In the **hw1\_p1.py** file, I implemented the Gram Schmitt orthogonalization algorithm.

By comparing the regression coefficient estimates based on the orthogonalization vs. that from multiple regression, I found that the last coefficients are exactly the same but the first a few coefficients are not exactly the same even though they are very close.

In the **hw1\_p2.py file**, I implemented the cyclic coordinate descent algorithm. I use the POS +/- 500000 to select the SNP. I choose the first outcome. The number of subjects is 358 and the number of predictors is 2178.

In the **hw1\_p2\_sklearn.py** file, I used the LASSO from sklearn package to check the correctness of my cyclic coordinate descent algorithm by using the same set of predictors and same lambda value **(0.1).** The lambda value is randomly select without any tuning. The coefficient estimations are the same. In the **hw1\_glmnet.R** file, I used the LASSO from glmnet package with R do check the coefficients. They’re very closed but not exactly the same.

I further used a toy data to check if I could get the same results with glmnet and sklearn and find out that I could exactly the same results. So the reason why I could slightly different results is due to the different decimal precision between R and python. After normalizing the X matrix, the data becomes quite different which results in the estimation discrepancy.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

The estimation using three methods are stored in **results.csv**. So, I conclude that the implementation of algorithm is correct.

Here are the main part of the code, please find the complete code in the .py files.

Algorithm:

def close\_form(rho,gamma,lambda\_i):  
 if rho > lambda\_i:  
 return (rho - lambda\_i)/gamma  
 elif rho < -lambda\_i:  
 return (rho + lambda\_i)/gamma  
 else:  
 return 0  
  
def coordinate\_descent\_lasso(X, y, lambda\_inp = 0.001, epsilon=0.00001):  
 # X = preprocessing.normalize(X.T).T # normalization  
 n\_sub = X.shape[0]  
 n\_pre = X.shape[1]  
 beta = np.ones((n\_pre,1))  
 beta\_old = np.copy(beta)  
 ite = 1  
 while ite == 1 or np.sum(np.square(beta\_old - beta)) > epsilon:  
 ite += 1  
 beta\_old = np.copy(beta)  
 if ite % 100 == 1:  
 # print(np.sum(np.square(beta\_old - beta)))  
 print(ite)  
 for j in range(n\_pre):  
 xj\_col\_vec = X[:,j].reshape(-1,1)  
 rho = (((xj\_col\_vec.T) @ (y - X @ beta + beta[j]\*xj\_col\_vec)))/n\_sub  
 gamma = (xj\_col\_vec.T) @ (xj\_col\_vec)/n\_sub  
  
 # rho = (((xj\_col\_vec.T) @ (y - X @ beta + beta[j]\*xj\_col\_vec))) / 1  
 # gamma = 1  
 beta[j] = close\_form(rho, gamma, lambda\_inp)  
  
 return beta

Data cleaning:

outcome = pd.read\_table('gene\_expression\_sample/GEUVADIS\_normalized\_expression\_chr20', sep ='\t')  
choose = 0  
start = outcome.loc[choose,'start']  
end= outcome.loc[choose,'end']  
outcome1 = outcome.loc[:, (outcome.columns.str.startswith('HG'))|(outcome.columns.str.startswith('NA'))]  
y = outcome1.loc[choose,:].to\_frame() # 358\*1  
  
raw = pd.read\_table('gene\_expression\_sample/GEUVADIS\_chr20\_processed.traw', sep ='\t')  
raw = raw[raw.POS.isin(range(start-500000, end+500000))]  
df1 = raw.loc[:, (raw.columns =='SNP') | (raw.columns.str.startswith('HG')) | (raw.columns.str.startswith('NA'))]  
df1 = df1.rename(columns=lambda s: s.split('\_')[0])  
df2 = df1.T # the covariate matrix, each row is a subject, each column is a variable  
df2 = df2.rename(columns=df2.iloc[0]).drop(df2.index[0])  
  
data = y.merge(df2, left\_index=True, right\_index=True)  
data.rename(columns={0: "y"}, inplace=True)  
  
y = data.iloc[:,0].to\_numpy().reshape(-1,1)  
X = data.iloc[:,1:data.shape[1]].to\_numpy()