Solution Ex: sheet 4

(By Shri Shalini Sekar)

Performance and convergence of PSGD

Parallel Version:

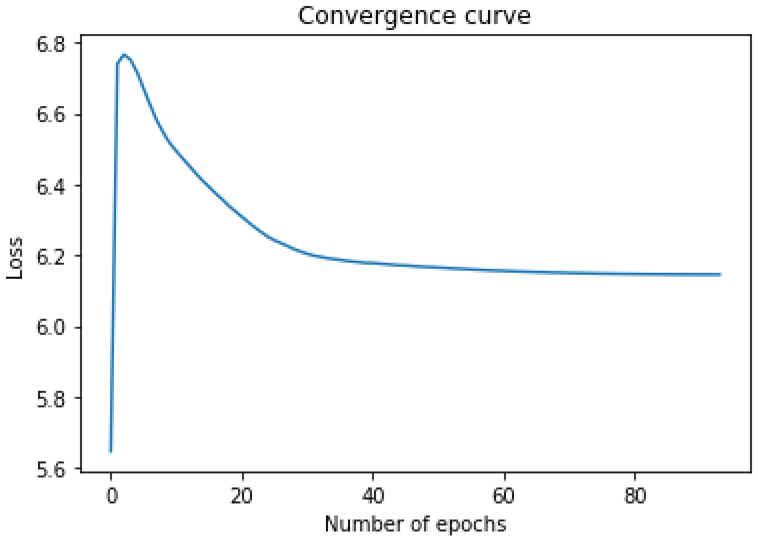
For number of processes = 8,

Time taken = 63.80923349992372

Number of epochs: 93

```
[5.646980297292352, 6.740464062938768, 6.766168187915817, 6.75152360739215,
6.716563503010813, 6.670180632762214, 6.623979591594544, 6.580227204208268,
6.546154049681842, 6.5172411229642195, 6.494634649301727, 6.473298837796135,
6.45242365256517, 6.431311845556447, 6.411214665484181, 6.393190109329815,
6.375101796719727, 6.358451522197263, 6.33997202734074, 6.324639943951855,
6.3089981365438605, 6.293759258315261, 6.278619842788125, 6.2651863972859445,
6.252386930042522, 6.24199678701919, 6.234207713160184, 6.225910370556843,
6.217105982884046, 6.210424213183161, 6.2040839092215405, 6.1987811984991446,
6.1954685948725885, 6.191698226076271, 6.189213750312316, 6.186827738447743,
6.183912778104477, 6.18208105167313, 6.1802083065148885, 6.178058844571696,
6.1779342155711845, 6.17608922936633, 6.1748090339220045, 6.17310719185453,
6.171449685730331, 6.17096832593964, 6.169508282879714, 6.1682414261390965,
6.166872855268927, 6.166431341244191, 6.165222616367872, 6.164188301702602,
6.163058199966849, 6.161960303318624,

6.161025320775041, 6.160004610672846, 6.159016084285834, 6.158060064050527,
6.157136865910141, 6.156246798858332, 6.155709982873777, 6.154913014441085,
6.154146810868609, 6.153603017663305, 6.152913823780921, 6.1522539915293315,
6.151878705077795, 6.151296658376957, 6.15074148483879, 6.15036281615407,
6.149871861463253, 6.149406250717426, 6.149095062587304, 6.14868816337429,
6.148305040810315, 6.148000026530408, 6.147770265443176, 6.147465492048662,
6.147182078084052, 6.147004145557878, 6.146805836494508, 6.146588947009276

, 6.146391030573614, 6.146304121943649, 6.1461448309093765, 6.1460021712506965,
6.145876168046199, 6.145786147027085, 6.145709262648734, 6.1456301750215125,
6.145566272689135,
```

6.14551756592142, 6.145484062552308, 6.1454657679735725]



For number of processes = 7,

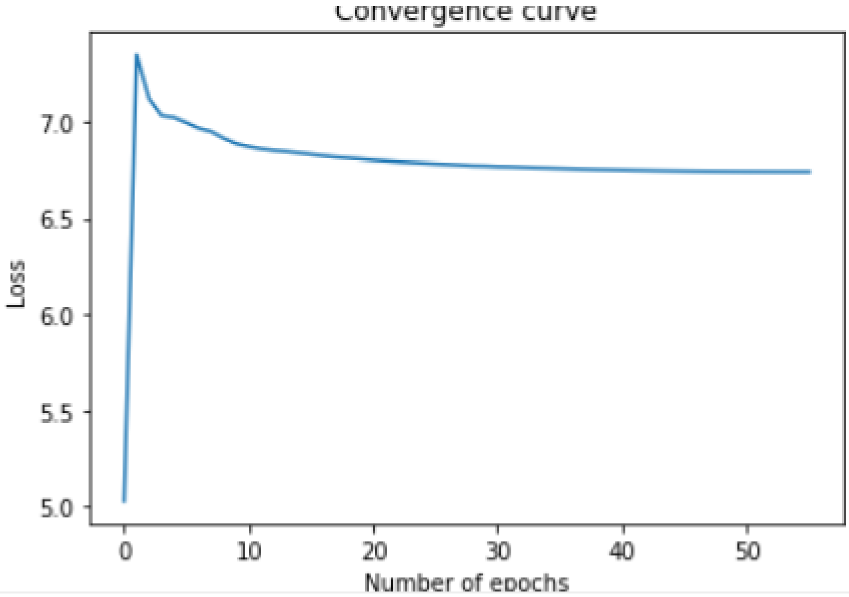
Time taken 48.503012700006366

Number of epochs = 55

epoch and costs associated

[5.030202679027441, 7.348886968769393, 7.11871145553583, 7.031584947511607, 7.021836588632796, 6.9941161200764554, 6.964600363977528, 6.9475732182423675, 6.91265898323451, 6.884527454154965, 6.8697802371171, 6.857854229868748, 6.85035754510302, 6.844545675903967, 6.8379284746313465, 6.830702396404406, 6.823405985877857, 6.816904847526152, 6.812091934387235, 6.8071158542338175, 6.801390991352567, 6.796938641436001, 6.792295701792934, 6.788648173821933, 6.785076529161629, 6.78155527680602, 6.778726250756865, 6.775527065944039, 6.772131572963846, 6.770891517846856, 6.768232843751999, 6.766852375489684, 6.764446666926227, 6.762006313651697, 6.760669654684888, 6.758879323442802, 6.757067159408842, 6.755232503095842, 6.753512284564597, 6.752276394627248, 6.7508122013115885, 6.749672443688785, 6.748740328916432, 6.747632310435597, 6.746793473196162

, 6.745895680661869, 6.745231719932319, 6.7445927406793835, 6.744093090132851, 6.743669199824952, 6.743276508874147, 6.743021702115192, 6.742741346461136, 6.742554929241885, 6.742431739476307, 6.742365251320805]6.742554929241885, 6.742431739476307, 6.742365251320805]



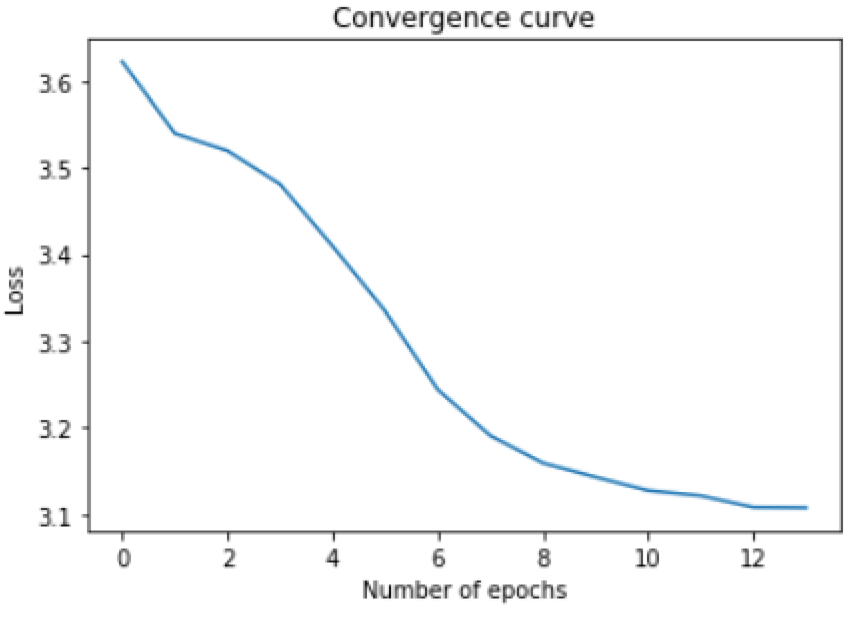
Convergence is better after the number of processes is less than 7

For Number of processes = 6,

time taken 10.407933499896899

Number of epochs = 4

epoch and costs associated [5.835449314065289, 6.409642969193518, 6.343774712805239, 6.167068297494485]

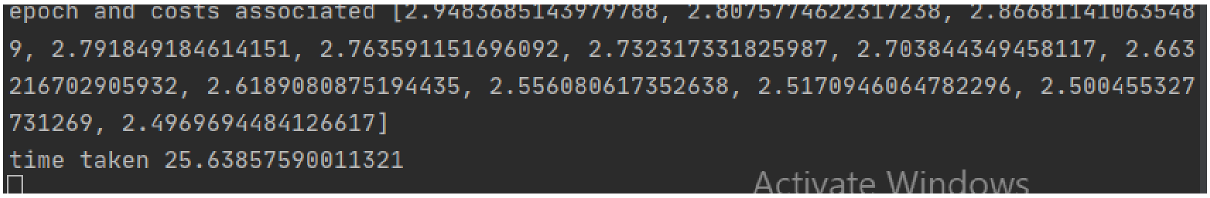


For number of processes = 5

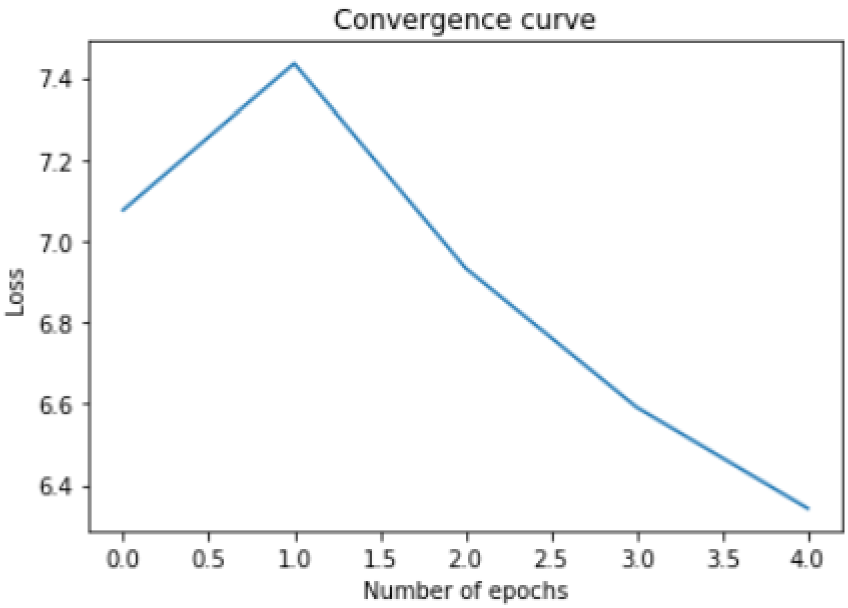
time taken 13.678639499936253

Number of epochs 5:

epoch and costs associated [7.076104253251086, 7.435453930656727, 6.9338547428957265, 6.591336904285984, 6.342413638027492]



```
epoch and costs associated [2.9483685143979788, 2.8075774622317238, 2.86681141063548
9, 2.791849184614151, 2.763591151696092, 2.732317331825987, 2.703844349458117, 2.663
216702905932, 2.6189080875194435, 2.556080617352638, 2.5170946064782296, 2.500455327
731269, 2.4969694484126617]
time taken 25.63857590011321
```

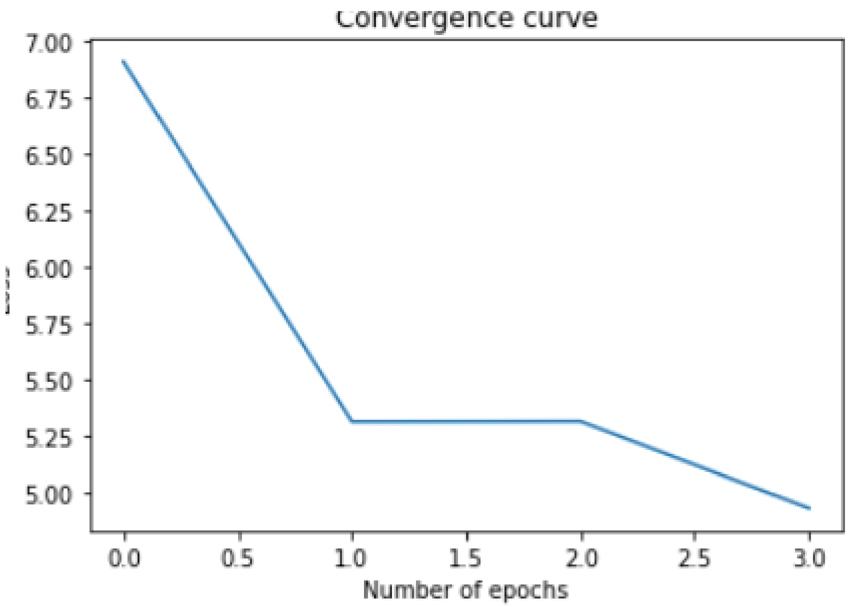


For number of processes = 4,

time taken 15.942531100008637

Number of epochs = 4

epoch and costs associated [6.908609167179904, 5.314706979637236, 5.315971505535835, 4.932973898721171]

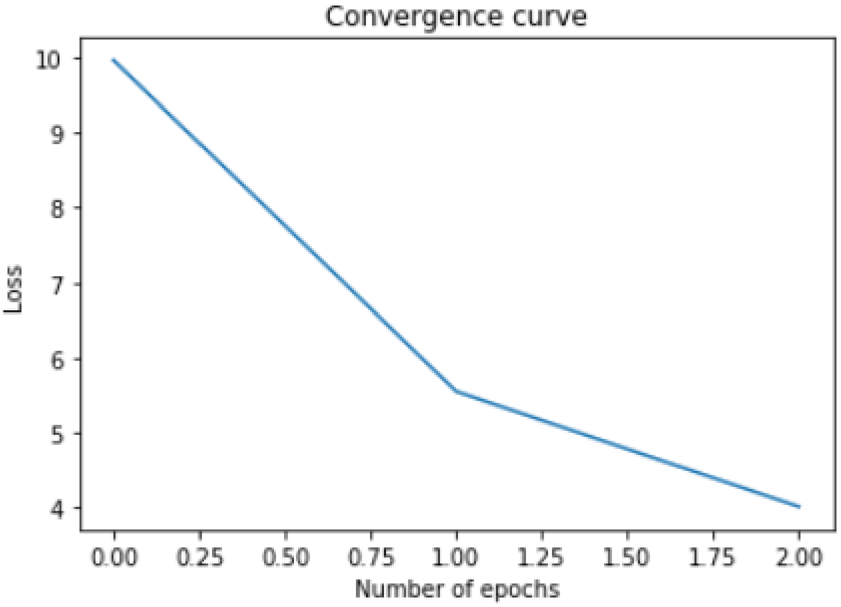


For number of processes = 3,

time taken time taken 24.53888399992138

epoch and costs [9.965371722747687, 5.551277214096245, 4.011283984773861]

number of epochs = 3



For number of processes = 2,

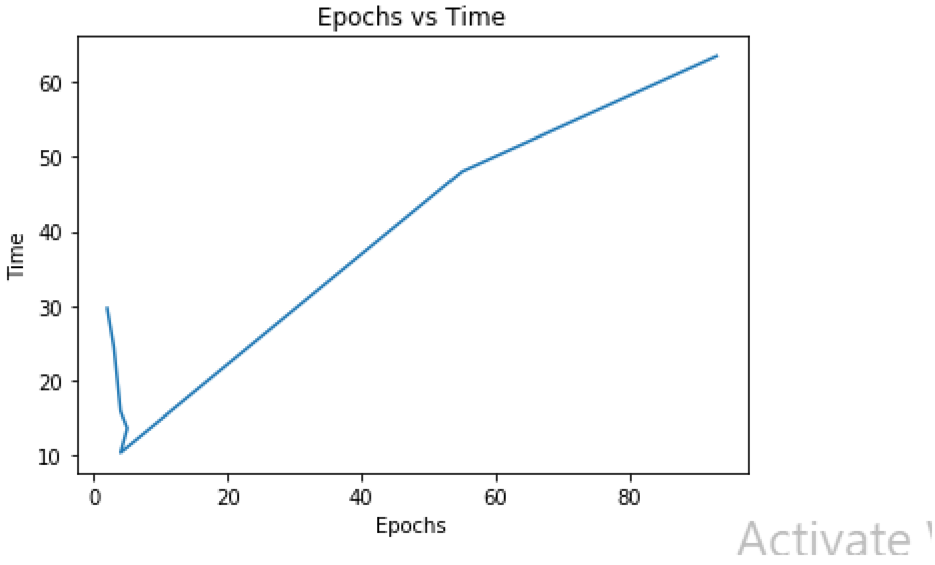
Number of epochs = 2

epoch and costs associated [15.232803497601855, 3.4267280751469187]

time taken = 29.691432799911126

Note: All the above results were obtained from executing PSGD on one of the data set files in "Dynamic Features of VirusShare Executables Data Set".

2. Epochs vs Time



3. Convergence and time performance improvement

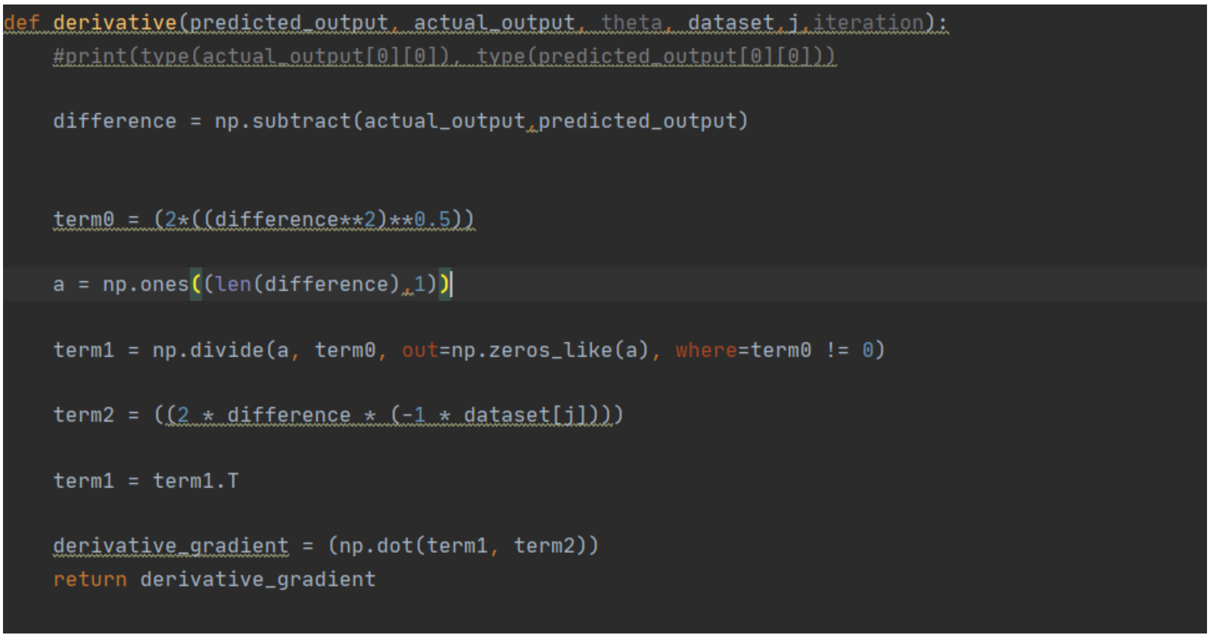
Time performance improvement for process 1 through 8:

1.321653600083664,29.691432799911126, 24.53888399992138, 15.942531100008637, 13.678639499936253, 10.407933499896899, 48.503012700006366, 63.80923349992372.

Time for serial execution using sklearn is the least. In manual implementation of PSGD, time constantly decreases until the number of processes is increased beyond 7

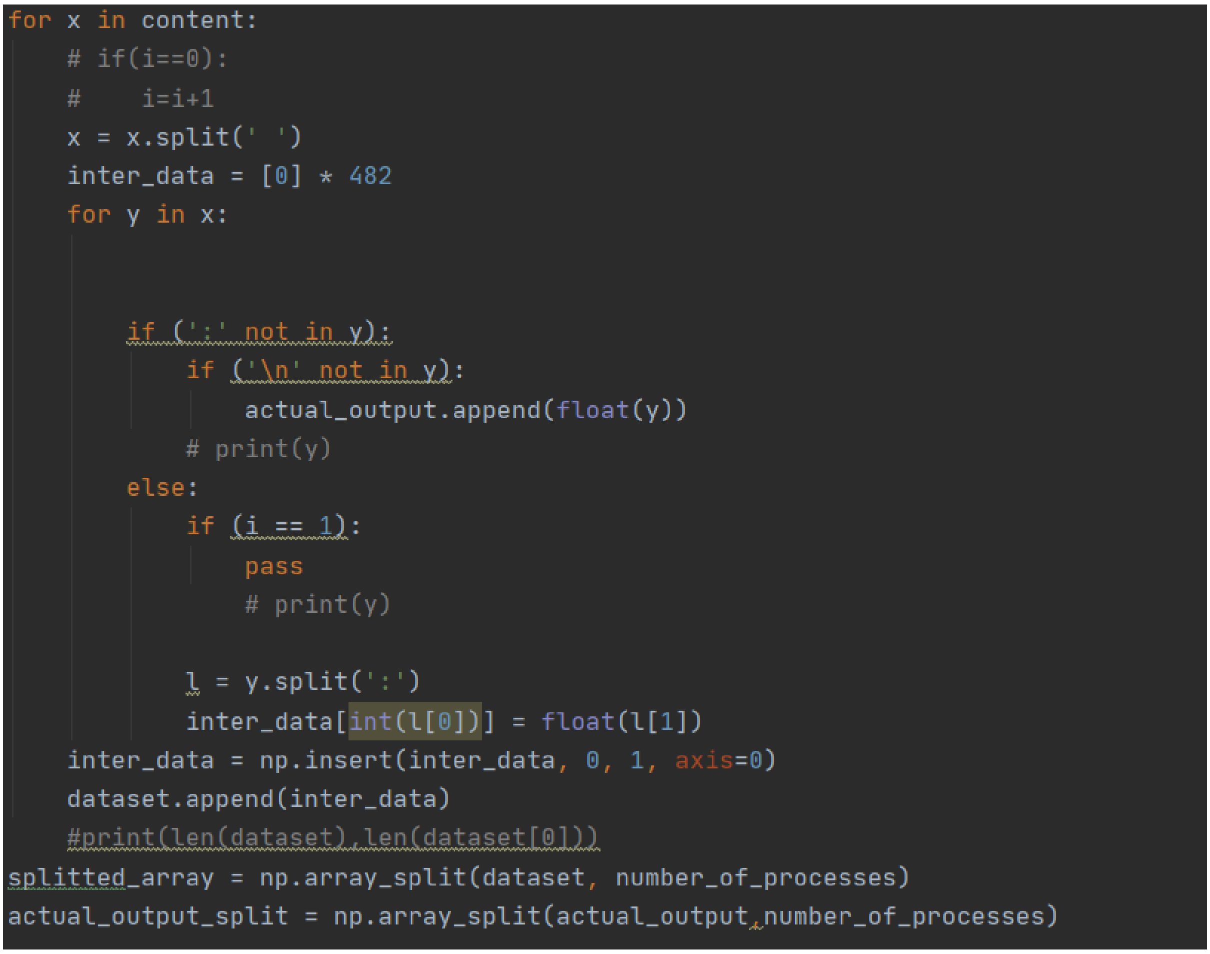
Code explanation:

Calculation of derivative:



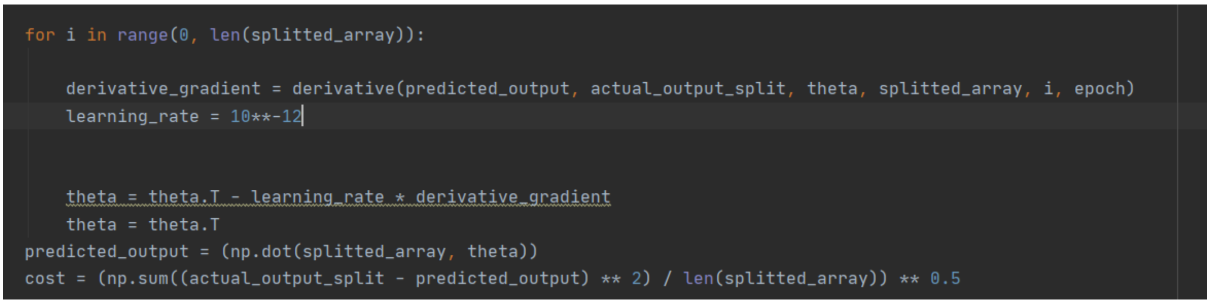
```
def derivative(predicted_output, actual_output, theta, dataset,j,iteration):
    #print(type(actual_output[0][0]), type(predicted_output[0][0]))

    difference = np.subtract(actual_output,predicted_output)


    term0 = (2*((difference**2)**0.5))

    a = np.ones((len(difference),1))

    term1 = np.divide(a, term0, out=np.zeros_like(a), where=term0 != 0)

    term2 = ((2 * difference * (-1 * dataset[j])))

    term1 = term1.T

    derivative_gradient = (np.dot(term1, term2))
    return derivative_gradient
```

Cleaning the input data:



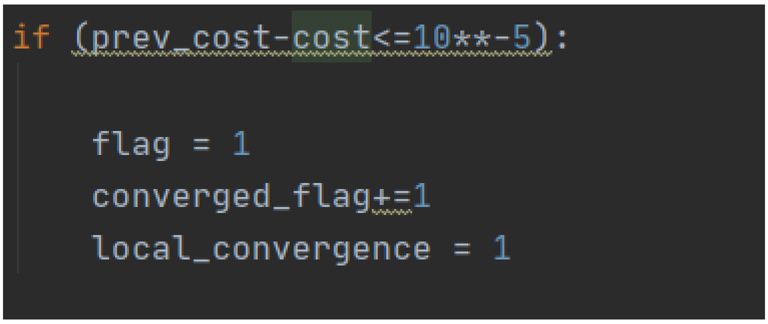
```
for x in content:
    # if(i==0):
    #    i=i+1
    x = x.split(' ')
    inter_data = [0] * 482
    for y in x:


        if (':' not in y):
            if ('\n' not in y):
                actual_output.append(float(y))
            # print(y)
        else:
            if (i == 1):
                pass
                # print(y)

            l = y.split(':')
            inter_data[int(l[0])] = float(l[1])
    inter_data = np.insert(inter_data, 0, 1, axis=0)
    dataset.append(inter_data)
    #print(len(dataset),len(dataset[0]))
splitted_array = np.array_split(dataset, number_of_processes)
actual_output_split = np.array_split(actual_output,number_of_processes)
```

Cost calculation:



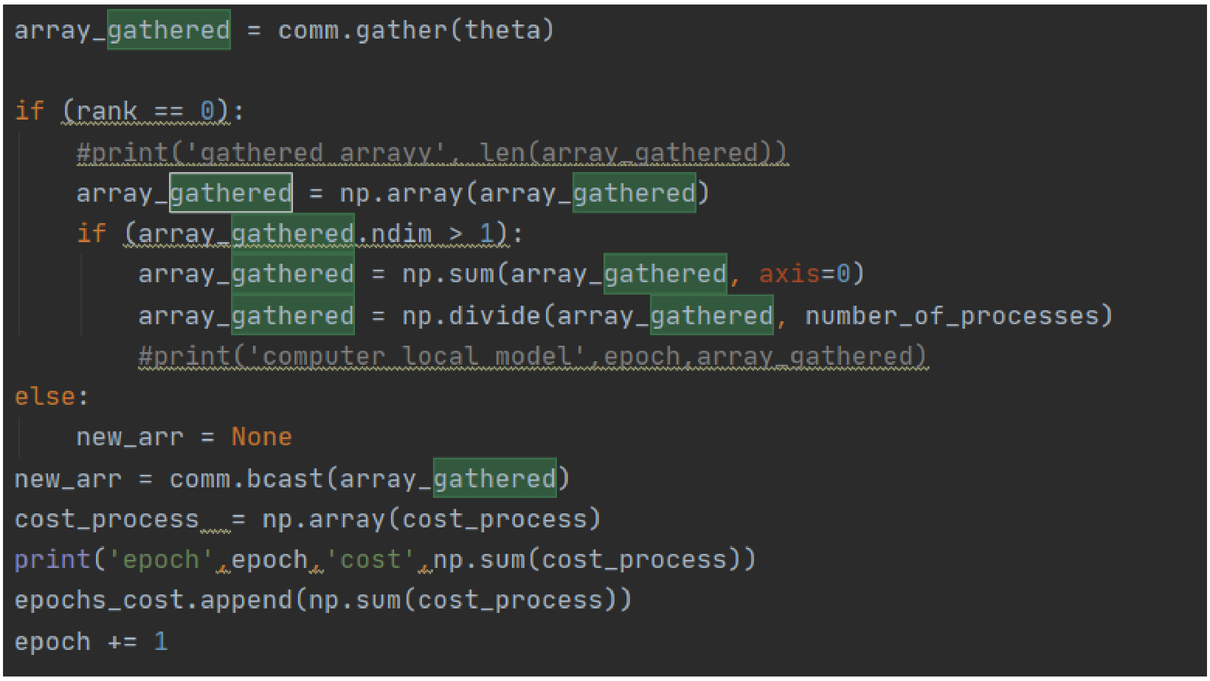
```
for i in range(0, len(splitted_array)):

    derivative_gradient = derivative(predicted_output, actual_output_split, theta, splitted_array, i, epoch)
    learning_rate = 10**-12


    theta = theta.T - learning_rate * derivative_gradient
    theta = theta.T
predicted_output = (np.dot(splitted_array, theta))
cost = (np.sum((actual_output_split - predicted_output) ** 2) / len(splitted_array)) ** 0.5
```

Convergence:



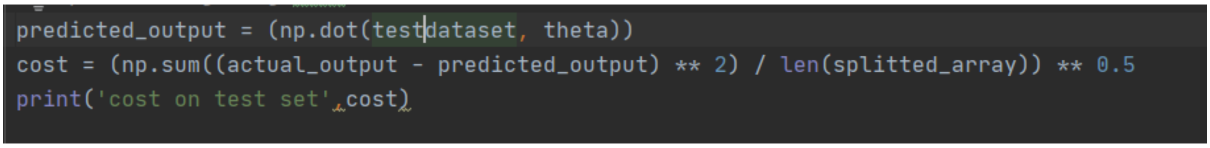
```
if (prev_cost-cost<=10**-5):

    flag = 1
    converged_flag+=1
    local_convergence = 1
```

Computing and distributing global theta from the local thetas:



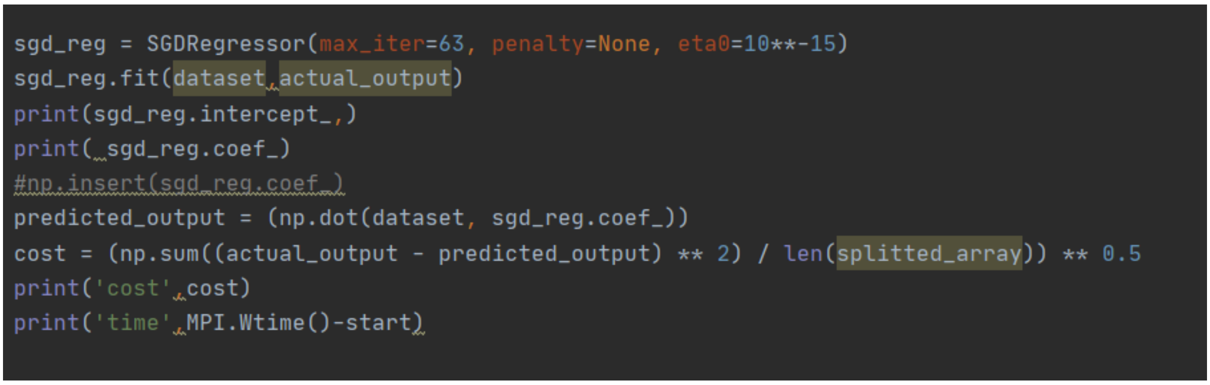
```
array_gathered = comm.gather(theta)

if (rank == 0):
    #print('gathered arrayy', len(array_gathered))
    array_gathered = np.array(array_gathered)
    if (array_gathered.ndim > 1):
        array_gathered = np.sum(array_gathered, axis=0)
        array_gathered = np.divide(array_gathered, number_of_processes)
        #print('computer local model',epoch,array_gathered)
else:
    new_arr = None
new_arr = comm.bcast(array_gathered)
cost_process  = np.array(cost_process)
print('epoch',epoch,'cost',np.sum(cost_process))
epochs_cost.append(np.sum(cost_process))
epoch += 1
```

Computing loss in test set:



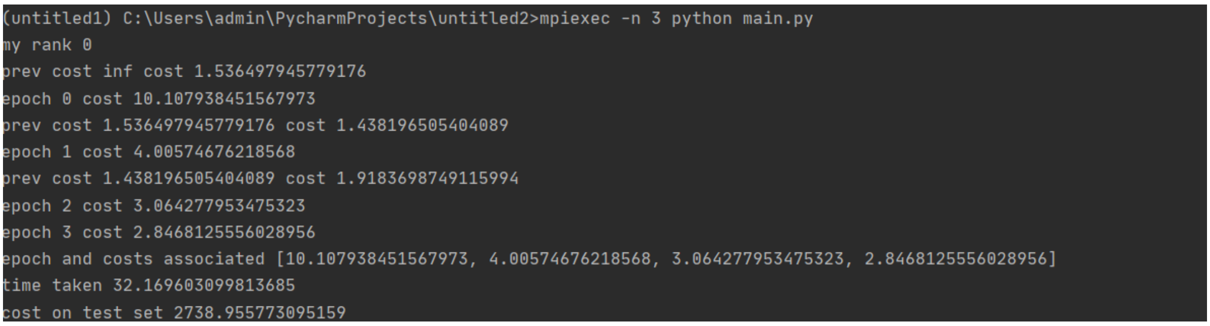
```
predicted_output = (np.dot(testdataset, theta))
cost = (np.sum((actual_output - predicted_output) ** 2) / len(splitted_array)) ** 0.5
print('cost on test set',cost)
```

For process size = 1, (using sklearn)



```
sgd_reg = SGDRegressor(max_iter=63, penalty=None, eta0=10**-15)
sgd_reg.fit(dataset,actual_output)
print(sgd_reg.intercept_,)
print( sgd_reg.coef_)
#np.insert(sgd_reg.coef_)
predicted_output = (np.dot(dataset, sgd_reg.coef_))
cost = (np.sum((actual_output - predicted_output) ** 2) / len(splitted_array)) ** 0.5
print('cost',cost)
print('time',MPI.Wtime()-start)
```

Outputs:



```
(untitled1) C:\Users\admin\PycharmProjects\untitled2>mpiexec -n 3 python main.py
my rank 0
prev cost inf cost 1.536497945779176
epoch 0 cost 10.107938451567973
prev cost 1.536497945779176 cost 1.438196505404089
epoch 1 cost 4.00574676218568
prev cost 1.438196505404089 cost 1.9183698749115994
epoch 2 cost 3.064277953475323
epoch 3 cost 2.8468125556028956
epoch and costs associated [10.107938451567973, 4.00574676218568, 3.064277953475323, 2.8468125556028956]
time taken 32.169603099813685
cost on test set 2738.955773095159
```