Part 1: Analyzing ancestry using principal components analysis (4 points)

Exercise 1

Option 1:

```
def perform_pca(ref_matrix_norm):
    """Perform PCA on normalized matrix

    Args:
        ref_matrix_norm (np.array): Float matrix of normalized SNPs, size numsnps x numsamples

    Returns:
        ref_matrix_projection (np.array): Float matrix of data projected onto PCs. size numsamples       x numdim (2)
    """
    ndim = 2
    # Get GRM (covariance matrix)
    grm = ref_matrix_norm.transpose().dot(ref_matrix_norm)/ref_matrix_norm.shape[0]
    # Get Eigendecomposition - column i is ith eigenvector
    evals, evecs = np.linalg.eig(grm)
    # sort eigenvalue in decreasing order
    idx = np.argsort(evals)[::-1]
    evecs = evecs[:,idx]
    # sort eigenvectors according to same index
    evals = evals[idx]
    # select the first n eigenvectors (n is desired dimension
    # of rescaled data array, or dims_rescaled_data)
    evecs = evecs[:, :ndim]
    return evecs
```

Option 2:

```
def perform_pca(ref_matrix_norm):
    """Perform PCA on normalized matrix

    Args:
        ref_matrix_norm (np.array): Float matrix of normalized SNPs, size numsnps x numsamples

    Returns:
        ref_matrix_projection (np.array): Float matrix of data projected onto PCs. size numsamples       x numdim (2)
    """
    ndim = 2
    # Get GRM (covariance matrix)
    grm = ref_matrix_norm.dot(ref_matrix_norm.transpose())/ref_matrix_norm.shape[1]
    # Get Eigendecomposition - column i is ith eigenvector
    evals, evecs = np.linalg.eig(grm)
    # sort eigenvalue in decreasing order
    idx = np.argsort(evals)[::-1]
    evecs = evecs[:,idx]
    # sort eigenvectors according to same index
    evals = evals[idx]
    # select the first n eigenvectors (n is desired dimension
    # of rescaled data array, or dims_rescaled_data)
    evecs = evecs[:, :ndim]
```

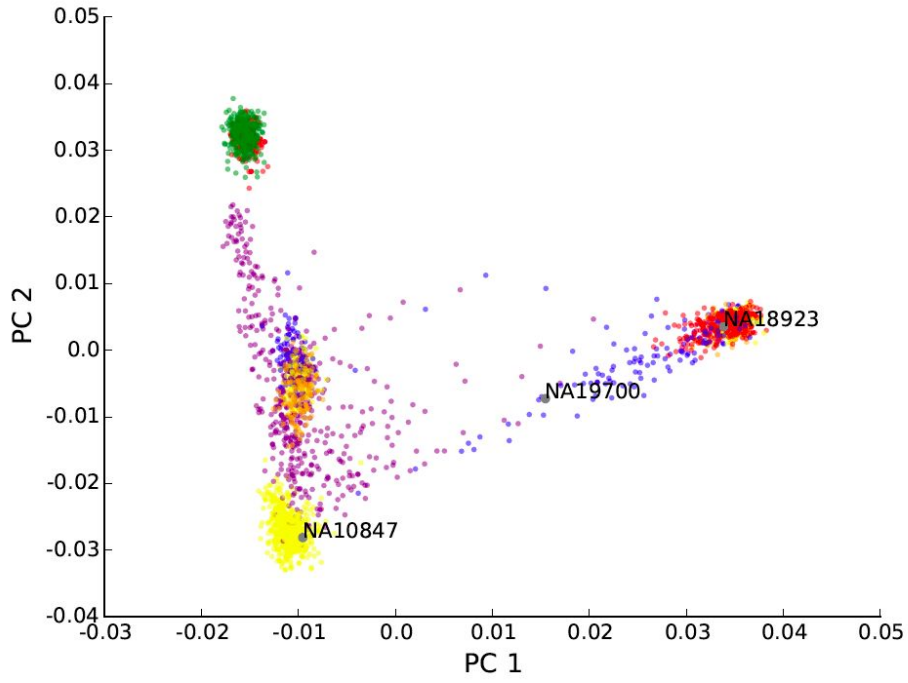
```
    ref_matrix_projection = np.dot(evecs.transpose(), ref_matrix_norm).transpose()
    return ref_matrix_projection
```

Option 3:

```
def perform_pca(ref_matrix_norm):
    """Perform PCA on normalized matrix

    Args:
        ref_matrix_norm (np.array): Float matrix of normalized SNPs, size numsnps x numsamples

    Returns:
        ref_matrix_projection (np.array): Float matrix of data projected onto PCs. size numsamples       x numdim (2)
    """
    pca = PCA(n_components=2)
    pca.fit(ref_matrix_norm.transpose())
    return pca.transform(ref_matrix_norm.transpose())
```



Exercise 2

yellow=Europe, green=Asia, red=Africa

1st PC separates African from non-African

2nd PC separates Asians and Europeans

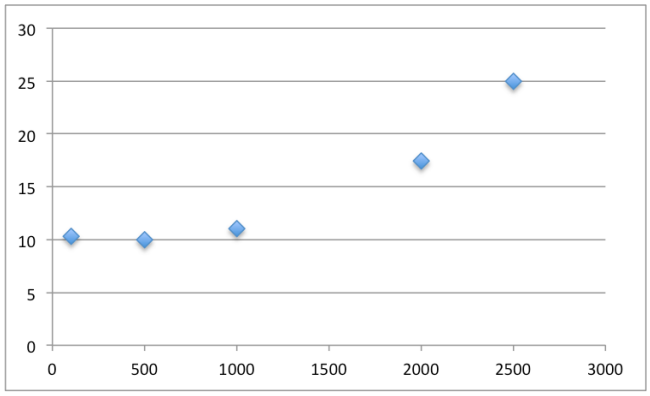
Exercise 3

NA10847: CEU

NA19700: ASW

NA18923: YRI

Exercise 4



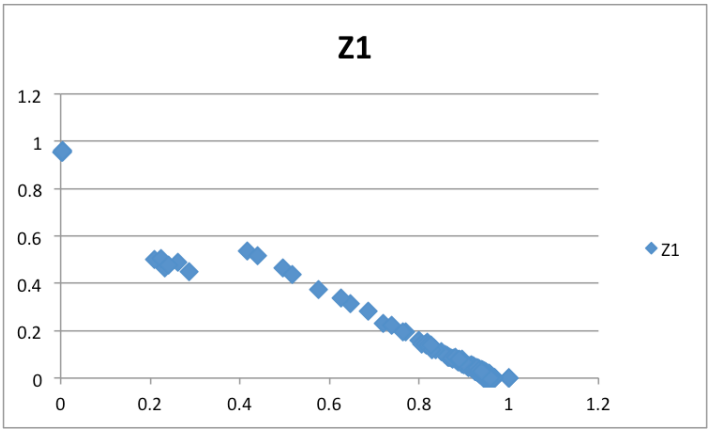
(xaxis is numsamples, yaxis is time in seconds)
Approximately quadratic

Part 2: Relative finding (2 points)

Exercise 1

Parent-child: IBD0=0%, IBD1=100%, IBD2=0%
Siblings: IBD0=25%, IBD1=50%, IBD2=25%
Cousins: IBD0=0%, IBD1=25%, IBD2=75%

Exercise 2



Xaxis is IBD=0, yaxis is IBD=1

Exercise 3

Parent-child:

NA19445	NA19453
NA19313	NA19331
NA19469	NA19470
NA19381	NA19382
NA19331	NA19334

Siblings:

NA19434	NA19444
NA19347	NA19352
NA19373	NA19374
NA19396	NA19397
NA19443	NA19470

Exercise 4

Background relatedness in the population. Just by chance, some segments will be shared.

Part 3: Imputing missing variants (4 points)

Exercise 1

I would expect the CEU+YRI panel to perform the best, since African Americans are admixed European/African.
It is important to include haplotypes that are similar to the population being analyzed so that haplotypes in the target set of individuals will match haplotypes in the reference panel.

Exercise 2

28,655
249,799

Exercise 3

ceu r2 = 0.67
yri r2=0.93
ceu_yri r2=0.94
not_asw r2=0.95

Exercise 4

Each row gives MAF, (r, pval)
The rarer the SNP, the harder to impute well.

0 (0.49871794871794878, 2.0846902146695781e-50) 1.0
0.0001 (0.49871794871794878, 2.0846902146695781e-50) 1.0
0.001 (0.32270074960257078, 0.0) 1.0
0.01 (0.74109616813472601, 0.0) 1.0
0.05 (0.89483461568238776, 0.0) 1.0
0.1 (0.91971408466608784, 0.0) 1.0
0.2 (0.92883551757497873, 0.0) 1.0
0.3 (0.94064234800328184, 0.0) 1.0
0.4 (0.94749999141094632, 0.0) 1.0
0.5 (0.95277683287371662, 0.0) 1.0