Q0

Below is the part print by help(PR.readData)

```
Help on function readData in module ParallelRegression:

readData(input_file, spark_context)
    Read data from an input file and return rdd containing pairs of the form:
                    (x,y)
    where x is a numpy array and y is a real value. The input file should be a
    'comma separated values' (csv) file: each line of the file should contain x
    followed by y. For example, line:

    1.0,2.1,3.1,4.5

    should be converted to tuple:

    (array(1.0,2.1,3.1),4.5)
(END)
```

Below is the part print by help(PR.f)

```
Help on function f in module ParallelRegression:

f(x, y, beta)
    Given vector x containing features, true label y,
    and parameter vector β, return the square error:

                f(β;x,y) =  (y - <x,β>)^2
(END)
```

These two parts is printed because they are between """…""" symbol under the function. And they will show when you type help

Below is one example of data in small.test. It is a set of numbers.

```
-0.0343076906387,-0.355072005669,-1.50352370378,-0.766225649675,-0.950713977291,1.05380419
197,-0.593194944805,-0.221718561863,-1.18912785271,-1.18774512899,2.56807110139,-0.8183288
90648,-0.662781516112,0.804097395871,0.227146264491,0.273034496873,-0.153341503055,0.55510
4682793,1.45425053511,-1.07126482738,-0.415101772764,0.0256166803652,0.105443209612,-1.209
26021244,-0.513497376729,1.18796367738,-0.264986918179,1.02635442726,-0.128253179206,-0.78
1220372001,0.0788227181649,1.09290930278,-0.0788870918038,1.79035517811,0.316103552002,-0.
0760382164356,0.0432155434594,-1.91082653798,0.215358030849,0.108923535945,-0.35114704108,
-0.75703747461,-1.02636455831,-0.0443045257197,-0.697358952359,0.888956723802,-0.476215861
246,0.768075213631,-1.71837588394,1.18811941685,5.69452261777
```

Below is one example of data in the RDD. It's a tuple with the form (x,y). It contains all the value except the last one as x and the last one as y

```
(array([-1.54605169, -0.44050394,  0.19322609,  0.07992739, -2.04398014,
        -0.18226367,  0.33157205,  0.13665696,  0.41624648, -0.18984357,
         0.25098238,  0.33334594,  1.91983396, -0.86581275,  1.09553347,
        -0.33123564, -0.94193872,  0.97487995,  2.26185742,  0.62938571,
        -0.69326367,  1.36792905,  1.11236515, -1.85539873,  0.85540653,
        -0.05805556, -1.06576905, -1.01761118,  0.68970422,  0.16244304,
        -0.28409818, -1.15820351,  0.44716698, -0.9948843 ,  0.42511979,
        -0.41264189,  0.29684205,  0.46531902,  0.30852392,  0.44002312,
        -0.16256279, -0.29633339,  1.17969165, -0.7898195 , -0.69792189,
        -1.29640209,  0.32377171, -1.04674736,  0.81126137,  0.25167934]), 27.2647885495)
```

Q1

a) Below is my code for predict

```
def predict(x,beta):
    """ Given vector x containing features and parameter vector β
        return the predicted value:

                        y = <x,β>

    """
    return np.dot(x,beta)
```

b) Below is my executing of predict

```
>>> import ParallelRegression as PR
>>> x = np.array([np.cos(t) for t in range(5)])
>>> beta = np.array([np.sin(t) for t in range (5)])
>>> print PR.predict(x,beta)
0.431218839971
```

Q2

a) $f(\beta; x, y) = (y - \beta^T x)^2$

$= (y - \sum \beta_i x_i)^2$

$$\nabla f(\beta; x, y) = \frac{\partial (y - \sum \beta_i x_i)^2}{\partial \beta}$$

$$= 2 * \left(y - \sum \beta_i x_i\right) * \left(\frac{\partial (-\sum \beta_i x_i)}{\partial \beta}\right)$$

$$= -2 * (y - \beta^T x) * x$$

b) Below is my code for function f

```
def f(x,y,beta):
    """ Given vector x containing features, true label y,
        and parameter vector β, return the square error:

                f(β;x,y) =  (y - <x,β>)^2

    """
    return np.square(y - np.dot(x, beta))
```

c) Below is my code for localGradient

```
def localGradient(x,y,beta):
    """ Given vector x containing features, true label y,
        and parameter vector β, return the gradient ∇f of f:

                ∇f(β;x,y) =  -2 * (y - <x,β>) * x

        with respect to parameter vector β.

        The return value is  ∇f.
    """
    inside = y - np.dot(x, beta)
    return -2.0 * inside *x
```

d) Below is the execution of localGradient and combine of f with estimateGrad

```
>>> y= 1.0
>>> x= np.array([np.cos(t) for t in range (5)])
>>> beta = np.array([np.sin(t) for t in range (5)])
>>> result1 = PR.estimateGrad(lambda beta :PR.f(x,y,beta),beta,0.000001)
>>> result = PR.localGradient(x,y,beta)
>>> result
array([-1.13756232, -0.61462754,  0.47339296,  1.12617816,  0.74356035])
>>> result1
array([-1.13756132, -0.61462725,  0.47339313,  1.12617914,  0.74356078])
>>>
```

As you can see the result is basically the same. So, that the localGradient is correct since the correctness already been tested by localGradient agrees with the estimate produced by estimateGrad When δ is small.

Q3

a) Below is my code for function F

```
def F(data,beta,lam = 0):
    """  Compute the regularized mean square error:

            F(β) = 1/n Σ_{(x,y) in data}    f(β;x,y)  + λ ||β ||_2^2
                 = 1/n Σ_{(x,y) in data} (y- <x,β>)^2 + λ ||β ||_2^2

         where n is the number of (x,y) pairs in RDD data.

         Inputs are:
            - data: an RDD containing pairs of the form (x,y)
            - beta: vector β
            - lam:  the regularization parameter λ


         The return value is F(β).

    """
    beta2 = np.sum(np.square(beta))
    sum, n = data.map(lambda (x,y): (f(x, y ,beta),1))\
                 .reduce(lambda x, y: (x[0] + y[0], x[1] + y[1]))
    return (1.0 / n) * sum + lam * beta2
                                                                    142,5
```

b) Below is my code for function gradient

```
def gradient(data,beta,lam = 0):
    """ Compute the gradient  ∇F of the regularized mean square error
                F(β) = 1/n Σ_{(x,y) in data} f(β;x,y) + λ ||β ||_2^2
                     = 1/n Σ_{(x,y) in data} (y- <x,β>)^2 + λ ||β ||_2^2

        where n is the number of (x,y) pairs in data.

        Inputs are:
             - data: an RDD containing pairs of the form (x,y)
             - beta: vector β
             - lam:  the regularization parameter λ


        The return value is an array containing ∇F.

    """
    gradSum,n = data.map(lambda(x,y): (localGradient(x,y,beta),1))\
                    .reduce(lambda x, y: (x[0] + y[0], x[1] + y[1]))
    return (1.0 / n) * gradSum + lam * 2 * beta
```

c) Below is my execucation for gradient and combine of estimateGrad with F

```
>>> import ParallelRegression as PR
>>> import numpy as np
>>> lam = 1.0
>>> beta = np.array([np.sin(t) for t in range (50)])
>>> data = PR.readData("data/small.test",sc)
```

```
>>> result = PR.gradient(data,beta,lam)
```

```
>>> result1 = PR.estimateGrad(lambda beta: PR.F(data,beta,lam),beta,0.0000001)
```

```
>>> result
array([ 4.39945352,  2.86050932,  4.50982604, -1.85170994, -6.19540742,
       -3.84463142, -2.80309467,  3.58914049,  5.23949536,  3.20436283,
       -1.13257134, -4.00822426, -3.97652387,  1.62077657,  8.53774366,
        3.76708615, -3.56305719, -6.7752212 , -7.21234235,  2.42757438,
        2.0236413 ,  5.2434361 , -1.23645832, -1.93962279, -4.89414425,
        0.33872292,  2.67334964,  5.74150933,  3.16590298, -0.2020467 ,
       -1.27129553, -1.11336197,  2.52013787,  1.58421622, -0.24287307,
       -2.71253267, -1.66387657, -1.93504448,  6.16984068,  2.62113744,
        5.2980497 ,  2.08366788, -6.20624771,  0.16714202, -0.13615823,
        2.54079899, -0.05016347,  1.36447249, -5.48897702, -1.47100687])
>>> result1
array([ 4.39945325,  2.8605092 ,  4.50982498, -1.85171075, -6.19540742,
       -3.84463135, -2.8030945 ,  3.58914178,  5.23949552,  3.204363  ,
       -1.13257101, -4.00822387, -3.976524  ,  1.62077697,  8.53774281,
        3.76708499, -3.56305804, -6.77522166, -7.21234244,  2.42757437,
        2.02364106,  5.24343591, -1.23645805, -1.93962251, -4.89414447,
        0.33872311,  2.67335054,  5.74150818,  3.1659016 , -0.20204823,
       -1.27129567, -1.11336249,  2.52013706,  1.58421642, -0.24287431,
       -2.71253271, -1.66387736, -1.9350432 ,  6.16984096,  2.62113701,
        5.29804993,  2.08366771, -6.20624803,  0.16714296, -0.13615818,
        2.5407985 , -0.05016318,  1.36447284, -5.48897674, -1.47100707])
```

As you can see result. Result and Result 1 are basically the same. Which can prove my correctness of gradient is the gradient of function F.

Q4

a) Below is my code for test

```
def test(data,beta):
    """ Compute the mean square error

                MSE(β) =  1/n Σ_{(x,y) in data} (y- <x,β>)^2

        of parameter vector β over the dataset contained in RDD data, where n is the size
of RDD data.

        Inputs are:
             - data: an RDD containing pairs of the form (x,y)
             - beta: vector β

        The return value is MSE(β).

    """
    return F(data,beta,0)
```

b) Below is my code for train

```
    tIni = time.time()
    beta = 1.0 *beta_0
    for k in range(max_iter):
        PresentFValue = F (data, beta, lam)
        GradF = gradient(data, beta, lam)
        normGradF = np.sqrt(np.sum(np.square(GradF)))
        if normGradF < eps:
            return beta,normGradF,k
        tFinal = time.time()
        print'Iteration Time', k+1, 'time',tFinal - tIni,'Function Value', PresentFValue,
'Norm', normGradF
        gamma = lineSearch(lambda x: F(data,x,lam), beta, GradF)
        beta = beta - gamma * GradF * 1.0
    return beta, normGradF,k
```

c) Below is the execution

```
Reading training data from data/small.train
Training on data from data/small.train with λ = 0.0 , ε = 0.01 , max iter =  100
Iteration Time 1 time 0.0956380367279 Function Value 220.564648376 Norm 10.7326121454
Iteration Time 2 time 0.298840045929 Function Value 196.950858491 Norm 5.21787375028
Iteration Time 3 time 0.53595495224 Function Value 192.202962818 Norm 0.981922719761
Iteration Time 4 time 0.70805311203 Function Value 192.030079227 Norm 0.695480877873
Iteration Time 5 time 0.912353992462 Function Value 191.9525186 Norm 0.168629000552
Iteration Time 6 time 1.06058192253 Function Value 191.947034509 Norm 0.126634425216
Iteration Time 7 time 1.2359559536 Function Value 191.944461359 Norm 0.0342334207718
Iteration Time 8 time 1.36379003525 Function Value 191.944201927 Norm 0.0250179753358
Algorithm ran for 8 iterations. Converged: True
Saving trained β in beta_small_0.0
Reading test data from data/small.test
Reading beta from beta_small_0.0
Computing MSE on data data/small.test
MSE is: 255.121147746
```

--silent will reduce the verbose runtime. And in the code

```
verbosity_group = parser.add_mutually_exclusive_group(required=False)
verbosity_group.add_argument('--verbose', dest='verbose', action='store_true')
verbosity_group.add_argument('--silent', dest='verbose', action='store_false')
parser.set_defaults(verbose=True)

args = parser.parse_args()

sc = SparkContext(appName='Parallel Ridge Regression')

if not args.verbose :
    sc.setLogLevel("ERROR")
```

We can see that there are two modes –silent and –verbose. We can only choose one of them. If we choose –silent the sc.setLogLevel will be ERROR Level which is different than the default value ALL

Q5

a) Below are the table for lambda = 0.0-20.0 for small.test and small.train

Lam=0.0	**1.0**	**2.0**	**3.0**	**4.0**	**5.0**	**6.0**
255.121	233.008	229.538	228.312	227.731	227.407	227.203
7.0	**8.0**	**9.0**	**10.0**	**11.0**	**12.0**	**13.0**
227.067	226.972	226.900	226.847	226.804	226.771	226.744
14.0	**15.0**	**16.0**	**17.0**	**18.0**	**19.0**	**20.0**
226.721	226.702	226.685	226.671	226.659	226.648	226.639

Vector β that attains the smallest test MSE is shown below

```
-0.017402507353,0.00797474217258,-0.0274240947991,0.0159458185151,-0.022518811717,-0.0
239316217949,0.0364235799041,-0.0227455624674,-0.0311777331392,-0.0135229518377,0.0064
162340321,0.0339924770738,0.0196718740877,0.0423553130949,-0.0435532651886,0.015658415
2513,-0.0285058292549,-0.0158214447749,0.0726666691996,0.0583111537736,0.0874434829294
,-0.0365586513321,-0.00343575922875,0.0424288008633,-0.0216981748667,0.00463211961379,
-0.052438977233,-0.0151045998635,0.044389229654,0.0479043633361,0.0251447153654,-0.005
3569524134,0.0606309128847,-0.0122600504909,-0.00918442247061,-0.0361993979921,0.04925
20143848,-0.0159157038413,0.0602865151194,-0.0761269591162,-0.0281943631548,0.00396622
694321,-0.0173751453971,0.00250281759287,0.0290921683854,0.022211985252,0.011609161284
6,0.0358589509738,-0.0521946977874,0.0304654818825
```

b) Below are the table for lambda = 0.0-10.0 for big.test and big.train

0.0	**1.0**	**2.0**	**3.0**	**4.0**	**5.0**
4151.494	4000.099	3977.838	3971.328	3968.868	3967.833
6.0	**7.0**	**8.0**	**9.0**	**10.0**	
3967.386	3967.214	3967.177	3967.208	3967.272	