#Minipro 4

16337266 徐原

Part 1: Principal Component Analysis (PCA)

(a) (Warm-up, do not submit.) Say we ran PCA on the binary matrix X above. What would be the dimension of the returned vectors?

Solution:
Nucleobase size=995*10100

(b) (6 points) We will examine the first 2 principal components of X. These components contain lots of information about our data set. Create a scatter plot with each of the 995 rows of X projected onto the first two principal components. In other words, the horizontal axis should be v1, the vertical axis v2, and each individual should be projected onto the subspace spanned by v1 and v2. Your plot must use a different color for each population and include a legend.

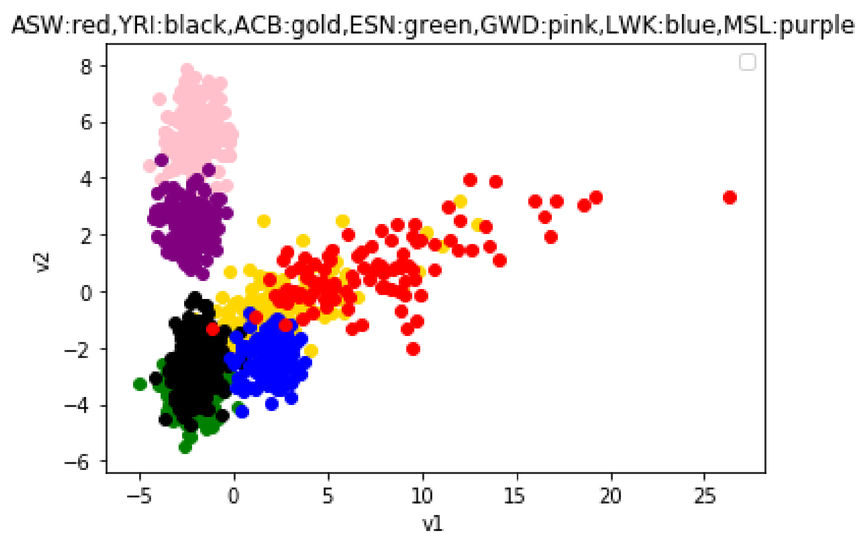
Firstly, process the initial data:

```
def processData(filename):
    ID = []
    sex = []
    population = []
    nucleobases = []
    file = open(filename,'r').readlines()
    for line in file:
        line = line.split(' ')
        ID.append(line[0])
        sex.append(line[1])
        population.append(line[2])
        line = line[3:-1]
        nucleobases.append(line)
    nucleobases = np.array(nucleobases)
    return ID,sex,population,nucleobases

def processNucleobases(nucleobases):
    most_fre_nuc = []
    for i in range(nucleobases.shape[1]):
        temp = nucleobases[:,i].tolist()
        nuc = Counter(temp)
        most_nuc,most_cnt = nuc.most_common(1)[0]
        most_fre_nuc.append(most_nuc)
    m,n = nucleobases.shape
    binary_matrix = np.zeros((m,n),dtype=int)
    for i in range(nucleobases.shape[1]):
        most_nuc = most_fre_nuc[i]
        for j in range(nucleobases.shape[0]):
            if nucleobases[j][i] != most_fre_nuc[i]:
                binary_matrix[j][i] = 1
            else:
                binary_matrix[j][i] = 0
    return binary_matrix
```

Secondly, ready to plot:

```
#part1 a b
ID,sex,population,nucleobase = processData("p4dataset2018.txt")
binary_matrix = processNucleobases(nucleobase)
print(binary_matrix.shape[0])
print(binary_matrix.shape[1])

#get the different kinds number
population_num = set(population)
print(len(population_num))
print(population_num)

def part1_b(binary_matrix,population,demesion):
    pca = PCA(demesion)
    nucleobase_pca = pca.fit_transform(binary_matrix)
    pkinds=['ASW','YRI','ACB','ESN','GWD','LWK','MSL']
    kinds = {'ASW':'red','YRI':'black','ACB':'gold','ESN':'green','GWD':'pink','LWK':'blue','MSL':'purple'}
    population_belong = {}
    for i in range(len(population)):
        population_belong[i] = population[i]
    fig, ax = plt.subplots()
    i = 0
    for individual in nucleobase_pca:
        popul = population[i]
        col = kinds[popul]
        ax.scatter(individual[0], individual[1], c = col)
        i+=1

    ax.legend()
    ax.set_title("ASW:red,YRI:black,ACB:gold,ESN:green,GWD:pink,LWK:blue,MSL:purple")
    ax.set_xlabel('v1')
    ax.set_ylabel('v2')
    plt.savefig("part1_b.png",format='png')
    plt.show()
    plt.close()

part1_b(binary_matrix,population,2)
```

Finally, get the result:



(c) (7 points) In two sentences, list 1 or 2 basic facts about the plot created in part (b). Can you interpret the first two principal components? What aspects of the data do the first two principal components capture? Hint: think about history and geography.

Solution:

Firstly, refer to related information, I got:

http://www.bubuko.com/infodetail-2021912.html

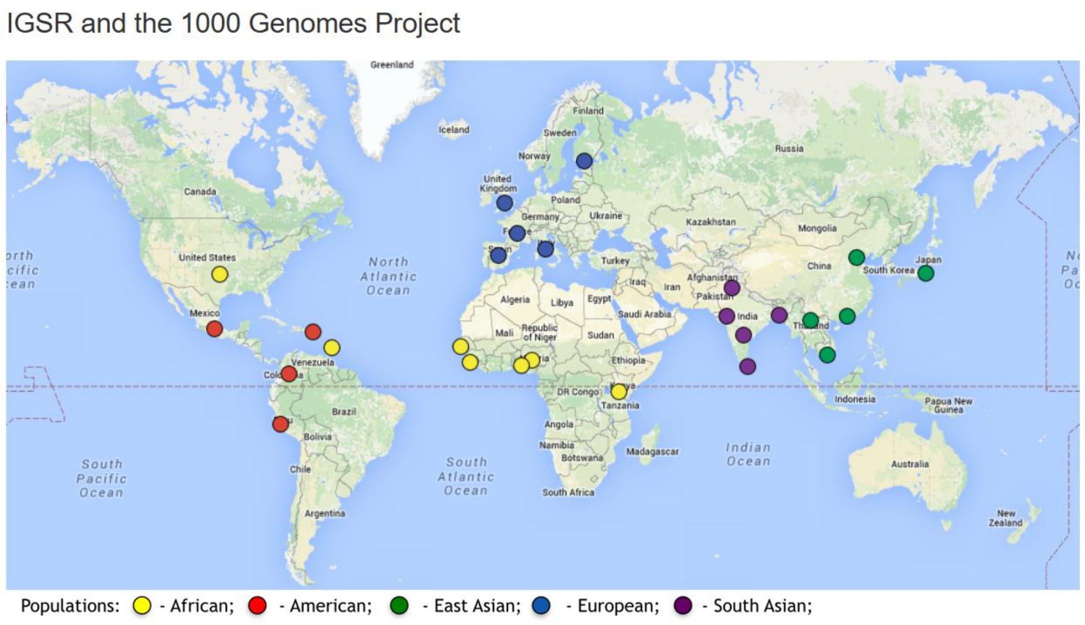
http://www.internationalgenome.org/faq/which-populations-are-part-your-study/

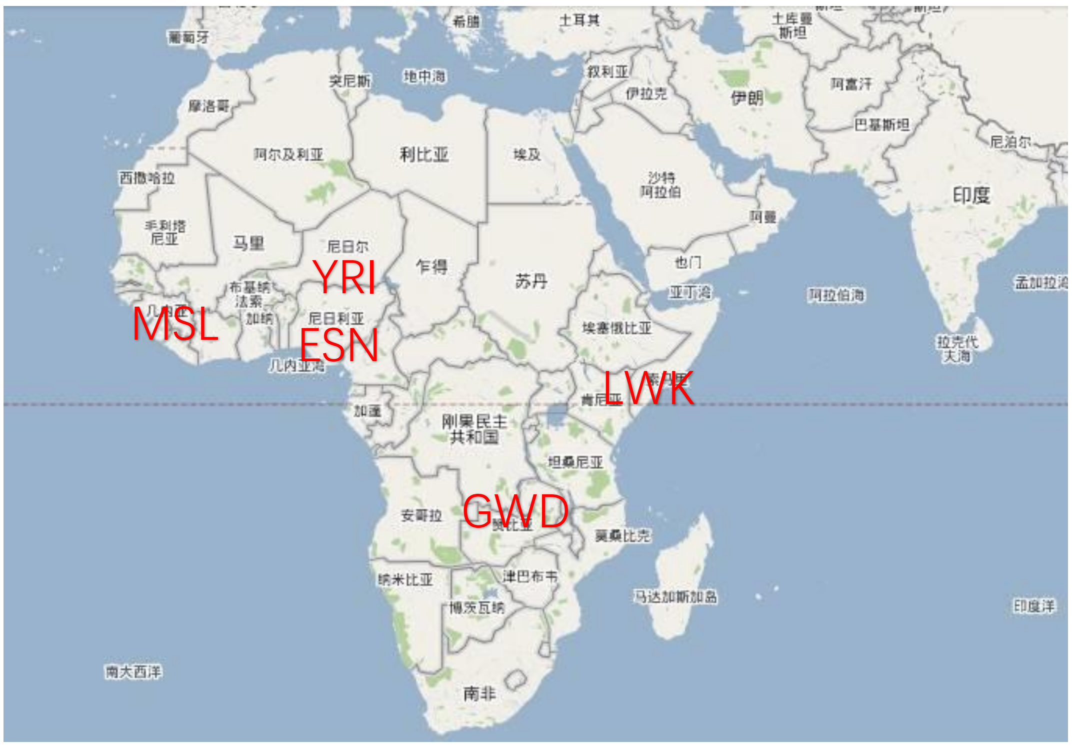
type	country	continent
ASW	非洲血统的美国人	America
YRI	约鲁巴，尼日利亚	Africa
ACB	非洲加勒比地区	North America
ESN	尼日利亚	Africa
GWD	冈比亚人	Africa
LWK	肯尼亚	Africa
MSL	塞拉利昂 曼德人	Africa

What I find from the table and plot:

It is clearly that red dots and golden dots scatter in the V1 direction, and the distribution of the two species is different from the other five. Looking at the table, it was found that the region corresponding to the red dot and the yellow dot is the region of the Americas, and the other five regions correspond to the region of Africa, indicating that the gene pools between each continent are different. It's true that Africa and America are geographically distributed.

Except for ASW and ACB, other five belong to Africa, and the difference among the four perform in V2. And we also can see green dots and black dots has the most overlapping parts. From table, we got both YRI and ESN have Nigeria people (it also prove that my PCA plot is right).


IGSR and the 1000 Genomes Project
Populations: ● - African; ● - American; ● - East Asian; ● - European; ● - South Asian;



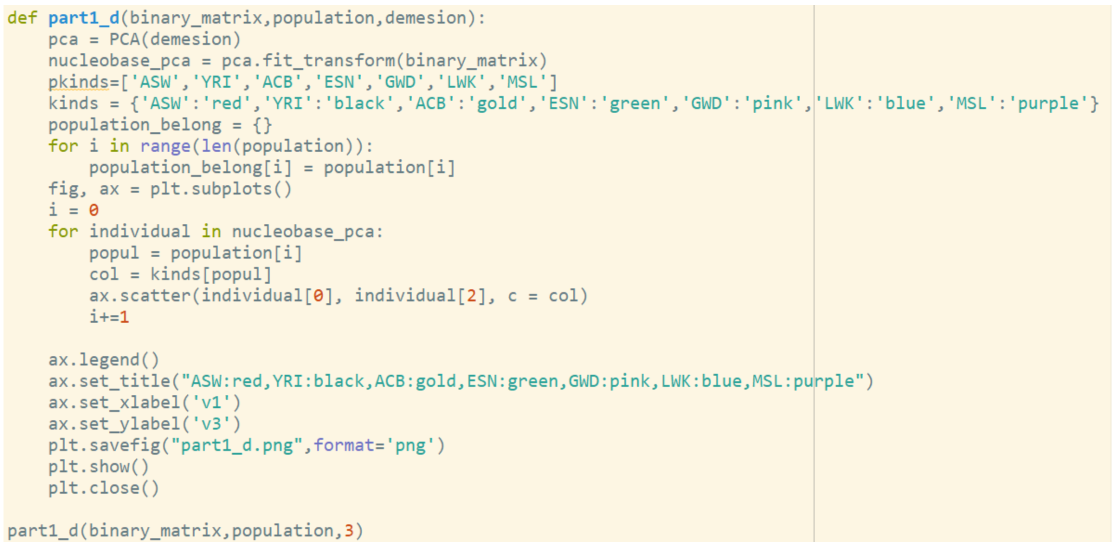
Conclusion:

So, I suppose V1 represents different continents (America and Africa) , V2 represents east and west of Africa.

(d) (5 points) We will now examine the third principal component of X. Create another scatter plot with each individual projected onto the subspace spanned by the first and third principal components. After plotting, play with different labeling schemes (with labels derived from the meta-data) to explain the clusters that you see. Your plot must include a legend.

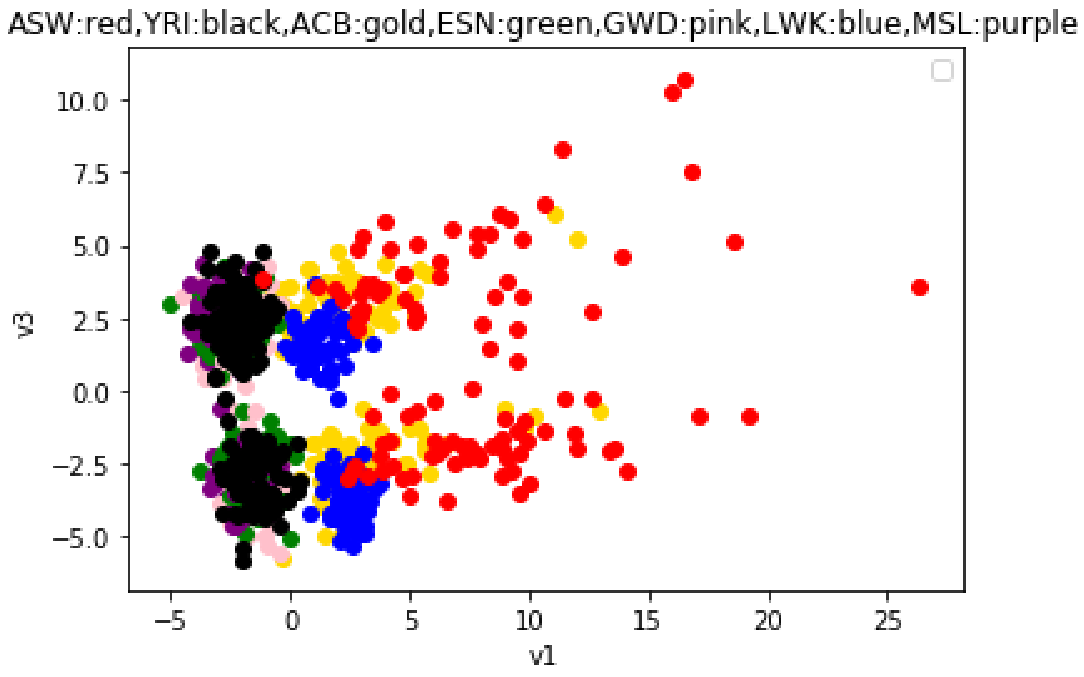
Solution:

Just need modify from part1_b:



```python
def part1_d(binary_matrix,population,demesion):
    pca = PCA(demesion)
    nucleobase_pca = pca.fit_transform(binary_matrix)
    pkinds=['ASW','YRI','ACB','ESN','GWD','LWK','MSL']
    kinds = {'ASW':'red','YRI':'black','ACB':'gold','ESN':'green','GWD':'pink','LWK':'blue','MSL':'purple'}
    population_belong = {}
    for i in range(len(population)):
        population_belong[i] = population[i]
    fig, ax = plt.subplots()
    i = 0
    for individual in nucleobase_pca:
        popul = population[i]
        col = kinds[popul]
        ax.scatter(individual[0], individual[2], c = col)
        i+=1

    ax.legend()
    ax.set_title("ASW:red,YRI:black,ACB:gold,ESN:green,GWD:pink,LWK:blue,MSL:purple")
    ax.set_xlabel('v1')
    ax.set_ylabel('v3')
    plt.savefig("part1_d.png",format='png')
    plt.show()
    plt.close()

part1_d(binary_matrix,population,3)
```

Finally, got result like this:



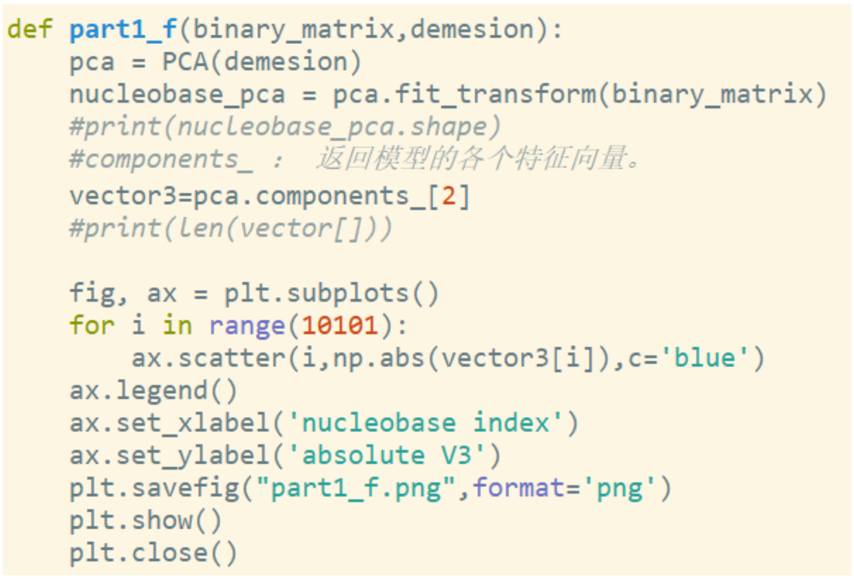
What I see from picture:

All of kinds are distributed to two parts. And for one kind, the number of two parts are nearly equal.

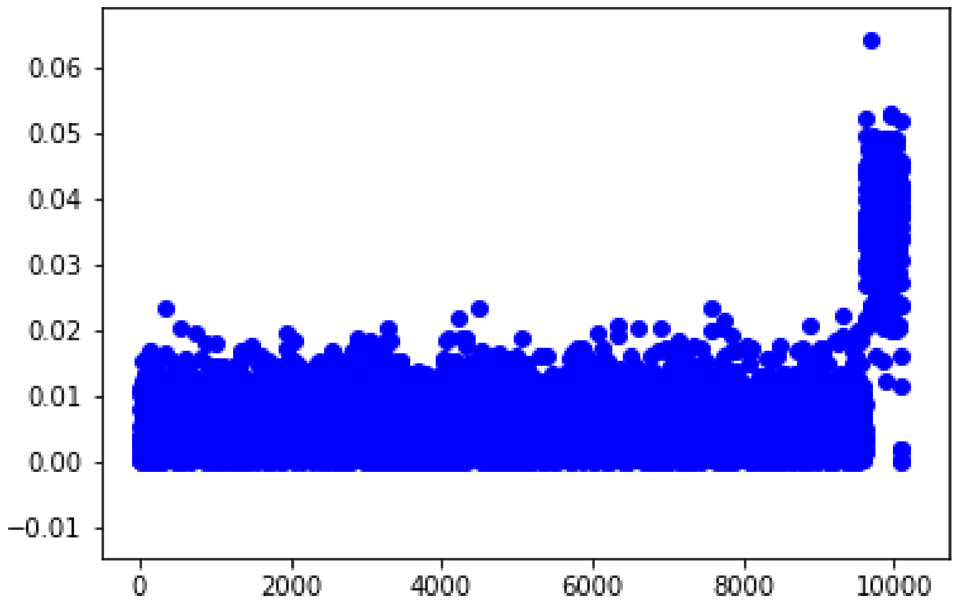
(e) (5 points) Something should have popped out at you in the plot above. In one sentence, what information does the third principal component capture?

Solution: It is so obvious that the third principal component is sex. (men or women)

(f) (4 points) In this part, you will inspect the third principal component. Plot the nucleobase index vs the absolute value of the third principal component. What do you notice? What's a possible explanation? Hint: think about chromosomes.



```python
def part1_f(binary_matrix,demesion):
    pca = PCA(demesion)
    nucleobase_pca = pca.fit_transform(binary_matrix)
    #print(nucleobase_pca.shape)
    #components_ :  返回模型的各个特征向量。
    vector3=pca.components_[2]
    #print(len(vector[]))

    fig, ax = plt.subplots()
    for i in range(10101):
        ax.scatter(i,np.abs(vector3[i]),c='blue')
    ax.legend()
    ax.set_xlabel('nucleobase index')
    ax.set_ylabel('absolute V3')
    plt.savefig("part1_f.png",format='png')
    plt.show()
    plt.close()
```



What I find:

There is a spike at the end of nucleobase index.

Explanation based on chromosomes:

There is a peak at the end, and we get the third dimension from the previous question to represent gender. So the peak may represent the chromosome that determines the gender. (XX or XY)

Part2

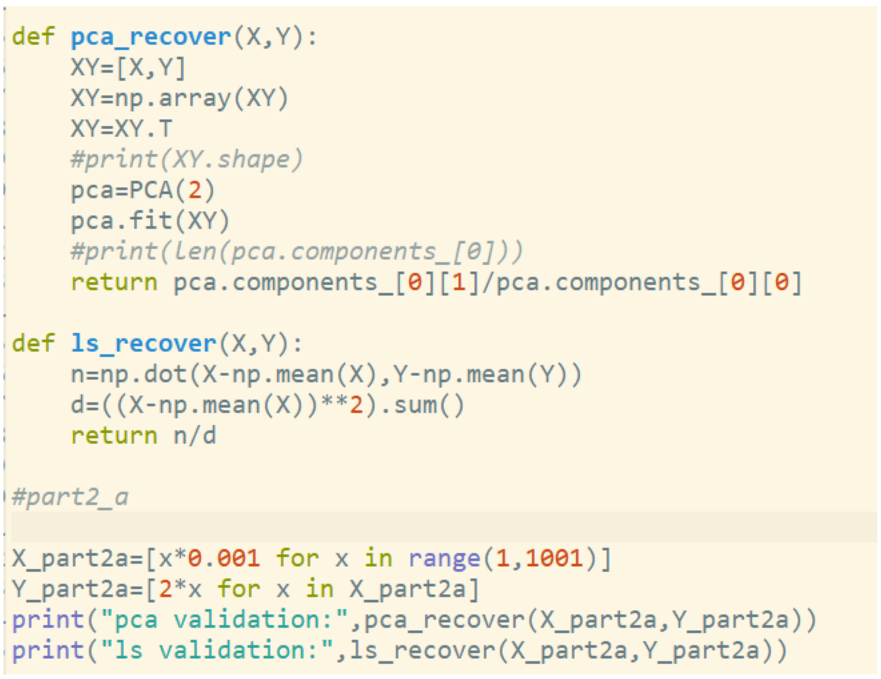
(a) Warm-up (do not submit):

• Write a routine pca-recover that takes a vector X of xi's and a vector Y of yi's and returns the slope of the first component of the PCA (namely, the second coordinate divided by the first).

• Write a routine ls-recover that takes X and Y and returns the slope of the least squares fit.

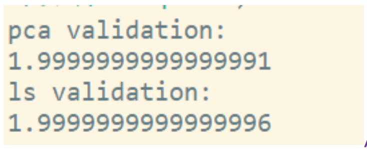
(Hint: since X is one dimensional, this takes a particularly simple form3: hX-X; Y -Y i=kX-Xk2 2,where X is the mean value of X.)

• Set X = [:001; :002; :003; : : : ; 1] and Y = 2X. Make sure both routines return 2.



```python
def pca_recover(X,Y):
    XY=[X,Y]
    XY=np.array(XY)
    XY=XY.T
    #print(XY.shape)
    pca=PCA(2)
    pca.fit(XY)
    #print(len(pca.components_[0]))
    return pca.components_[0][1]/pca.components_[0][0]

def ls_recover(X,Y):
    n=np.dot(X-np.mean(X),Y-np.mean(Y))
    d=((X-np.mean(X))**2).sum()
    return n/d

#part2_a

X_part2a=[x*0.001 for x in range(1,1001)]
Y_part2a=[2*x for x in X_part2a]
print("pca validation:",pca_recover(X_part2a,Y_part2a))
print("ls validation:",ls_recover(X_part2a,Y_part2a))
```

Result(validation):



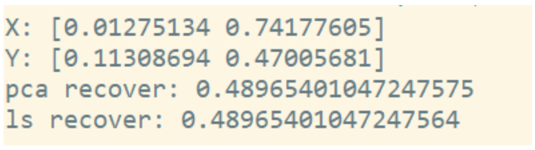
```
pca validation:
1.9999999999999991
ls validation:
1.9999999999999996
```

Approximately equal to 2

(b) (4 points) Say the elements of X and Y were chosen identically and independently at random (e.g.every element is uniformly distributed in the square [0; 1]×[0; 1]). What would PCA recover, and what would LS recover?

Solution:

I choose size=2, pca result and ls result like this:



```
X: [0.01275134 0.74177605]
Y: [0.11308694 0.47005681]
pca recover: 0.48965401047247575
ls recover: 0.48965401047247564
```

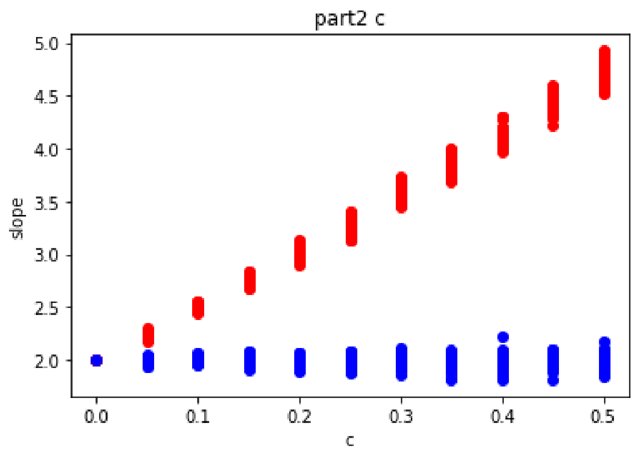
(c) (5 points) We first consider the case where x is an independent (a.k.a. explanatory) variable, and we get noisy measurements of y. Fix X = [x1; x2; : : : ; x1000] = [:001; :002; :003; : : : ; 1]. For a given noise level c, let ^ yi ~ 2xi + N (0; c) = 2i=1000 + N (0; c), and Yb = [^ y1; y^2; : : : ; y^1000]. Make a scatter plot with c on the horizontal axis, and the output of pca-recover and ls-recover on the vertical axis.For each c in [0; 0:05; 0:1; : : : ; :45; :5], take a sample Yb, plot the output of pca-recover as a red dot, and the output of ls-recover as a blue dot. Repeat 30 times. You should end up with a plot of 660 dots, in 11 columns of 60, half red and half blue. Hint: in both numpy and matlab, randn(1000)*σ generates an array of 1000 independent samples from N (0; σ2). In python you'll also need to add from numpy. random import randn.

Solution:

We first can code like this:

```
def part2_c_XY(c):
    X=[xx*0.001 for xx in range(1,1001)]
    X=np.array(X)

    Y=[2*X[i] for i in range(0,1000)]
    Y=np.array(Y)

    Xnoise=random.randn(1000)*(math.sqrt(c))
    Ynoise=random.randn(1000)*(math.sqrt(c))

    Xnoise+=X
    Ynoise+=Y

    return X,Xnoise,Ynoise

def part2_c():
    c=[cc*0.05 for cc in range(0,11)]
    times=30
    for i in range(times):
        for j in c:
            X,Xnoise,Ynoise=part2_c_XY(j)
            '''
            if j==0.05:
                print(Xnoise-X)
                break
            '''
            recover_pca=pca_recover(X,Ynoise)
            recover_ls =ls_recover(X,Ynoise)
            plt.plot(j, recover_pca, 'ro', label = 'pca',alpha=1)
            plt.plot(j, recover_ls, 'bo',label='ls',alpha=1)
    plt.title("part2 c")
    plt.xlabel("c")
    plt.ylabel("slope")
```

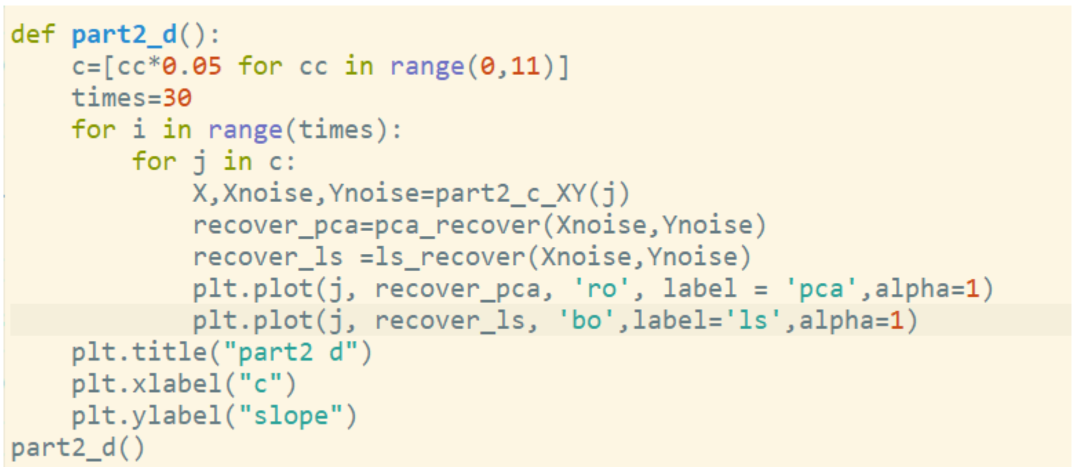
Then, we get the picture:

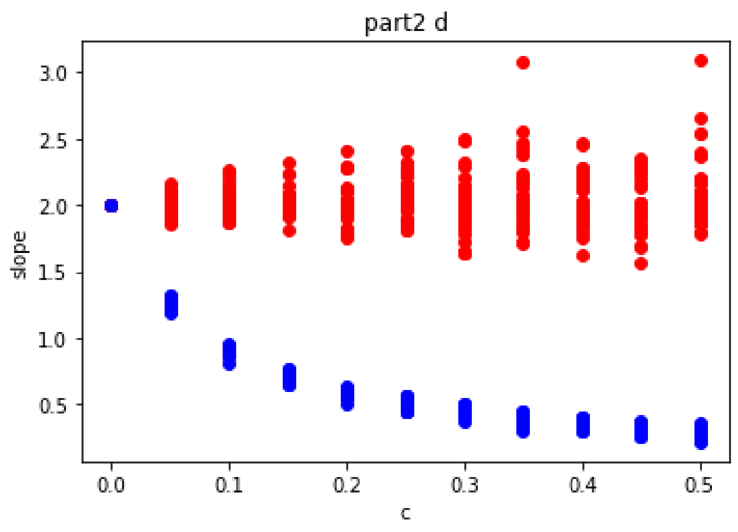


What I can find in this result:

What we can see from the picture is that ls_recover has better approximate mode. And with the noise become larger, the slope of pca_recover also become larger, which means it behaves worse.

(d) (5 points) We now examine the case where our data consists of noisy estimates of both x and y. For a given noise level c, let ^ xi ~ xi + N (0; c) = i=1000 + N (0; c) and ^ yi ~ yi + N (0; c) = 2i=1000 +N (0; c). Similar to (b), for each c in [0; 0:05; 0:1; : : : ; :45; :5], take a sample Xb and Yb, plot the output of pca-recover as a red dot, and the output of ls-recover as a blue dot. Repeat 30 times. You should have a plot with 330 red dots and 330 blue dots.


```
def part2_d():
    c=[cc*0.05 for cc in range(0,11)]
    times=30
    for i in range(times):
        for j in c:
            X,Xnoise,Ynoise=part2_c_XY(j)
            recover_pca=pca_recover(Xnoise,Ynoise)
            recover_ls =ls_recover(Xnoise,Ynoise)
            plt.plot(j, recover_pca, 'ro', label = 'pca',alpha=1)
            plt.plot(j, recover_ls, 'bo',label='ls',alpha=1)
    plt.title("part2 d")
    plt.xlabel("c")
    plt.ylabel("slope")
part2_d()
```



What I can find in this result:

Firstly, when c=0, which means there is no noise, both pca _recover and ls_recover return '2', it is correspondent to supposition. Secondly, we can see with the noise becoming larger, both methods have more errors. Specially ls recover method, it behaves worse.

(e) (9 points) Why does PCA do better in one, and least squares in the other? (No need to repeat the discussion above about latent vs. known independent variables.) Split this into three questions:

(i)Why does PCA do poorly with noise in only Y? (ii) Why does PCA do well with noise in X and Y?

(iii) Why does LS do poorly with noise in X and Y?

Solution:

1. When PCA only has Y noise, it adjusts the direction according to the distribution of Y noise, because it cares about the vertical distance to that boundary, which is the size of the covariance. So when there is only Y noise, PCA does not perform well.
2. Since both X and Y have noise and the noise is positive, from the linear space, all the points move in one direction, which does not affect the slope line where the vertical projection is found.
3. The least squares method works well with only Y noise, because the core of the least squares is the calculation of the variance, and the addition of noise does not affect the calculation of the minimum variance (noise is reduced). But when there are both X and Y noise, both factors are changing, the variance is changing, so the error is getting bigger.