In [ ]:

```
import numpy as np
```

In [1]:

```
def coordinate_descent(y,X, Lambda,tolerence,w_init=None):
    '''
    n is the number of record
    d is

    y is a n-by-1 vector

    X is composed of [x1;x2;x3...;xn], where xn is 1-by-d

    w is is a d-by-1 vector

    '''
    n,d = X.shape
    b=0
    ak=0
    ck = 0
    ek = 1000;
    if(w_init is None):
        w_curr=np.zeros((d,))
    else:
        w_curr = w_init
    loop_count =0
    print("++++++++++++++++++++++++++++++Calculating for Lambda = {}++++++++++++++
++++++++++++++".format(Lambda))
    while (ek > tolerence):
        if(loop_count > 10000):
            print("didn't converge")
            break
        loop_count += 1
        w_prev = np.copy (w_curr)
        b0=np.dot(w_prev.T,X.T)
        #print("the shape of b0 is {}".format(b0.shape))
        c = np.zeros((d,))
        b = 1/n * (np.sum(y -b0))
        #print("the shape of b is {}".format(b.shape))
        #print("b is {}".format(b))
        #print("b shape {}".format(b.shape))
        a = 2*np.sum(np.square(X), axis=0)
        for k in range(0,d):
#             selector = [j for j in range(d) if j != k]
#             p1=np.dot(X[:, selector], w_curr[selector])+b
#             p2 = y-p1
#             c[k] = 2 * np.dot(X[:,k], p2 )
            c[k] = 2*np.dot(X[:, k], y - (b + np.dot(w_curr.T, X.T) - w_curr[k]*X
[:, k]))
            #print("the shape of ck is {}".format(ck.shape))
            #print(ck)
            #print(ck.shape)
            if (c[k] < -Lambda):
                w_curr[k] = (c[k]+ Lambda) / a[k]
            elif (c[k] > Lambda):
                w_curr[k] = (c[k]- Lambda) / a[k]
            else:
                w_curr[k] = 0
            #if (w_curr[k]!= 0):
```

```
                #print(w_curr[k])

        #print(np.linalg.norm(w_curr))
        #print(np.linalg.norm(w_prev))
        ek = np.max(np.abs(w_curr - w_prev))
    print("..................... it takes {} iterations to converg
e.................".format(loop_count))
    print("........error is {}............".format(ek))
    #Lambda  = Lambda/2
    return b,w_curr
```

In [2]:

```
def maxLambda(X,y):
    '''
    y is a n-by-1 vector

    X is composed of [x1;x2;x3...;xn], where xn is 1-by-d

    '''
    return  2*np.max( np.abs(np.dot(X.T, (y-np.mean(y)).T)))
```

In [ ]: