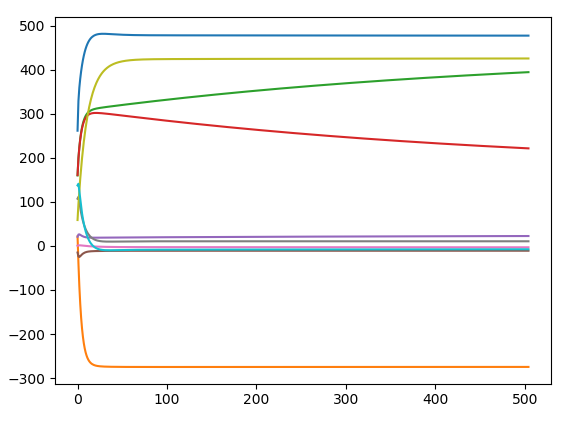
**Report by Anak wannaphaschaiyong and Linie Zamir**

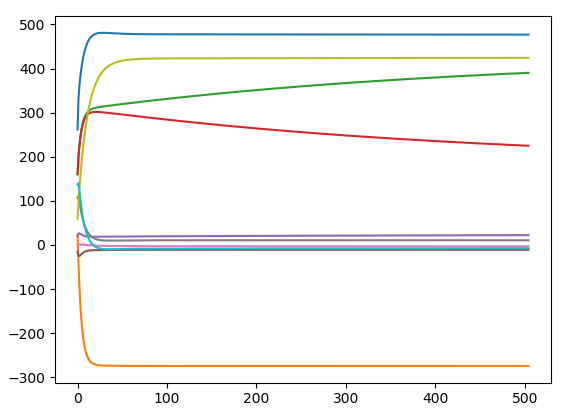
**Deliverable 1:**

Beta\_coeff vs epoch

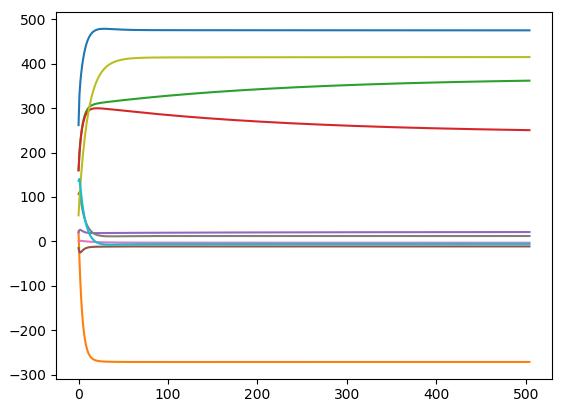
Lambda = 0.01



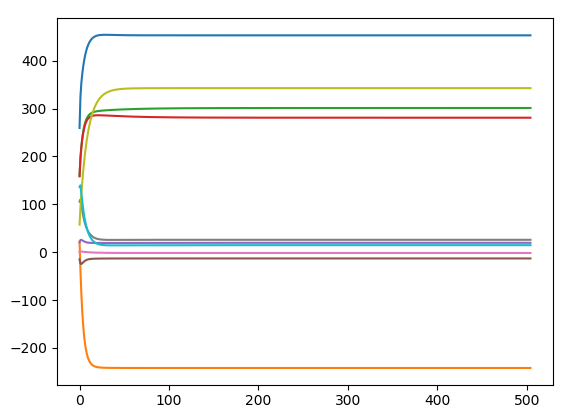
Lambda = 0.1



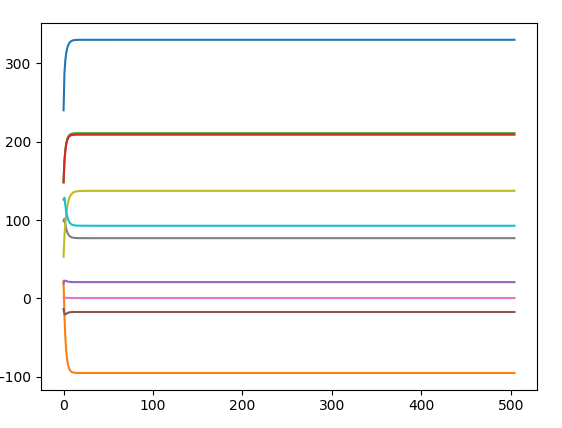
Lambda = 1



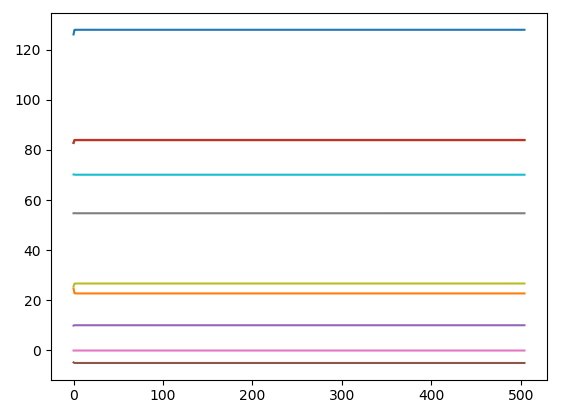
Lamda = 10 (shrinking pattern starts to be noticeable)



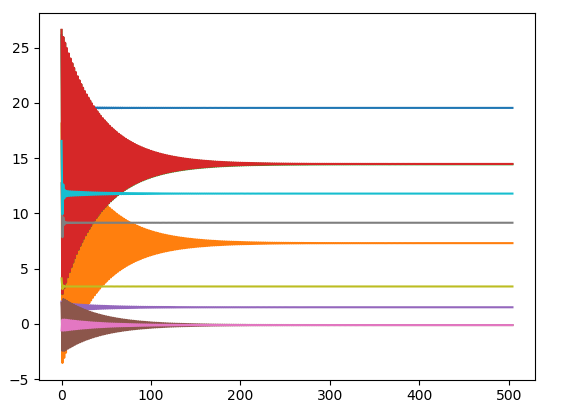
Lambda = 100



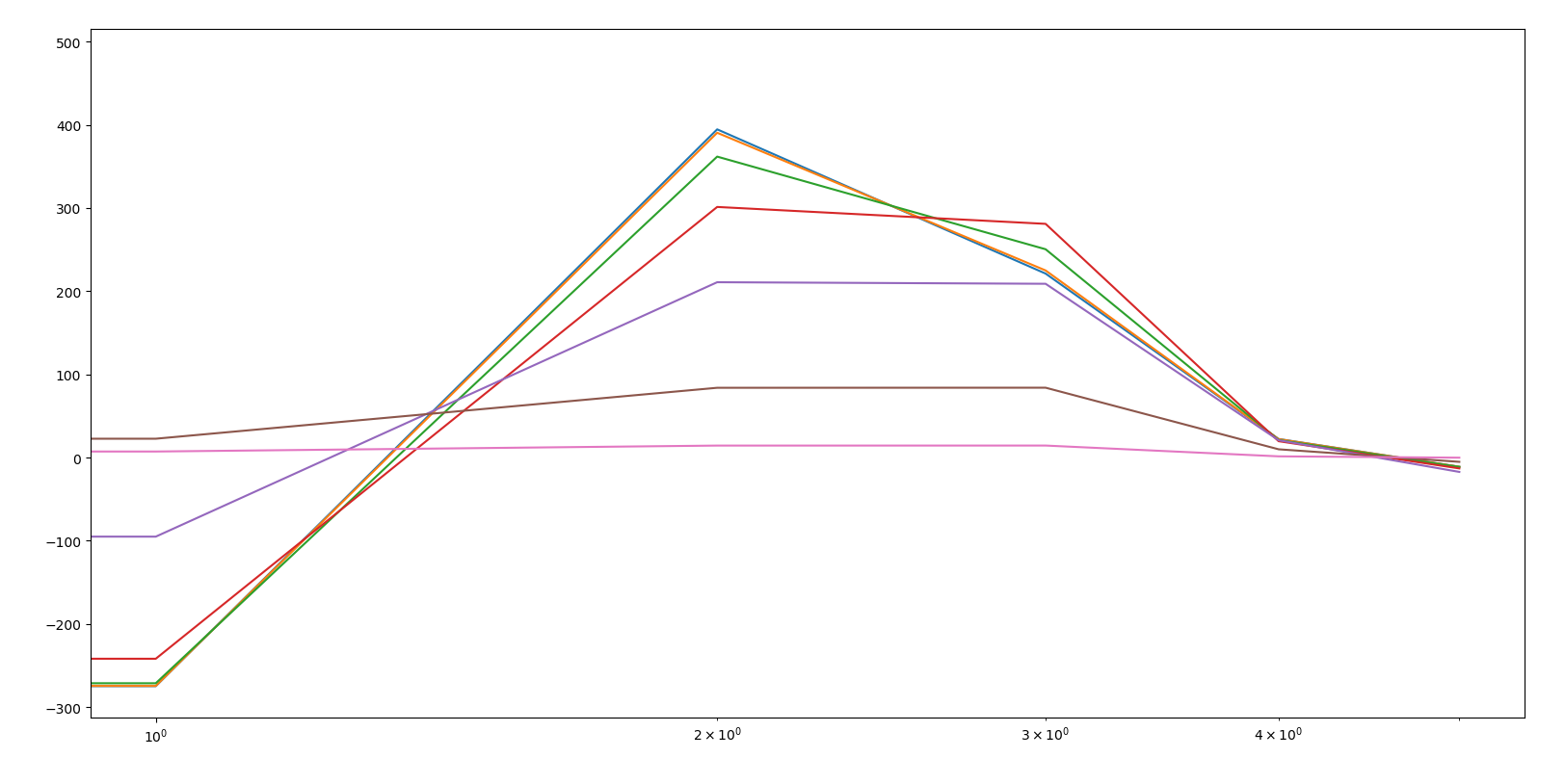
Lambda =1000



Lambda = 10000 (interesting convergence pattern)

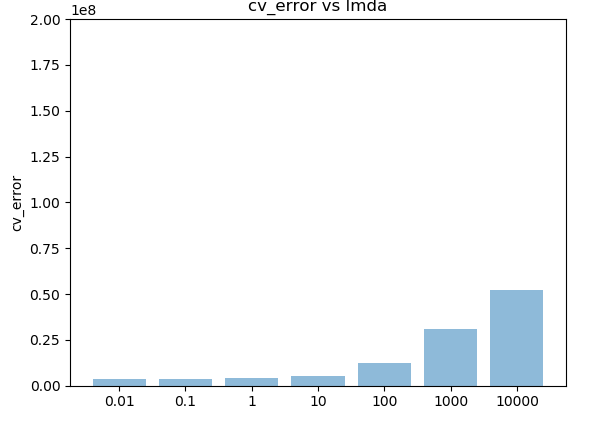


Beta\_coeff vs lamda



**Delivery 2**

Cv error vs lamda.



**Delivery3**

Lambda =0.01 obtains best performance as shown below pair with loss\_value

Lamda = 0.01 => 3819168.561472533 (best performance)

Lamda = 0.1 => 3849967.1505458215

Lamda = 1 => 4062346.1892320784

Lamda = 10 => 5524696.074350899

Lamda = 100 => 12621942.384689316

Lamda = 1000 => 30847784.959911693

Lamda = 10000 => 52257044.9267018

**Deliverable 4:**

Run best parameter on all dataset (no cross validation just prediction)

**Note: loss value is shown on the last printed line after code is run.**

From Deliverable 3, **lamda = 0.01, learning rate =0.00009, batch\_size = 32 are used, epoch = 1000**

**Loss\_val = 3808498.1779229348**

**Deliverable 5:**

Code can be pull from github <https://github.com/Anak2016/ridge_regr_with_bgd>

Command line to use I structure the file as I would structure I for any of my machine learning/deep learning projects. To run the code go to, models/ridge\_regr.py and use the following command

--verbose --epochs 100 --bs 32 --lmda 0.01 --lr 0.00009

Observation

Epoch is chosen to be 100 instead of 1000 because it is shown to converge at the fast rate. One can notice from the output “loss\_diff”,where loss\_diff = loss\_val\_i+1 - loss\_val\_i. loss\_diff start of at about 2-5 percent of the loss\_val and,at epoch 100, loss\_diff is insignificant it is about 1e-15 percent (loss\_diff / loss\_val)

Explaination of convergence and learning rate

Learning rate = lambda =0.00009 always converge

Learning rate = lambda =0.0009 converge up to 1 and it starts to diverge at lambda = 10 because punishment and gradient increase iteratively and reinforce divergence. (depends on topology of the learned dataset)

