Allenrolfea_Analysis_Pipeline

Carol Rowe

1/12/2018

Samples

Sample Information: Note: Samples from population 8543 were removed as they were not included in the final study due to low reads.

<table>
<thead>
<tr>
<th>Population</th>
<th>Longitude</th>
<th>Latitude</th>
<th>n</th>
<th>Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-119.349</td>
<td>40.697</td>
<td>5</td>
<td>wetland</td>
</tr>
<tr>
<td>2</td>
<td>-119.349</td>
<td>40.697</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>3</td>
<td>-117.945</td>
<td>41.405</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>4</td>
<td>-118.013</td>
<td>40.24</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>5</td>
<td>-118.018</td>
<td>40.239</td>
<td>5</td>
<td>wetland</td>
</tr>
<tr>
<td>6</td>
<td>-110.567</td>
<td>40.583</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>7</td>
<td>-116.514</td>
<td>40.523</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>8</td>
<td>-114.422</td>
<td>39.29</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>9</td>
<td>-114.422</td>
<td>39.291</td>
<td>4</td>
<td>wetland</td>
</tr>
<tr>
<td>10</td>
<td>-114.376</td>
<td>39.32</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>11</td>
<td>-114.38</td>
<td>39.32</td>
<td>5</td>
<td>wetland</td>
</tr>
<tr>
<td>12</td>
<td>-114.078</td>
<td>38.755</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>13</td>
<td>-117.189</td>
<td>39.49</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>14</td>
<td>-118.015</td>
<td>39.837</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>15</td>
<td>-118.013</td>
<td>39.857</td>
<td>5</td>
<td>wetland</td>
</tr>
<tr>
<td>16</td>
<td>-118.013</td>
<td>39.858</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>17</td>
<td>-118.576</td>
<td>39.342</td>
<td>10</td>
<td>upland</td>
</tr>
<tr>
<td>18</td>
<td>-118.573</td>
<td>39.342</td>
<td>10</td>
<td>wetland</td>
</tr>
<tr>
<td>19</td>
<td>-118.539</td>
<td>39.617</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>20</td>
<td>-118.555</td>
<td>39.601</td>
<td>5</td>
<td>upland</td>
</tr>
<tr>
<td>21</td>
<td>-118.555</td>
<td>39.599</td>
<td>5</td>
<td>wetland</td>
</tr>
<tr>
<td>8542</td>
<td>-112.764</td>
<td>39.647</td>
<td>3</td>
<td>upland</td>
</tr>
<tr>
<td>8544</td>
<td>-113.339</td>
<td>39.834</td>
<td>3</td>
<td>wetland</td>
</tr>
<tr>
<td>8545</td>
<td>-113.386</td>
<td>39.82</td>
<td>2</td>
<td>upland</td>
</tr>
<tr>
<td>8546</td>
<td>-113.326</td>
<td>39.817</td>
<td>1</td>
<td>upland</td>
</tr>
<tr>
<td>8547</td>
<td>-113.331</td>
<td>39.998</td>
<td>1</td>
<td>upland</td>
</tr>
<tr>
<td>8548</td>
<td>-113.321</td>
<td>39.985</td>
<td>1</td>
<td>upland</td>
</tr>
<tr>
<td>8549</td>
<td>-114.027</td>
<td>40.923</td>
<td>2</td>
<td>wetland</td>
</tr>
<tr>
<td>8550</td>
<td>-112.734</td>
<td>40.896</td>
<td>3</td>
<td>upland</td>
</tr>
<tr>
<td>8551</td>
<td>-112.817</td>
<td>41.781</td>
<td>2</td>
<td>upland</td>
</tr>
</tbody>
</table>

Software versions

Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.10.5 (Yosemite)
ipyrad (v.0.5.15)

R version 3.3.1 (2016-06-21) Bug in Your Hair
{ Note: in R you can use the following to get version info: sessionInfo() }
library("ade4") # ade4_1.7-5
library("adegenet") # adegenet_2.0.1
library("ape") # ape_4.1
library("genetics") # genetics_1.3.8.1
library("ggplot2") # ggplot2_2.2.1
library(MASS) # MASS_7.3-45 library("pegas") # pegas_0.9
library("poppr") # poppr_2.4.1
library("seqinr") # seqinr_3.3-3

Python 3.6.2
import pandas as pd # version: 0.20.3
import seaborn as sns # version 0.8.1
import numpy as np # version 1.13.1
from geopy.distance import vincenty # geopy (1.11.0)
import re # 2.2.1
import scipy.stats as stats # scipy (0.19.1)

FOR MAP-MAKING ONLY:
Python 2.7.13
from mpl_toolkits.basemap import Basemap # mpl_toolkits '1.0.7'
import matplotlib.pyplot as plt # matplotlib '1.5.1'
from PIL import Image # PIL '1.1.7'
from matplotlib.collections import PatchCollection # matplotlib '1.5.1'
import matplotlib.patches as patches # matplotlib '1.5.1'

**Barcodes file used for ipyrad**

03_02 ATTCTGGAA
05_03 AAGTTGCTAA
07_04 AATCCAGA
10_02 AGCAATGAA
21_05 ATAGTCGTAA
08_02 CCTAGAGA
04_05 ACCTCGTCCA
17_10 CCAGTCTA
01_05 TAACGGTAA
18_06 CGAGGAAGCA
20_05 ATATAGTA
02_04 AATCCTTAA
17_05 AACGCAA
03_05 CCAGAACC
05_01 CGTCCAA
10_03 CTGGATGAA
03_01 TGCACCTAA
02_05 GGAATAGA
16_02 CAACTTGA
14_04 AAGCACA
16_05 TTCGTAGA
08_01 AAGAGTA
21_03 AGGCGCAGCA
17_01 TTCAATT
09_03 CATCTCAGCA
06_01 CCGCGTAA
10_01 GGTAAACCA
03_03 TGAAGCCAA
17_09 AGACGCAA
13_03 AACGGCAA
11_03 TCGCGACCAA
14_05 CTTCAATAA
09_01 CTTGCGGA
13_01 AAGGAATAA
02_03 CGCTCAACCA
19_03 GACCGCGA
19_05 TTCTCATAA
13_05 AATTGGAGCA
15_02 CCTCAAAC
12_02 GTCAGACCA
20_02 CAGGAACGCA
18_08 TTTCGGAAC
20_04 AGACGACCA
06_03 ATCGGACCAA
05_02 TAGATCAA
01_01 CTGATCCAA
11_05 CATTGACCAA
07_02 ATAAGAACCA
15_04 ACGTTGGA
15_03 GATAGATAA
09_05 CTTCAAACCA
03_04 TATAAGGA
13_02 CCAGGATAA
10_05 AACTCTCGCA
05_04 CCTATACCA
07_03 AATGCAGGCA
21_02 ACCTGAAC
02_01 CTGAGGCCA
11_03 TTGGACGGCA
18_02 CTGATACCAA
07_01 AATAAGGCAA
16_01 ACCAAGCCAA
10_02 CAGCAGAAA
01_01 GTCTCAACCA
16_01 CTGATGGA
04_01 GGCTTATAAA
18_09 CTGCAGACCA
16_03 TGGACTGGA
10_04 AGATTATAAA
17_04 AATATAAAC
17_02 ACTCCGCCA
18_07 TTAATTGCCA
12_01 GGGTAAAC
19_01 GGAGGCGCA
17_08 CATAATTGCA
11_01 TGAGAGAA
05_05 TGAGGCAAA
08_02 TAGAGCCAA
07_04 GTCTGGA
11_05 CTTATTACCA
04_02 ACTGAATA
01_02 GTAACCTAAA
20_03 ACCGTACCA
18_05 TAGTAATA
16_04 CAGTCCTAA
01_04 CTTAAGAC
18_10 CCGAGGCGCA
14_02 AGAACTTGGCA
17_03 GACGAGAC
12_03 ATAAATGCCA
14_01 CAATTAATCA
15_01 GGTAGGAA
18_01 CCAACGCAA
04_02 GATACGCCAA
02_02 CCAACGCA
08_04 CAGAGGCGCA
08_05 TAGCATA
11_02 ACCTAGTAA
06_02 CATTACGCCA
01_03 ATCCGATA
06_04 TGGAGTAAAA
21_01 GGGACGAC
19_04 GCGTTGCGCA
21_04 TTATGCATCA
11_04 TATTGGAC
14_03 TAGACTCCA
04_04 TATTCTATCA
18_03 GGCCTATAA
04_03 ATCGGGA
05_01 TTAACTCCAA
21_03 ATGGATAA
13_04 CTATCGGCCCA
08_03 GCCTTATA
06_05 GCTCGTAA
12_05 TGAGCTGCCA
17_06 TGCTGCTA
20_01 CTCAGGTAA
09_02 CGCTTGAC
07_05 GATGGTCCA
15_05 GCTTGCTCA
02_02 TAGGCTAC
18_04 CTCTTTCCA
19_02 AGCATGCTCA
12_04 CATACTAA
17_07 CCTTATCAA
01_03 TGCGAAGCAA
04_05 TTACCTAA
8542_1A GCCGGCTCA
8544_3C CTCTGATCGA
8550_9A CTATGCTCA
8545_4A ATGGGAGGA
8542_1B GAGATCTCA
8550_9B GGCAAGTCA
8542_1C ATATGCTGGA
8550_9C GGAATCTGGA
8545_4C TAGCATATGA
8551_10A GGCTCAAGA
8546_5B TTATATCTGA
8549_8A GAGGTAAGA
8551_10C TGCGGCAATA
8544_3A GCCGACGATA
8548_7C TATAACTATA
8544_3B CCGCCTACTA
8547_6A GGGGCAG
8549_8C AACTTCAGA
8548_7C TGAGTACCTA
**ipyrad params file**

----------  ipyrad params file (v.0.5.15)------------------------------------------

**AllenrFINAL**  ## [0] [assembly_name]: Assembly name. Used to name output directories for assembly steps

```
./  ## [1] [project_dir]: Project dir (made in curdir if not present)
./Allenr_01_S4_L008_R1_001.fastq.gz  ## [2] [raw_fastq_path]: Location of raw non-demultiplexed fastq files
./AllenrFINALbars.txt  ## [3] [barcodes_path]: Location of barcodes file

denovo  ## [5] [assembly_method]: Assembly method (denovo, reference, denovo+reference, denovo-reference)

## [6] [reference_sequence]: Location of reference sequence file
ddram  ## [7] [datatype]: Datatype (see docs): rad, gbs, ddrad, etc.

CAATTC,GTAA  ## [8] [restriction_overhang]: Restriction overhang (cut1,) or (cut1,cut2)

4  ## [9] [max_low_qual_bases]: Max low quality base calls (Q<20) in a read

33  ## [10] [phred_Qscore_offset]: phred Q score offset (33 is default and very standard)

6  ## [11] [mindepth_statistical]: Min depth for statistical base calling

6  ## [12] [mindepth_majrule]: Min depth for majority-rule base calling

1000  ## [13] [maxdepth]: Max cluster depth within samples

0.90  ## [14] [clust_threshold]: Clustering threshold for de novo assembly

0  ## [15] [max_barcode_mismatch]: Max number of allowable mismatches in barcodes

2  ## [16] [filter_adapters]: Filter for adapters/primers (1 or 2= stricter)

40  ## [17] [filter_min_trim_len]: Min length of reads after adapter trim

2  ## [18] [max_alleles_consens]: Max alleles per site in consensus sequences

2, 2  ## [19] [max_Ns_consens]: Max N’s (uncalled bases) in consensus (R1,R2)

4, 4  ## [20] [max_Hs_consens]: Max Hs (heterozygotes) in consensus (R1,R2)

40  ## [21] [min_samples_locus]: Min # samples per locus for output

20, 20  ## [22] [max_SNPs_locus]: Max # SNPs per locus (R1, R2)

5, 5  ## [23] [max_Indels_locus]: Max # of indels per locus (R1, R2)

0.5  ## [24] [max_shared_Hs_locus]: Max # heterozygous sites per locus (R1, R2)

0, 0, 0, 0  ## [25] [edit_cutsites]: Edit cut-sites (R1, R2) (see docs)

*  ## [27] [output_formats]: Output formats (see docs)

## [28] [pop_assign_file]: Path to population assignment file

**Loci selection from ipyrad .ustr file**

Due to varying lengths of reads, randomly selecting a single SNP from each locus does not always result in maintaining the "Min # samples per locus for output" (##21 of ipyrad params file). To check this, and to change this value to a minimum of 100 samples per locus, I ran the following python script.
NOTE: As seen from the script below, even though the ipyrad results suggest I should have 28,899 loci present in a min. of 40 samples, I actually have 26604 loci for min_samples_locus. Difference of 2,295 loci.

Note: There are 132 samples in this dataset. The .ustr file from ipyrad contains 28,899 loci.

python code:

```python
import pandas as pd

# IMPORT .ustr FILE:
# There seems to be wierd combos of spaces or tabs as separators in the .ustr file
# Thus, use the generic delim_whitespace=True to delimit on any kind of space
my_ustr = pd.read_csv('AllenrFINAL.ustr', delim_whitespace=True, header=None, na_values='-9')
# print(my_ustr.iloc[0:5, 0:9]) # File imported into pandas correctly.
# print(my_ustr.shape) # (264, 28900) rows by columns - as expected

# Counts of non-missing data by column
my_counts = my_ustr.count()
# Since there are 2 lines (or rows) per sample:
my_counts_per_sample = my_counts/2

# Don't forget that row 0 is the sample names, so should have count of 264 (132 samples)
# print(len(my_counts_per_sample))
# print(my_counts[0:10])
# print(my_counts_per_sample[0:10])

# CREATE PLOTS TO LOOK AT YOUR DATA:
import seaborn as sns
import numpy as np
# Can change width of bins by changing numer: bins=np.arange(min, max, bin_width)
# Or you could tell it how many bins you want, for example: bins=80
sns_plot = sns.distplot(my_counts_per_sample, bins=np.arange(0,132, 2))
sns_plot.set(xlabel='Number Samples', ylabel='Proportion of Total Loci')
fig = sns_plot.get_figure()
fig.savefig("proportion_loci_per_sample.png") # see plot below

# SELECT FOR LOCI WHERE DATA EXISTS IN AT LEAST 100 SAMPLES AND SAVE TO NEW FILE:
# count will be >= 200 as there are two lines per individual
# Make a new .ustr based upon a cut-off number of existing data
new_ustr = my_ustr.loc[:,my_ustr.count() >= 200]
```
#print(new_ustr.shape)  # (264, 1384)
#print(new_ustr.iloc[0:5,0:5])

# Just for curiosity. Supposed to have 28,889 loci in a min of 40 samples...
ustr_40 = my_ustr.loc[:,my_ustr.count() >= 80]
print(ustr_40.shape)  # (264, 26605)
#Hence, you can see that ipyrad said I have 28,899 (subtract 1 for column name) loci, where in reality, I have 26604 loci for min_samples_locus. Difference of 2,295 loci.

# Put NaN to -9
new_ustr = new_ustr.fillna('-9')
#print(new_ustr.iloc[0:5, 0:9])  # Looks correct.
# Make sure that your columns stay as integers because:
# This is a "gotcha" in pandas (Support for integer NA), where integer columns with NaNs are converted to floats.
new_ustr.iloc[:,1:] = new_ustr.iloc[:,1:].astype(int)
new_ustr.to_csv('AllenrFINAL_100samptolocus.ustr', index=False, sep=' ', header=False)

Plot of Proportion of Loci Per Sample

Note the peak at 40 samples, which is what we expect given the settings in ipyrad for a min. of 40 samples per locus.
Genetic Subdivision Analysis

I highly recommend reading:
Jombart, T. and Collins, C. (2017) A tutorial for Discriminant Analysis of Principal Components (DAPC) using adegenet 2.1.0. Imperial College London. MRC Centre for Outbreak Analysis and Modelling
Jombart and Collins link

Read the *str file onto a genind format in R. To do this, you first need to know how many individuals (n.ind) in your file, the number of loci (n.loc), and whether there are column headers (col.lab)

If you want to look at the first 12 rows (head -n 12) and the first 12 columns in your file:
head -n 12 AllenrFINAL_100samptolocus.ustr | cut -d' ' -f1-12

In editing the .ustr file, we know that we have 132 samples and 1383 loci. You can double check this by:

n.ind = 132
Get this by typing the following from command line:
w -l AllenrFINAL_100samptolocus.ustr
>> returns 164. There are 2 lines per individual: 164/2 = 132 individuals

n.loc = 1383
Get this by typing the following from command line:
head -n 1 AllenrFINALout.str | awk '{print NF}'
>> returns 1384. Subtract 1 for the row name = 1383. (Don't forget to subtract IF you have population info columns, etc.)
( NOTE: you could also run this command: awk '{print NF; exit}' AllenrFINALout.str )

col.lab = 1 Since column 1 has names of the samples
col.pop = 0 since I do not have a column containing a priori population info.
Otherwise give the column number
row.markernames = 0 Since there is no header in my .ustr file
onerowperind = FALSE Because I have two rows per individual
NA_char = '-9' Since -9 is what our missing data are.

R code

library("ape")
library("genetics")
library("pegas")
library("seqinr")
library("ggplot2")
library("adegenet")

########## READ IN DATA   #ADegenet link
Allen_obj1 <- read.structure("AllenrFINAL_100samptolocus.stru", n.ind =
# For kicks, you can run the following two lines just for partial confirmation of proper importation of file:
indNames(Allen_obj1)  # should return a list of the 132 sample names
ploidy(Allen_obj1)  # should return 2 since we gave it 2 alleles for each marker

# For kicks, you can run the following two lines just for partial confirmation of proper importation of file:
indNames(Allen_obj1)  # should return a list of the 132 sample names
ploidy(Allen_obj1)  # should return 2 since we gave it 2 alleles for each marker

# IDENTIFY OPTIMAL NUMBER OF CLUSTERS TO BEST EXPLAIN THE DATA

R code

# find.clusters
# Start with zoomed-out perspective. max.n.clust=30
# When you run the command below, it will ask you how many PCs to retain and then how many clusters
# I retained 200 PCs, and retained 30 clusters (I have 30 populations and 132 samples, so figure this will be ample.)

BIG_grps <- find.clusters(x=Allen_obj1, stat="BIC",choose.n.clust=TRUE,
max.n.clust=80, n.iter=100000, n.start=100, scale=FALSE, truenames=TRUE)

# Based upon BIG_grps results, we can narrow the scale:
# I am "zooming in " on the initial dip from the above output - around 0-10 clusters

grps <- find.clusters(x=Allen_obj1, stat="BIC",choose.n.clust=TRUE,
max.n.clust=10, n.iter=100000, n.start=100, scale=FALSE, truenames=TRUE)

# retained 200 PCs and 10 clusters
Note: The BIC vs # clusters with 30 clusters is inset with in the "zoomed-in" version showing the plot with 10 clusters.

Let’s look at the actual assignment of samples to the optimal 3 clusters:

R code

```r
# To see what info we can access:
names(grps)
# "Kstat" "stat" "grp" "size"
grps$size  # tells how many individuals in each cluster
# 42 34 56
grps$grp  # gives list of individuals assigned to each cluster
# output is list of sample names and what group they belong to
```
GROUP ASSIGNMENTS:

Describing Clusters with DAPC

Finding correct number of PCs to retain.

R code

```
# ?dapc
# n.da (number of discriminant functions to retain) and n.pc (number of principal components to retain) within the dapc function, or you can enter them manually at the screen prompt when you run the command
# for n.da, if k < 10, then use k-1 (3-1=2)
# suggests n.pc <= N/3 (N is number samples), hence: 132/3 = 44

test1_dapc <- dapc(Allen_obj1, pop=grps$grp, n.pc = 44, n.da = 2)
summary(test1_dapc) # following for output
```

```
#$n.dim
#2
#$n.pop
#3
#$assign.prop
# 0.9772727
#$assign.per.pop
#1 2 3
#0.9285714 1.0000000 1.0000000
#$prior.grp.size
```
test1_dapc # following is the output

#########################################################
# Discriminant Analysis of Principal Components #
#########################################################
#class: dapc
#$call: dapc.genind(x = Allen_obj1, pop = grps$grp)
#
#$n.pca: 44 first PCs of PCA used
#$n.da: 2 discriminant functions saved
#$var (proportion of conserved variance): 0.66
#
#$eig (eigenvalues): 1159 353.6 vector length content
#1 $eig 2 eigenvalues
#2 $grp 132 prior group assignment
#3 $prior 3 prior group probabilities
#4 $assign 132 posterior group assignment
#5 $pca.cent 2780 centring vector of PCA
#6 $pca.norm 2780 scaling vector of PCA
#7 $pca.eig 131 eigenvalues of PCA
#
#data.frame nrow ncol content
#1 $tab 132 44 retained PCs of PCA
#2 $means 3 44 group means
#3 $loadings 44 2 loadings of variables
#4 $ind.coord 132 2 coordinates of individuals (principal components)
#5 $grp.coord 3 2 coordinates of groups
#6 $posterior 132 3 posterior membership probabilities
#7 $pca.loadings 2780 44 PCA loadings of original variables
#8 $var.contr 2780 2 contribution of original variables

#DETERMINE CORRECT NUMBER PCs TO RETAIN:
mat <- tab(Allen_obj1, NA.method=“mean”)
xval <- xvalDapc(mat, grps$grp, n.pca.max = 300, training.set = 0.9, result = “groupMean”, center = TRUE, scale = FALSE, n.pca = NULL, n.rep = 50, xval.plot = TRUE)
xval[2:6]

OUTPUT:
$Median and Confidence Interval for Random Chance
2.5% 50% 97.5%
0.2521213 0.3324580 0.4094363
$\text{Mean Successful Assignment by Number of PCs of PCA}$

<table>
<thead>
<tr>
<th>Number of PCs</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.9766667</td>
</tr>
<tr>
<td>20</td>
<td>0.9616667</td>
</tr>
<tr>
<td>30</td>
<td>0.9672222</td>
</tr>
<tr>
<td>40</td>
<td>0.9727778</td>
</tr>
<tr>
<td>50</td>
<td>0.9600000</td>
</tr>
<tr>
<td>60</td>
<td>0.9438889</td>
</tr>
<tr>
<td>70</td>
<td>0.9405556</td>
</tr>
<tr>
<td>80</td>
<td>0.8533333</td>
</tr>
<tr>
<td>90</td>
<td>0.8072222</td>
</tr>
<tr>
<td>100</td>
<td>0.7088889</td>
</tr>
<tr>
<td>110</td>
<td>0.5072222</td>
</tr>
</tbody>
</table>

$\text{Number of PCs Achieving Highest Mean Success}$

[1]'10'

$\text{Root Mean Squared Error by Number of PCs of PCA}$

<table>
<thead>
<tr>
<th>Number of PCs</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.04409586</td>
</tr>
<tr>
<td>20</td>
<td>0.06273105</td>
</tr>
<tr>
<td>30</td>
<td>0.05624571</td>
</tr>
<tr>
<td>40</td>
<td>0.04795188</td>
</tr>
<tr>
<td>50</td>
<td>0.06161409</td>
</tr>
<tr>
<td>60</td>
<td>0.07807483</td>
</tr>
<tr>
<td>70</td>
<td>0.09035253</td>
</tr>
<tr>
<td>80</td>
<td>0.18002057</td>
</tr>
<tr>
<td>90</td>
<td>0.0.24085111</td>
</tr>
<tr>
<td>100</td>
<td>0.31656919</td>
</tr>
<tr>
<td>110</td>
<td>0.51224211</td>
</tr>
</tbody>
</table>

$\text{Number of PCs Achieving Lowest MSE}$

[1]'10'
Jombart and Collins (2017) recommend "using the number of PCs associated with the lowest RMSE as the 'optimum' n.pca in the DAPC analysis". In our case, this is: 10.

Let’s rerun the DAPC retaining 10 PCs:

R code

```r
k3_PC10_dapc <- dapc(Allen_obj1, pop=grps$grp, n.pca = 10, n.da = 2 )
# look at the dapc results: retained 10 clusters and 2 eigen values
summary(k3_PC20_dapc)
```

```r
$n.dim
[1] 2
$n.pop
[1] 3
$assign.prop
```
[1] 0.9772727
$assign.per.pop
  1 2 3
0.9285714 1.0000000 1.0000000
$prior.grp.size
  1 2 3
42 34 56
$post.grp.size
  1 2 3
39 34 59
'''
k3_PC10_dapc
'''

#################################################
# Discriminant Analysis of Principal Components #
#################################################
class: dapc
$call: dapc.genind(x = Allen_obj1, pop = grps$grp, n.pca = 10, n.da = 2)

$n.pca: 10 first PCs of PCA used
$n.da: 2 discriminant functions saved
$var (proportion of conserved variance): 0.312

$eig (eigenvalues): 765.4 236.9 vector length content
1 $eig 2 eigenvalues
2 $grp 132 prior group assignment
3 $prior 3 prior group probabilities
4 $assign 132 posterior group assignment
5 $pca.cent 2780 centring vector of PCA
6 $pca.norm 2780 scaling vector of PCA
7 $pca.eig 131 eigenvalues of PCA

data.frame nrow ncol content
1 $tab 132 10 retained PCs of PCA
2 $means 3 10 group means
3 $loadings 10 2 loadings of variables
4 $ind.coord 132 2 coordinates of individuals (principal components)
5 $grp.coord 3 2 coordinates of groups
6 $posterior 132 3 posterior membership probabilities
7 $pca.loadings 2780 10 PCA loadings of original variables
8 $var.contr 2780 2 contribution of original variables
'''

SCATTER PLOT
R code

# Choose 3 colors for the clusters:
myCol <- c("#66C2A5", "#FC8D62", "#8DA0CB"
SCATTERPLOT OF THE THREE CLUSTERS: n.pca = 10, n.da = 2

```
scatter(k3_PC10_dapc, posi.da="bottomleft", bg="white", pch=17:22, scree.pca=TRUE, posi.pca="topleft", col=myCol)
```

STRUCTURE-LIKE PLOT:

R code

```
# with sample labels
compoplot(k3_PC10_dapc, posi="topright",txt.leg=paste("Cluster", 1:3),ncol=1, col = myCol, cex.lab=1)
# or without sample labels
compoplot(k3_PC10_dapc, posi="topright",txt.leg=paste("Cluster", 1:3),ncol=1, col = myCol, cex.lab=1, lab='', xlab='individuals')
```

```
# Look at the most admixed individuals: no more than 90% probability of membership to a single cluster
temp90 <- which(apply(k3_PC10_dapc$posterior,1, function(e)
```
all(e<0.9)))
temp90 # 8 individuals:

<table>
<thead>
<tr>
<th></th>
<th>03_05</th>
<th>05_02</th>
<th>07_01</th>
<th>07_02</th>
<th>07_03</th>
<th>18_01</th>
<th>8547_6A</th>
<th>8551_10A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15</td>
<td>22</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>90</td>
<td>124</td>
<td>131</td>
</tr>
</tbody>
</table>

# Note: temp70 included: 03_05, 05_02, 07_02, 8547_6A. temp95 had 12 indiv. and temp99 had 17 indiv.
# Structure-like with just the subset of data from temp90
compoplot(k3_PC10_dapc, subset=temp90, posi="topright", txt.leg=paste("Cluster", 1:3), ncol=1, col=myCol)

Structure-like plot without individual labels:
Structure-like plot of most admixed individuals: < 90% probability of membership to a single cluster:

Tree:
R code

```r
# NJ tree with colored nodes:
D <- dist(tab(Allen_obj1))
tre <- nj(D)
# Get group assignments:
# Note that k3_dapc$grp is identical to grps$grp (At least in my case. Is this always true?)
clust3 = k3_PC10_dapc$grp
# tree types can be: “phylogram”, “cladogram”, “fan”, “unrooted”, or “radial”
plot(x=tre, type = "radial", edge.color="black", edge.width=2, edge.lty=1, cex=0.8, tip.color = myCol[clust3], label.offset = 0.0 )
```
Neighbor-joining Tree (radial)

Genetic Distance by Geographic Distance

Load the needed libraries and then read in your data file into structure
R code

```r
library("poppr")
library("pegas")
library("ape")
library("adegenet")
library("ade4")
library("poppr")

#Read in data
Allen_obj1 <- read.structure("AllenrFINAL_100samptolocus.stru", n.ind =
```
Get Euclidean (genetic) distance

R code

```r
# Genetic Distance as a matrix
# p is for ploidy
# when using genind (individuals) vs genpop (pre-defined populations),
# euclidean is only method choice
Genetic_Dist <- dist(Allen_obj1, method = "euclidean", p = 2)
print(length(Genetic_Dist)) # 8646
hist(Genetic_Dist)
print(min(Genetic_Dist)) # 8.418918
print(max(Genetic_Dist)) # 43.4546
```

Histogram of Euclidean Genetic Distance
Get Geographic Distance from Lat. and Long.

Python code

```python
import pandas as pd
from itertools import combinations

# Note: All samples within a population have the same lat. and long.

# Import a table which has just one column with your sample names. (I
can not extract names from my master_table from below, as I did not use
all of the original samples in my final data analyses. Hence, I created
this table.)
samples = pd.read_csv("FinalSamplesList.csv")
print(samples.head())

'''
Samples
0  01_01
1  01_02
2  01_03
3  01_04
4  01_05
'''

# Making the column to a list so that I can use itertools.combinations
sampleList = samples.Samples.tolist()
# print(len(sampleList)) # 132 - as expected

# Create a list of all sample combinations (no duplicates)
SampleBySample2 = list(combinations(sampleList, 2))
# Turn the list of tuples into a dataframe
SS2 = pd.DataFrame(data=SampleBySample2, columns=['Sample1',
'Sample2'])
print(SS2.shape) # (8646, 2)
print(SS2.head())

'''
Sample1  Sample2
0  01_01  01_02
1  01_01  01_03
2  01_01  01_04
3  01_01  01_05
4  01_01  02_01
'''

# First import the master list with sample, lat., long., and population
assignment:
master_table = pd.read_csv("SampleLatLong.csv")
print(master_table.head()) # (144, 4) Note, this table has extra
samples not used in final data set. (144 vs 132)
'''
```
```
| ID  Longitude  Latitude  Population |
|-----|-------------|-------------|----------------|
| 0   01_01    -119.349059 40.697272  pop01 |
| 1   01_02    -119.349059 40.697272  pop01 |
| 2   01_03    -119.349059 40.697272  pop01 |
| 3   01_04    -119.349059 40.697272  pop01 |
| 4   01_05    -119.349059 40.697272  pop01 |
```

# Now, we need to combine tables so that we get columns for Long. and Lat. for each of the 2 sample combinations per row.
# I will do this in two steps: 1) add lat. long. for Sample1, then 2) add lat. long. for Sample2
# Combine sample-by-sample table to master table using only samples in final data set.
combo_table = pd.merge(SS2, master_table, left_on='Sample1', right_on='ID')
print(combo_table.shape) # (8646, 6)
combo_table2 = pd.merge(combo_table, master_table, left_on='Sample2', right_on='ID')
print(combo_table2.shape) # (8646, 10)
print(combo_table2.head())
```
```
<table>
<thead>
<tr>
<th>Sample1 Sample2</th>
<th>ID_x Longitude_x Latitude_x Population_x</th>
<th>ID_y</th>
</tr>
</thead>
<tbody>
<tr>
<td>01_01</td>
<td>-119.349059 40.697272 pop01</td>
<td>01_02</td>
</tr>
<tr>
<td>01_02</td>
<td>-119.349059 40.697272 pop01</td>
<td>01_03</td>
</tr>
<tr>
<td>01_03</td>
<td>-119.349059 40.697272 pop01</td>
<td>01_04</td>
</tr>
<tr>
<td>01_04</td>
<td>-119.349059 40.697272 pop01</td>
<td>01_05</td>
</tr>
</tbody>
</table>
```

# Let's tidy this table up: Select columns we want, sort by samp1 then samp2, reset the index, and then rename columns
combo = combo_table2[['Sample1', 'Sample2', 'Longitude_x', 'Latitude_x', 'Longitude_y', 'Latitude_y']]
combo_sort = combo.sort_values(['Sample1', 'Sample2'], ascending=[True, True])
combo_sort = combo_sort.reset_index(drop=True)
combo_sort.columns = ['Sample1', 'Sample2', 'Long_1', 'Lat_1', 'Long_2', 'Lat_2']
print(combo_sort.head())
```
```
| Sample1 Sample2 Long_1 Lat_1 Long_2 Lat_2 |
|--------|------------|-------|-------|-------|-------|
| 01_01  | 01_02      | -119.349059 | 40.697272 | -119.349059 | 40.697272 |
| 01_01  | 01_03      | -119.349059 | 40.697272 | -119.349059 | 40.697272 |
```
# Now get the geographic distance via vincenty
from geopy.distance import vincenty

# Calculate vincenty for each row; use lambda function for this
combo_sort["Geo_Dist_km"] = combo_sort.apply(lambda x:
    vincenty((x['Lat_1'], x['Long_1']), (x['Lat_2'], x['Long_2'])), axis =1)

# the Geo_Dist_km column is of dtype object. Need to change to string
# and then split to remove " km"
combo_sort["Geo_Dist_km"] = combo_sort["Geo_Dist_km'].astype('str')
combo_sort["Geo_Dist_km"] = combo_sort["Geo_Dist_km"].apply(lambda x:
    x.split(' km')[0])
print(combo_sort.head())

# Finally, save this to a csv file and put into R for graphing!
combo_sort.to_csv("AllenrFIANAL_GeoDist.csv", index=False)

Load Geographic Location Data
R code

ggeog_dist <- read.csv("AllenrFIANAL_GeoDist.csv", header=TRUE)
#print a summary of geographic distances
print(head(ggeog_dist))
\texttt{nrow}(\texttt{geog\_dist}) \ # \ 8646 \\
\texttt{ncol}(\texttt{geog\_dist}) \ # \ 7 \\
\texttt{max\_dist} <- \texttt{max}(\texttt{geog\_dist}\$\texttt{Geo\_Dist\_km}, \texttt{na.rm} = \texttt{TRUE}) \\
\texttt{print}(\texttt{max\_dist}) \ # \ 572.7349 \\
\texttt{max\_row} <- \texttt{geog\_dist}[\texttt{which}\_\texttt{max}(\texttt{geog\_dist}\$\texttt{Geo\_Dist\_km}),] \\
\texttt{print}(\texttt{max\_row}) \\
\ldots \\
\texttt{Sample1} \ \texttt{Sample2} \ \texttt{Long\_1} \ \texttt{Lat\_1} \ \texttt{Long\_2} \ \texttt{Lat\_2} \ \texttt{Geo\_Dist\_km} \\
754 \ \ 02\_01 \ 8542\_1A \ -119.3491 \ 40.69745 \ -112.7644 \ 39.64667 \ 572.7349 \\
\ldots \\

\# \texttt{Now find the lowest non-0 distance:} \\
\texttt{submin} <- \texttt{geog\_dist}[\texttt{which}(\texttt{geog\_dist}\$\texttt{Geo\_Dist\_km}>0.0),] \\
\#\texttt{print(submin)} \\
\texttt{min} <- \texttt{submin}[\texttt{which}\_\texttt{min}(\texttt{submin}\$\texttt{Geo\_Dist\_km}),] \\
\texttt{print(min)} \\
\ldots \\
\texttt{Sample1} \ \texttt{Sample2} \ \texttt{Long\_1} \ \texttt{Lat\_1} \ \texttt{Long\_2} \ \texttt{Lat\_2} \ \texttt{Geo\_Dist\_km} \\
5 \ \ 01\_01 \ 02\_01 \ -119.3491 \ 40.69727 \ -119.34 \ 40.69745 \ 0.02061151 \\
\ldots \\

In summary:
Max dist btwn pops: 572.7349km between pops 02 and 8542
Min dist btwn pops: 0.02061151km between pops 01 and 02

Plotting Geographic vs Genetic Distance:

\texttt{R} code

\#plot geographic distance vs genetic distance \\
\# First Plot - rather dull, I'll show the second plot \\
\# Get this geog. dist. as a list \\
\texttt{Geog\_Dist} <- \texttt{geog\_dist}\$\texttt{Geo\_Dist\_km} \\
\texttt{print(length(Geog\_Dist))} \ # \ 8646 \\
\# Plot and then add regression line \\
\texttt{plot(Geog\_Dist, Genetic\_Dist, main}"\texttt{Genetic vs geographic distance for each pair of samples"}, xlab="\texttt{Geographic distance (km)}", ylab="\texttt{Genetic distance (Euclidian)}") \\
\texttt{abline(lm(Geog\_Dist, Genetic\_Dist), col=}"red",lty=2) \\

\# Second Plot: with heatmap \\
\texttt{library(MASS)} \\
\# the \texttt{n=} is the resolution of the heatmap (number of squares) \\
\# \texttt{lims: The limits of the rectangle covered by the grid as c(xl, xu, yl, yu).} \\
\# for the \texttt{x limits: we are dealing with distances of 0km to 573km} \\
\# for the \texttt{y limits: we have genetic distances between 8.418918 and 43.4546} \\
\# \texttt{h: vector of bandwidths for x and y directions. Defaults to normal reference bandwidth (see bandwidth.nrd). A scalar value will be taken}
to apply to both directions.
# play with h vlaues to change the spread of density colors

dens <- kde2d(Geog_Dist, Genetic_Dist, n=900, lims = c(-5,580, 5,48))
myPal <- colorRampPalette(c("white","blue","gold", "orange", "red"))
plot(Geog_Dist, Genetic_Dist, pch=20,cex=.5,xlab="Geographic distance (km)", ylab="Genetic distance (Euclidian)")
image(dens, col=transp(myPal(900),0.5), add=TRUE)
abline(mod <- lm(Genetic_Dist~Geog_Dist))

title("Isolation by distance plot")

# Summary stats of regression
print(summary(mod))

Summary of Regression Line:
Call:
lm(formula = Genetic_Dist ~ Geog_Dist)

Residuals:
Min 1Q Median 3Q Max
-21.5938 -2.2320 0.1865 2.2895 14.6185

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.644e+01 6.071e-02 435.49 <2e-16 ***
Geog_Dist 7.373e-03 2.168e-04 34.01 <2e-16 ***
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.196 on 8644 degrees of freedom
Multiple R-squared: 0.116, Adjusted R-squared: 0.1179
F-statistic: 1157 on 1 and 8644 DF, p-value: < 2.2e-16
Isolation by Distance Plot:

![Isolation by distance plot](image)

**MAP**

Mapping Samples with Colors for Each Cluster

Using Python 2.7.13 for this: (Already had Basemap for py2, and had problems getting it for py3.)

**NOTE: THREE OF THE 30 POPULATIONS HAD MIXED CLUSTERING.**

For the map, I assigned color to majority rule.

Population name, cluster assignment with (individual counts)

- Pop3 Cluster1 (4 individuals) and Cluster3 (1 individual)
- Pop7 Cluster1 (1 individual) and Cluster3 (4 individuals)
- Pop17 Cluster1 (1 individual) and Cluster2 (9 individuals)

************ NICE HELP PAGES: ************

Simple_basemap_example.py by Brian Blaylock:
https://gist.github.com/blaylockbk/79658bdde8c1334ab88d3a67c6e57477

Selection from "Python Data Science Handbook" by Jake VanderPlas- Geographic Data with Basemap

Don't forget to check out the tutorial too! And youtube.com videos

youtube: "Python Geographic Maps with Matplotlib Basemap by Ryan Noonan":
https://www.youtube.com/watch?v=goFY_W7T0z4

youtube: Matplotlib tutorials #26 - ? of pythonproraming.net series

Python code
import mpl_toolkits
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import pandas as pd
from PIL import Image
from matplotlib.collections import PatchCollection  # for making the legend
import numpy as np  # for when I was going to add lat and long lines on map
import matplotlib.patches as mpatches  # creating my own legend

# IMPORT YOUR CSV FILE WITH LAT., LONG., NAME INFORMATION, ETC.
# Import your .csv file which has lat/long coords, names, color indicators, etc.
my_latlong = pd.read_csv("./AllenrFINAL100_mapinfo.csv")
print(my_latlong.head())

Sample_prefix  Longitude  Latitude  Indiv_PRE  Indiv_POST
Designation
0 1 -119.349059  40.697272  5  5
wetland
1 2 -119.349124  40.697451  5  5
upland
2 3 -117.944959  41.405421  5  5
upland
3 4 -118.012679  40.240277  5  5
upland
4 5 -118.017707  40.239242  5  5
wetland

Extraction  Cluster_Assignment  k3_count  k3_majority
0  Qiagen  1  5  Cluster_1
1  Qiagen  1  5  Cluster_1
2  Qiagen  1,3  4,1  Cluster_1
3  Qiagen  1  5  Cluster_1
4  Qiagen  1  5  Cluster_1

# create column with colors for each of the cluster designations of column "k2_color":
# color ideas: https://stackoverflow.com/questions/22408237/named-colors-in-matplotlib

def color_code(row):
    if row[\'k3_majority\'] == "Cluster_1":
        return "#66C2A5"
    if row[\'k3_majority\'] == "Cluster_2":
        return "#FC8D62"
    if row[\'k3_majority\'] == "Cluster_3":
        return "#8DA0CB"

# Apply the above function to create a new column called 'k2_col' in
your table; axis=1 for rows:
# This is for if you want to color code your population names
my_latlong['k3_color'] = my_latlong.apply(lambda row: color_code (row), axis=1)
# print(my_latlong.head())

# create figure with axis so we can eventually save this
# opting not to do this as it's easier to work with the plt.show()
# image rather than the saved .png file
fig, ax = plt.subplots()

# *********** GET BOUNDARIES FOR CREATING YOUR MAP ***************
# Here, I am getting the boundaries of my map of interest from input .csv file
# Adding as a fudge-factor (+/- 1) so points aren't on edge of map
# llclon = lower left corner longitude, llclat = lower left corner latitude,
# urclon = upper rt corner long., urclat = upper right corner lat.
llclon = min(my_latlong['Longitude'] - 1)
llclat = min(my_latlong['Latitude'] - 1)
urclon = max(my_latlong['Longitude'] + 1)
urclat = max(my_latlong['Latitude'] + 1)
# This is about the center of my map
lat0 = (max(my_latlong['Latitude']) - min(my_latlong['Latitude'])) / 2
long0 = (max(my_latlong['Longitude']) - min(my_latlong['Longitude'])) / 2

# *********** INITIALIZING YOUR MAP WITH YOUR BOUNDARIES
# Creating the initial map using my lat/longs as above:
# resolution: l = low, i = intermediate, h = high, f = full
# projection='tmerc'
map = Basemap(llcrnrlon=llclon, llcrnrlat=llclat, urcrnrlon=urclon, urcrnrlat=urclat, resolution='f', epsg=32007, lat_0=lat0, lon_0=long0)

# *********** DRAW SOME MAP ELEMENTS TO YOUR MAP ***************
map.drawrivers(color='blue')
map.drawstates(linewidth=0.9, color='gray')
# you can drawcounties(color='blue'), draw country lines. etc.

# *********** ADDING TOPOGRAPHY TO YOUR MAP ***************
# When using python Basemap to plot maps, a nice background would be a big plus.
# But when using map.bluemarble(), map.etopo(), or map.shadedrelief(), we can not zoom in to a smaller region,
# since it will generate a blur image. The best way to create a high
resolution background image
#(either topography, street map, etc.) is using arcgisimage method. See example list below.

# Drawing ArcGIS Basemap (only works with cylc projections??) ONTO YOUR BASEMAP PROJECTION/SIZE
# Examples of what each map looks like can be found here:
# http://kbkb-wx-python.blogspot.com/2016/04/python-basemap-background-image-from.html

maps = ['ESRI_Imagery_World_2D', 'ESRI_Satellite_World_2D', 'NatGeo_World_Map', 'NGS_Topos_US_2D', 'Ocean_Basemap', 'USA_Topo_Maps', 'World_Imagery', 'World_Physical_Map', 'World_Shaded_Relief', 'World_Street_Highway', 'World_Terrain_Base', 'World_Topo_Map']

# Alter the xpixels to get better and better resolution! ypixels will default based on xpixels value
map.arcgisimage(service=maps[8], xpixels=2000, verbose=False)

# ********** GETTING YOUR LAT/LONG POINTS TO PLOT ONTO MAP AND PLOTTING THEM ************
pt_lat = my_latlong['Latitude'].values
pt_long = my_latlong['Longitude'].values
pt_names = my_latlong['Sample_prefix'].values

# x is longitudes, y is latitudes.
x, y = map(pt_long, pt_lat)
# 'bo' means b for blue and o .... k for black
# or you could write as: map.plot(x, y, markerfacecolor='black', marker='o', markersize=5)
map.plot(x, y, 'ko', markersize=5)

# ********** ADDING NAMES TO YOUR PLOTTED LOCATIONS ************
# The x_offset and y_offset lists are for off-setting the names so they don't overlap.
# Had downloaded and tried from adjustText import adjust_Text with is like R/ggplot2 ggrepel. Didn't work well for me!!
x_offset = [20000, -30000, 20000, 20000, -10000, 30000, 30000, -20000,
color = my_latlong['k3_color'].tolist()
wet_dry = my_latlong['Designation'].tolist()
count = 0
for label, xpt, ypt, x_offset, y_offset in zip(pt_names, x, y, x_offset, y_offset):
    # wetland gets circles with white border
    if wet_dry[count] == "wetland":
        plt.annotate(label, fontsize = 10, xy=(xpt,ypt),
        xytext=(xpt+x_offset, ypt+y_offset), color='black',
        arrowprops=dict(arrowstyle="-"), bbox=dict(boxstyle = 'circle',
        facecolor = color[count], edgecolor = "black")
        count += 1
    elif wet_dry[count] == "upland":
        plt.annotate(label, fontsize = 10, xy=(xpt,ypt),
        xytext=(xpt+x_offset, ypt+y_offset), color='black',
        arrowprops=dict(arrowstyle="-"), bbox=dict(facecolor = color[count],
        edgecolor = color[count]))
        count += 1

# ******** CREATING MY OWN LEGEND ***************
# Get a subset of the .csv file with just the cluster name ('k3_majority') and corresponding color ('k3_color') and sort
# Then get unique values by dropping the duplicates
my_legend = my_latlong[['k3_majority',
            'k3_color']].sort_values(['k3_majority'])
my_legend2 = my_legend.drop_duplicates()
# Create the 3 color and name "patches" for the legend
# iloc gets the physical location of value from a cell in the table
patch1 = mpatches.Patch(color=my_legend2.iloc[0,1], label = my_legend2.iloc[0,0])
patch2 = mpatches.Patch(color=my_legend2.iloc[1,1], label = my_legend2.iloc[1,0])
patch3 = mpatches.Patch(color=my_legend2.iloc[2,1], label = my_legend2.iloc[2,0])
# location or name: 'best'=0, 'upper right'=1, 'upper left'=2, 'lower left'=3, 'lower right'=4
# 'right'=5, 'center left'=6, 'center right'=7, 'lower center'=8, 'upper center'=9, 'center'=10
plt.legend(handles=[patch1, patch2, patch3], loc=8)

# Did NOT add lat and long lines:
# ********** DRAWING LATITUDE AND LONGITUDE LINES
# The following draw the dotted (dashes=[1,3]) gray (color='gray') lines for lats and longs every 2 degrees within the range given and that of your map
# The labels=[0,0,0,0] is for where you want labels (1) or not (0) in the rt, left, top, and bottom orientation - I think
# parallels are your latitudes; meridians are longitudes
#map.drawparallels(np.arange(0.,60.,2.),color='gray',dashes=[1,3],labels=[1,0,0,0])
#map.drawmeridians(np.arange(0.,360.,2.),color='gray',dashes=[1,3],labels=[0,0,0,1])

# ********* WANTED TO DRAW A SCALE BAR FOR DISTANCE
# map scale does NOT work with projection "cyl"
map.drawmapscale(-118, 38.4, -116, 40, 100, barstyle='fancy')

# *********** GETTING THE OUTPUT ******************************************
plt.show()

# SAVING YOUR MAP:
# Working from WingIDE......
# When you get the image, then adjust with the envelope icon with the arrows. "configure subplots"
# Adjust left to 0, bottom = 0, right = 1.00, and top = 1.00. Then save it from there as .png.

# This method (below) of saving below gives a poorer quality map.
# plt.savefig('AllenrFINAL_map.png', bbox_inches='tight')
SNP summary from ipyrad .loci file

Using the .loci file from the ipyrad output, we can observe the following:
There are 33,538 loci in the file, with 28,899 containing at least one SNP.
There are an average of 2.84 SNPs across the 33,538 loci.
The maximum number of SNPs per locus is 19.
The distribution of SNPs across the loci will be shown below.

NOTE: If BayPass results in SNPs that differentiate samples by ecotype, you will need the saved .csv file ('AllenrFINAL_loci_to_SNP.csv') to easily extract the corresponding sequences from the .loci file. The SNP number will tell us which is the corresponding locus number. python code

```python
import pandas as pd

# Read .loci file by line, count # of the SNPs (number of * and - in the line starting with //)
# Add the Locus #, number SNPs, and Line # from the input .loci file to a dataframe

# Create empty dataframe to add info to:
my_df = pd.DataFrame(columns=['locus', 'num_SNPs', 'line_num'])

with open("AllenrFINAL.loci","r") as fi:
```
count=0 # row number to enter in dataframe
lines=1 # to get line number from the .loci file
for line in fi:
    rec = line.strip()
    if rec.startswith("//"):
        locus = line.split()[-1]
        SNPs_a = line.count('*')
        SNPs_b = line.count('-')
        SNPs = SNPs_a + SNPs_b
        my_df.loc[count] = [locus, SNPs, lines]
        count += 1
        lines += 1
    else:
        lines += 1

print(my_df.head(6))
'''
    Locus num_SNPs line_num
0 |31|  5  45
1 |65|  6 149
2 |66|  1 216
3 |84|  7 269
4 |104|  3 312
5 |112| 11 358
'''
# Adding cumulative sum of SNPs to a new column...think this will be helpful in future analysis
my_df['cum_sum_SNPs'] = my_df.num_SNPs.cumsum()
my_df.head()
'''
    Locus num_SNPs line_num cum_sum_SNPs
0 |31|  5  45     5
1 |65|  6 149    11
2 |66|  1 216    12
3 |84|  7 269    19
4 |104|  3 312    22
'''
# Get number of loci in file
print(my_df.shape) # (33538, 4) - meaning 33,538 loci
# Get avg number Loci across the 33538 Loci
print(my_df.num_SNPs.mean()) # 2.83940604687
# Get number of loci with at least 1 SNP
print(my_df.num_SNPs[my_df.num_SNPs > 0].count()) # 28899
# Get max number SNPs per locus
print(my_df.num_SNPs.max()) # 19

# Plot distribution of number SNPs across the loci:
import seaborn as sns
p = sns.distplot(my_df.num_SNPs, bins=19, kde=False,
BayPass:

Our results clearly show that given the data, our Allenrolfea samples segregate by geographical distance NOT by upland/wetland habitat. But, amongst all the loci, are there any that differentiate according to ecotype? Here, we test for such loci that may segregate by ecotype.
Converting ipyrad .u.geno file to a .geno file for BayPass

ipyrap .u.geno file AND the BayPass geno file are both set up where columns are samples and rows are for each locus.

The ipyrad .u.geno output file contains one column per sample for allele counts: 2, 1, 0, or -9. 2 if both alleles match the "reference", 1 if one allele matches the reference, 0 for no matching alleles, and -9 for missing data.

The BayPass genotype file requires TWO columns for each sample, where each column has a count for each of two alleles in a diploid organism.

Hence, the conversion of a single ipyrad number to the two for BayPass would be:

ipyrap BayPass
2 2
1 1
0 0

Below, I use the ipyrad .u.geno file to calculate the sum of each allele ACROSS each population.

python code

""
3 files to import:
AllenrFINALpops.csv : this has samples listed by population (01, 02, etc.) with designations (0 for wetland, and 1 for upland)
AllenrFINALall.csv : this lists all samples as they should be in the u.geno file
AllenrFINAL.u.geno : this is the ipyrad .u.geno output file

This will be the name of my final saved file for use in BayPass
AllenrFINAL100.u.geno : this is the ipyrad output file
""

import pandas as pd
AllenrFINALpops = pd.read_csv("AllenrFINALpops.csv")
AllenrFINALall = pd.read_csv("./AllenrFINALall.csv")
print(AllenrFINALpops.head())
print(AllenrFINALall.head())

<table>
<thead>
<tr>
<th>sample</th>
<th>pop</th>
<th>designation</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01_</td>
<td>wetland</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>02_</td>
<td>upland</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>03_</td>
<td>upland</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>04_</td>
<td>upland</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>05_</td>
<td>wetland</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sample</th>
<th>pop</th>
<th>designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01_01</td>
<td>wetland</td>
</tr>
<tr>
<td>1</td>
<td>01_02</td>
<td>wetland</td>
</tr>
<tr>
<td>2</td>
<td>01_03</td>
<td>wetland</td>
</tr>
</tbody>
</table>
# creating a list of all sample names

gen_cols = AllenrFINALall['sample'].tolist()
print(len(gen_cols)) # 132

ipyrad u.geno output file has no separators for columns
example:
2109219921
9092212191
etc.
Thus, we read the file in with pd.read_fwf where column width is 1 and there are 132 for them (number for each sample)

AllenrFINALgeno = pd.read_fwf("AllenrFINAL.u.geno", widths=[1]*132, header=None, na_values='9', names=gen_cols)
# print first 5 rows and 5 cols
print(AllenrFINALgeno.iloc[0:5, 0:5])
print(AllenrFINALgeno.shape) # (28899, 132)

Before I proceed, I'd like to make my selection of loci more stringent. As I did in the previous analyses, I am going to SELECT FOR LOCI WHERE DATA EXISTS IN AT LEAST 100 SAMPLES.

python code

# Counts of non-missing data across rows (axis=1). i.e. counts of non-NaN per locus across samples
# Running next two lines just to make sure I am doing this correctly
my_counts = AllenrFINALgeno.count(axis=1)
print(my_counts[0:10])
Now, save these selected rows into a new dataframe
AllenrFINAL100geno = AllenrFINALgeno.loc[AllenrFINALgeno.count(axis=1) >= 100, :]
print(AllenrFINAL100geno.shape)  # (1381, 132)

Now, we have a dataframe with 1381 loci across 132 samples. These remaining loci have non-missing data for at least 100 of the 132 samples.

Continue creating our new .geno file for BayPass...

```python
# Get the list of samples by population:
# Should have 30 populations
FINAL_pops = AllenrFINALpops['sample'].tolist()
print(FINAL_pops)
print(len(FINAL_pops))  # 30 populations

['01_','02_','03_','04_','05_','06_','07_','08_','09_','10_','11_','12_','13_','14_','15_','16_','17_','18_','19_','20_','21_','8542_','8544_','8545_','8546_','8547_','8548_','8549_','8550_','8551_']
30

# New let's do the file conversion!
outDF = pd.DataFrame()
for pop in FINAL_pops:
    # matching columns with header starting with pop and any characters after
    pattern = pop + ".*"
    # this returns a subset of the AllenrFINALgeno with just headers matching the pop
    sorted_df = AllenrFINAL100geno.filter(regex=(pattern))
    #print(sorted_df.head())
    # Making allele counts across the selected population; axis 1 is single row
    outDF[pop] = sorted_df.sum(axis=1)
    outDF[pop+"NaN"] = sorted_df.isnull().sum(axis=1)
    num_col = sorted_df.shape[1]
    outDF[pop+"b"] = (num_col*2) - (outDF[pop]) - (2*outDF[pop+"NaN"])

# Remove the columns where I did the NaN count
outDF = outDF[outDF.columns.drop(list(outDF.filter(regex='NaN')))]
# If cells contain NaN, I need to convert those to 0 for BayPass:
outDF = outDF.fillna(0)
# Make the numbers all to integers
outDF = outDF.astype(int)
print(outDF.iloc[0:5, 0:10])
```
print(outDF.shape)
# (1381, 60). Meaning, 1381 loci and 30 populations. There are two columns for each population, hence 30 populations.

# Want this in the same order as when I create the ecotypes file in the following section:
outDF_names = outDF.columns.tolist()
print(outDF_names)
""
['01_','01_b','02_','02_b','03_','03_b','04_','04_b','05_','05_b','06_','06_b','07_','07_b','08_','08_b','09_','09_b','10_','10_b','11_','11_b','12_','12_b','13_','13_b','14_','14_b','15_','15_b','16_','16_b','17_','17_b','18_','18_b','19_','19_b','20_','20_b','21_','21_b','8542_','8542_b','8544_','8544_b','8545_','8545_b','8546_','8546_b','8547_','8547_b','8548_','8548_b','8549_','8549_b','8550_','8550_b','8551_','8551_b']
""
# NOTE: you could always do a sort on your column headers if needed to ensure order:
#outDF = outDF.sort_index(axis=1)
# Remove header row and remove the index column and save to file
outDF.to_csv('AllenrFINAL100_geno.txt', header=False, index=False, sep=' ')

# I want to keep a file with the index numbers in it. These are the SNP reference numbers.
# If BayPass results in SNPs differing by ecotype, you will need to access the original SNP #.
# This saved file will be for SNP reference if I need it....
outDF1 = outDF.copy()
outDF1['SNP_index'] = outDF1.index
print(outDF1.shape)  # (1381, 61) - as expected
outDF1 = outDF1.reset_index(drop=True)  # drop=True so it doesn't get saved as a column again
outDF1['BP_index'] = outDF1.index
print(outDF1.shape)  # (1381, 62)

#print(outDF1.iloc[0:5, 56:])
#print(outDF1.iloc[0:5, 0:5])
# Want to keep the BayPass index and SNP index columns. Else is just the data
```python
outDF2 = outDF1.loc[:, ['BP_index', 'SNP_index']]
print(outDF2.head())

# Save to file
outDF2.to_csv('AllenrFINAL100_geno_INDEX.csv', header=True, index=False)
```

Hooray, now we have our geno file to use in BayPass: `AllenrFINAL100_geno.txt`

Still need to create an ecotype file to use in BayPass:

```python
# Need the ecotype file: 0 for wetland, and 1 for upland
# NOTE: these are in the SAME POPULATION ORDER as in AllenrFINAL100_geno.txt
eo_pops = AllenrFINALpops['designation'].tolist()
# Confirm samples are in the correct order:
eo_names = AllenrFINALpops['sample'].tolist()
print(len(eco_pops))  # 30

with open('AllenrFINAL_ecotype.txt', 'w') as text_file:
    text_file.write(eco_out + '
')
```

Now that we have our ecotypes file, `AllenrFINAL_ecotype.txt`, we are ready to run BayPass

**Files for running on CHPC**

For running BayPass on the CHPC, I used two files:
1.) CR_BayPassFork.pl (perl script so I can run multiple chains at once)
2.) Perl_BayPass_SLURM.txt (slurm script which runs the above perl script)
BayPass settings:

-npilot 20
-nthreads 1
-npop 30
-nval 3,000
-thin 1,000
-burnin 160,000
-seed $ch  # Need this since I am running in parallel via perl script. If I don't have a different seed number for each run, then seed would be the same for each run and hence outputs for each run would be idnetical!
-gfile ./AllenrFINAL100_geno.txt
-efile ./AllenrFINAL_ecotype.txt
-outprefix ./BP_out_AllenrFINAL100_$ch

python code

import os
from glob import glob
import pandas as pd
import re

# Enter pathway to directory with files
my_path = './BP_Burnin160_AllenrFINAL100/'

# Want to select the files as two batches: 1-48 and 49-96:
# Here's for 1-47:
first_files_a = glob(os.path.join(my_path, 'BP_out_AllenrFINAL100_[1-9]_summary_betai_reg.out'))
first_files_b = glob(os.path.join(my_path, 'BP_out_AllenrFINAL100_[1][0-9]_summary_betai_reg.out'))
first_files_c = glob(os.path.join(my_path, 'BP_out_AllenrFINAL100_2[0-9]_summary_betai_reg.out'))
first_files_d = glob(os.path.join(my_path, 'BP_out_AllenrFINAL100_3[0-9]_summary_betai_reg.out'))
first_files_e = glob(os.path.join(my_path, 'BP_out_AllenrFINAL100_4[0-8]_summary_betai_reg.out'))
first_files = first_files_a + first_files_b + first_files_c + first_files_d + first_files_e

# Here's for 48-96:
second_files_a = glob(os.path.join(my_path, "BP_out_AllenrFINAL100_49_summary_betai_reg.out"))
second_files_b = glob(os.path.join(my_path, "BP_out_AllenrFINAL100_5[0-9]_summary_betai_reg.out"))
second_files_c = glob(os.path.join(my_path, "BP_out_AllenrFINAL100_6[0-9]_summary_betai_reg.out"))
second_files_d = glob(os.path.join(my_path, "BP_out_AllenrFINAL100_7[0-9]_summary_betai_reg.out"))
second_files_e = glob(os.path.join(my_path, 'BP_out_AllenrFINAL100_8[0-9]_summary_betal_reg.out'))
second_files_f = glob(os.path.join(my_path, 'BP_out_AllenrFINAL100_9[0-6]_summary_betal_reg.out'))
second_files = second_files_a + second_files_b + second_files_c + second_files_d + second_files_e + second_files_f

print(len(first_files)) # 48
print(len(second_files)) # 48

# Enter column names as list, and then create two empty pandas dataframes with just column names:
col_names = ['COVARIABLE', 'MRK', 'M_Pearson', 'SD_Pearson', 'BF(dB)', 'Beta_is', 'SD_Beta_is', 'eBPis']
first_DF = pd.DataFrame(columns=col_names)
second_DF = pd.DataFrame(columns=col_names)

# Enter each of the first (1-48) files into a single dataframe:
for f in first_files:
    data = pd.read_csv(f, delim_whitespace=True)
    first_DF = pd.concat([first_DF, data]) # adding each new df to the bottom

# Enter each of the second (49-96) files into a single dataframe:
for ff in second_files:
    data2 = pd.read_csv(ff, delim_whitespace=True)
    second_DF = pd.concat([second_DF, data2]) # adding each new df to the bottom

# Observe initial results to make sure all looks correct:
print(first_DF.head())

COVARIABLE  MRK  M_Pearson  SD_Pearson  BF(dB)  Beta_is  SD_Beta_is  eBPis
0  1  1 -0.001984  0.181445 -10.176260  0.000835  0.008399
0.008399  1  2 -0.035352  0.184981 -12.184625  0.000225  0.005719
0.005719  2  3  0.006277  0.169172 -6.547445  0.002663  0.015632
0.015632  3  4  0.015866  0.165232 -6.383850 -0.001505  0.022622
0.022622  4  5 -0.038747  0.176829 -8.034721 -0.000895  0.011823
0.011823  0  0.035815  1  0.013840  2  0.063127
```python
print(first_DF.shape)  # (66288, 8)
print(second_DF.shape)  # (66288, 8)

# BF(dB) is the value we are interested in. We will average these across each individual for each of the 24 runs in each data set
# Group by locus ('MRK') and take the average of the BF(dB) for each locus across the 24 chains
# Do this for the first 24 and then the second 24
first_mean = first_DF[['MRK', 'BF(dB)']].groupby('MRK').mean()
# This resets the index and labels the mean column
first_mean = first_mean.add_suffix('Mean_1').reset_index()
print(first_mean.shape)  # (1381, 2)
print(first_mean.head())

second_mean = second_DF[['MRK', 'BF(dB)']].groupby('MRK').mean()
# Reset the index and label the mean column
second_mean = second_mean.add_suffix('Mean_2').reset_index()
print(second_mean.shape)  # (1381, 2)

# Merge the two dataframes
corr_df = pd.merge(first_mean, second_mean, on='MRK', how='outer')
print(corr_df.head())
print(corr_df.shape)  # (1381, 3)

# Get Pearson correlation with corresponding p-value
import scipy.stats as stats
corr_out = stats.pearsonr(corr_df['BF(dB)Mean_1'], corr_df['BF(dB)Mean_2'])
print(corr_out)  # (0.98270135654146695, 0.0)

# Want to see if there are loci in common between the two dataframes
```
with high BF(dB) values
# Look at BF(dB) values greater than ..... 10
first_DF_over10 = first_DF[first_DF['BF(dB)'] >= 10]
print(first_DF_over10.shape)  # (45, 8) - hence, 48 loci
second_DF_over10 = second_DF[second_DF['BF(dB)'] >= 10]
print(second_DF_over10.shape)  # (39, 8) - 39 loci in this batch

# Get info on any loci with >= 10 BF(dB) that are shared between the two sets of runs
over_10 = pd.merge(first_DF_over10, second_DF_over10, on='MRK', how='inner')
print(over_10)  # ONLY 2 loci in common: MRK 1146 and MRK 556

COVARIABLE_x  MRK  M_Pearson_x  SD_Pearson_x  BF(dB)_x  Beta_is_x
\0  1  1146  0.089602  0.167036  10.805242  0.006313
1  1  556  0.002912  0.179544  11.144949  0.000520

SD_Beta_is_x  eBPis_x  COVARIABLE_y  M_Pearson_y  SD_Pearson_y
BF(dB)_y  \
0  0.015931  0.159953  1  0.082061  0.173104
17.197884
1  0.018064  0.010083  1  0.010086  0.176687
11.415238

Beta_is_y  SD_Beta_is_y  eBPis_y
\0  0.005614  0.015900  0.140236
1  0.001178  0.017869  0.023457

In summary:
The Pearson correlation coefficient between the two groups of BayPass data was: 0.983
There were only two loci shared between the two data groups with BP(dB) values
>= 10.
Matcing loci are at markers: 556, and 1146.
I suspect, with this few loci, it is a matter of chance rather than due to ecotypes.

Let’s get the corresponding locus sequences anyway...
python code
import pandas as pd
my_ref = pd.read_csv('AllenrFINAL100_geno_INDEX.csv')
print(my_ref.head())
# Using BP_Index values of 556 and 1146, get corresponding Seq_Index
# Don't get confused by the default index in pandas! (ie 555 vs 556)
locus1 = my_ref[my_ref.BP_index == 556]
print(locus1)  # Seq_Index = 9672
locus2 = my_ref[my_ref.BP_index == 1146]
print(locus2)  # Seq_Index = 24414

# Now open the AllenrFINAL_loci_to_SNP.csv we created a bit back
# This file has line# and locus name for the ipyrad output file .loci
lookup = pd.read_csv('AllenrFINAL_loci_to_SNP.csv')
print(lookup.head())

# Get the cumulative sums of 9672 and 24414.
# First part gets subset of dataframe above the desired value. Print
# just the row interested in.
location1 = lookup[lookup.cum_sum_SNPs >= 9672]
location2 = lookup[lookup.cum_sum_SNPs >= 24414]
print(location1.iloc[0, :])
print(location2.iloc[0, :])

# Interesting, these were from two loci with very high number SNPs!
# Use the line number or locus name to find the corresponding seqs in
# the .loc file
# Do that from shell
From your shell, navigate to your .loci file. Then type in a command to get some lines before and up to the line number you want. Then make sure the locus name and number of SNPs match just as a check. Here, I opt to get about 30 lines...226804 through 226834

```
   sed -n '226804,226834p' AllenrFINAL.loci
```

Just printing the last 4 lines from the sed command to give an idea of output. Note, the html of this will remove all whitespaces from each line. Everything looks great. Blast one of the sequences...lucky if you get it to match anything!

```
8546_5B
CATCTAGTAGACAATCTAAACGCAACCTCCCAACAGATGCAGATATGGTACACAGGCTCAAGAAATCGACAAACAAACCCAACA--
8550_9A
CATCTAGTAGACAATCTAAAYNCAACCTCCCAACARATGCAGATATGTMACAGGCTCAAGAAATYGACAAACAAACCCAACA--
8550_9B
CATCTAGTAGACAATCTAAACGCAACCTCCCAACAGATGCAGATATGGTAMACAGGCTCAAGAAATCGACAAACAAACCCAACA--
```

Again, I’m just pasting the last 4 of the 30 lines I printed.

```
8548_7C
TTACTAAGGTCGAAAAACGGCAAGGCTTTTTAGAGATAGAGGGCCCTGCGGACCTCTTCCCCTTCCATGCTACGAAGGAGCTTGTA--
8550_9A
TTACTAAGGTCGAAAAACGGCAAGGCTTTTTAGAGATAGAGGGCCCTGCGGACCTCTTCCCCTTCCATGCTACRAAGGAGCTTGAC--
8550_9B
TTACTAAGGTCRCAAACGGCAAGGCTTTTTTAGAGATAGAGGGCCCTGCGGACCTCTTCCCCTTCCATGCTACRRGAGGGRCTTGAC--
```

**NOTE**: I believe these loci do not really reflect ecotypes. These are both outlier loci as they have 9 and 11 SNPs respectively. Recall, there are an average of 2.84 SNPs across the 33,538 loci. Look at the distribution plot from section: SNP summary from ipyrad .loci file
As of 24 Jan. 2018, there are NO HITS in genbank.