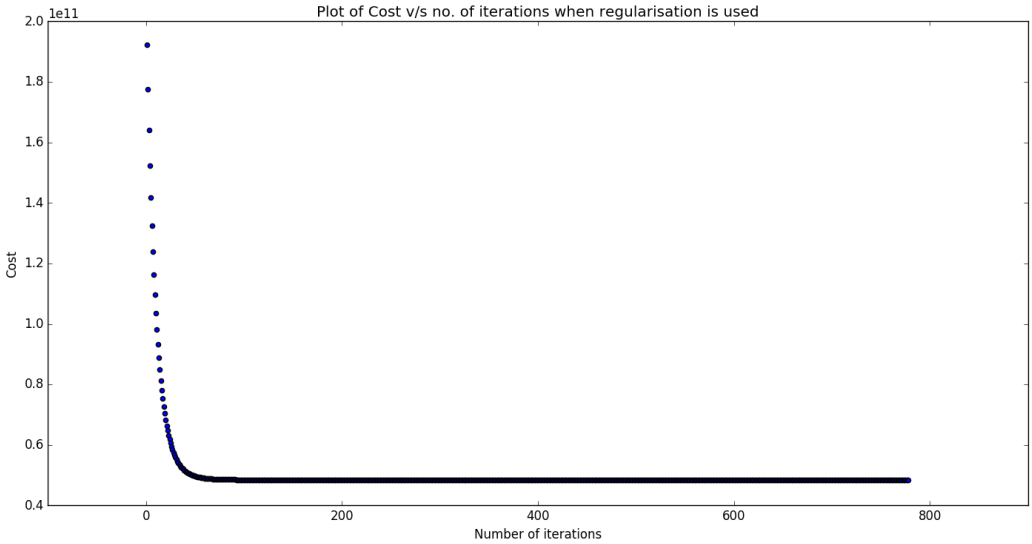
Report

Although the main function in each file is written in such a way to understand the program control and sequence very easily, I would also like to provide a concise report here:

Part (a)



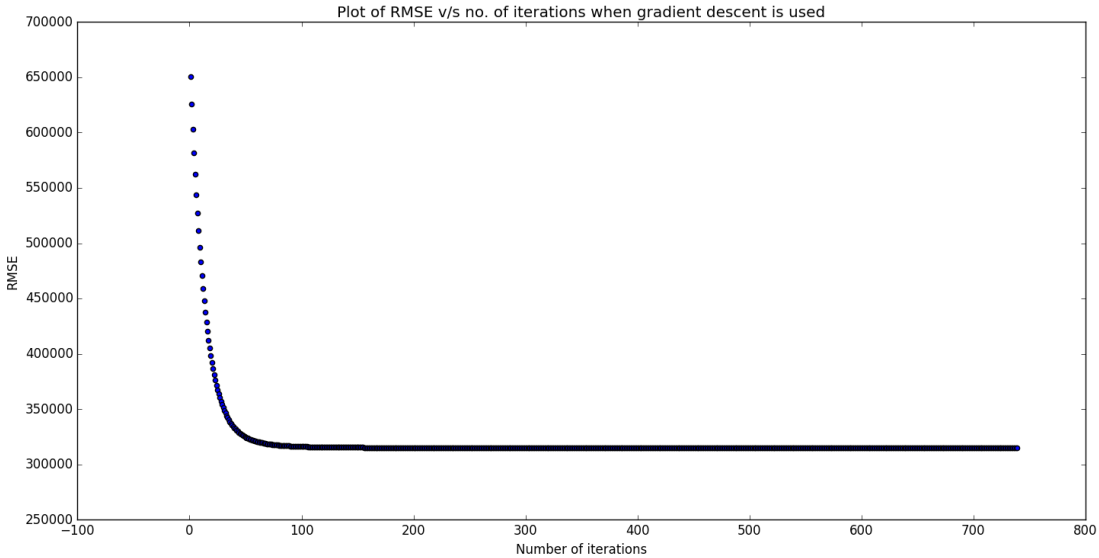


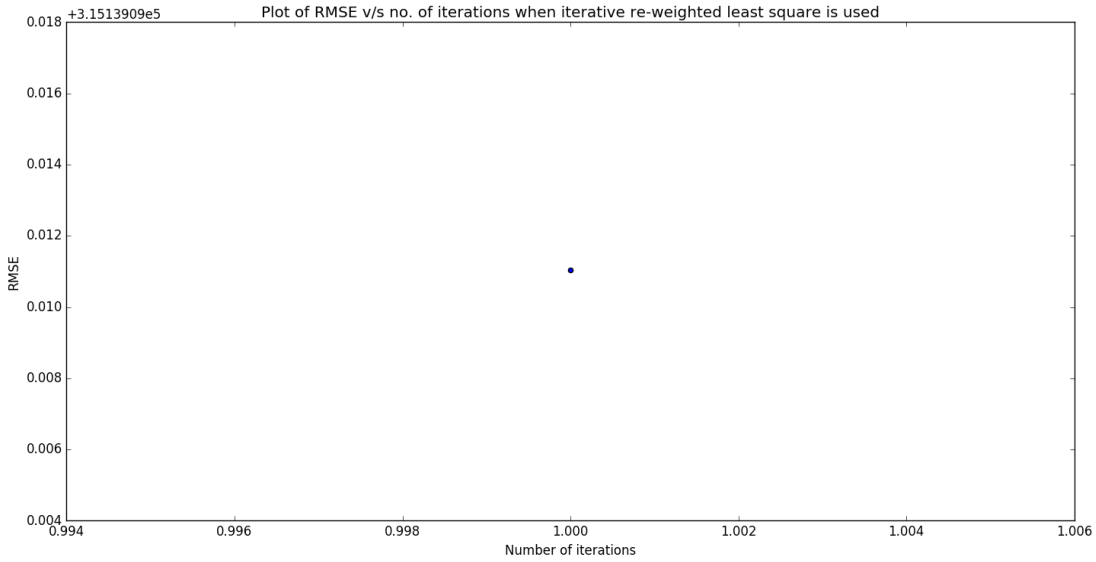


problem without regularization, alpha = 0.05
Learned values of parameters:
[[533343.83233083]
[15704.69396639]
[10273.73321201]
[16360.12429035]
[176366.33755878]]

problem with regularization, alpha = 0.05, lambda = 0.01
Learned values of parameters:
[[533343.83233082]
[15704.69294901]
[10273.77045451]
[16360.15846264]
[176366.21702372]]

Part (b):



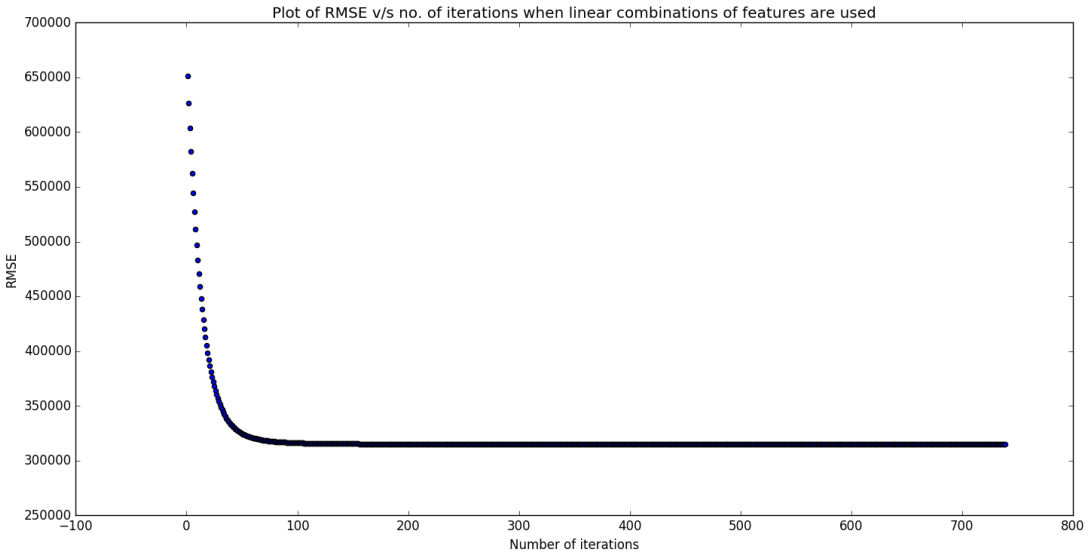


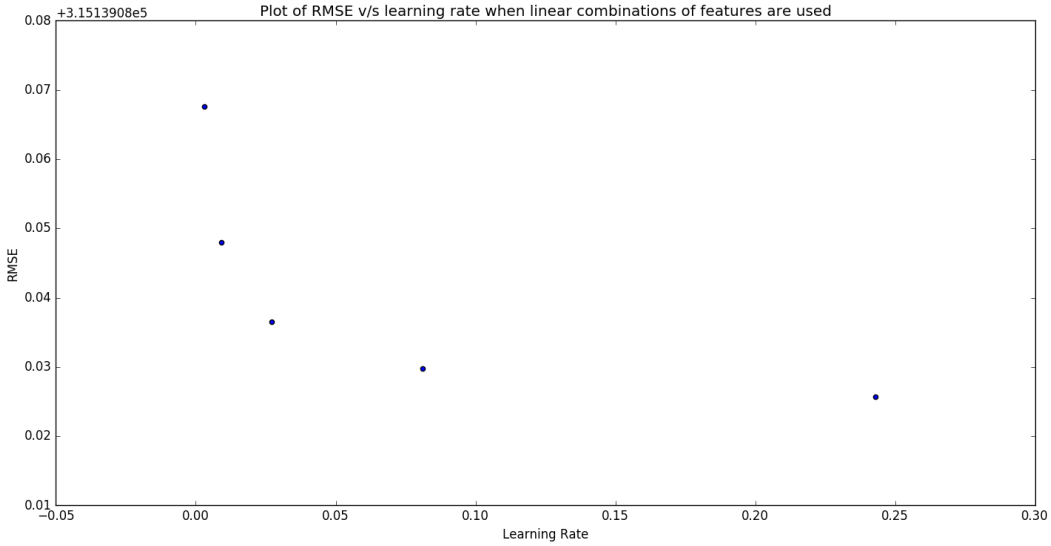
Gradient Descent:
Learned values of parameters:
[[533343.83233083]
[15704.6939717]
[10273.73324342]
[16360.12432329]
[176366.33750576]]

Iterative re-weighted least square method:
Learned Parameters:
[[533343.83233083]
[15704.68461889]
[10273.67793273]
[16360.06629425]
[176366.43090329]]

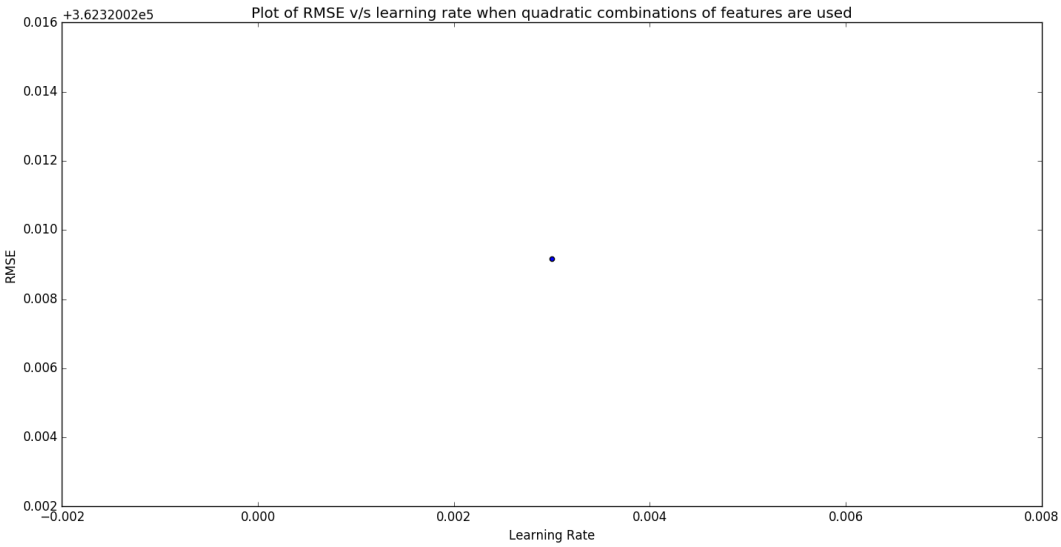
For the given problem, I would prefer the iterative re-weighted least square method because it gives out the values of parameters in just one iteration and that too exactly. IRLS is not used in situations where there are million of features to account for because inversion of such a big resulting matrix in the algorithm would take a lot of time.

Part (c):



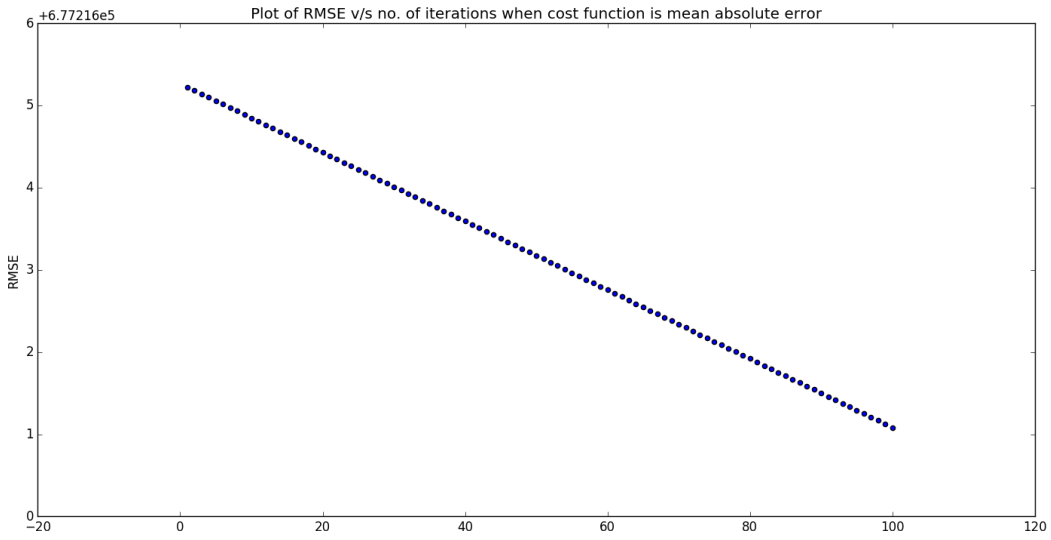


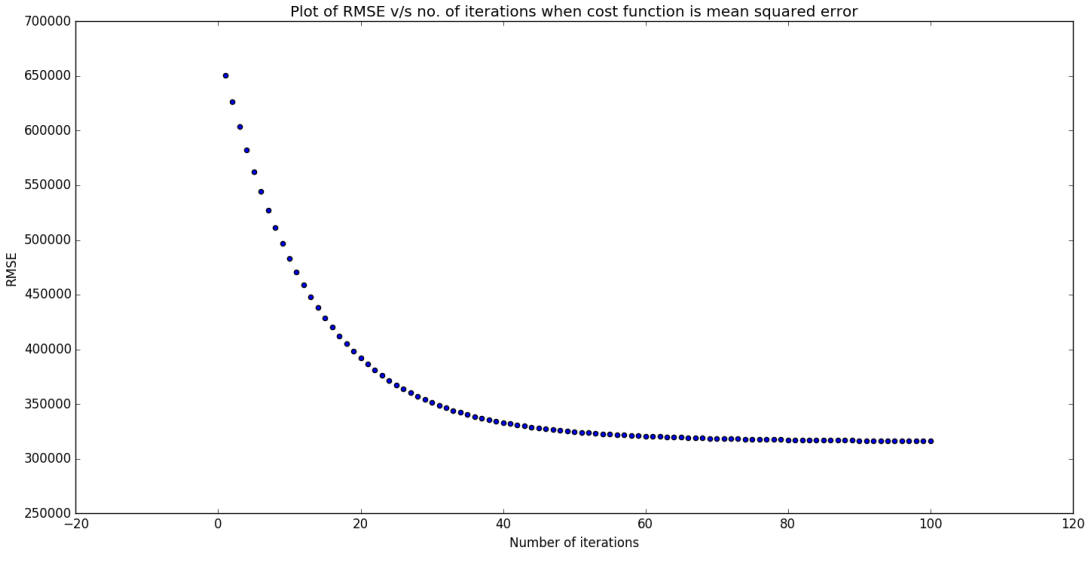
Learned parameter values(only linear features model converged):
[[533343.83233083]
[15704.69381031]
[10273.73228902]
[16360.12332199]
[176366.33911735]]

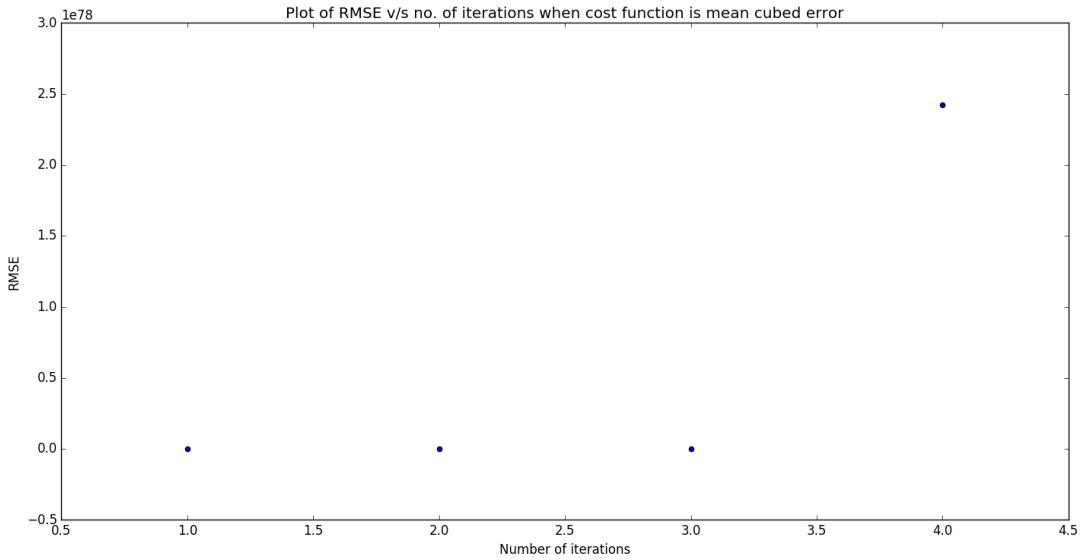


From the above graphs, we observe that the linear features converge for all alpha values, the quadratic features only converge for alpha=0.003 out of the given values and the cubic features don't converge for any alpha value of the given values.
Since for the given alpha = 0.05, only linear combinational features were leading to convergence, so for this problem, I would choose linear combinations.

Part (d)







Learned parameter values (mean abs error)
[[38.31935559]
[774.08139936]
[328.4022212]
[284.47319895]
[157.15195473]]
Learned values of parameters:(mean squared error)
[[530186.645475]
[17059.13265849]
[18683.43992716]
[25044.60315712]
[162278.1264686]]
Mean cubed error actually resulted in runtime error due to the huge values which were generated while performing gradient descent at alpha = 0.05.

For this problem, I would prefer the mean squared error cost function as it decays more rapidly than mean absolute error and does not get to huge values like in case of mean cubed error.