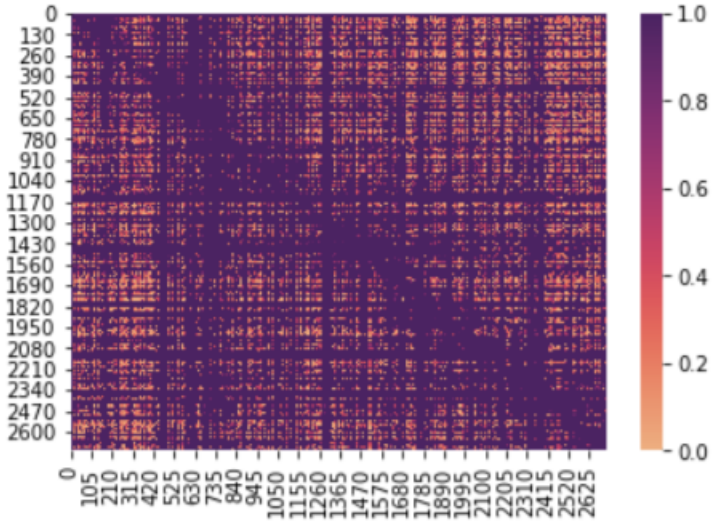
CSE 280A HW 1

Sharad Venkateswaran

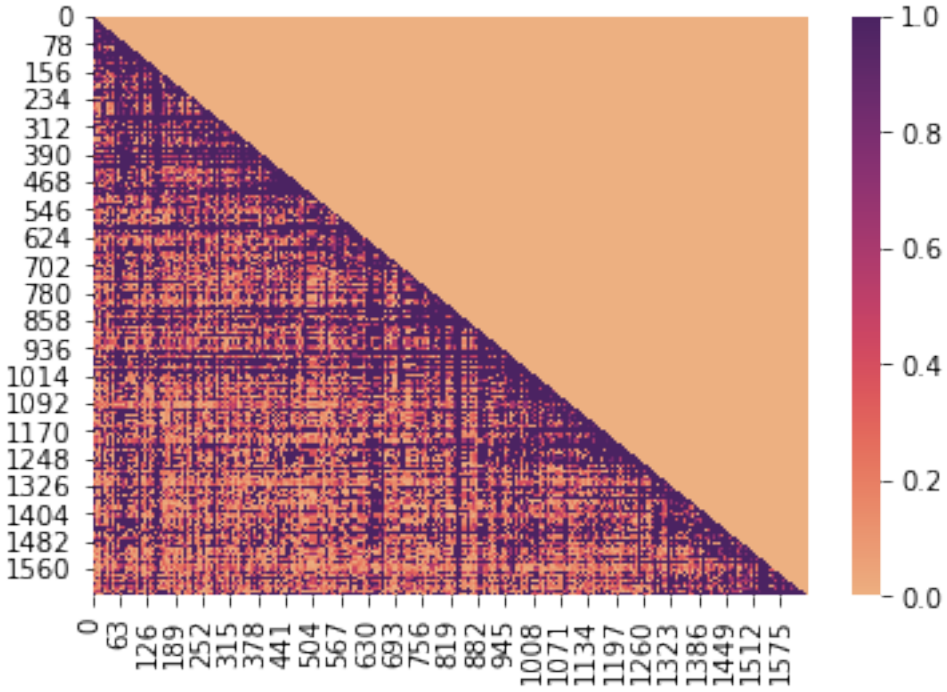
January 2021

Problem 1

I could not figure out how to plot the LD in the same way as the graphic, with the indices along the base of the triangle, so I instead plotted a lower-triangular heatmap of the matrix. Originally I plotted the full matrix using all the data, which resulted in this heatmap:

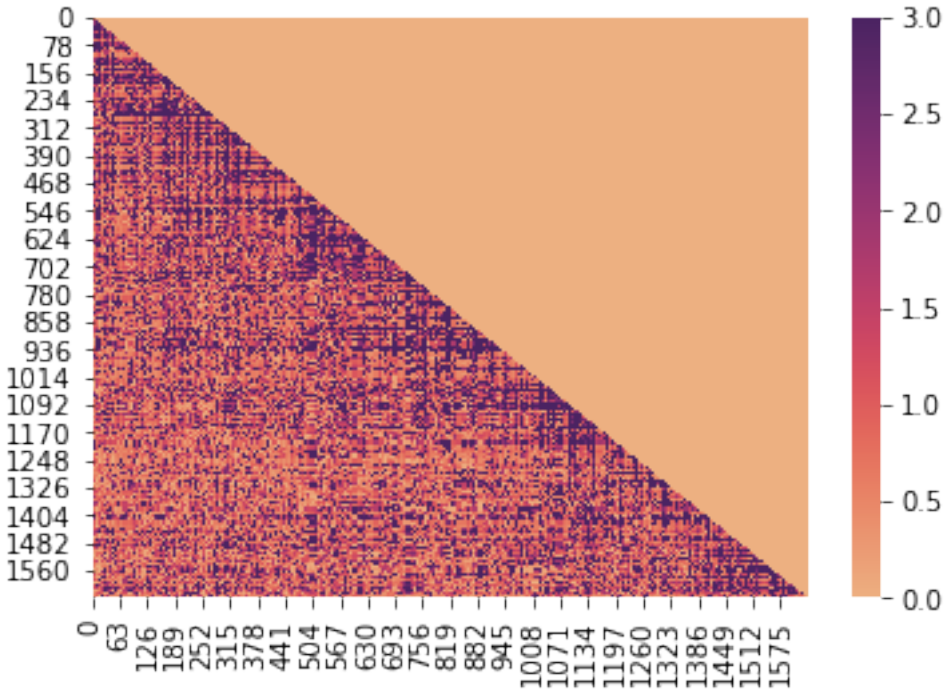


However, this heatmap is difficult to interpret - the mirrored nature makes it difficult to reason about behavior near the diagonal, and there are many straight columns and rows of dark purple which seem anomalous at first. These represent degenerate SNPs with very low minor allele frequency - since their MAF is so low their variability is low enough that a change at any other site will also predict the allele at these sites, since their values essentially don't change at all. As a result the LD of these sites is very high with every other site. To better see the signal (at the professor's recommendation) I removed these sites and plotted only the lower half of the resulting heatmap:



As we would expect, we see dark purple all along the diagonal since LD will tend to be higher between sites that are closer together. However, there are several patches along the diagonal that indicate larger areas with low recombination. For example, between 390 and 468 there is a patch of dark purple at the diagonal, which indicates that this may be a region of low recombination - by contrast 702 to 780 is multicolored throughout, and is therefore not indicative of lower recombination rates. The entire region from 936 to 1248 stands out as having more dark purple, which indicates that there may be several blocks in this region that have lower recombination rates. Similar applies to 1326 to 1482. Note that since I removed some of the sites due to their low MAF the site positions pictured here will not correlate directly with the original site positions in the datafile.

I also recomputed this matrix using $-log(p)$, shown below:



In this case p was found by computing $r^2 * N$ and treating this as a χ^2 value, then finding the corresponding p. Note that a $-log(p)$ value of ≈ 3 correlates with a p values of ≈ 0.05, which is generally considered the point of statistical significance. Therefore I capped my $-log(p)$ values at 3 for this plot, so any value at or below 3 will appear as dark purple here. The results are similar to the results using D', but seem more noisy. Values near the diagonal tend to be more significant than values elsewhere, which does not seem out of the ordinary, but there is little other rhyme or reason to the pattern in this plot.

Problem 2

For this problem we will consider '1' as the minor allele, because if 0 was the minor allele with frequency 0.01 then we would expect many instances of the 11 haplotype. Given that the allele frequency of 1 is 0.01 then $P_{1*} = 0.01$ and $P_{*1} = 0.01$. By extension, $P_{0*} = 0.99$ and $P_{*0} = 0.99$. Then we can compute $D = P_{11} - P_{1*} * P_{*1} = 0 - 0.01 * 0.01 = -0.0001$. With this we can compute $r = \frac{D}{\sqrt{P_{1*} * P_{*1} * P_{0*} * P_{*0}}} = \frac{-0.0001}{\sqrt{0.01 * 0.01 * 0.99 * 0.99}} = \frac{-0.0001}{0.0099}$. Then when $N = 10$, $r^2 N = (\frac{-0.0001}{0.0099})^2 * 10 = 0.00102$. We can treat this as a χ^2 statistic with 1 degree of freedom and compute a p-value from this. The corresponding p-value is ≈ 0.97, which is much greater than 0.05, so in this case we cannot reject the null hypothesis H_0, which is that the sites are in linkage equilibrium. However, when $N = 100000$ we instead find $r^2 N = (\frac{-0.0001}{0.0099})^2 * 100000 = 10.2$, for which

the corresponding p-value is 0.0014, which is less than 0.05, and therefore we can conclude that the sites are in LD. However, this is quite a large sample size, so when researching these relationships in situations where our sample size may be limited, it is much more difficult to make statistically significant conclusions about sites where the minor allele frequency is very low. In sites where the minor allele frequency is larger we can make statistically significant conclusions from much smaller samples, because the likelihood of not seeing any 11 haplotypes, for example, decreases dramatically as the frequency of allele 1 increases, and is therefore more readily explained by LD than by chance.

Problem 3

3a

0 corresponds to the "reference" allele, so I believe this also corresponds to the "ancestral" allele, but I may be mistaken. The data is phased, as denoted by the vertical bars | dividing each individuals' entry at each site. This divides the entry into the individuals' paternal and maternal alleles.

3b

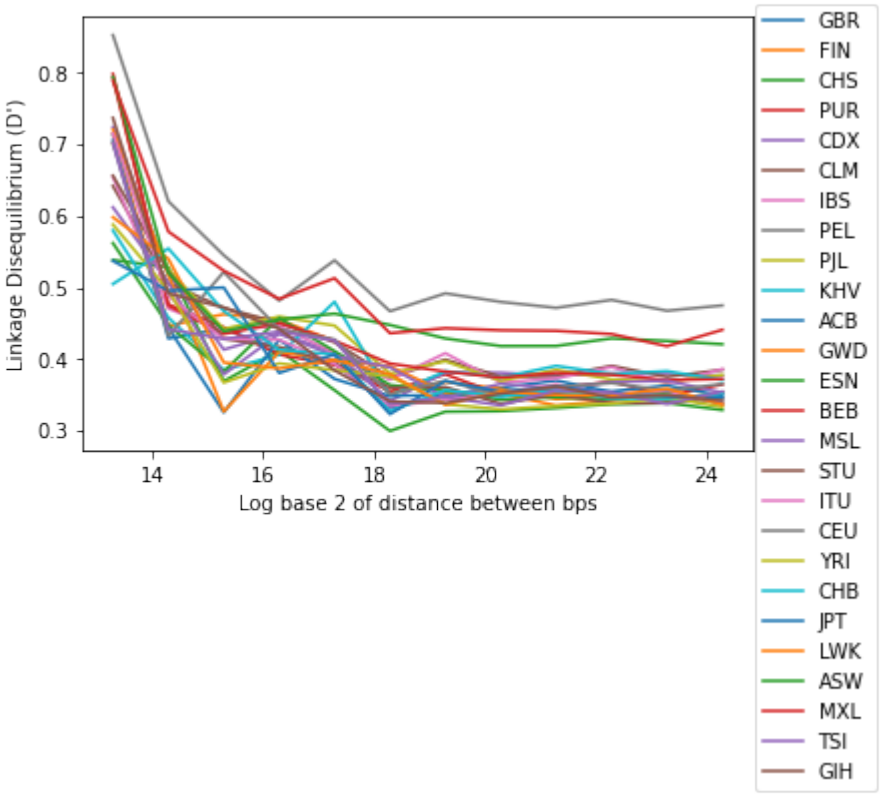
The number of haplotypes is 5008, with each individual contributing two haplotypes for a total of 2504 individuals. I'm not sure about unique haplotypes as I was not able to load the entire dataset of genotypes at once to analyze this.

3c

The number of all sites considered in the vcf file is 1103547, while the number of SNP sites specifically is 1060388. The highest position of all sites in the file is 51244237, while the lowest position of all sites is 16050075. Therefore the total number of base pairs covered in this file is $51244237 - 16050075 = 35194162$. The number of SNPs per base pair is then $\frac{1060388}{35194162} \approx 0.03$, so the number of SNPs per 1000 bps is ≈ 30. Across the human genome the average number of SNPs per 1000bps for an individual is around 1, however since our dataset is composed of 2504 individuals we should expect many more than this number of SNPs across the entire dataset. On the other hand, many individuals likely have variations at the same site, so we should not expect there to be 2504 times as many SNPs as there are in the average individual's genome. 30 SNPs per 1000 bp seems like a reasonable amount when taking this into account.

3d

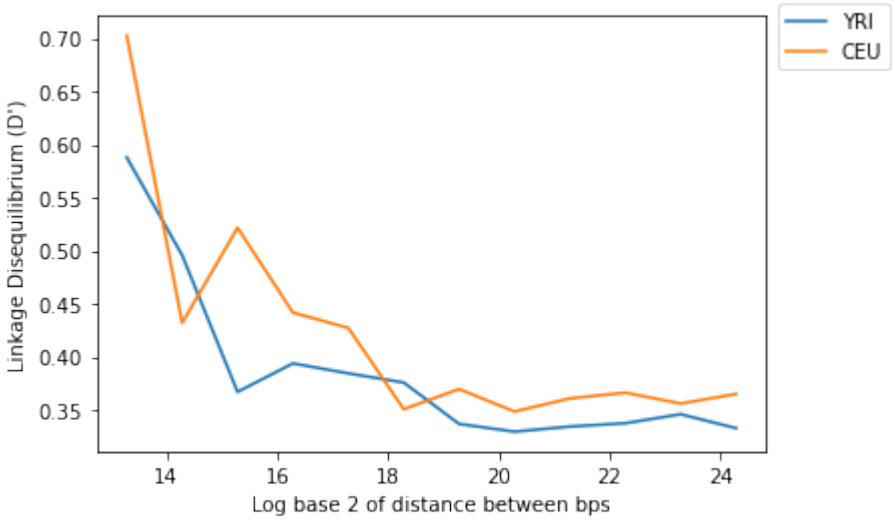
My plot for all populations is shown below (see code for implementation):



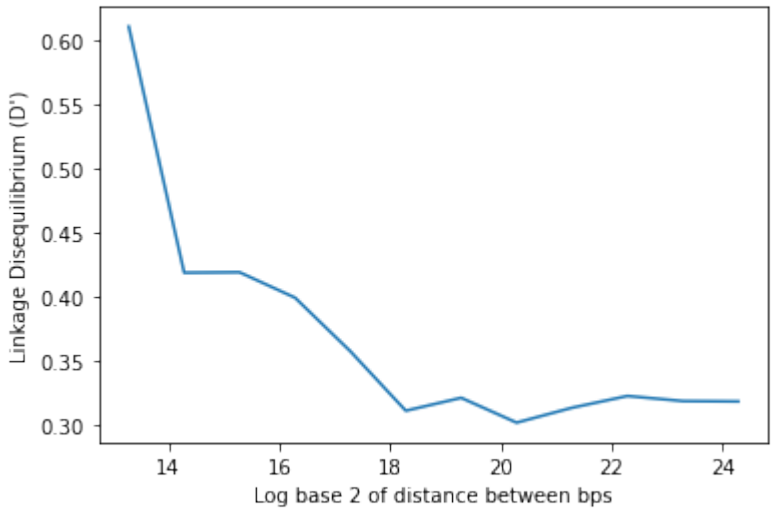
The results somewhat match the plots shown in class - the gray line at the top of the plot is CEU, which we expect to have high LD due to a more recent common ancestor, and therefore less time for recombination events that would increase LD. Similarly, YRI, i.e. the Yoruban population line, appears towards the bottom of the plot (though it is hard to find due to the sheer number of populations pictured). This makes sense as the Yoruban population's most recent common ancestor was further in the past than that of the other populations, and therefore they have had more time to accumulate recombination events and would therefore have lower LD. All populations LD decreases as distance between base pairs increases, which also matches expectations. However, we might expect GBR and FIN to also have similar trends to the CEU plot but they do not - this may be due to how I selected which data to use in the plot, or may be due to the CEU population simply having a more recent common ancestor than these populations. The number of sites I chose for this plot was only 340, due to the computational complexity of computing the D' measure, however even this small number of data points leads to many pairwise comparisons, which lead to an adequately rich dataset to demonstrate these trends.

3e

With the two populations separate CEU clearly has a higher LD in general compared to YRI. This matches our expectations, as the CEU population will have a more recent common ancestor and therefore will have had less chance for recombinations that would lead to linkage equilibrium. This relationship is not as stark as was shown in class - this is likely due to the limited amount of data I was able to work with at once while maintaining reasonable runtimes for my code, but the general trend is still apparent.



If we mix the two populations then we would expect a less spiky line that somewhat averages these two curves out. The resultant plot is shown below:



Indeed this result appears to match expectations. The starting point of the curve is somewhere between the two starting points, though it is closer to the Yoruban curve's starting point. The sharp discrepancy at $x \approx 15$ has split the difference between these two extremes and resulted in a smoother curve down to $x = 18$, where both of the original curves met. At a first glance the curve appears to then follow the original CEU curve, but the y-axis is not the same for both plots. In fact the mixed curve seems closer to the Yoruban curve towards the end, but in general when the data is mixed the results appear similar to the average of the two original curves. This is maybe most apparent at $x = 24$ - when the CEU curve trends upwards and the YRI curve trends downwards the mixed curve is straight and steady.

Problem 4

Hint part 1

Hint: Given a lineage in epoch k the probability that this lineage has i descendants is equal to the probability that the other $k-1$ lineages together cover the other $n-i$ descendants in the sample. The probability of this can be expressed as the number of ways for these $k-1$ lineages to cover these $n-i$ descendants divided by the number of possible configurations of the n descendants over all k lineages. The number of ways to distribute $n-i$ descendants among $k-1$ lineages is simply the number of ways to divide $n-i$ items into $k-1$ non-empty groups which we can think of as the number of ways to place $k-2$ identical barriers between $n-i$ items. There are $n-i-1$ spaces for the $k-2$ barriers, so this quantity is $\binom{n-i-1}{k-2}$. The number of ways to distribute all n individuals among all k lineages is then $\binom{n-1}{k-1}$ We then reach the desired form $p_{ki} = \frac{\binom{n-i-1}{k-2}}{\binom{n-1}{k-1}}$

Hint part 2

The goal is now to show that $\frac{\binom{n-i-1}{k-2}}{\binom{n-1}{k-1}} = \frac{\binom{n-k}{i-1}}{\binom{n-1}{i}} * \frac{k-1}{i}$. I will first rewrite each side according to the definition of $\binom{n}{r}$, then perform several algebraic rewrites to show equality (below **WTS** means "want to show")

$$\textbf{WTS} \quad \frac{\binom{n-i-1}{k-2}}{\binom{n-1}{k-1}} = \frac{\binom{n-k}{i-1}}{\binom{n-1}{i}} * \frac{k-1}{i}$$

$$\textbf{WTS} \quad \frac{\frac{(n-i-1)!}{(k-2)!(n-i-k+1)!}}{\frac{(n-1)!}{(k-1)!(n-k)!}} = \frac{\frac{(n-k)!}{(i-1)!(n-i-k+1)}}{\frac{(n-1)!}{i!(n-i-1)!}} * \frac{k-1}{i}$$

$$\textbf{WTS} \quad \frac{(n-i-1)!(k-1)!(n-k)!}{(k-2)!(n-i-k+1)!(n-1)!} = \frac{(n-k)!(n-i-1)!(i)!}{(n-1)!(i-1)!(n-k-i+1)!} * \frac{k-1}{i}$$

$$\textbf{WTS} \quad \frac{(n-i-1)!(k-1)(k-2)!(n-k)!}{(k-2)!(n-i-k+1)!(n-1)!} = \frac{(n-k)!(n-i-1)!i(i-1)!}{(n-1)!(i-1)!(n-k-i+1)!} * \frac{k-1}{i}$$

WTS $\frac{(n-i-1)!(n-k)!(k-1)}{(n-i-k+1)!(n-1)!} = \frac{(n-k)!(n-i-1)!(k-1)}{(n-1)!(n-k-i+1)!}$

This is clearly true, so the statement $\frac{\binom{n-i-1}{k-2}}{\binom{n-1}{k-1}} = \frac{\binom{n-k}{i-1}}{\binom{n-1}{i}} * \frac{k-1}{i}$ is true as well.

$Exp(m_i)$

Given the second formula for p_{ki} given in the homework, we can demonstrate that $Exp(m_i) = \frac{\theta}{i}$ as follows: To find the number of columns with exactly i ones we want to find the number of mutations that lie on lineages in the topology that have exactly i descendants.

We can express this as the sum over all epochs k of the sum over all branches j in that epoch of the probability that branch j has i descendants times the expected number of mutations on branch j.

We have that the likelihood that the branch has i descendants is $\frac{\binom{n-k}{i-1}}{\binom{n-1}{i}} * \frac{k-1}{i}$

We have from class that the expected number of mutations on a branch in epoch k is $\frac{4N\mu}{k(k-1)} = \frac{\theta}{k(k-1)}$.

Then $Exp(m_i) = \sum_{k=2}^{n} \sum_{j-1}^{k} \frac{\binom{n-k}{i-1}}{\binom{n-1}{i}} * \frac{k-1}{i} * \frac{\theta}{k(k-1)}$.

Since j does not appear in the inner sum we can instead multiply the inner term by k to get $Exp(m_i) = \sum_{k=2}^{n} k * \frac{\binom{n-k}{i-1}}{\binom{n-1}{i}} * \frac{k-1}{i} * \frac{\theta}{k(k-1)}$.

Cancelling terms we then get $Exp(m_i) = \sum_{k=2}^{n} \frac{\binom{n-k}{i-1}}{\binom{n-1}{i}} * \frac{\theta}{i}$.

What is left to show is that $\sum_{k=2}^{n} \frac{\binom{n-k}{i-1}}{\binom{n-1}{i}} = 1$, or alternatively, that

$$\sum_{k=2}^{n} \binom{n-k}{i-1} = \binom{n-1}{i}$$

. Note that when $k > n-i+1, n-k < i = 1$, so $\binom{n-k}{i-1} = 0$. Therefore we can rewrite our goal as

$$\sum_{k=2}^{n-i+1} \binom{n-k}{i-1} = \binom{n-1}{i}$$

. From Pascal's Rule we get that $\binom{n-1}{i} = \binom{n-2}{i-1} + \binom{n-2}{i}$. We can apply this again to get $\binom{n-2}{i-1} + \binom{n-2}{i} = \binom{n-2}{i-1} + \binom{n-3}{i-1} + \binom{n-3}{i}$. Recursively applying this until the new term is equal to 0 gives us the desired sum such that $\binom{n-1}{i} = \sum_{k=2}^{n-i+1} \binom{n-k}{i-1}$.

In [1]:
```
import string
import time
import math
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```

In [4]:
```
# Note - I removed nonbinary lines and the last empty line in the pop2.txt file
def lineSplit(line):
    retList = []
    for c in line:
        retList.append(int(c))

    return retList

data1 = open("./pop2.txt")
data = [lineSplit(line) for line in data1.read().splitlines()]
data_np = np.array(data)
print(data_np.shape)
columnsToDelete = []
print(data_np.shape[1])

# At the professor's recommendation I removed sites with very low MAF,
# in this case those sites with less than 5 occurrences of the minor allele.
for i in range(data_np.shape[1]):
    if sum(data_np[:,i]) < 5:
        columnsToDelete.append(i)
data_np = np.delete(data_np, columnsToDelete, axis=1)
print(data_np.shape)

```

```
(200, 2718)
2718
(200, 1637)
```

In [3]:

```
def LD(col1, col2):
    P0_ = 0
    P_0 = 0
    P00 = 0
    P11 = 0
    N = len(col1)

    for i in range(N):
        if col1[i] == 0:
            P0_ += 1
            if col2[i] == 0:
                P00 += 1
                P_0 += 1
        else:
            if col2[i] == 1:
                P11 += 1
            else:
                P_0 += 1

    P0_ = float(P0_) / N
    P_0 = float(P_0) / N
    P00 = float(P00) / N
    P11 = float(P11) / N
    P1_ = 1 - P0_
    P_1 = 1 - P_0

    #print("P0* = ", P0_)
    #print("P*0 = ", P_0)
    #print("P00 = ", P00)
    #print("P11 = ", P11)
    #print("P1* = ", P1_)
    #print("P*1 = ", P_1)

    D = P00 - P0_*P_0
    D_max = 0

    if(D >= 0):
        D_max = min(P0_*P_1,P1_*P_0)
    else:
        D_max = min(P0_*P_0, P1_*P_1)
```

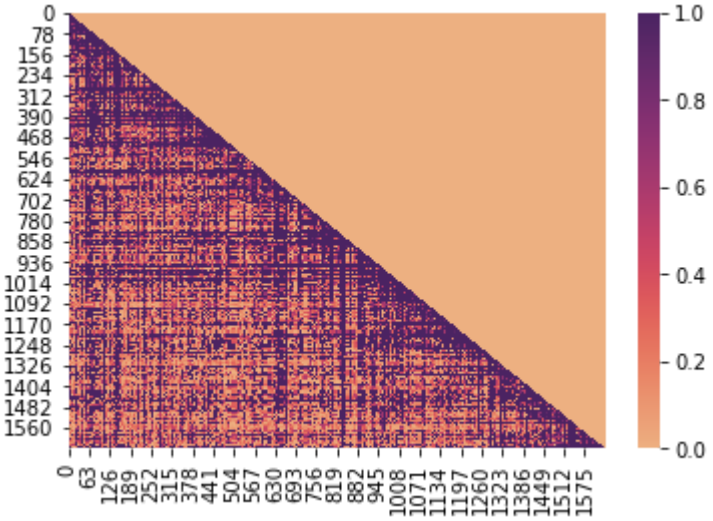
```
    D_prime = float(abs(D) / D_max)

    r = D / math.sqrt(P1_*P0_*P_1*P_0)
    p_val = stats.chi2.sf(math.pow(r,2)*N, 1)


    return (D_prime, math.log(p_val) * -1)
    #return D
```

In [7]:

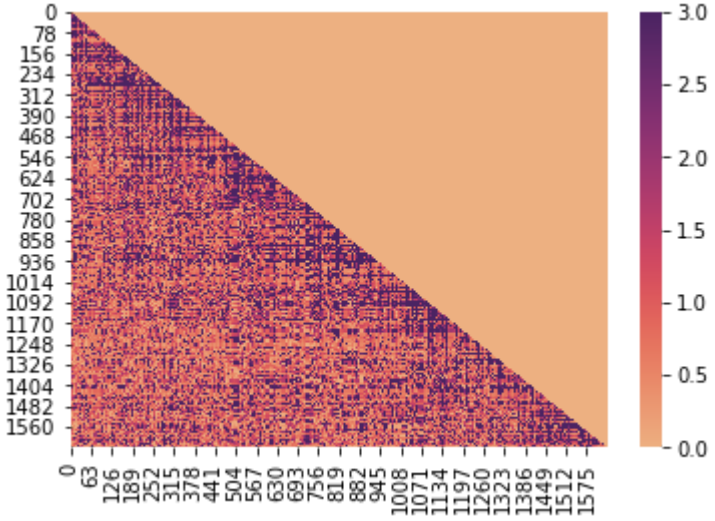
```
LD_mat_D = np.zeros((data_np.shape[1], data_np.shape[1]))
LD_mat_P = np.zeros((data_np.shape[1], data_np.shape[1]))
try:
    LD_mat_D = np.load("Problem1LD_D.npy")
    LD_mat_P = np.load("Problem1LD_P.npy")

except:
    for i in range(LD_mat_D.shape[0]):
        if i % 100 == 0:
            print(i)
        for j in range(0, i):
            LD_mat_D[i][j], LD_mat_P[i][j] = LD(data_np[:,i], data_np[:,j])


    np.save("Problem1LD_D", LD_mat_D)
    np.save("Problem1LD_P", LD_mat_P)
```

```
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
```

In [8]:
```
1 ax1 = sns.heatmap(LD_mat_D, cmap="flare")
```



In [9]:
```
# p < 0.05 will result in -log(p) > 3, so cap heatmap range there
# to find statistically significant data
ax2 = sns.heatmap(LD_mat_P, cmap="flare", vmax=3)
```



In [2]:
```
import allel
import numpy as np
import csv
import math
import matplotlib.pyplot as plt
import scipy.stats as stats
import random
from collections import defaultdict
```

In [3]:
```
# Read only the data needed for parts a through c
data = allel.read_vcf("ALL.chr22.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf",fields=['va
```

In [4]:
```
print(data.keys())
```

```
dict_keys(['variants/AA', 'variants/AF', 'variants/AN', 'variants/POS', 'variants/VT'])
```

In [5]:
```
print("Earliest position in file: ", min(data['variants/POS']))
print("Latest position in file: ", max(data['variants/POS']))

#This file considers SNPs between the positions 16050075 - 51244237
#Number of base pairs covered is then
numBasePairs = 51244237 - 16050075
print("Number of base pairs covered: ", numBasePairs)

#The total number of sites is 1103547
numSites = len(data['variants/VT'])
print("Number of sites (not just SNPs): ", numSites)

#While the number of SNP sites is 1060388
numSNPS = list(data['variants/VT']).count("SNP")
print("Number of SNP sites: ", numSNPS)

#Then the number of SNP sites per 1000 base pairs is
print("Number of SNPs per 1000 base pairs: ", numSNPS/numBasePairs * 1000)
```

```
Earliest position in file:  16050075
Latest position in file:  51244237
Number of base pairs covered:  35194162
Number of sites (not just SNPs):  1103547
Number of SNP sites:  1060388
Number of SNPs per 1000 base pairs:  30.129656162860194
```

In [6]:
```
# Get sample names from vcf file
_,s,_,_ = allel.iter_vcf_chunks("ALL.chr22.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf",
                                fields=['variants/POS','variants/VT','variants/AN','variants/AA','variants/
                                chunk_length=1)
```

In [7]:
```
# Read tsv file to get the population code for each sample
tsv_file = open("igsr_samples.tsv")
read_tsv = csv.reader(tsv_file, delimiter="\t")
sample_to_population = {}
next(read_tsv)
for line in read_tsv:
    sampleName = line[0]
    populationCode = line[3]
    sample_to_population[sampleName] = populationCode

```

In [8]:
```
# Map each sample in the vcf to its population code according to tsv mapping
# Also find the indices of all members of each population so we can extract individual
# populations from the matrix later
unique_populations = []
population_to_samples = defaultdict(list)
population_indices = defaultdict(list)
for i in range(len(s)):
    sample = s[i]
    populationCode = sample_to_population[sample]
    population_indices[populationCode].append(i)
    population_to_samples[populationCode].append(sample)
```

In [12]:

```
# Iterator for vcf file
_,_,_,data_it = allel.iter_vcf_chunks("ALL.chr22.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.v
                                      fields=['variants/POS','variants/VT','variants/AN','variants/AA','variants/A
                                      chunk_length=1)
totalVariants = 1103547
random.seed(0)
def getNext(dataIterator):
    nextTup = next(dataIterator)
    return nextTup[0]
count = 0
population_data = defaultdict(list)

positions = []

for i in range(totalVariants):
    data_tup = getNext(data_it)

    # Only consider SNPs
    if(data_tup['variants/VT'][0] != 'SNP'):
        continue

    # Ignore SNPs with low MAF
    if(data_tup['variants/AF'][0][0] > 0.05):

        # To reduce amount of data considered without biasing towards earlier sites
        if math.floor(random.random() * 300) != 1:
            continue
        count += 1
        genotype = data_tup['calldata/GT'][0]
        # Split the genotype into  paternal / maternal haplotypes, since data is phased
        haplotype1 = genotype[:,0].astype('int')
        haplotype2 = genotype[:,1].astype('int')

        position = data_tup['variants/POS']

        positions.append(position[0])
        positions.append(position[0]) # Each haplotype is at the same position

        # For each population, store the samples belonging to that population
        for key in population_indices.keys():
            population_data[key].append(np.take(haplotype1, population_indices[key], 0))
            population_data[key].append(np.take(haplotype2, population_indices[key], 0))
```

```
43
44 print("Considering " + str(count) + " SNP sites")
45 # Number of variants with MAF > 0.05 = 114322
```

```
Considering 1 SNP sites
```

In [357]:

```
1 # x-values for plot. For each candidate pair of columns, check if their
2 # distance is close to one of these points.
3 datapoints = [10000,20000,40000,80000,160000,320000,640000, 1280000, 2560000, 5120000, 10240000, 20480000]
4 def checkDatapoint(distance, threshold):
5     for datapoint in datapoints:
6         if abs(datapoint - distance) < (datapoint * threshold):
7             return datapoint
8     return -1
```

In [371]:

```
population_LD_x = defaultdict(list)
population_LD_y = defaultdict(list)
population_LD = defaultdict(lambda: defaultdict(list))
threshold = 0.1
for popCode in population_data.keys():
    data = np.array(population_data[popCode]).T
    for i in range(data.shape[1]):
        # Don't compare haplotypes at the same position
        for j in range(i + 2 - i%2, data.shape[1]):
            distance = positions[j] - positions[i]
            # Do these sites fit in one of the distance buckets?
            dp = checkDatapoint(distance, threshold)
            if dp == -1:
                continue
            # Compute LD for these sites using D' measure
            D_prime = LD(data[:,i], data[:,j])[0]
            # D_max may be 0 if all values in both sites are the same due to population split
            if D_prime == -1:
                continue
            population_LD[popCode][dp].append(D_prime)

#How many datapoints do we have for each bucket?
for i in range(len(datapoints)):
    print(datapoints[i])
    print((population_LD_x['GBR'].count(datapoints[i])))
```

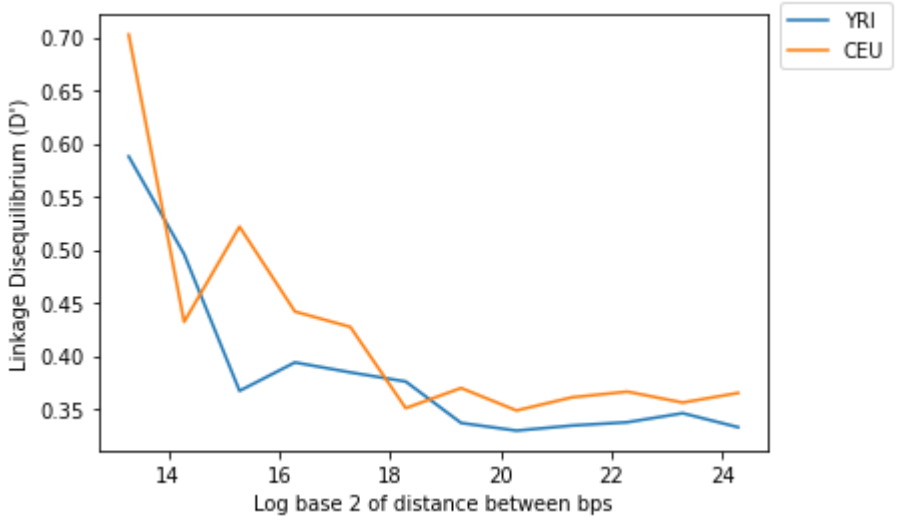
```
10000
27
20000
60
40000
90
80000
207
160000
367
320000
722
640000
1224
1280000
2822
```

```
2560000
4861
5120000
8743
10240000
13165
20480000
18635
```

In [372]:

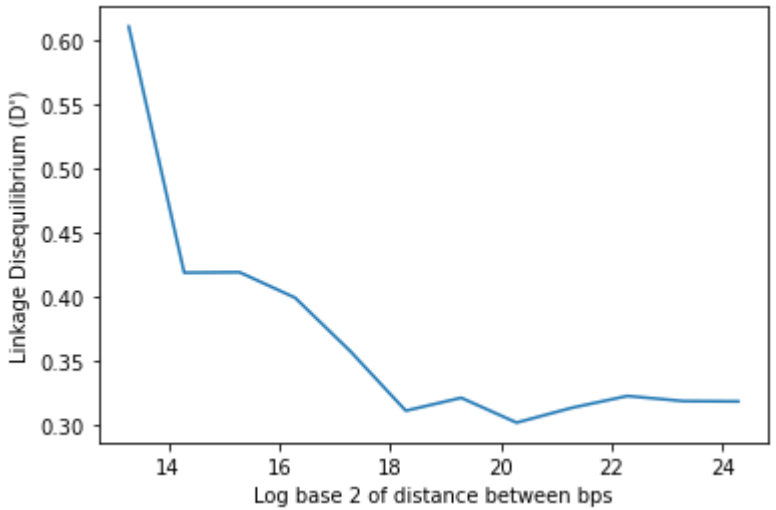
```
def LD(col1, col2):
    P0_ = 0
    P_0 = 0
    P00 = 0
    P11 = 0
    N = len(col1)

    for i in range(N):
        if col1[i] == 0:
            P0_ += 1
            if col2[i] == 0:
                P00 += 1
                P_0 += 1
        else:
            if col2[i] == 1:
                P11 += 1
            else:
                P_0 += 1

    P0_ = float(P0_) / N
    P_0 = float(P_0) / N
    P00 = float(P00) / N
    P11 = float(P11) / N
    P1_ = 1 - P0_
    P_1 = 1 - P_0

    D = P00 - P0_*P_0
    D_max = 0

    if(D >= 0):
        D_max = min(P0_*P_1,P1_*P_0)
    else:
        D_max = min(P0_*P_0, P1_*P_1)
    if D_max == 0:
        return (-1,-1)
    D_prime = float(abs(D) / D_max)

    r = D / math.sqrt(P1_*P0_*P_1*P_0)
    p_val = stats.chi2.sf(math.pow(r,2)*N, 1)


    return (D_prime, math.log(p_val) * -1)
```

In [387]:
```
x_vals = [math.log(datapoint, 2) for datapoint in datapoints]
y_vals_dict = defaultdict(lambda: defaultdict(list))
for popCode in ['YRI','CEU']: # change to population_LD.keys() to plot all
    for datapoint in datapoints:
        y_vals_dict[popCode][datapoint] = sum(population_LD[popCode][datapoint]) / len(population_LD[popCod
    y_vals = [y_vals_dict[popCode][datapoint] for datapoint in datapoints]
    plt.plot(x_vals, y_vals, label = popCode)
    plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1.05))
    plt.ylabel("Linkage Disequilibrium (D\')")
    plt.xlabel("Log base 2 of distance between bps")
```



In [396]:
```
# Problem 3e mixing YRI and CEU
data = np.concatenate((np.array(population_data['CEU']).T, np.array(population_data['YRI']).T))
for i in range(data.shape[1]):
    for j in range(i + 2 - i%2, data.shape[1]):
        distance = positions[j] - positions[i]
        dp = checkDatapoint(distance, threshold)
        if dp == -1:
            continue
        D_prime = LD(data[:,i], data[:,j])[0]
        if D_prime == -1:
            continue
        population_LD[popCode][dp].append(D_prime)

```

In [400]:
```
x_vals = [math.log(datapoint, 2) for datapoint in datapoints]
y_vals_dict = defaultdict(lambda: defaultdict(list))
for popCode in ['CEU']:
    for datapoint in datapoints:
        y_vals_dict[popCode][datapoint] = sum(population_LD[popCode][datapoint]) / len(population_LD[popCod
    y_vals = [y_vals_dict[popCode][datapoint] for datapoint in datapoints]
    plt.plot(x_vals, y_vals, label = popCode)
    plt.ylabel("Linkage Disequilibrium (D\')")
    plt.xlabel("Log base 2 of distance between bps")
```



In [ ]:
```
1
```