**Fahad Fiaz – (**303141**) – G2**

**System Info:**

|  |  |
| --- | --- |
| Processor | i7-5500U , 2.40GHz |
| Cores | 4 |
| Operating system | Windows 64 Bit |
| Ram | 8GB |
| Programming Language | Python 3.7.7 |

**Exercise 1: Parallel Linear Regression?**

*Steps:*

1. Consider 1st process as master node and other as slave node.

2. Master process will first read data from files. Since there are multiple files in dataset so we use glob library to read names of multiple files at once. Then the dataset is randomly shuffled and preprocessed to get X as Feature matrix and Y as Prediction Vector. Then we split data into 70% train and 30% test data. Then master process divides the training data into equal chunks according to number of workers and scatters it to child Nodes.

# Pre\_processing function to convert data to specific format  
def pre\_processing*(*feature\_example*)*:  
 total\_features = np.zeros*(*482*)* y = 0.0  
 processed\_features = feature\_example.split*(*" "*)* if *(*processed\_features*[*0*]*.strip*()* != ''*)*:  
 y = float*(*processed\_features*[*0*])* for i in range*(*len*(*processed\_features*))*:  
 split\_features = processed\_features*[*i*]*.split*(*":"*)* if *(*len*(*split\_features*)* == 2*)*:  
 total\_features*[*int*(*split\_features*[*0*])]* = int*(*split\_features*[*1*])* return total\_features, y

Total\_data = *[]* path = "dataset"  
 All\_Files = glob.glob*(*path + "/\*.txt"*)* for name in All\_Files:  
 File = open*(*name*)* File\_data = File.read*()*.split*(*"\n"*)* Total\_data.append*(*File\_data*)* np.random.shuffle*(*Total\_data*)* # shuffle complete dataset randomly  
 for i in range*(*len*(*Total\_data*))*:  
 for j in range*(*len*(*Total\_data*[*i*]))*:  
 x, y = pre\_processing*(*Total\_data*[*i*][*j*])* X.append*(*x*)* Y.append*(*y*)* X = np.array*(*X*)* # feature matrix  
 Y = np.array*(*Y*)* # prediction vector  
 FullDataset = X.shape*[*0*]* # splitting Dataset in Test/Train format  
 X\_Train = X*[*:*(*FullDataset \* 70*)* // 100, :*]* X\_Test = X*[(*FullDataset \* 70*)* // 100:, :*]* Y\_Train = Y*[*:*(*FullDataset \* 70*)* // 100*]* Y\_Test = Y*[(*FullDataset \* 70*)* // 100:*]* # partitioning dataset  
 data\_size = len*(*X\_Train*)* partition = data\_size // size  
 for process in range*(*0, size*)*: # partition data in equal portions  
 start = process \* partition  
 end = *(*process + 1*)* \* partition  
 if process == size - 1 and end != data\_size:  
 end = data\_size  
 X\_Chunk.append*(*X\_Train*[*start:end*])* Y\_Chunk.append*(*Y\_Train*[*start:end*])*

3. Slave processes received chunk of Feature Matrix and Prediction Vector. Then they use SGD algorithm to find parameters (**Beta**) of model.

# scattering data to child processes  
Feature\_chunk = np.array*(*comm.scatter*(*X\_Chunk*))*Prediction\_chunk = np.array*(*comm.scatter*(*Y\_Chunk*))*

Parameters = SGD*(*Feature\_chunk, Prediction\_chunk, alpha*)*

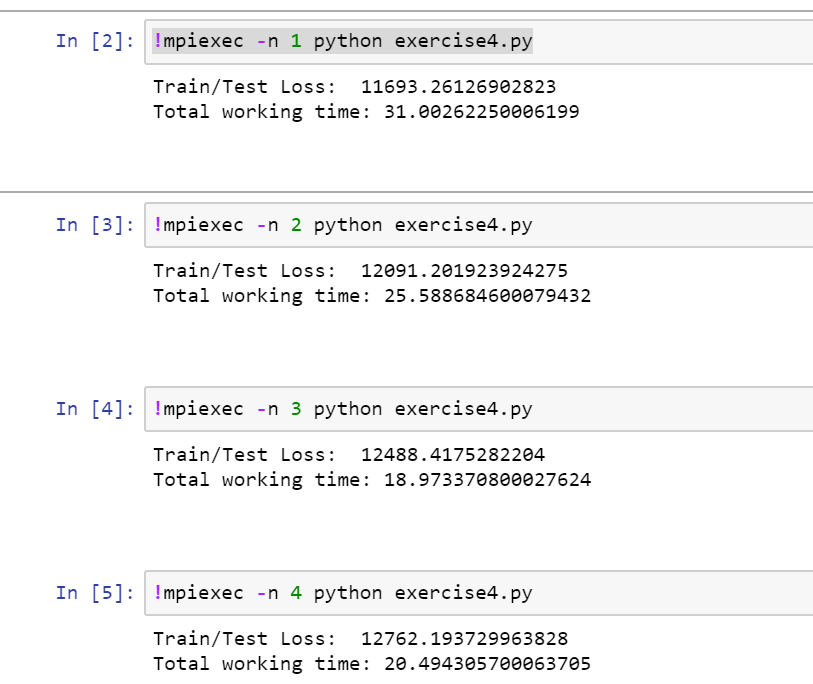
def SGD*(*X\_Train, Y\_Train, alpha*)*: # function to apply Stochastic Gradient Descent algorithm  
 if flag == False: # checking if you are running sgd for 1st epoch else Parameters will get value from broadcasting  
 Parameters = np.zeros*(*len*(*X\_Train*[*0*]))* for i in range*(*len*(*X\_Train*))*:  
 my\_prediction = np.dot*(*Parameters.T, X\_Train*[*i*])* error = Y\_Train*[*i*]* - my\_prediction  
 Parameters = Parameters + *(*2 \* *(*alpha \* error*))* return Parameters

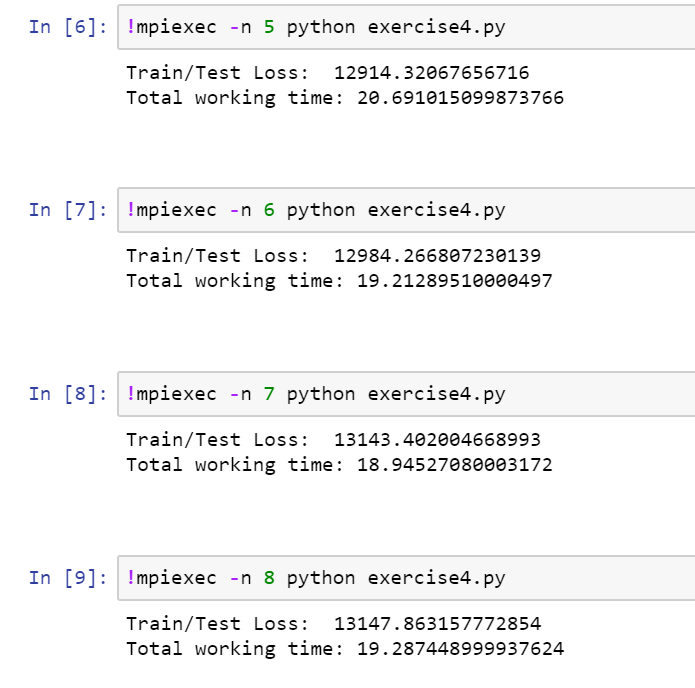
4. Master process then gathers the local parameters (**Beta**) of model from slave workers and averages them and broadcast the new global model parameters (**Beta**) to each worker for the next epoch. We also calculate RMSE loss on Test data using averaged model parameters.

# Master node gathering parameters from child processes  
All\_Local\_Parameters = comm.gather*(*Parameters, root=Master*)*if *(*rank == Master*)*:  
 if *(*size > 1*)*: # checking if there were multiple child processes  
 Global\_Parameters\_Mean = np.mean*(*All\_Local\_Parameters,  
 axis=0*)* # Master node calculate Global mean fo parameters  
 Loss.append*(*RMSE*(*X\_Test, Y\_Test, Global\_Parameters\_Mean*))* # calculating Loss  
 else:  
 Global\_Parameters\_Mean = All\_Local\_Parameters # If there was single process  
else:  
 Global\_Parameters\_Mean = None  
  
Global\_Parameters\_Mean = comm.bcast*(*Global\_Parameters\_Mean, root=Master*)* # sending average of parameters to child proceses for next epoch  
  
if len*(*Global\_Parameters\_Mean*)* > 0:  
 flag = True  
 Parameters = Global\_Parameters\_Mean  
Epoch = Epoch + 1

def RMSE*(*X, Y, Global\_Parameters\_Mean*)*: # function to calculate RMSE  
 Predictions = *[]* for i in range*(*len*(*X*))*:  
 Predictions.append*(*np.dot*(*X*[*i*]*, Global\_Parameters\_Mean.T*))* Final\_Predictions = np.reshape*(*np.array*(*Predictions*)*, -1*)* Error = sum*((*Y - Final\_Predictions*)* \*\* 2*)* return Error

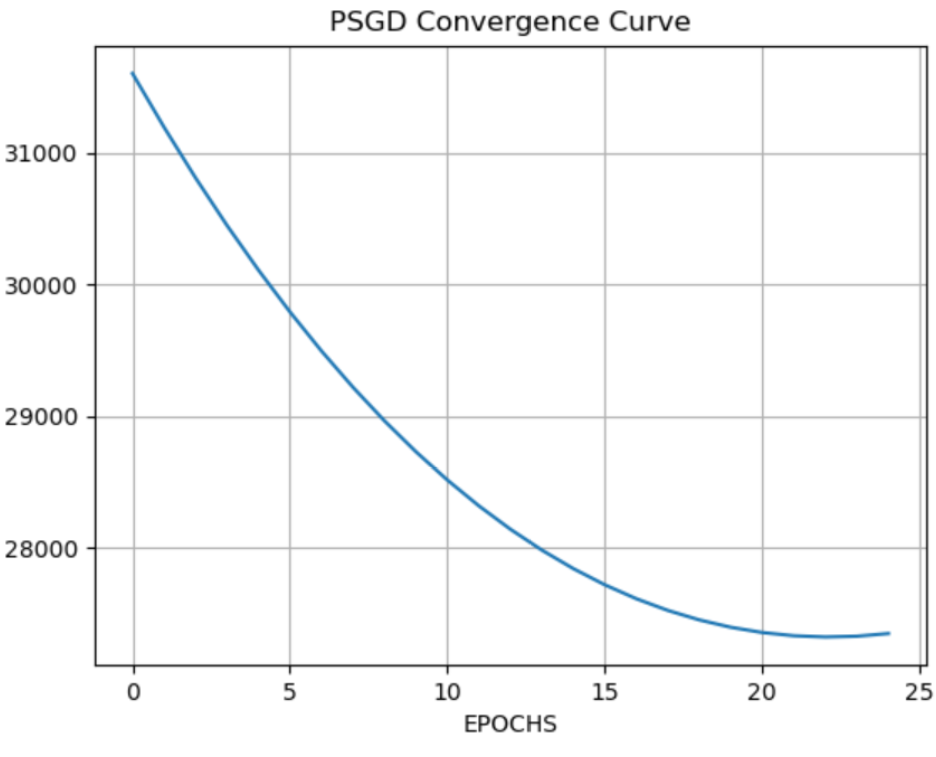
**Code working for arbitrary number of clusters and doing parallel processing:**

****

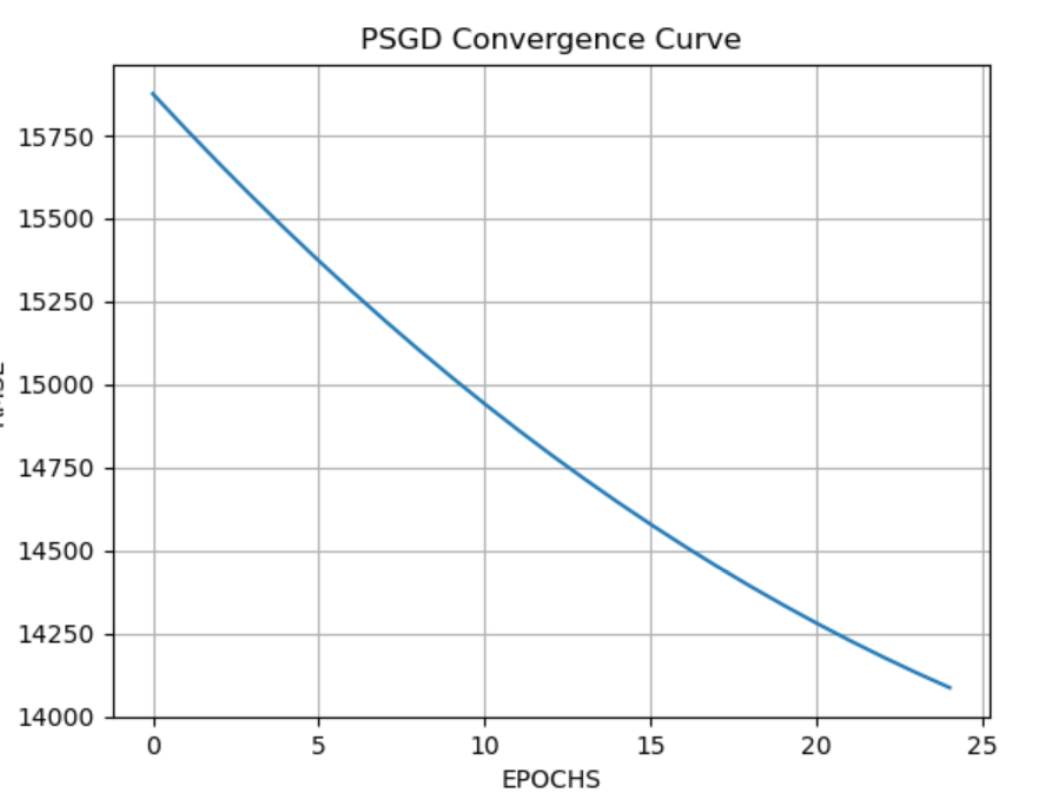
****

**Convergence Graphs:**

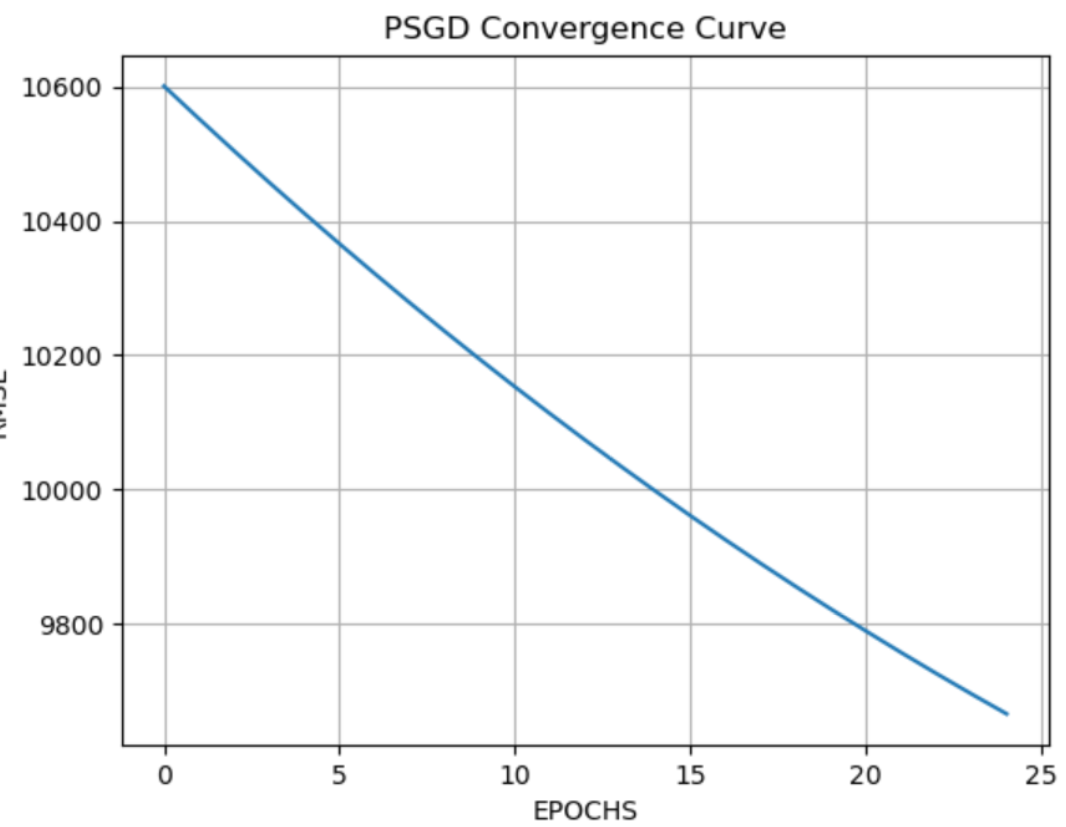
**P=1**



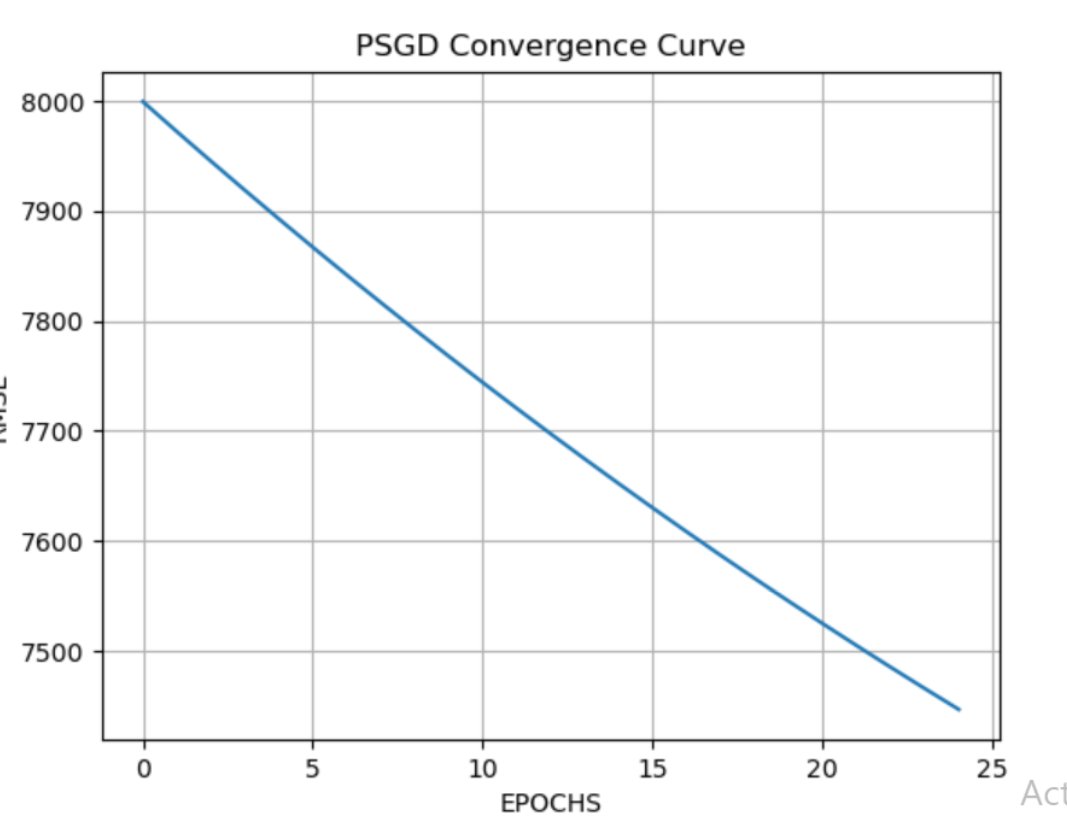
**P=2**

****

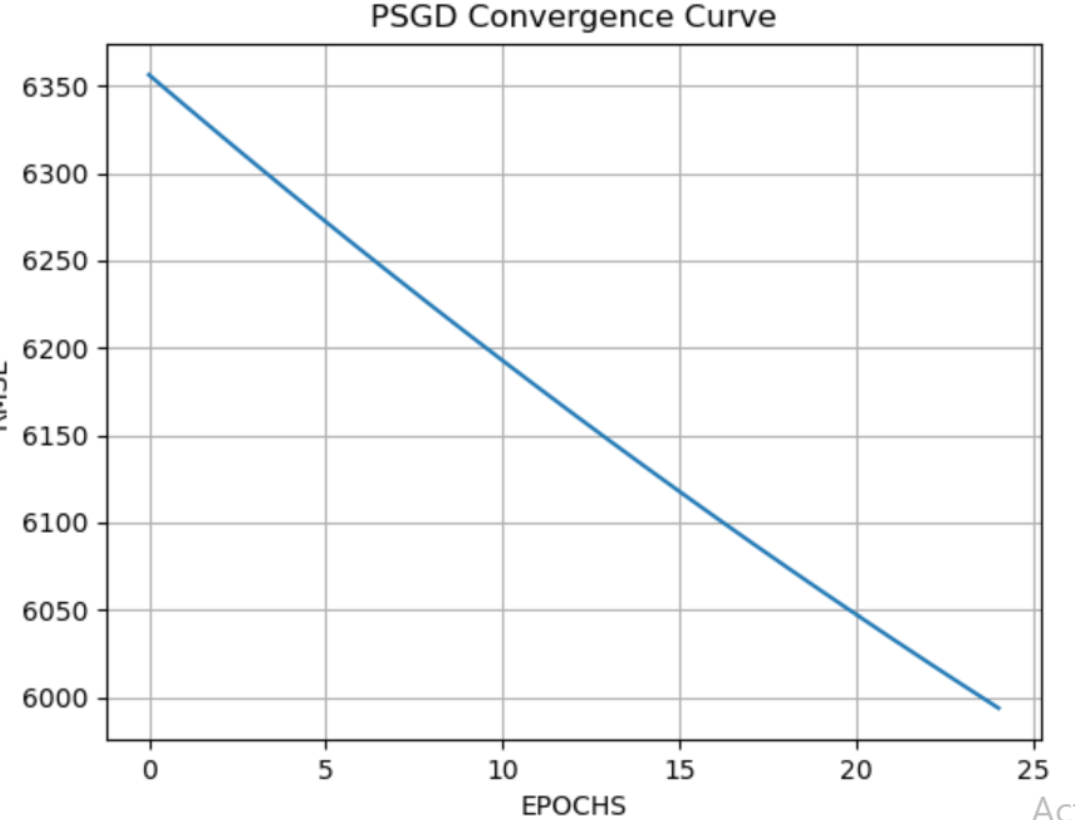
**P=3**

****

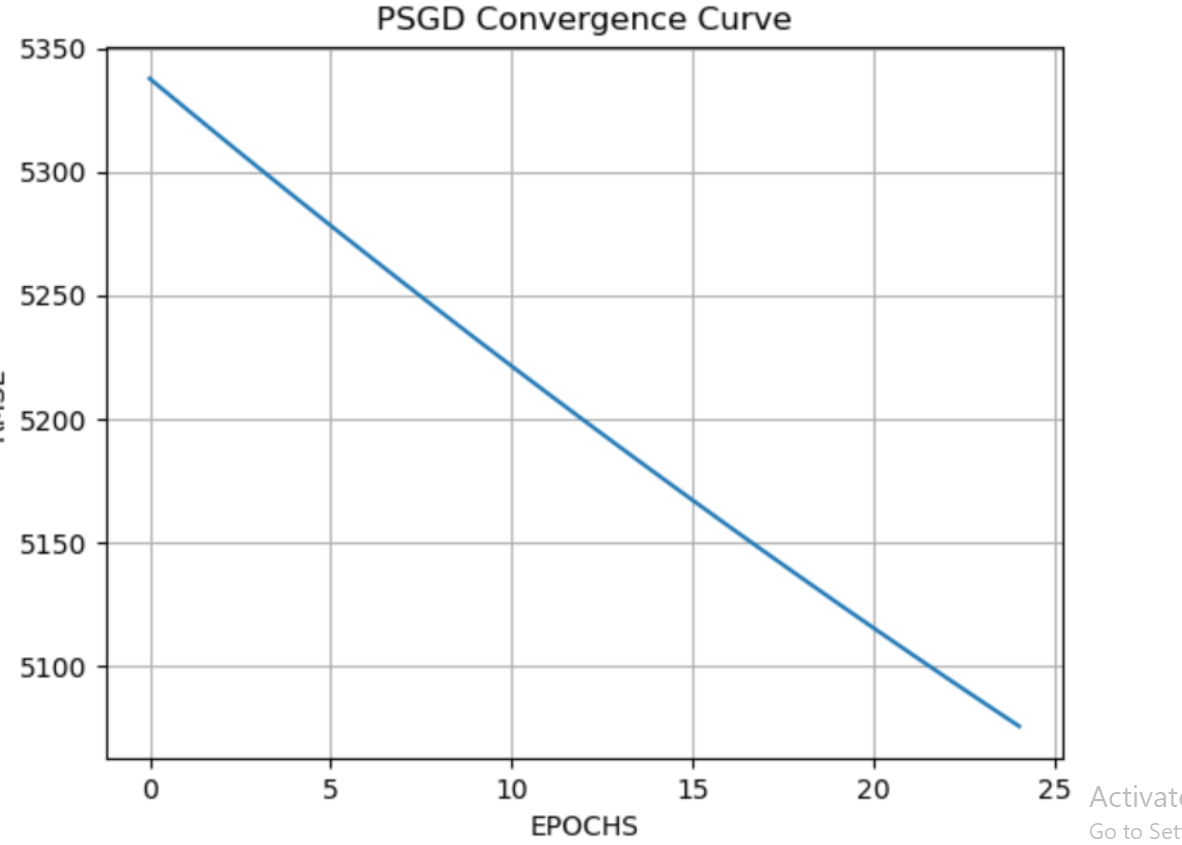
**P=4**

****

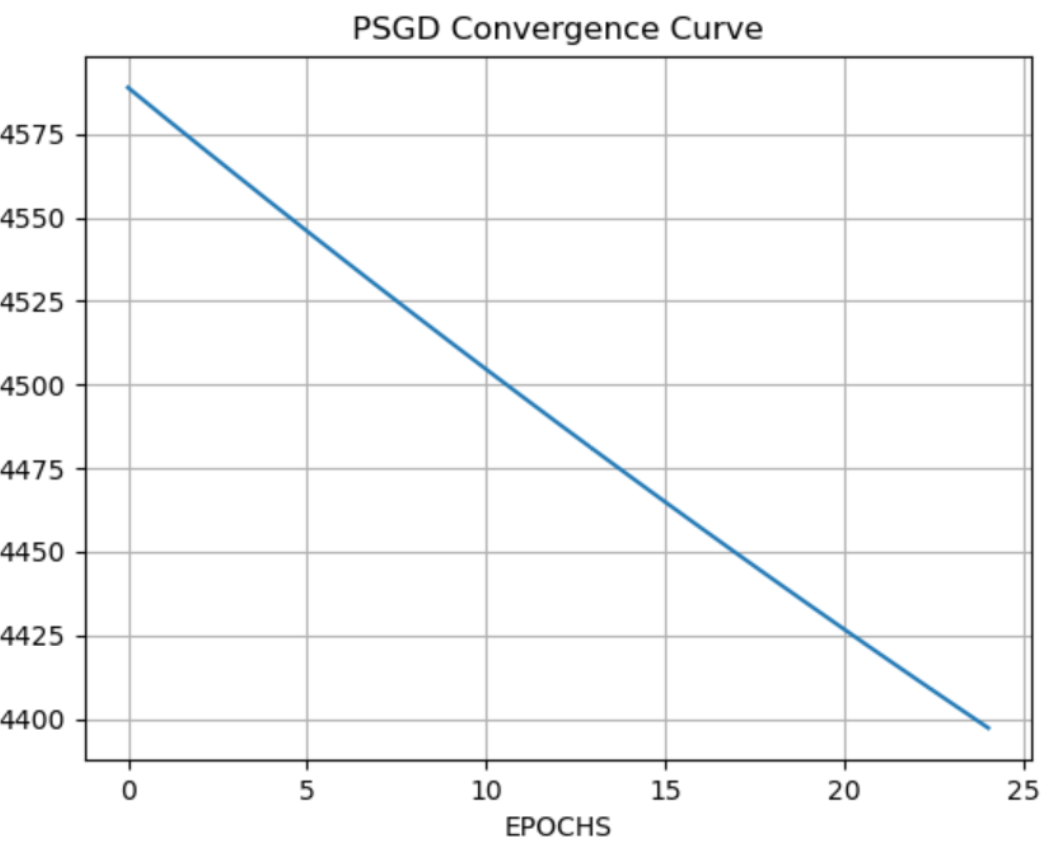
**P=5**

****

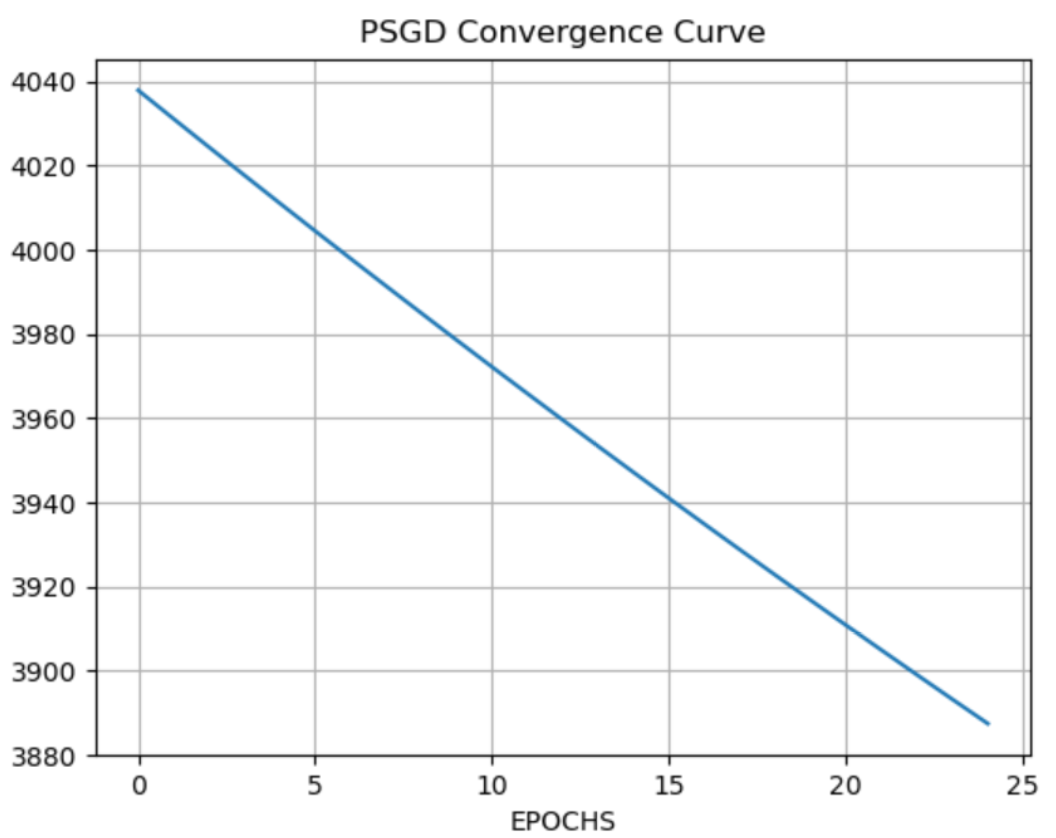
**P=6**

****

**P=7**

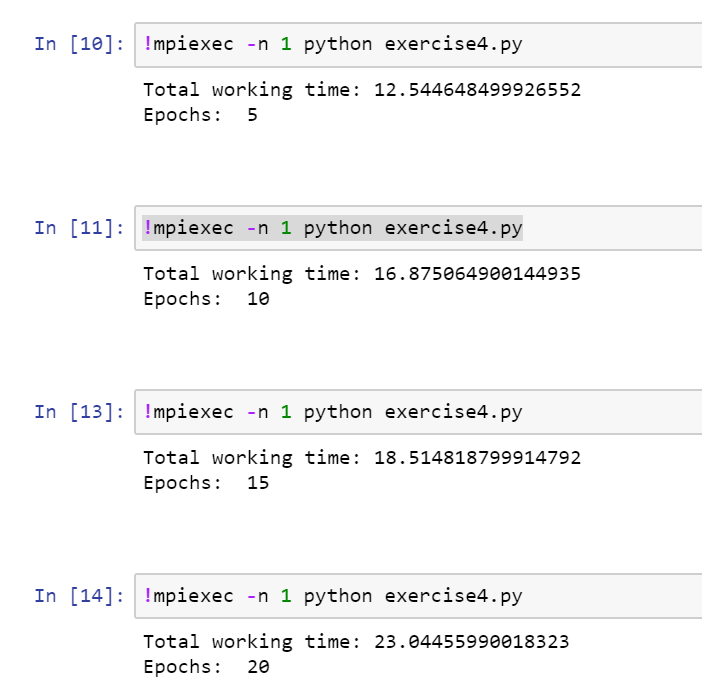
****

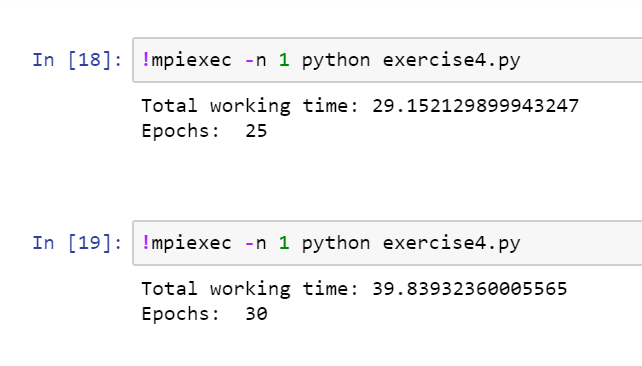
**P=8**

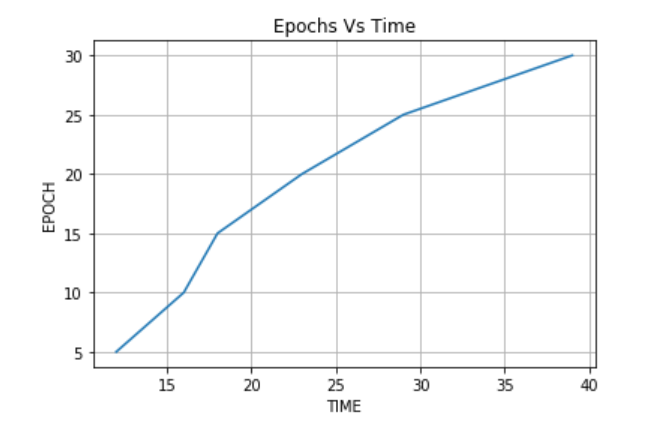
****

**Performance analysis by epochs vs time graph:**

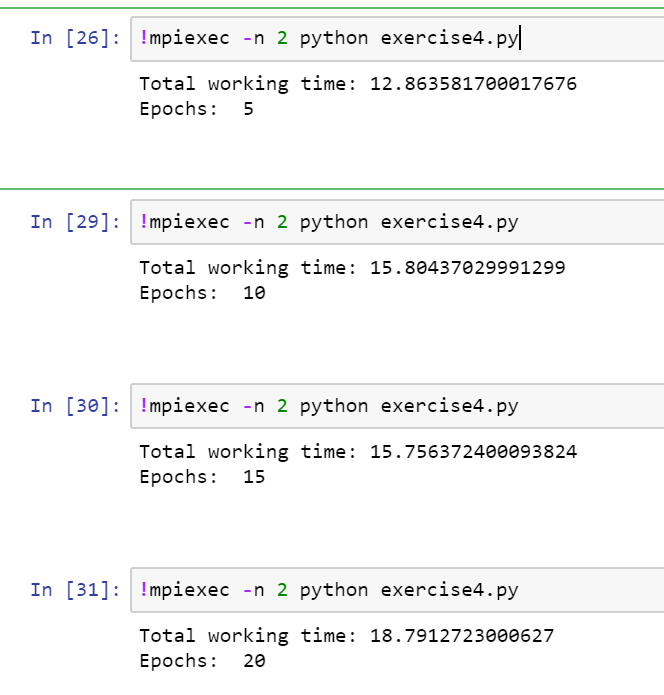
**P=1:**

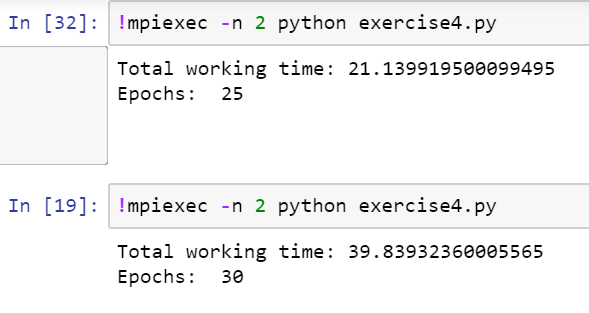
****

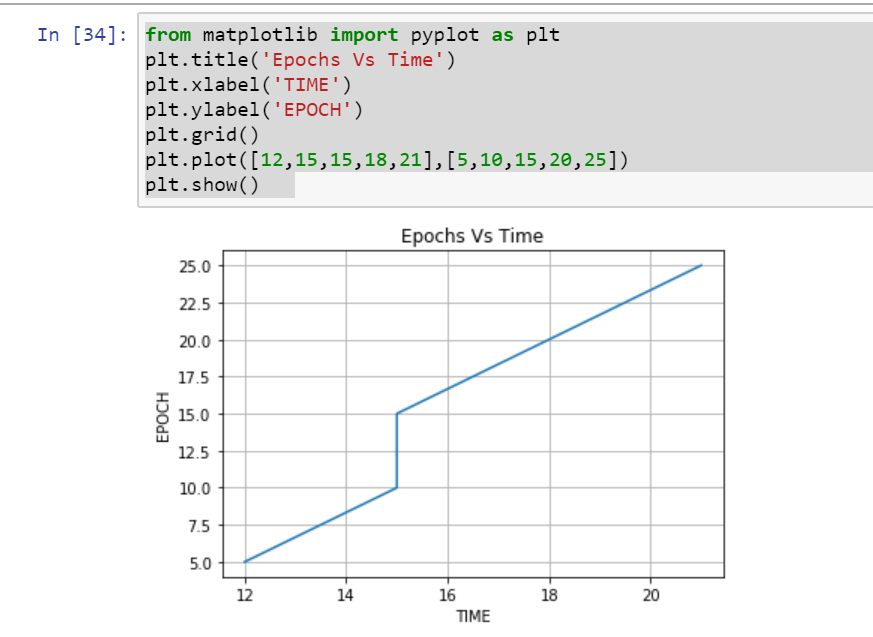
****

****

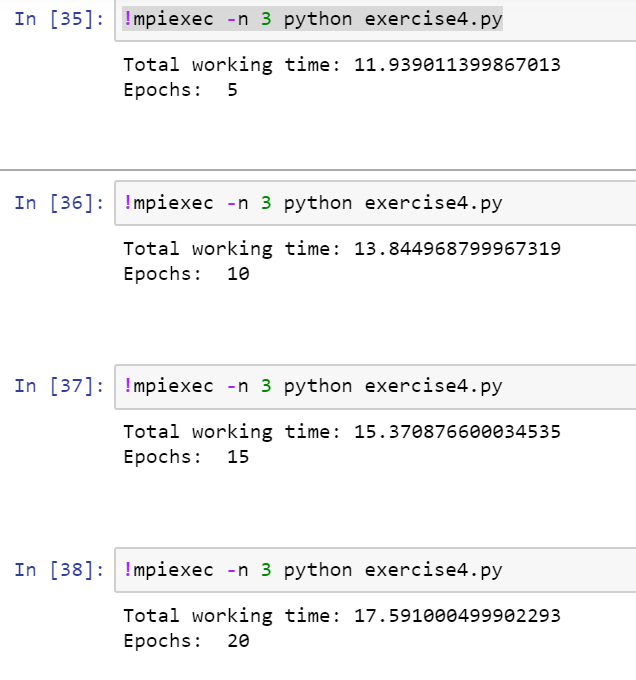
**P=2:**

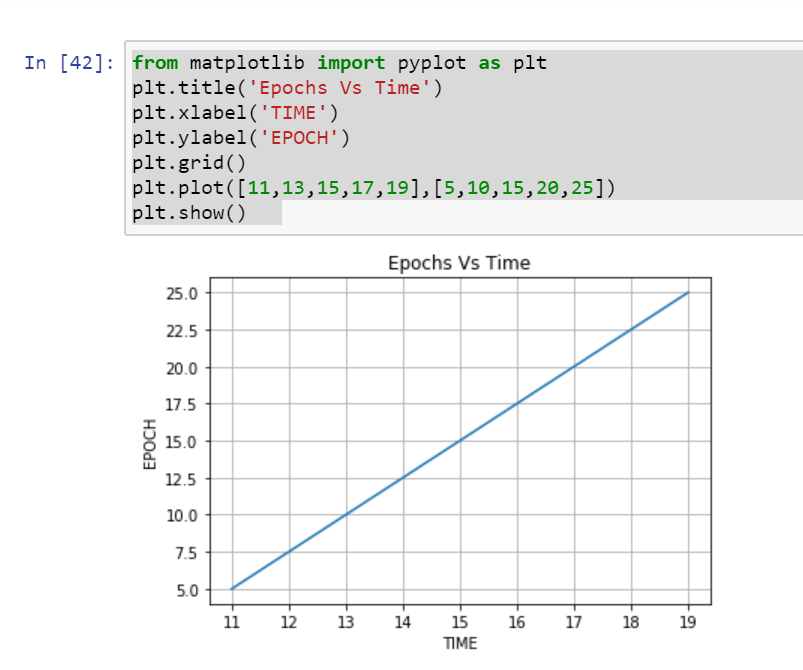
****

****

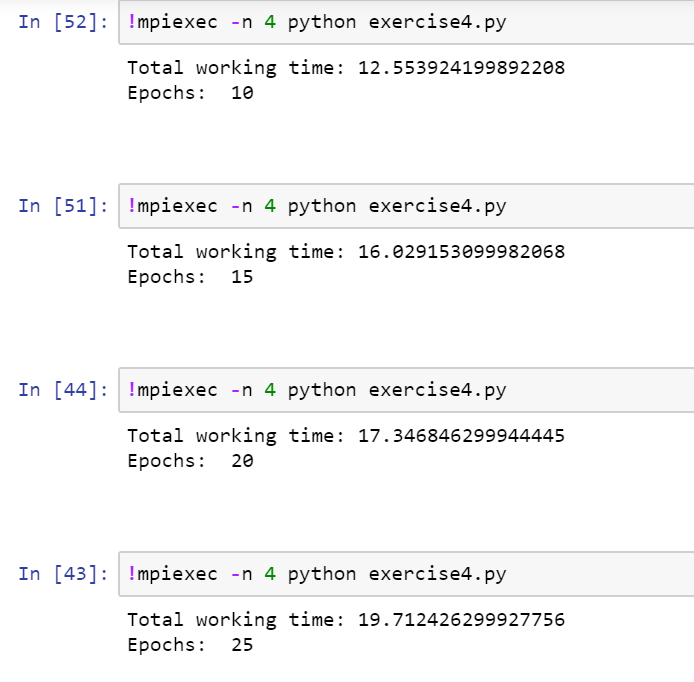
****

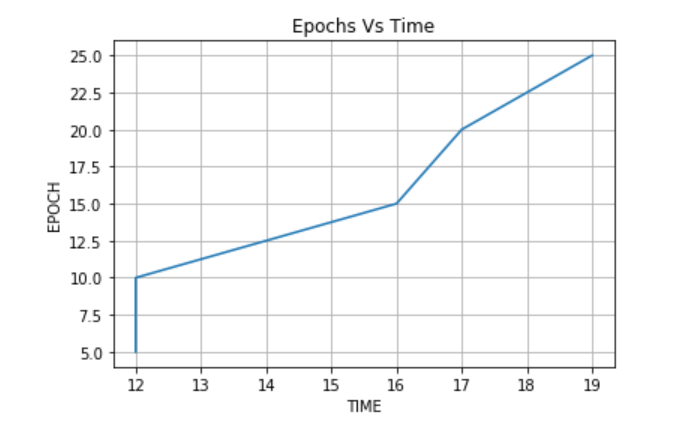
**P=3:**

****

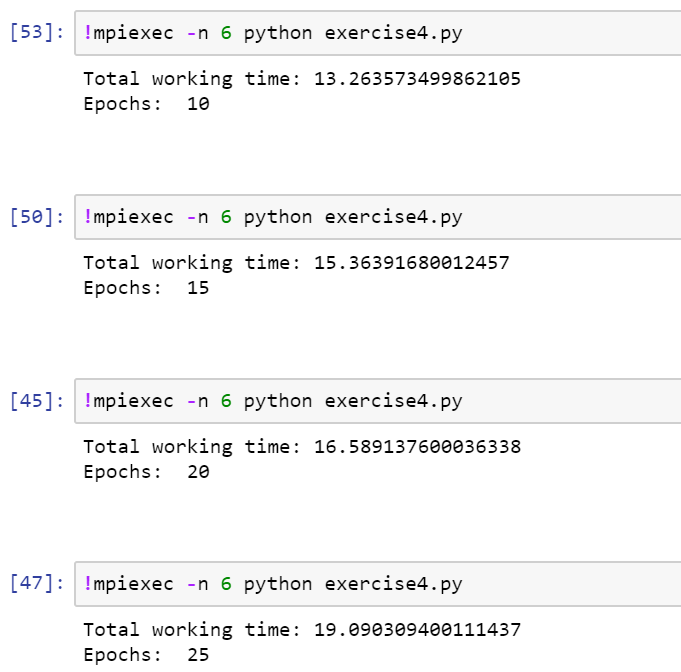
****

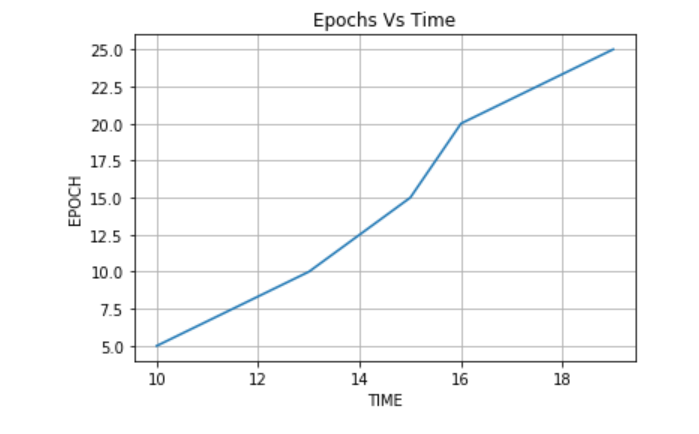
**P=4:**

****

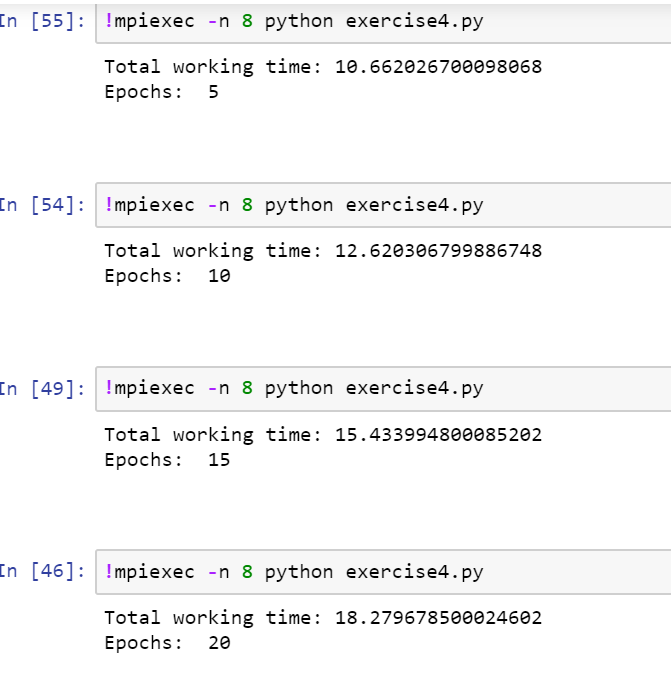
****

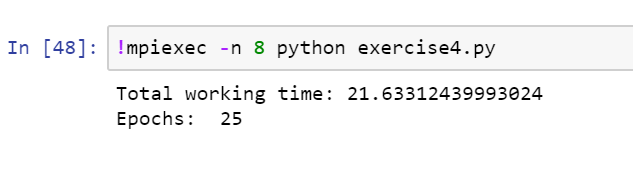
**P=6:**

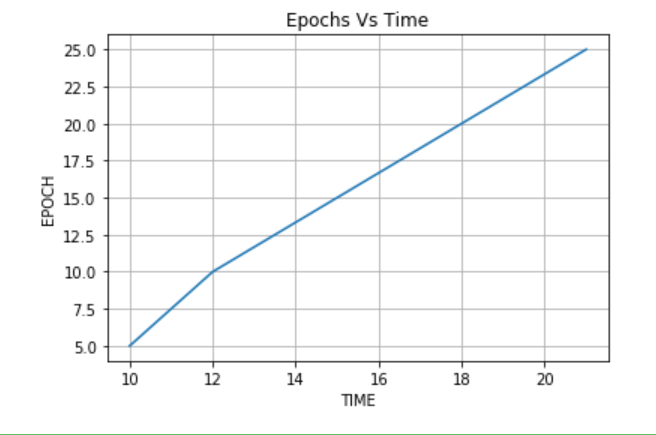
****

****

**P=8:**

****

****

****

**All these above graphs shows that more epochs per unit time can be achieved using a parallel parallel processing.**