**Appendix**

**Reading the data in parallel- sample checking code**

import numpy as np

from numpy import arange,array,ones,linalg

from mpi4py import MPI

wt1 = MPI.Wtime()

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

mpisize = comm.Get\_size()

def is\_number(s):

try:

float(s)

return s

except ValueError:

return 0

infile=open('dft\_vs\_dft\_elumo\_dump.dat')

num\_lines = sum(1 for line in infile)

print num\_lines

start=rank\*(num\_lines-2)/mpisize+1

end=(rank+1)\*(num\_lines-2)/mpisize

lines=end-start+1

print rank, start,end,lines

x1=np.zeros(lines)

x2=np.zeros(lines)

x3=np.zeros(lines)

x4=np.zeros(lines)

x5=np.zeros(lines)

x6=np.zeros(lines)

x7=np.zeros(lines)

x8=np.zeros(lines)

x9=np.zeros(lines)

infile=open('dft\_vs\_dft\_elumo\_dump.dat')

for \_ in xrange(start+1):

next(infile)

for i,line in enumerate(infile):

if i==(lines):

break

values = line.strip().split(',')

x1[i] = is\_number((values[2]))

x2[i] = is\_number((values[3]))

x3[i] = is\_number((values[4]))

x4[i] = is\_number((values[5]))

x5[i] = is\_number((values[6]))

x6[i] = is\_number((values[7]))

x7[i] = is\_number((values[8]))

x8[i] = is\_number((values[9]))

x9[i] = is\_number((values[10]))

wt2 = MPI.Wtime()

if rank==0:

print mpisize,wt2-wt1

**Code written for HPC project**

*Note*- Please note that this code only contain the part of the serial code which was parallelized. The serial code that is attached later in appendix is a more generalized code which would compute correlations between different properties and also plot the data and write the output into text file.

**Python code for parallel gradient descent**

import numpy as np

from numpy import arange,array,ones,linalg

from mpi4py import MPI

wt1 = MPI.Wtime()

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

mpisize = comm.Get\_size()

def is\_number(s):

try:

float(s)

return s

except ValueError:

return 0

infile=open('dft\_vs\_dft\_elumo\_dump.dat')

num\_lines = sum(1 for line in infile)

start=rank\*(num\_lines-2)/mpisize+1

end=(rank+1)\*(num\_lines-2)/mpisize

lines=end-start+1

x1=np.zeros(lines)

x2=np.zeros(lines)

x3=np.zeros(lines)

x4=np.zeros(lines)

x5=np.zeros(lines)

x6=np.zeros(lines)

x7=np.zeros(lines)

x8=np.zeros(lines)

x9=np.zeros(lines)

infile=open('dft\_vs\_dft\_elumo\_dump.dat')

for \_ in xrange(start+1):

next(infile)

for i,line in enumerate(infile):

if i==(lines):

break

values = line.strip().split(',')

x1[i] = is\_number((values[2]))

x2[i] = is\_number((values[3]))

x3[i] = is\_number((values[4]))

x4[i] = is\_number((values[5]))

x5[i] = is\_number((values[6]))

x6[i] = is\_number((values[7]))

x7[i] = is\_number((values[8]))

x8[i] = is\_number((values[9]))

x9[i] = is\_number((values[10]))

X\_norm=np.array([np.ones(lines),x1])

X\_norm=X\_norm.transpose()

alpha = 0.01

theta1= np.ones(2)

theta2= np.ones(2)

theta3= np.ones(2)

theta4= np.ones(2)

theta5= np.ones(2)

theta6= np.ones(2)

theta7= np.ones(2)

theta8= np.ones(2)

def sums(x, y, theta, m):

xTrans = x.transpose()

hypothesis = np.dot(x, theta)

loss = hypothesis - y

gradient = np.dot(xTrans, loss) / m

return gradient

def errors(x, y, theta, m):

hypothesis = np.dot(x, theta)

loss\_sq = sum((hypothesis - y)\*\*2)

return loss\_sq

for i in xrange(1000):

grad\_sum1= sums(X\_norm, x2, theta1, num\_lines)

grad\_sum2= sums(X\_norm, x3, theta2, num\_lines)

grad\_sum3= sums(X\_norm, x4, theta3, num\_lines)

grad\_sum4= sums(X\_norm, x5, theta4, num\_lines)

grad\_sum5= sums(X\_norm, x6, theta5, num\_lines)

grad\_sum6= sums(X\_norm, x7, theta6, num\_lines)

grad\_sum7= sums(X\_norm, x8, theta7, num\_lines)

grad\_sum8= sums(X\_norm, x9, theta8, num\_lines)

error1 = errors(X\_norm, x2, theta1, num\_lines)

error2 = errors(X\_norm, x3, theta2, num\_lines)

error3 = errors(X\_norm, x4, theta3, num\_lines)

error4 = errors(X\_norm, x5, theta4, num\_lines)

error5 = errors(X\_norm, x6, theta5, num\_lines)

error6 = errors(X\_norm, x7, theta6, num\_lines)

error7 = errors(X\_norm, x8, theta7, num\_lines)

error8 = errors(X\_norm, x9, theta8, num\_lines)

if rank==0:

totals1 = np.zeros\_like(grad\_sum1)

totals2 = np.zeros\_like(grad\_sum2)

totals3 = np.zeros\_like(grad\_sum3)

totals4 = np.zeros\_like(grad\_sum4)

totals5 = np.zeros\_like(grad\_sum5)

totals6 = np.zeros\_like(grad\_sum6)

totals7 = np.zeros\_like(grad\_sum7)

totals8 = np.zeros\_like(grad\_sum8)

total\_e1=np.zeros(1)

total\_e2=np.zeros(1)

total\_e3=np.zeros(1)

total\_e4=np.zeros(1)

total\_e5=np.zeros(1)

total\_e6=np.zeros(1)

total\_e7=np.zeros(1)

else:

totals1 = None

totals2 = None

totals3 = None

totals4 = None

totals5 = None

totals6 = None

totals7 = None

totals8 = None

total\_e1=None

total\_e2=None

total\_e3=None

total\_e4=None

total\_e5=None

total\_e6=None

total\_e7=None

total\_e8=None

comm.Reduce([grad\_sum1, MPI.DOUBLE],[totals1, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([grad\_sum2, MPI.DOUBLE],[totals2, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([grad\_sum3, MPI.DOUBLE],[totals3, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([grad\_sum4, MPI.DOUBLE],[totals4, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([grad\_sum5, MPI.DOUBLE],[totals5, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([grad\_sum6, MPI.DOUBLE],[totals6, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([grad\_sum7, MPI.DOUBLE],[totals7, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([grad\_sum8, MPI.DOUBLE],[totals8, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([error1, MPI.DOUBLE],[total\_e1, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([error2, MPI.DOUBLE],[total\_e2, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([error3, MPI.DOUBLE],[total\_e3, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([error4, MPI.DOUBLE],[total\_e4, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([error5, MPI.DOUBLE],[total\_e5, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([error6, MPI.DOUBLE],[total\_e6, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([error7, MPI.DOUBLE],[total\_e7, MPI.DOUBLE],op = MPI.SUM,root = 0)

comm.Reduce([error8, MPI.DOUBLE],[total\_e8, MPI.DOUBLE],op = MPI.SUM,root = 0)

if rank==0:

theta1=theta1 - alpha\*totals1

theta2=theta2 - alpha\*totals2

theta3=theta3 - alpha\*totals3

theta4=theta4 - alpha\*totals4

theta5=theta5 - alpha\*totals5

theta6=theta6 - alpha\*totals6

theta7=theta7 - alpha\*totals7

theta8=theta8 - alpha\*totals8

totals1=0

totals2=0

totals3=0

totals4=0

totals5=0

totals6=0

totals7=0

totals8=0

theta1=comm.bcast(theta1, root=0)

theta2=comm.bcast(theta2, root=0)

theta3=comm.bcast(theta3, root=0)

theta4=comm.bcast(theta4, root=0)

theta5=comm.bcast(theta5, root=0)

theta6=comm.bcast(theta6, root=0)

theta7=comm.bcast(theta7, root=0)

theta8=comm.bcast(theta8, root=0)

wt2 = MPI.Wtime()

if rank==0:

print 'C1',theta1,'C2',theta2,'C3',theta3,'C4',theta4,'C5',theta5,'C6',theta6,'C7',theta7,'C8',theta8

print 'Elapsed wall time = ',wt2-wt1

print mpisize, wt2-wt1, theta1

**Serial code**

import matplotlib

matplotlib.use('Agg')

from matplotlib import pyplot as plt

import numpy as np

from scipy import polyval, polyfit

from numpy import arange,array,ones,linalg

from pylab import plot,show

import argparse

#######################################################################

#this function is used to test the dump file's contents to make sure their actual

#data and not blank or invalid formats

def is\_number(s):

try:

float(s)

return s

except ValueError:

return 0

#Used to extend the linear regression and compound plot lines so that they go past

#the max and min of their data sets

def extended(ax, x, y, \*\*args):

xlim = ax.get\_xlim()

ylim = ax.get\_ylim()

x\_ext = np.linspace(xlim[0], xlim[1], 100)

p = np.polyfit(x, y , deg=1)

y\_ext = np.poly1d(p)(x\_ext)

ax.plot(x\_ext, y\_ext, \*\*args)

ax.set\_xlim(xlim)

ax.set\_ylim(ylim)

return ax

## Save LR data into file

def write(m,b,r\_squared,property,x,y):

mm = str(m)

bb = str(b)

rr = str(r\_squared)

txt = open('projectionvalues.txt', 'a')

txt.write(property),txt.write("\_"),txt.write(x),txt.write("\_"),

txt.write(y),txt.write(" "),txt.write(mm),txt.write(" "),

txt.write(bb),txt.write(" "),txt.write(rr),txt.write("\n")

txt.close()

def write\_outlier(m,b,r\_squared,property,x,y,percentage):

mm = str(m)

bb = str(b)

rr = str(r\_squared)

txt = open('projectionvalues.txt', 'a')

txt.write(property),txt.write("\_"),txt.write(x),txt.write("\_"),

txt.write(y),txt.write("\_outlier\_"),txt.write(percentage),txt.write("%"),

txt.write(" "),txt.write(mm),txt.write(" "),

txt.write(bb),txt.write(" "),txt.write(rr),txt.write("\n")

txt.close()

def write\_remained(m,b,r\_squared,property,x,y,percentage):

mm = str(m)

bb = str(b)

rr = str(r\_squared)

txt = open('projectionvalues.txt', 'a')

txt.write(property),txt.write("\_"),txt.write(x),txt.write("\_"),

txt.write(y),txt.write("\_remained\_"),txt.write(percentage),txt.write("%"),

txt.write(" "),txt.write(mm),txt.write(" "),

txt.write(bb),txt.write(" "),txt.write(rr),txt.write("\n")

txt.close()

###############################################################################

### the HelpFormatter and provide a special intro for the options that should be handled "raw"

### Any other calls to .add\_argument() where the help does not start with R| will be wrapped as normal.

class SmartFormatter(argparse.HelpFormatter):

def \_split\_lines(self, text, width):

# this is the RawTextHelpFormatter.\_split\_lines

if text.startswith('R|'):

return text[2:].splitlines()

return argparse.HelpFormatter.\_split\_lines(self, text, width)

parser = argparse.ArgumentParser(description="Do Linear Regression to the data",

formatter\_class=SmartFormatter)

parser.add\_argument("infile", type=str, help="input the name of data file")

parser.add\_argument("property", type=int, choices=[1,2,3,4],

help="R|choose the options of property:\n"

"1 = HOMO 2 = LUMO 3 = Dipole 4 = Gap")

parser.add\_argument("percent\_outlier", type=float, help="R|define the outlier\n"

"enter the percentage of data taken out to be outliers\n"

"input only number part")

parser.add\_argument("-x", "--x\_flavor", type=int,

choices=[1,2], help="R|choose the flavor for x-axis:\n"

"1 = BP86/SVP 2 = B3LYP/SVP")

parser.add\_argument("-y", "--y\_flavor", type=int,

choices=[1,2,3,4,5,6,7,8,9], help="R|choose the flavor for y-axis:\n"

"1 = BP86/SVP 2 = B3LYP/SVP 3 = PBE0/SVP \n4 = BH&HLYP/SVP 5 = M06-2X/SVP"

" 6 = HF/SVP \n7 = BP86/TZVP 8 = B3LYP/TZVP 9 = PBE0/TZVP")

parser.add\_argument("-rb","--result\_box",action='store\_false',default=True,

help="Hide the result display box in all plots")

parser.add\_argument("-rb1","--result\_box1",action='store\_false',default=True,

help="Hide the result display box only in original data pool")

parser.add\_argument("-rb2","--result\_box2",action='store\_false',default=True,

help="Hide the result display box only in outlier")

parser.add\_argument("-rb3","--result\_box3",action='store\_false',default=True,

help="Hide the result display box only in remaining pool")

parser.add\_argument("-lr","--lr\_line",action='store\_false',default=True,

help="Hide all the LinearReg lines.")

parser.add\_argument("-lr1","--lr\_line1",action='store\_false',default=True,

help="Hide the LinearReg line for original data pool.")

parser.add\_argument("-lr2","--lr\_line2",action='store\_false',default=True,

help="Hide the LinearReg line for outliers.")

parser.add\_argument("-lr3","--lr\_line3",action='store\_false',default=True,

help="Hide the LinearReg line for remaining pool.")

parser.add\_argument("-his","--histogram",action='store\_false',default=True,

help="Do not generate historgram of the projection discrepancy")

args = parser.parse\_args('dft\_vs\_dft\_elumo\_dump.dat 1 5 -x 1 -y 4 -lr2 -lr3'.split())

#args = parser.parse\_args()

###percent\_outlier

num\_percentage = str(args.percent\_outlier)

### property

if args.property == 1:

pro\_name = "HOMO"

line\_color = "Blue"

num1 = -9.00 #for ax.set\_xlim and ax.set\_ylim

num2 = -3.00 #for ax.set\_xlim and ax.set\_ylim

elif args.property == 2:

pro\_name = "LUMO"

line\_color = "Red"

num1 = -6.00 #for ax.set\_xlim and ax.set\_ylim

num2 = 3.00 #for ax.set\_xlim and ax.set\_ylim

elif args.property == 3:

pro\_name = "Dipole"

line\_color = "Violet"

num1 = -1.00 #for ax.set\_xlim and ax.set\_ylim

num2 = 14.00 #for ax.set\_xlim and ax.set\_ylim

elif args.property == 4:

pro\_name = "Gap"

line\_color = "Green"

num1 = 0.00 #for ax.set\_xlim and ax.set\_ylim

num2 = 10.00 #for ax.set\_xlim and ax.set\_ylim

### x-flavor, y-flavor

if args.x\_flavor == 1:

x\_name = "BP86s"

x\_fname = "BP86/SVP"

#Used to get rid of obvious outliers that have been brought into the data

if args.property == 1: #HOMO

if args.y\_flavor != 6: # for all y flavors except HF #HOMO,BP86s

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.24) and i[0]<=(-0.12):

if i[1]>=(-0.30) and i[1]<=(-0.14):

return True

else:

return False

else: # only for HF (y flavor) #HOMO,BP86s

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.24) and i[0]<=(-0.12):

if i[1]>=(-0.60) and i[1]<=(-0.14):

return True

else:

return False

####

elif args.property == 2: #LUMO,BP86s

if args.y\_flavor != 6: # for all y flavors except HF #LUMO,BP86s

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.20) and i[0]<=(0.0):

if i[1]>=(-0.20) and i[1]<=(0.05):

return True

else:

return False

else: # only for HF (y flavor) #LUMO,BP86s

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.20) and i[0]<=(0.10):

if i[1]>=(-0.20) and i[1]<=(0.30):

return True

else:

return False

###

elif args.property == 3: #Dipole,BP86s

if args.y\_flavor != 6: # for all y flavors except HF #Dipole,BP86s

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.10) and i[0]<=(13.0):

if i[1]>=(-0.10) and i[1]<=(13.0):

return True

else:

return False

else: # only for HF (y flavor) #Dipole,BP86s

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.10) and i[0]<=(13.0):

if i[1]>=(-0.10) and i[1]<=(13.0):

return True

else:

return False

###

elif args.property == 4: #Gap,BP86s

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(0.0) and i[0]<=(0.15):

if i[1]>=(0.0) and i[1]<=(0.40):

return True

else:

return False

elif args.x\_flavor == 2:

x\_name = "B3LYPs"

x\_fname = "B3LYP/SVP"

#Used to get rid of obvious outliers that have been brought into the data

if args.property == 1: #HOMO,B3LYPs

if args.y\_flavor != 6: # for all y flavors except HF #HOMO

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.28) and i[0]<=(-0.14):

if i[1]>=(-0.30) and i[1]<=(-0.14):

return True

else:

return False

else: # only for HF (y flavor) #HOMO

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.24) and i[0]<=(-0.12):

if i[1]>=(-0.60) and i[1]<=(-0.14):

return True

else:

return False

###

elif args.property == 2: #LUMO,B3LYPs

if args.y\_flavor != 6: # for all y flavors except HF #LUMO,B3LYPs

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.20) and i[0]<=(0.05):

if i[1]>=(-0.20) and i[1]<=(0.05):

return True

else:

return False

else: # only for HF (y flavor) #LUMO,B3LYPs

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.20) and i[0]<=(0.10):

if i[1]>=(-0.20) and i[1]<=(0.30):

return True

else:

return False

###

elif args.property == 3: #Dipole,B3LYPs

if args.y\_flavor != 6: # for all y flavors except HF #Dipole,B3LYPs

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.10) and i[0]<=(13.0):

if i[1]>=(-0.10) and i[1]<=(13.0):

return True

else:

return False

else: # only for HF (y flavor) #Dipole,B3LYPs

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(-0.10) and i[0]<=(13.0):

if i[1]>=(-0.10) and i[1]<=(13.0):

return True

else:

return False

###

elif args.property == 4: #Gap,B3LYPs

def lrboundaries(s,t):

#i[0] is x, i[1] is y, first term is min, second is max

if i[0]>=(0.0) and i[0]<=(0.20):

if i[1]>=(0.0) and i[1]<=(0.40):

return True

else:

return False

elif args.x\_flavor == 3:

x\_name = "PBE0s"

elif args.x\_flavor == 4:

x\_name = "BHHLYPs"

elif args.x\_flavor == 5:

x\_name = "M06s"

elif args.x\_flavor == 6:

x\_name = "HFs"

elif args.x\_flavor == 7:

x\_name = "BP86t"

elif args.x\_flavor == 8:

x\_name = "B3LYPt"

elif args.x\_flavor == 9:

x\_name = "PBE0t"

if args.y\_flavor == 1:

y\_name = "BP86s"

y\_fname = "BP86/SVP"

elif args.y\_flavor == 2:

y\_name = "B3LYPs"

y\_fname = "B3LYP/SVP"

elif args.y\_flavor == 3:

y\_name = "PBE0s"

y\_fname = "PBE0/SVP"

elif args.y\_flavor == 4:

y\_name = "BHHLYPs"

y\_fname = "BH&HLYP/SVP"

elif args.y\_flavor == 5:

y\_name = "M06s"

y\_fname = "M06/SVP"

elif args.y\_flavor == 6:

y\_name = "HFs"

y\_fname = "HF/SVP"

elif args.y\_flavor == 7:

y\_name = "BP86t"

y\_fname = "BP86/TZVP"

elif args.y\_flavor == 8:

y\_name = "B3LYPt"

y\_fname = "B3LYP/TZVP"

elif args.y\_flavor == 9:

y\_name = "PBE0t"

y\_fname = "PBE0/TZVP"

lr\_plot = "lr\_%s\_%s\_%s.png" % (pro\_name, x\_name, y\_name)

lr\_histogram = "Hist\_%s\_%s\_%s.png" % (pro\_name, x\_name, y\_name)

num5 = str(int(args.percent\_outlier \* 100))

lr\_plot\_outlier = "lr\_%s\_%s\_%s\_outlier\_%s.png" % (pro\_name, x\_name, y\_name, num5)

lr\_plot\_remained = "lr\_%s\_%s\_%s\_remained\_%s.png" % (pro\_name, x\_name, y\_name, num5)

lr\_text\_outlier = "lr\_%s\_%s\_%s\_outlier\_%s" % (pro\_name, x\_name, y\_name, num5)

projection\_discrepancy = "discrepancy\_%s\_%s\_%s"% (pro\_name, x\_name, y\_name)

#used to count the amount of lines in my file, -2 is for the lines that dont't matter, the heading

num\_lines = sum(1 for line in open(args.infile))

x\_array = np.zeros(num\_lines-2)

y\_array = np.zeros(num\_lines-2)

with open(args.infile) as f:

for \_ in xrange(2):

next(f)

for i,line in enumerate(f):

values = line.strip().split(',')

x\_array[i] = is\_number((values[args.x\_flavor + 1]))

y\_array[i] = is\_number((values[args.y\_flavor + 1]))

f.close()

##########################################################################

z = zip(x\_array,y\_array)

x1=[]

y1=[]

for i in z:

if i[0] and i[1]!=0:

#line[0] is x and line[1] is y

if lrboundaries(i[0],i[1])==True:

if args.property == 3: #only for Dipole

x1.append(i[0])

y1.append(i[1])

else:

x1.append(i[0]\*27.21139570)

y1.append(i[1]\*27.21139570)

b1=x1

print 'x'

print x1

print 'y'

print y1

#linear regression using stats.linregress

#slope, intercept, r\_value, p\_value, std\_err = stats.linregress(x1,y1)

#

#

#this will find your y-intercept and slope of your linear regression line and

#then will create points where a line is plotted on

(w,r\_value,a1,b1,c1)=polyfit(x1,y1,1,full=True)

m=w[0]

b=w[1]

r\_squared= r\_value\*\*2

yp=polyval(w,x1)

yp1=yp

stand = (np.std(x1) + np.std(y1))/2

print 'r-squared: ', r\_squared

print 'stand: ', stand

plt.rcParams['font.family'] = 'sans-serif'

plt.rcParams['mathtext.default'] = 'regular'

fig= plt.figure()

ax = fig.add\_subplot(111, aspect='equal')

ax.scatter(x1,y1,s=0.5,color='Black')

textstr = '$R^2:\ \ \ \ \ \ \ %.4f$\n$slope:\ \ \ %.4f$\n$shift:\ \ \ \ \ %.4f$'%(r\_squared, m, b)

props = dict(boxstyle='round', facecolor= 'White', alpha=1.0)

if args.result\_box == True and args.result\_box1 == True:

ax.text(0.05, 0.95, textstr, transform=ax.transAxes, fontsize=11,

verticalalignment='top', bbox=props)

write(m,b,r\_squared,pro\_name,x\_name,y\_name)

if args.property == 1:

ax.set\_xlabel(r'$\epsilon\ $HOMO (%s) [eV]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\epsilon\ $HOMO (%s) [eV]'%(y\_fname), fontsize='x-large')

elif args.property == 2:

ax.set\_xlabel(r'$\epsilon\ $LUMO (%s) [eV]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\epsilon\ $LUMO (%s) [eV]'%(y\_fname), fontsize='x-large')

elif args.property == 3:

ax.set\_xlabel(r'$\mu\ $ (%s) [D]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\mu\ $ (%s) [D]'%(y\_fname), fontsize='x-large')

elif args.property == 4:

ax.set\_xlabel(r'$\Delta\ \epsilon\ $ (%s) [eV]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\Delta\ \epsilon\ $ (%s) [eV]'%(y\_fname), fontsize='x-large')

ax.tick\_params(labelsize='large')

ax.set\_xlim((num1),(num2))

ax.set\_ylim((num1),(num2))

if args.lr\_line == True and args.lr\_line1 == True:

ax = extended(ax, x1, yp, lw=2.4, color = line\_color)

#ax.plot(x1,yp,'green',linewidth=1.0)

ax.grid(True)

plt.savefig(lr\_plot)

#plt.show(ax)

##########################################################################

# save outliers into a list

def\_outlier = args.percent\_outlier \* 0.01 #define what percentage of datas are taken out

dis = []

dis\_raw = []

# turn list x1 and y1 into array

z2 = zip(x1,y1)

x\_and\_y = np.asarray(z2)

#def predicted\_y(x):

# predicted\_y\_value = slope \* x + intercept

# return predicted\_y\_value

# calculte the distance from each point to regression line

for (x,y) in x\_and\_y:

distance = m\*x+b - y

dis\_raw = np.append(dis\_raw,[distance])

dis = np.absolute(dis\_raw)

##save discrepancy into a python array file

#np.save(projection\_discrepancy, dis\_raw)

# get an array (x,y,distance) and make it sorted by order of distance (low to high)

x\_y\_dis= np.concatenate((x\_and\_y,dis[:,np.newaxis]), axis=1)

x\_y\_dis\_sorted = x\_y\_dis[x\_y\_dis[:,2].argsort()]

# extract the outliers of specified percentage from the data

num3 = len(x\_y\_dis\_sorted) - int(len(x\_y\_dis\_sorted) \* def\_outlier)

x\_y\_dis\_remained = x\_y\_dis\_sorted[0:num3, :]

x\_y\_dis\_outlier = x\_y\_dis\_sorted[num3:len(x\_y\_dis\_sorted), :]

##save outlier into file

#if args.property == 3: #only for Dipole

# np.save(lr\_text\_outlier, x\_y\_dis\_outlier)

#else:

# array1 = np.array([27.21139570,27.21139570,1], float)

# x\_y\_dis\_outlier2 = x\_y\_dis\_outlier / array1

# np.save(lr\_text\_outlier, x\_y\_dis\_outlier2)

x\_xydisoutlier = x\_y\_dis\_outlier[:,0].tolist()

y\_xydisoutlier = x\_y\_dis\_outlier[:,1].tolist()

x\_xydisremained = x\_y\_dis\_remained[:,0].tolist()

y\_xydisremained = x\_y\_dis\_remained[:,1].tolist()

###linear regression for outliers

#slope, intercept, r\_value, p\_value, std\_err = stats.linregress(x\_xydisoutlier,y\_xydisoutlier)

#

#this will find your y-intercept and slope of your linear regression line and

#then will create points where a line is plotted on

(w2,r\_value,a2,b2,c2)=polyfit(x\_xydisoutlier,y\_xydisoutlier,1,full=True)

m2=w2[0]

b2=w2[1]

r\_squared= r\_value\*\*2

yp2=polyval(w2,x\_xydisoutlier)

yp1=yp2

stand = (np.std(x\_xydisoutlier) + np.std(y\_xydisoutlier))/2

#print 'r-squared: ', r\_squared

print 'stand: ', stand

plt.rcParams['font.family'] = 'sans-serif'

plt.rcParams['mathtext.default'] = 'regular'

fig= plt.figure()

ax = fig.add\_subplot(111, aspect='equal')

ax.scatter(x\_xydisoutlier,y\_xydisoutlier,s=0.5,color='Black')

textstr = '<Outliers>\n$R^2:\ \ \ \ \ \ \ %.4f$\n$slope:\ \ \ %.4f$\n$shift:\ \ \ \ \ %.4f$'%(r\_squared, m2, b2)

props = dict(boxstyle='round', facecolor= 'White', alpha=1.0)

if args.result\_box == True and args.result\_box2 == True:

ax.text(0.05, 0.95, textstr, transform=ax.transAxes, fontsize=11,

verticalalignment='top', bbox=props)

write\_outlier(m2,b2,r\_squared,pro\_name,x\_name,y\_name,num\_percentage)

if args.property == 1:

ax.set\_xlabel(r'$\epsilon\ $HOMO (%s) [eV]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\epsilon\ $HOMO (%s) [eV]'%(y\_fname), fontsize='x-large')

elif args.property == 2:

ax.set\_xlabel(r'$\epsilon\ $LUMO (%s) [eV]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\epsilon\ $LUMO (%s) [eV]'%(y\_fname), fontsize='x-large')

elif args.property == 3:

ax.set\_xlabel(r'$\mu\ $ (%s) [D]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\mu\ $ (%s) [D]'%(y\_fname), fontsize='x-large')

elif args.property == 4:

ax.set\_xlabel(r'$\Delta\ \epsilon\ $ (%s) [eV]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\Delta\ \epsilon\ $ (%s) [eV]'%(y\_fname), fontsize='x-large')

ax.tick\_params(labelsize='large')

ax.set\_xlim((num1),(num2))

ax.set\_ylim((num1),(num2))

if args.lr\_line == True and args.lr\_line2 == True:

ax = extended(ax, x\_xydisoutlier, yp2, lw=2.4, color = line\_color)

#ax.plot(x1,yp,'green',linewidth=1.0)

ax.grid(True)

plt.savefig(lr\_plot\_outlier)

#plt.show(ax)

#########################################################################

###linear regression for the remained

#slope, intercept, r\_value, p\_value, std\_err = stats.linregress(x\_xydisremained,y\_xydisremained)

#

#this will find your y-intercept and slope of your linear regression line and

#then will create points where a line is plotted on

(w3,r\_value,a3,b3,c3)=polyfit(x\_xydisremained,y\_xydisremained,1,full=True)

yp2=polyval(w3,x\_xydisremained)

m3=w3[0]

b3=w3[1]

r\_squared= r\_value\*\*2

yp1=yp2

stand = (np.std(x\_xydisremained) + np.std(y\_xydisremained))/2

print 'r-squared: ', r\_squared

print 'stand: ', stand

plt.rcParams['font.family'] = 'sans-serif'

plt.rcParams['mathtext.default'] = 'regular'

fig= plt.figure()

ax = fig.add\_subplot(111, aspect='equal')

ax.scatter(x\_xydisremained,y\_xydisremained,s=0.5,color='Black')

textstr = '<Remained>\n$R^2:\ \ \ \ \ \ \ %.4f$\n$slope:\ \ \ %.4f$\n$shift:\ \ \ \ \ %.4f$'%(r\_squared, m3, b3)

props = dict(boxstyle='round', facecolor= 'White', alpha=1.0)

if args.result\_box == True and args.result\_box3 == True:

ax.text(0.05, 0.95, textstr, transform=ax.transAxes, fontsize=11,

verticalalignment='top', bbox=props)

write\_remained(m3,b3,r\_squared,pro\_name,x\_name,y\_name,num\_percentage)

if args.property == 1:

ax.set\_xlabel(r'$\epsilon\ $HOMO (%s) [eV]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\epsilon\ $HOMO (%s) [eV]'%(y\_fname), fontsize='x-large')

elif args.property == 2:

ax.set\_xlabel(r'$\epsilon\ $LUMO (%s) [eV]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\epsilon\ $LUMO (%s) [eV]'%(y\_fname), fontsize='x-large')

elif args.property == 3:

ax.set\_xlabel(r'$\mu\ $ (%s) [D]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\mu\ $ (%s) [D]'%(y\_fname), fontsize='x-large')

elif args.property == 4:

ax.set\_xlabel(r'$\Delta\ \epsilon\ $ (%s) [eV]'%(x\_fname), fontsize='x-large')

ax.set\_ylabel(r'$\Delta\ \epsilon\ $ (%s) [eV]'%(y\_fname), fontsize='x-large')

ax.tick\_params(labelsize='large')

ax.set\_xlim((num1),(num2))

ax.set\_ylim((num1),(num2))

if args.lr\_line == True and args.lr\_line3 == True:

ax = extended(ax, x\_xydisremained, yp2, lw=2.4, color = line\_color)

#ax.plot(x1,yp,'green',linewidth=1.0)

ax.grid(True)

plt.savefig(lr\_plot\_remained)

#plt.show(ax)

#########################################################################

# Create a histogram for the projection discrepancy

if args.histogram == True:

fig= plt.figure()

ax = fig.add\_subplot(111)

plt.rcParams['font.family'] = 'sans-serif'

plt.rcParams['mathtext.default'] = 'regular'

ax.tick\_params(labelsize='large')

ax.grid(True)

if args.property == 1:

ax.set\_xlabel("$\epsilon\ $ $\Delta$%s Values [eV]" %(pro\_name), fontsize='x-large')

elif args.property == 2:

ax.set\_xlabel("$\epsilon\ $ $\Delta$%s Values [eV]" %(pro\_name), fontsize='x-large')

elif args.property == 3:

ax.set\_xlabel(r'$\mu\ $%s Values [D]' %(pro\_name), fontsize='x-large')

elif args.property == 4:

ax.set\_xlabel("$\Delta\ \epsilon\ $%s Values [eV]" %(pro\_name), fontsize='x-large')

ax.set\_ylabel('Frequency', fontsize='x-large')

plt.yscale('log', nonposy='clip')

props = dict(boxstyle='round', facecolor= 'White', alpha=1.0)

if args.property == 1:

textstr= "$\epsilon\ $%s \n (%s) [eV] \n vs. \n (%s) [eV]" %(pro\_name,x\_fname,y\_fname)

elif args.property == 2:

textstr= "$\epsilon\ $%s \n (%s) [eV] \n vs. \n (%s) [eV]" %(pro\_name,x\_fname,y\_fname)

elif args.property == 3:

textstr= "$\mu\ $%s \n (%s) [eV] \n vs. \n (%s) [eV]" %(pro\_name,x\_fname,y\_fname)

elif args.property == 4:

textstr= "$\Delta\ \epsilon\ $%s \n (%s) [eV] \n vs. \n (%s) [eV]" %(pro\_name,x\_fname,y\_fname)

ax.text(0.14, 0.97, textstr, transform=ax.transAxes, fontsize=11,

verticalalignment='top', bbox=props,ha='center')

ax.hist(dis\_raw,bins=20,range=[-3.0,3.0],histtype='bar')

ax.set\_xlim(-3.0,3.0)

plt.savefig(lr\_histogram)

###################################################

**Slurm script written for running parallel code**

#!/bin/sh

#SBATCH --partition=general-compute

#SBATCH --time=26:00:00

#SBATCH --job-name="para\_grad"

#SBATCH --output=para\_grad\_120core.out

#SBATCH --mail-user=m27@buffalo.edu

#SBATCH --mail-type=ALL

#SBATCH --nodes=15

#SBATCH --cpus-per-task=8

# ====================================================

# For 8-core nodes

# ====================================================

#SBATCH --constraint=CPU-L5520

tic=`date +%s`

echo "Start Time = "`date`

echo "Loading modules ..."

module load python

ulimit -s unlimited

module list

echo "SLURM job ID = "$SLURM\_JOB\_ID

echo "Working Dir = "`pwd`

echo "Compute Nodes = "`nodeset -e $SLURM\_NODELIST`

echo "Number of Processors = "$SLURM\_NPROCS

echo "Number of Nodes = "$SLURM\_NNODES

# for i in $(seq 12 12 120); do

# echo $i

# mpiexec -np $i python readpara.py

# done

mpiexec -np 120 python readpara3.py

mpiexec -np 96 python readpara3.py

mpiexec -np 80 python readpara3.py

mpiexec -np 72 python readpara3.py

mpiexec -np 60 python readpara3.py

mpiexec -np 48 python readpara3.py

mpiexec -np 36 python readpara3.py

mpiexec -np 24 python readpara3.py

mpiexec -np 12 python readpara3.py

mpiexec -np 10 python readpara3.py

mpiexec -np 8 python readpara3.py

mpiexec -np 6 python readpara3.py

mpiexec -np 4 python readpara3.py

mpiexec -np 2 python readpara3.py

mpiexec -np 1 python readpara3.py

echo "All Done!"

echo "End Time = "`date`

toc=`date +%s`

elapsedTime=`expr $toc - $tic`

echo "Elapsed Time = $elapsedTime seconds"