This file contains all relevant code used in the study "Finding the adaptive needles in a population-structured haystack: a case study in a New Zealand mollusc" Citations for all software and scripts are in the original article. The code below is free to be used, just cite our work (Salloum et al. 2022).

Notes on terminology:

- "all" - generally referring to a dataset containing all SNPs for a set of populations
- "neutral" - referring to a dataset containing non-outlier SNPs. It does not mean that the SNPs are truly evolutionary neutral
- "outliers" - referring to a dataset containing outlier SNPs. It does not mean that the SNPs are adaptive.
- "NZ-wide" and "all_conserved" refer to the same set of populations (all populations sampled across New Zealand)
- "NI" refers to North Island
- "SI" refers to South Island
- "Southern" refers to South and Sub-antarctic Islands/ Please refer to the original study for more information on terminology.

SNP calling using stacks and the same parameters used in the first submission of man 3 (m=3, M=2, n=4)

    #calling SNPs only for the North Island populations

```
cd /nesi/nobackup/uoa00538/programs/Stacks_oneg
mkdir samples_all
cd samples_all
cp AdaptationInStrucutredPops/Stacks/samples_all/* ./

cd ..
mkdir NI
nano ustacks_NI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=ustacksNI
#SBATCH --account=uoa00538
#SBATCH --partition=long
#SBATCH --time=80:00:00
#SBATCH --mem=24G
#SBATCH --cpus-per-task=16
#SBATCH --output=ustacks_NI%j.out

module purge
module load Stacks/2.2-gimkl-2017a

src=./samples_all
files="OnegEC01
OnegEC02
OnegEC03
OnegEC04
OnegEC05
OnegEC06
OnegEC07
OnegEC08
OnegEC09
OnegNC02
OnegNC03
OnegNC04
OnegNC05
OnegNC06
OnegNC07
OnegNC08
OnegNC09
OnegNC10
OnegNC11
OnegRU03
OnegRU04
OnegRU05
OnegRU06
OnegRU07
OnegRU08
OnegRU09
OnegRU13
OnegRU14
OnegRU15
OnegRU16
OnegRU17
OnegRU18
OnegRU19
OnegTI09
OnegTI10
OnegTI11
OnegTI12
OnegTI13
OnegTI14
OnegTI15
OnegTI16
OnegTI17
OnegTI18
OnegTI21
OnegTI22"


id=1

for sample in $files

do

ustacks -f ./samples_all/$sample.fq.gz \
-o ./NI \
-i $id \
--name $sample \
-m 3 \
-M 2 \
-p 16

let "id+=1"

done
```

nano cstacks_NI.sh

```
#!/bin/bash -e
#SBATCH --job-name=cstacksNI
#SBATCH --account=uoa00538
#SBATCH --time=48:00:00
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH --mem=24G
#SBATCH --output=cstacks_NI%j.out
#SBATCH --error=cstacks_NI%j.err
module purge
module load Stacks/2.2-gimkl-2017a
cstacks -P ./ -M ./popmapNI_bypop.txt -n 4 -p 16
```

```
nano popmapNI_bypop.txt
```

```
OnegEC01        EC
OnegEC02        EC
OnegEC03        EC
OnegEC04        EC
OnegEC05        EC
OnegEC06        EC
OnegEC07        EC
OnegEC08        EC
OnegEC09        EC
OnegNC02        NC
OnegNC03        NC
OnegNC04        NC
OnegNC05        NC
OnegNC06        NC
OnegNC07        NC
OnegNC08        NC
OnegNC09        NC
OnegNC10        NC
OnegNC11        NC
OnegRU03        RU
OnegRU04        RU
OnegRU05        RU
OnegRU06        RU
OnegRU07        RU
OnegRU08        RU
OnegRU09        RU
OnegRU13        RU
OnegRU14        RU
OnegRU15        RU
OnegRU16        RU
OnegRU17        RU
OnegRU18        RU
OnegRU19        RU
OnegTI09        TI
OnegTI10        TI
OnegTI11        TI
OnegTI12        TI
OnegTI13        TI
OnegTI14        TI
OnegTI15        TI
OnegTI16        TI
OnegTI17        TI
OnegTI18        TI
OnegTI21        TI
OnegTI22        TI
```

```
nano sstacks_NI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=sstacksNI
#SBATCH --account=uoa00538
#SBATCH --time=36:00:00
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH --mem=24G
#SBATCH --output=sstacks_NI%j.out
#SBATCH --error=sstacks_NI%j.err
module purge
module load Stacks/2.2-gimkl-2017a
sstacks -P ./ -M ./popmapNI_bypop.txt -p 16
```

```
nano tsv2bam_NI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=tsv2bamNI
#SBATCH --account=uoa00538
#SBATCH --time=02:00:00
#SBATCH --cpus-per-task=16
#SBATCH --mem=24G
#SBATCH --ntasks=1
#SBATCH --output=tsv2bam_NI%j.out
#SBATCH --error=tsv2bam_NI%j.err
module purge
module load Stacks/2.2-gimkl-2017a
tsv2bam -P ./ -M ./popmapNI_bypop.txt -t 16
```

```
nano gstacks_NI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=gstacks_NI
#SBATCH --account=uoa00538
#SBATCH --time=02:00:00
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH --output=gstacks_NI%j.out
#SBATCH --error=gstacks_NI%j.err
module purge
module load Stacks/2.2-gimkl-2017a
gstacks -P ./ -M ./popmapNI_bypop.txt -t 16
```

```
nano populations_NI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=populationsNI
#SBATCH --account=uoa00538
#SBATCH --time=24:00:00
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH --mem=24G
#SBATCH --output=pop_NI%j.out
#SBATCH --error=pop_NI%j.err
module purge
module load Stacks/2.2-gimkl-2017a
populations -P ./ -M ./popmapNI_bypop.txt -t 16 -p 3 -r 0.8 --min_maf 0.05 --write_random_snp --hwe --fstats --fasta_loci --fasta_samples --vcf --vcf_haplotypes --genepop --structure --phylip --fasta_samples_raw --log_fst_comp
```

calling SNPs only for the South Island populations

```
cd ..
mkdir SI
nano ustacks_SI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=ustacksSI
#SBATCH --account=uoa00538
#SBATCH --partition=long
#SBATCH --time=80:00:00
#SBATCH --mem=24G
#SBATCH --cpus-per-task=16
#SBATCH --output=ustacks_SI%j.out

module purge
module load Stacks/2.2-gimkl-2017a

src=./samples_all
files="OnegAK06
OnegAK10
OnegAK11
OnegAK12
OnegAK13
OnegAK14
OnegAK15
OnegAK16
OnegAK17
OnegAK18
OnegCR04
OnegCR09
OnegCR10
OnegCR11
OnegCR12
OnegCR13
OnegCR14
OnegCR15
OnegCU01
OnegCU03
OnegCU04
OnegCU05
OnegCU07
OnegDU04
OnegDU08
OnegDU09
OnegDU10
OnegDU13
OnegDU14
OnegDU15
OnegDU16
OnegDU20
OnegDU21
OnegDU47_lo
OnegDU48_lo
OnegDU49_lo
OnegDU52_lo
OnegDU53_lo
OnegDU55_hi
OnegDU56_hi
OnegDU57_hi
OnegDU58_hi
OnegDU59_hi
OnegST05
OnegST07
OnegST09
OnegST10
OnegST11
OnegST12
OnegST13
OnegST14"

id=1

for sample in $files

do

ustacks -f ./samples_all/$sample.fq.gz \
-o ./SI \
-i $id \
--name $sample \
-m 3 \
-M 2 \
-p 16

let "id+=1"

done
```

```
nano cstacks_SI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=cstacksSI
#SBATCH --account=uoa00538
#SBATCH --time=48:00:00
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH --mem=24G
#SBATCH --output=cstacks_SI%j.out
#SBATCH --error=cstacks_SI%j.err
module purge
module load Stacks/2.2-gimkl-2017a
cstacks -P ./ -M ./popmapSI_bypop.txt -n 4 -p 16
```

```
nano popmapSI_bypop.txt
```

```
OnegAK06        AK
OnegAK10        AK
OnegAK11        AK
OnegAK12        AK
OnegAK13        AK
OnegAK14        AK
OnegAK15        AK
OnegAK16        AK
OnegAK17        AK
OnegAK18        AK
OnegCR04        CR
OnegCR09        CR
OnegCR10        CR
OnegCR11        CR
OnegCR12        CR
OnegCR13        CR
OnegCR14        CR
OnegCR15        CR
OnegCU01        CU
OnegCU03        CU
OnegCU04        CU
OnegCU05        CU
OnegCU07        CU
OnegDU04        DU
OnegDU08        DU
OnegDU09        DU
OnegDU10        DU
OnegDU13        DU
OnegDU14        DU
OnegDU15        DU
OnegDU16        DU
OnegDU20        DU
OnegDU21        DU
OnegDU47_lo     DU
OnegDU48_lo     DU
OnegDU49_lo     DU
OnegDU52_lo     DU
OnegDU53_lo     DU
OnegDU55_hi     DU
OnegDU56_hi     DU
OnegDU57_hi     DU
OnegDU58_hi     DU
OnegDU59_hi     DU
OnegST05        ST
OnegST07        ST
OnegST09        ST
OnegST10        ST
OnegST11        ST
OnegST12        ST
OnegST13        ST
OnegST14        ST
```

```
nano sstacks_SI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=sstacksSI
#SBATCH --account=uoa00538
#SBATCH --time=36:00:00
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH --mem=24G
#SBATCH --output=sstacks_SI%j.out
#SBATCH --error=sstacks_SI%j.err
module purge
module load Stacks/2.2-gimkl-2017a
sstacks -P ./ -M ./popmapSI_bypop.txt -p 16
```

```
nano tsv2bam_SI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=tsv2bamSI
#SBATCH --account=uoa00538
#SBATCH --time=02:00:00
#SBATCH --cpus-per-task=16
#SBATCH --mem=24G
#SBATCH --ntasks=1
#SBATCH --output=tsv2bam_SI%j.out
#SBATCH --error=tsv2bam_SI%j.err
module purge
module load Stacks/2.2-gimkl-2017a
tsv2bam -P ./ -M ./popmapSI_bypop.txt -t 16
```

```
nano gstacks_SI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=gstacks_SI
#SBATCH --account=uoa00538
#SBATCH --time=02:00:00
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH --output=gstacks_SI%j.out
#SBATCH --error=gstacks_SI%j.err
module purge
module load Stacks/2.2-gimkl-2017a
gstacks -P ./ -M ./popmapNI_bypop.txt -t 16
```

```
nano populations_SI.sh
```

```
#!/bin/bash -e
#SBATCH --job-name=populationsSI
#SBATCH --account=uoa00538
#SBATCH --time=24:00:00
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH --mem=24G
#SBATCH --output=pop_SI%j.out
#SBATCH --error=pop_SI%j.err
module purge
module load Stacks/2.2-gimkl-2017a
populations -P ./ -M ./popmapSI_bypop.txt -t 16 -p 3 -r 0.8 --min_maf 0.05 --write_random_snp --hwe --fstats --fasta_loci --fasta_samples --vcf --vcf_haplotypes --genepop --structure --phylip --fasta_samples_raw --log_fst_comp
```

Filtering datasets to remove individuals with more than 90% missing data with the SNP_GBS filter pipeline (Alexander 2018) - https://github.com/laninsky/GBS_SNP_filter What is needed:

- GBS_SNP_filter.txt with the parameters to filter
- script to slurm (script_snps_filter.txt)
- population map
- GBS_SNP_filter.sh (copy from github above)
- GBS_SBP_filter_chrom_modifier.R (copy from github above)
- GBS_SBO_filter_HWE.R (copy from github above)
- GBS_SNP_filter.rsq.R (copy from github above)
- VCF file to filter (output from Stack population module)

```
less GBS_SNP_fiter.txt

populations.snps.vcf #name of your VCF file (goes in the first line)
0.8 # proportion of samples that has to be equalled or exceeded for a SNP to be retained in the dataset - I make it the same as the r value on Stacks (goes in the second line)
0.9 # proportion of missing loci for individual samples (more than that and the sample is removed) (goes in the third line)
0.00000000000001 # p-value cut-off for determining loci in HWE within a population (I made it very low because I did not want to filter for HWE) (goes in the fourth line)
0.00000000000001 # r^2 cut-off for determining SNPs in LD with each other within a population (I also made it very low because I only kept biallelic loci) (goes in the fifth line)
1 # number of populations a locus has to be found out of HWE/in LD so it is discarded (goes in the sixth line)
#CHROM # column eader for where the locus id is placed in your vcf file (goes in the seventh line)
  #any regular expression needed to modify the contents of the column header specified in line 7 (I did not need to specify any regular expression) (goes in the eigth line)
```

```
less script_snps_filter.txt

#!/bin/bash -e
#SBATCH --job-name=filter_snps
#SBATCH --account=uoa00538
#SBATCH --mem=3G
#SBATCH --output=snps_filter.out
#SBATCH --error=snps_filter.err
module purge
module load Plink
module load VCFtools
module load HTSlib
module laod R
library(dplyr)
library(readr)
library(tibble)
library (stringr)
bash GBS_SNP_filter.sh
```

```
less popmap.txt

OnegAK10        AK
OnegAK11        AK
OnegAK12        AK
OnegAK13        AK
OnegAK14        AK
OnegAK15        AK
OnegAK16        AK
OnegAK17        AK
OnegAK18        AK
```

Filtering datasets to remove loci with more than 30% missing data with VCFtools What is needed:

- VCF file comming from the GBS_SNP filter (so individuals with more than 90% missing data would already have been removed)
- VCFtools (load the module and type the command below locally to filter out any loci missing in more than 30% of all individuals - Exclude sites on the basis of the proportion of missing data (defined to be between 0 and 1, where 0 allows sites that are completely missing and 1 indicates no missing data allowed).

```
vcftools --vcf populations.snps.vcf --max-missing 0.7 --recode --recode-INFO-all --out vcftools.out
```

Use R to make a plot showing the the genotypes (0/0, 0/1 ,1/1 and NA) and percentage of missing per individual and locus (NA)

```
#define directory to work
setwd()
getwd()

#load required packages
library(vcfR)
library(poppr)
library(reshape2)
library(tidyr)
library(ggpubr)

#Start with the file that is output from Stacks population module, and repeat for every filter to compare the proportion of NA between files
filename = "populations.snps.vcf"
popfile = "indID_popID.csv"
#VCF output from populations module on Stacks
#read vcf data into vcfR
vcf = read.vcfR(filename)

#check that the correct number of variants was read into R, by counting the number of SNPs in the original vcf file
length(count.fields(filename))

#query specific information from the vcf file
queryMETA(vcf)
queryMETA(vcf, element = "GT") #focusing on the genotypes field

#query specific samples
vcf@gt[120:4, 120:6] #will give genotype information for that sample - in case OnegEC08

#tidy data into a table that R can work with
tidy_data = vcfR2tidy(vcf) #the vcf file has to have been read in first, with read.vcfR()
summary(tidy_data)
show(tidy_data)
head(tidy_data)

# extract the genotypes from vcf (already read into vcfR with read.vcfR())
gt = extract.gt (vcf)
head (gt)
summary(gt)
dim(gt) #check if all samples and all variants (snps) are read in
gt_summary = summary(gt)
head(gt_summary)
#save table
#write.table (gt_summary, "gt_summary_notFiltered_all2020.tsv", sep = "\t")
#Open table in Notepad and remore "" and extra spaces. Copy and paste in Excel, then save as CVS file and load back into R
tidyfilename = "gt_summary_notFiltered_all_2020.csv"
gt_summary_table = read.csv(tidyfilename, header = T)
head(gt_summary_table)

#reshape table for plotting
long_gt = melt(gt_summary_table)
head(long_gt)
#write.table (long_gt, "long_gt_summary_notFiltered.csv", sep = ",", row.names = FALSE)
#long_gt = read.csv("long_gt_summary_notFiltered.csv", header = T)
#save the total number of snps in an object, used in the plot to transform the data to percentages
total = length(count.fields(filename))

class(long_gt$ID)
class(long_gt$variable)
class(long_gt$value)

names = long_gt[!duplicated(long_gt$variable),]
names = names$variable
names
#write.table (names, "indID.csv", row.names = FALSE)

pops = read.csv(popfile, header = T)

textcol = "black"
long_gt$pop = factor (long_gt$pop, level = c("RU", "TI", "NC", "EC", "WE", "CP", "CR", "DU", "AK", "CU", "ST", "AU", "CA", "AN", "BO", "CH"))
ggplot(long_gt, aes(x = pop, y = (value/total)*100)) +
  geom_bar (aes(fill = ID), stat = "identity", position = position_fill()) +
  labs (x = "Population", y = "Proportion of Genotype", title = "") +
  theme(legend.position = "none") +
  scale_colour_manual(values = c("orange", "blue", "green", "grey")) +
  scale_fill_manual (values = c("orange", "blue", "green", "grey")) +
  theme_classic() +
  guides(fill=guide_legend(title="Genotype", size = 18)) +
  theme(axis.title=element_text(size=18,face="bold"),
        axis.text.x=element_text(size=18,colour=textcol),
        axis.text.y=element_text(size = 18, vjust=0.2,colour=textcol),
        legend.position="right",legend.direction="vertical",
        legend.title=element_text(colour=textcol, size = 18),
        legend.margin=margin(grid::unit(0,"cm")),
        legend.text=element_text(colour=textcol,size=18,face="bold"),
        legend.key.height=grid::unit(0.5,"cm"),
        legend.key.width=grid::unit(0.5,"cm"))
#Safe to ignore the warning - seven CR samples with NA for the number of NAs
#Rinse and repeat: plot graph with NAs genotypes filtered (using the filtered VCF file output by SNP_GBS_filter and VCFtools as input)---------------------------------------
#Change the following and re-run code
filename = "populations.snps.vcf" #use the filtered vcf file name
popfile = "indID_popID.csv" #if individuals were removed during filtering, the population map will need to be updated
tidyfilename = "gt_summary_notFiltered_all_2020.csv" #after modifying the tidyfile in Notepad to remove "" as specified in the code
```

Splitting the NZ-wide SNP dataset using vcftools

For splitting the NZ-wide SNP dataset into Southern (South Island + Subantarctics), South Island, and North Island, the NZ-wide dataset has to be ready (split it just after finising the filtering steps above)

Southern

```
cd /nesi/nobackup/uoa00538/programs/revisions_man3/splitting_SNPdataset

module load VCFtools/0.1.15-GCC-9.2.0-Perl-5.30.1

cd southern

nano keep_southern.txt
OnegAK06
OnegAK10
OnegAK11
OnegAK12
OnegAK13
OnegAK14
OnegAK15
OnegAK16
OnegAK17
OnegAK18
OnegAN01
OnegAN03
OnegAN05
OnegAN06
OnegAN07
OnegAN08
OnegAN09
OnegAU02
OnegAU16
OnegAU17
OnegAU18
OnegAU23
OnegAU32
OnegAU34
OnegAU35
OnegAU36
OnegAU37
OnegBO01
OnegBO02
OnegBO03
OnegBO04
OnegBO05
OnegBO06
OnegBO07
OnegBO08
OnegBO09
OnegBO10
OnegCA02
OnegCA03
OnegCA19
OnegCA21
OnegCA22
OnegCA23
OnegCA24
OnegCA25
OnegCA26
OnegCA27
OnegCH03
OnegCH04
OnegCH08
OnegCH17
OnegCH18
OnegCH19
OnegCH20
OnegCH21
OnegCH27
OnegCU01
OnegCU03
OnegCU04
OnegCU05
OnegCU07
OnegDU04
OnegDU08
OnegDU09
OnegDU10
OnegDU13
OnegDU14
OnegDU15
OnegDU16
OnegDU20
OnegDU21
OnegDU47_lo
OnegDU48_lo
OnegDU49_lo
OnegDU52_lo
OnegDU53_lo
OnegDU55_hi
OnegDU56_hi
OnegDU57_hi
OnegDU58_hi
OnegDU59_hi
OnegST05
OnegST07
OnegST09
OnegST10
OnegST11
OnegST12
OnegST13
OnegST14


# Filtering for specific individuals (keep a list of individuals saved in a file, with one individual name per row and no header - names must match vcf file)
vcftools --vcf ../conserved.vcf --keep keep_southern.txt --recode --recode-INFO-all --out southern.out
#Filtering out alleles with count smaller than 1 within a clade
vcftools --vcf southern.out.recode.vcf --mac 1 --recode --recode-INFO-all --out southern_mac1
```

South island

```
cd ../south

nano keep_SI.txt
OnegAK06
OnegAK10
OnegAK11
OnegAK12
OnegAK13
OnegAK14
OnegAK15
OnegAK16
OnegAK17
OnegAK18
OnegCU01
OnegCU03
OnegCU04
OnegCU05
OnegCU07
OnegDU04
OnegDU08
OnegDU09
OnegDU10
OnegDU13
OnegDU14
OnegDU15
OnegDU16
OnegDU20
OnegDU21
OnegDU47_lo
OnegDU48_lo
OnegDU49_lo
OnegDU52_lo
OnegDU53_lo
OnegDU55_hi
OnegDU56_hi
OnegDU57_hi
OnegDU58_hi
OnegDU59_hi
OnegST05
OnegST07
OnegST09
OnegST10
OnegST11
OnegST12
OnegST13
OnegST14

# Filtering for specific individuals (keep a list of individuals saved in a file, with one individual name per row and no header - names must match vcf file)
vcftools --vcf ../conserved.vcf --keep keep_SI.txt --recode --recode-INFO-all --out SI.out
#Filtering out alleles with count smaller than 1 within a clade
vcftools --vcf SI.out.recode.vcf --mac 1 --recode --recode-INFO-all --out SI_mac1
```

North Island

In [ ]:

```
cd ../north

nano keep_NI.txt
OnegEC01
OnegEC02
OnegEC03
OnegEC04
OnegEC05
OnegEC06
OnegEC07
OnegEC08
OnegEC09
OnegNC02
OnegNC03
OnegNC04
OnegNC05
OnegNC06
OnegNC07
OnegNC08
OnegNC09
OnegNC10
OnegNC11
OnegRU03
OnegRU04
OnegRU05
OnegRU06
OnegRU07
OnegRU08
OnegRU09
OnegRU13
OnegRU14
OnegRU15
OnegRU16
OnegRU17
OnegRU18
OnegRU19
OnegTI09
OnegTI10
OnegTI11
OnegTI12
OnegTI13
OnegTI14
OnegTI15
OnegTI16
OnegTI17
OnegTI18
OnegTI21
OnegTI22

# Filtering for specific individuals (keep a list of individuals saved in a file, with one individual name per row and no header - names must match vcf file)
vcftools --vcf ../conserved.vcf --keep keep_NI.txt --recode --recode-INFO-all --out NI.out

#Filtering out alleles with count smaller than 1 within a clade
vcftools --vcf NI.out.recode.vcf --mac 1 --recode --recode-INFO-all --out NI_mac1
```

BayeScan

From the directory where BayeScan was installed, sbatch the script below.

What is needed:

- BayeScan installed locally (http://cmpg.unibe.ch/software/BayeScan/download.html)
- VCF file converted to bayescan format (done using PGD spider, code at the end)
- The slurm script below to run BayeScan
- The R script below to plot results and identify outliers (the cut-off used was a q-value <= 0.05. The q-value is in one of Bayescan's output file, the one ending with _fst.txt)
- The outputs from the Bayescan run (_fst.txt, which will have the results and .sel, which can be used to check the run for convergence)
- A csv file with ID info from the vcf file (loci_info_vcfStyle.csv), for matching the locus ID of the outliers found with Bayescan with the locus ID of the vcf file

In [ ]:

```
less bayescenv.sh

#!/bin/bash -e
#SBATCH --job-name=bayescnev
#SBATCH --account=uoa00538
#SBATCH --time=72:00:00
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH --mem=24G
#SBATCH --output=slurm.out
#SBATCH --error=slurm.err

./bayescan_2.1 allConservedTObayescan.txt -o all_ConservednotNeutral -out_freq
```

In [ ]:

```
#        This file is used to plot figures for the software Bayescan in R.

#    This program, BayeScan, aims at detecting genetics markers under selection,
#        based on allele frequency differences between population.
#    Copyright (C) 2010  Matthieu Foll
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.

#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>.

# Arguments:
# - file is the name of your file ex: "output_fst.txt"
# - the q-value threshold corresponding to the target False Discovery Rate (FDR)
# - size is the size of the points and text labels for outliers
# - pos is the distance between the points and the labels
# - highlight is a optional list of marker indices to display in red.
# - name_highlighted alows to write the indices of highlighted markers instead of using a point like the other markers
# - add_text adds the indices of the outlier markers

# Output:
# This function returns different paremeters in a list
# - outliers: the list of outliers
# - nb_outliers: the number of outliers

# Typical usage:
# - load this file into R (file/source R code)
# - in R, go to the directory where "output_fst.txt" is (file/change current dir)
# - at the R prompt, type
# > plot_bayescan("output_fst.txt",0,FDR=0.05)
# if you save the output in a variable, you can recall the different results:
# results<-plot_bayescan("output_fst.txt",0,FDR=0.05)
# results$outliers
# results$nb_outliers

#
# plotting posterior distribution is very easy in R with the output of BayeScan:
# first load the output file *.sel produced by BayeScan
# > mydata=read.table("bi.sel",colClasses="numeric")
# choose the parameter you want to plot by setting for example:
# parameter="Fst1"
# then this line will make the plot for:
# > plot(density(mydata[[parameter]]),xlab=parameter,main=paste(parameter,"posterior distribution"))
# you can plot population specific Fst coefficient by setting
# parameter="Fst1"
# if you have non-codominant data you can plot posterior for Fis coefficients in each population:
# parameter="Fis1"
# if you test for selection, you can plot the posterior for alpha coefficient for selection:
# parameter="alpha1"
# you also have access to the likelihood with:
# parameter="logL"
# if you have the package "boa" installed, you can very easily obtain Highest Probability
# Density Interval (HPDI) for your parameter of interest (example for the 95% interval):
# > boa.hpd(mydata[[parameter]],0.05)


plot_bayescan<-function(res,FDR=0.05,size=1,pos=0.35,highlight=NULL,name_highlighted=F,add_text=T)
{
if (is.character(res))
  res=read.table(res)

colfstat=5
colq=colfstat-2

highlight_rows=which(is.element(as.numeric(row.names(res)),highlight))
non_highlight_rows=setdiff(1:nrow(res),highlight_rows)

outliers=as.integer(row.names(res[res[,colq]<=FDR,]))

ok_outliers=TRUE
if (sum(res[,colq]<=FDR)==0)
        ok_outliers=FALSE;

res[res[,colq]<=0.0001,colq]=0.0001

# plot
plot(log10(res[,colq]),res[,colfstat],xlim=rev(range(log10(res[,colq]))),xlab="log10(q value)",ylab=names(res[colfstat]),type="n")
points(log10(res[non_highlight_rows,colq]),res[non_highlight_rows,colfstat],pch=19,cex=size)

if (name_highlighted) {
        if (length(highlight_rows)>0) {
                text(log10(res[highlight_rows,colq]),res[highlight_rows,colfstat],row.names(res[highlight_rows,]),col="red",cex=size*1.2,font=2)
        }
}
else {
        points(log10(res[highlight_rows,colq]),res[highlight_rows,colfstat],col="red",pch=19,cex=size)
        # add names of loci over p and vertical line
        if (ok_outliers & add_text) {
                text(log10(res[res[,colq]<=FDR,][,colq])+pos*(round(runif(nrow(res[res[,colq]<=FDR,]),1,2))*2-3),res[res[,colq]<=FDR,][,colfstat],row.names(res[res[,colq]<=FDR,]),cex=size)
        }
}
lines(c(log10(FDR),log10(FDR)),c(-1,1),lwd=2)

return(list("outliers"=outliers,"nb_outliers"=length(outliers)))
}

# My data ----------------------------------------------------------------

#1. Read the function above into R
#Plot the Fst results
fst_results = plot_bayescan("all_ConservednotNeutral_fst.txt", 0, FDR=0.05)
plot_bayescan("all_ConservednotNeutral_fst.txt")
#Identify outliers
outliers = fst_results$outliers
length(outliers)
fst_results$nb_outliers #this just gives you the length of the outliers
write.table(outliers, 'outliers.csv', sep = ",")

# plotting posterior distribution of the parameters
#read the file with the .sel extension in R
my_sel = read.table("all_ConservednotNeutral.sel", colClasses = "numeric")
#define which paramter to plot
parameter = "Fst1" #This is the Fst in population 1
plot(density(my_sel[[parameter]]), xlab = parameter, main = paste(parameter,"posterior distribution"))
#obtain Highest Probability Density Interval (HPDI) for your parameter of interest (example for the 95% interval):
library(boa)
boa.hpd(my_sel[[parameter]], 0.05)

#plotting the above for all populations
fsts = c("Fst1","Fst2","Fst3","Fst4","Fst5","Fst6","Fst7","Fst8","Fst9","Fst10","Fst11","Fst12","Fst13","Fst14","Fst15","Fst16 ")
for (i in fsts) {
    parameter = i
    plot1 = plot(density(my_sel[[parameter]]), xlab = parameter, main = paste(parameter,"posterior distribution"))
    plot2 = boa.hpd(my_sel[[parameter]], 0.05)
    print (plot1)
    print (plot 2)
}


# if you test for selection, you can plot the posterior for alpha coefficient for selection:
# parameter="alpha1" ##There was no alpha in my .sel output file, only in the _fst.txt output file.

# you also have access to the likelihood with:
# parameter="logL"
parameter = "logL"
plot(density(my_sel[[parameter]]), xlab = parameter, main = paste(parameter,"posterior distribution"))
boa.hpd(my_sel[[parameter]], 0.05)


##Customised code and plots for finding outliers in the Bayescan output
library(ggplot2)
results = read.table("all_ConservednotNeutral_fst.txt", header = TRUE)
#save a table with the q-values for each locus
locus = 1:10987
qvalues = results$qval
df = as.data.frame(cbind(locus, qvalues))
colnames(df) = c('locus', 'qvalue')
#write.table(df, "all_loci_qvalues.csv", sep = ",", row.names = FALSE)
ggplot(results, aes(x=fst, y=alpha)) + geom_point()
ggplot(results, aes(x=fst, y=alpha)) + geom_text(label = rownames(results))
ggplot(results, aes(x=fst, y=alpha, colour = qval)) + geom_point() + scale_colour_continuous()

#Classic Bayescan plot
plot_bayescan("all_ConservednotNeutral_fst.txt",1, FDR = 0.05)
ggplot(results, aes(x = -log10(qval), y = fst)) +
  geom_point() +
  geom_segment(aes(y=min(fst), yend=max(fst)+0.2, x=1.3, xend=1.3), color="black") +
  labs(x = "-log10(qvalue)", y = "Fst") +
    theme_classic()

ggplot(results, aes(x=qval, y=alpha)) + geom_point()


which(results$alpha<0)
length(which(results$alpha<0))

#Distributions (all loci)
hist(results$qval)
hist(results$prob)
hist(results$log10.PO.)
hist(results$alpha)
hist(results$fst)


#Match Bayescan outliers with the loci ID in the vcf file
#Read a csv file with ID info from the vcf file
all_loci = read.csv("loci_info_vcfStyle.csv", header = T)
#read a csv file with the outlier position detected by Bayescan (in the MOD dataset (in which I added line numbers under the ID column in the vcf file), the outlier position matches the line number, which matches the SNP id in the vcf)
bayscanOUT = read.csv("outliers.csv", header = T)

head(all_loci)
head(bayscanOUT)

#Id which loci are outliers for Bayescan and mark them with TRUE
all_loci$bayscanOUT = all_loci$ID %in% bayscanOUT$outliers

#add that column to the table with vcf information
write.table(all_loci, "loci_info_vcfStyle.csv", sep = ",")


#make a list with the real vcf ID of the Bayscan outliers
only_outs = all_loci[all_loci$bayscanOUT==TRUE,]
dim(only_outs)

outs_id = only_outs$CHROM
length(outs_id)

write.table(outs_id, 'vcfID_bayscanOutliers.csv', sep = ",")
```

PCAdapt

PCAdapt runs in R. Create a directory to save all tables and pltos from PCAdapt.

What is needed:

- PCAdapt, qvalue, tidyverse and ggplot2 installed. For PCAdapt tutorial: https://bcm-uga.github.io/pcadapt/articles/pcadapt.html
- Filtered VCF file
- The R script below to run analyses, plot results and identify outliers (the cut-off used was a q-value <= 0.05.)
- A csv file with ID info from the vcf file (loci_info_vcfStyle.csv), for matching the locus ID of the outliers found with PCAdapt with the locus ID of the vcf file

In [ ]:

```
#Load packages
library(pcadapt)
library(qvalue)
library(ggplot2)

#load data (in vcf fotmat)
path_to_file <- "../../files_vcf-env-popmap/All_conserved/conserved_MOD.vcf"

#read popmap info file (this file has information on individual ID, population ID, group or clade ID (e.g., North, Central and Southern))
info = read.csv("id_pop_env.csv", header = T)

#You should use the read.pcadapt function to convert your genotype file to the pcadapt format.
#pcadapt files should have individuals in columns, SNPs in lines, and missing values should be
#encoded with 9's.
filename <- read.pcadapt(path_to_file, type = "vcf")

#Perform PCA to choose the number of K (should be done with a large enough number of PCS)
#The function will perform a PCA and also compute test statistcs and p-values based on the
#correlation between SNPs and the principal components
#Data is assumed to be diploid by default (but use ploidy = 2 or ploidy =1 to specify it)
x = pcadapt(filename, K = 50)

#to check the proportion of variance of each PC, look at x$singular.values. This is the squared root of the proportion of
#variance explained by each PC (https://bcm-uga.github.io/pcadapt/articles/pcadapt.html). Thus, to find the proportion of variance, do:
proportion_of_variance = (x$singular.values)^2
proportion_of_variance
class(proportion_of_variance)

#make a scree plot to display in decreasing order the percentage of variance explained by each PC.
par(bg="white")
plot(x, option = "screeplot") + theme_classic() + scale_colour_manual (values = "black") + theme(text = element_text(size=20))
plot(x, option = "screeplot", K = 10) + theme_classic() + scale_colour_manual (values = "black") #reduce the number of PCs ploted to only the 10 first

#Make scores plots to display population structure (PC1 vs.2 is default, but can be changed to
#display other PCs as well, and should be done to investigate until which PC there is population
#structure in the dataset)
poplist.int = info$pop #create a vector with the populations of each individual
grouplist.int = info$group # create a vector with the group of each individual
plot(x, option = "scores", pop = poplist.int)
plot(x, option = "scores", pop = grouplist.int)
plot(x, option = "scores", i = 3, j = 4, pop = poplist.int) # i sets the PC in the X axis and j sets the PC in the y axis
plot(x, option = "scores", i = 3, j = 4, pop = grouplist.int)
plot(x, option = "scores", i = 5, j = 6, pop = poplist.int)
plot(x, option = "scores", i = 5, j = 6, pop = grouplist.int)
plot(x, option = "scores", i = 7, j = 8, pop = poplist.int)
plot(x, option = "scores", i = 7, j = 8, pop = grouplist.int)
plot(x, option = "scores", i = 9, j = 10, pop = poplist.int)
plot(x, option = "scores", i = 9, j = 10, pop = grouplist.int)
plot(x, option = "scores", i = 11, j = 12, pop = poplist.int)
plot(x, option = "scores", i = 11, j = 12, pop = grouplist.int)
plot(x, option = "scores", i = 13, j = 14, pop = poplist.int)
plot(x, option = "scores", i = 13, j = 14, pop = grouplist.int)
plot(x, option = "scores", i = 15, j = 16, pop = poplist.int)
plot(x, option = "scores", i = 15, j = 16, pop = grouplist.int)

#CUSTOM PLOTS
dat = as.data.frame(x$scores)
#Defining a different colour for each population
#Specify the factor levels in the order you want
poplist.int = factor(info$pop, levels = c("RU", "TI", "NC", "EC", "WE", "CP", "CR", "DU", "AK", "CU", "ST", "AU", "CA", "AN", "BO", "CH"))
#Create a custom color scale (called cols_pop_scale)
cols_pop = c("#cc0000", "#ff0000", "#990066", "#ff33cc", "#ff9900", "#ffcc00", "#003300", "#006600", "#339933", "#33ff33", "#99cc66", "#3399ff", "#33ccff", "#99cccc", "#669999", "#000099")
names(cols_pop) = levels(poplist.int)
cols_pop_scale = scale_colour_manual(name = "poplist.int", values = cols_pop)
#plot data using the custum colour scale
#PC1 vs PC2
ggplot(dat, aes(x = dat[,1], y = dat[,2], colour = poplist.int)) +
  geom_point(size = 3, alpha = 1, position=position_dodge2(width=.01, padding = .2, preserve = "total")) +
  labs(x = "PC1",
       y = "PC2",
       title = "PC1 versus PC2, complete dataset") +
  cols_pop_scale +
  theme_classic()
#PC9 vs PC10
ggplot(dat, aes(x = dat[,9], y = dat[,10], colour = poplist.int)) +
  geom_point(size = 6) +
  labs(x = "PC9",
       y = "PC10",
       title = "PC9 versus PC10, complete dataset") +
  cols_pop_scale +
  theme_classic()

#Defining a different colour for each group/clade
#Specify the factor levels in the order you want
grouplist.int = factor(grouplist.int, levels = c("North_Is", "Central_NZ", "South_Is", "Subantarctics"))
#Create a custom color scale (called cols_group_scale)
cols_group = c("#ff3333", "#ff9900", "#33ff00", "#3333ff")
names(cols_group) = levels(grouplist.int)
cols_group_scale = scale_colour_manual(name = "grouplist.int", values = cols_group)
#plot data using the custum colour scale
#PC1 vs PC2
ggplot(dat, aes(x = dat[,1], y = dat[,2], colour = grouplist.int)) +
  geom_point(size = 6) +
  labs(x = "PC1",
       y = "PC2",
       title = "PC1 versus PC2, complete dataset") +
  cols_group_scale +
  theme_classic()
# PC9 vs PC10
ggplot(dat, aes(x = dat[,9], y = dat[,10], colour = grouplist.int)) +
  geom_point(size = 6) +
  labs(x = "PC9",
       y = "PC10",
       title = "PC9 versus PC10, complete dataset") +
  cols_group_scale +
  theme_classic()

#Computing the test statistic based on PCA
#For a given SNP, the test statistic is based on the z-scores obtained when regressing SNPs
#with the K principal components. The test statistic for detecting outlier SNPs is the
#Mahalanobis distance, which is a multi-dimensional approach that measures how distant
#is a point from the mean. Once divided by a constant (lambda) called the genomic inflation factor, the
#scaled squared distances D2j/lambda should have a chi-square distribution with K degrees of
#freedom under the assumption that there are no outlier.

#If  K=9 or K=3 corresponds to the optimal choice of the number of PCs, do:
x <- pcadapt(filename, K = 9)
x3 <- pcadapt(filename, K = 3)
#In addition to the number K of principal components to work with, the user can also set the
#parameter min.maf that corresponds to a threshold of minor allele frequency. By default,
#the parameter min.maf is set to 5%. P-values of SNPs with a minor allele frequency smaller
#than the threshold are not computed (NA is returned). My dataset was filtered for MAF, so min.maf did not make a difference in this analysis.

summary(x)
summary(x3)
#We assume in the following that there are n individuals and L markers.
 #stat is a vector of size L containing squared Mahalanobis distances by default.
 #pvalues is a vector containing L p-values.
 #maf is a vector of size L containing minor allele frequencies.
 #gif is a numerical value corresponding to the genomic inflation factor estimated from stat.
 #chi2.stat is a vector of size L containing the rescaled statistics stat/gif that follow a chi-squared distribution with K degrees of freedom.
 #scores is a (n,K) matrix corresponding to the projections of the individuals onto each PC.
 #loadings is a (L,K) matrix containing the correlations between each genetic marker and each PC.
 #singular.values is a vector containing the K ordered squared root of the proportion of variance explained by each PC.
 #zscores is a (L,K) matrix containing the z-scores.
#All of these elements are accessible using the $ symbol. For example, the p-values are contained in x$pvalue

#Plot chi-squared distribution of p-values that are rescaled using stat/gif (chi2.stat)
d = density(x3$chi2.stat,  na.rm = TRUE)
d
plot(d)
plot(d, xlim = c(-3,100))
hist(x$chi2.stat)

#Manhattah Plot
#A Manhattan plot displays -log10 of the p-values
plot(x , option = "manhattan")
plot(x3 , option = "manhattan") + geom_segment(aes(x=0, xend=length(x3$pvalues), y=1.3, yend=1.3), color="red", size = 1) + theme_classic() #adds a red line to p-value = 0.05 (-log10(0.05) = 1.3)


#Q-Q Plot
#Check the distribution of the p-values using a Q-Q plot
plot(x, option = "qqplot", threshold = 0.05)

plot(x3, option = "qqplot", threshold = 0.05) +
  theme_classic() + scale_colour_manual(values = "black") +
  geom_segment(aes(x=1.3, xend=1.3, y=-1, yend=50), color="blue", size = 1) +
  coord_cartesian(xlim = c(0, 5), ylim = c(0, 50)) +
  theme_classic() #adds a blue line to expected p-value = 0.05 (which is -log10(1.3)). On the right side of this line, there will be the top 5% observed p-values
# On the right side of the grey dotted line are the top 5% lowest p-values.
#This plot shows if the p-values follow the expected uniform distribution, and if the smallest
#p-values are smaller than expected (confirming the presence of outliers)

#Histograms of the test statistic and of the p-values
#A histogram of p-values shows if most of the p-values follow the uniform distribution,
#and that the excess of small p-values indicates the presence of outliers.
hist(x$pvalues, xlab = "p-values", main = NULL, breaks = 50, col = "orange")
hist(x3$pvalues, xlab = "p-values", main = NULL, breaks = 50, col = "orange")
#The presence of outliers is also visible when plotting a histogram of the test statistic Dj.
plot(x, option = "stat.distribution")
plot(x3, option = "stat.distribution")

#Choosing a cutoff for outlier detection
#To provide a list of outliers, use the R package qvalue, transforming the
#p-values into q-values.
#For a given alpha (real valued number between 0 and 1), SNPs with q-values less than alpha will be
# considered as outliers with an expected false discovery rate bounded by alpha. The false discovery
# rate is defined as the percentage of false discoveries among the list of candidate SNPs.
# To provide a list of candidate SNPs with an expected false discovery rate lower than 5%:
qval <- qvalue(x$pvalues)$qvalues
alpha <- 0.05
outliers <- which(qval < alpha)
length(outliers)

qval3 <- qvalue(x3$pvalues)$qvalues
qvalues3= qvalue(x3$pvalues)
alpha3 <- 0.05
outliers3 <- which(qval3 < alpha3)
length(outliers3)
#[1] 2974
hist(qval3, xlab = "q-values", main = NULL, breaks = 50, col = "orange")
write.table(outliers3, 'outliers_qval005.csv', sep = "," )
#save qvalues for each loci in a table
locus = 1:10987
qval3
df = cbind(locus, qval3)
class(df)
df = as.data.frame(df)
colnames(df) = c('locus', 'qvalue')
write.table(df, "all_loci_qvalues.csv", sep = ",", row.names = FALSE)

#Manhattan plot with q-values
qval_plot = -log10(qval3)
locus = c(1:length(qval_plot))
qval_plot = data.frame (qval_plot, locus)
length (qval_plot)
plot(qval_plot) + geom_segment(aes(x=0, xend=9275, y=1.3, yend=1.3), color="red", size = 1) + theme_classic() #adds a red line to q-value = 0.05 (-log10(0.05) = 1.3)
#or
ggplot(qval_plot, aes(x = locus, y = qval_plot)) +
  geom_point() +
  geom_segment (aes(x=0, xend = length(x3$pvalues), y = 1.3, yend=1.3), color = "red", size = 1) +
  theme_classic() +
  labs(title = "Manhattan plot", y = "-log10(qvalue)", x = "locus")
  axis.title = element_text(size = 12, family = "arial") #adds a red line to q-value = 0.05 (-log10(0.05) = 1.3)

#distribution and summary
hist(qvalues3$lfdr)
summary(qvalues3$lfdr)

#To know which principal components are actually the most correlated with the oulier SNPs, use
#the function get.pc()
snp_pc <- get.pc(x, outliers)
write.table(snp_pc, "outliers_pcs.tsv", sep = "\t")
snp3_pc <- get.pc(x3, outliers)
write.table(snp3_pc, "outliers3_pcs.tsv", sep = "\t")

###___####
#Match PCAdapt outliers with the loci ID in the vcf file
#Read a csv file with ID info from the vcf file
all_loci = read.csv("loci_info_vcfStyle.csv", header = T)
#read a csv file with the outlier position detected by PCAdapt (in the compelte dataset, the outlier position matches the line number, which matches the SNP id in the vcf)
pcadapt_out = read.csv("outliers_qval005.csv", header = T)

head(all_loci)
head(pcadapt_out)

#Id which loci are outliers for pcadapt and mark them with TRUE
all_loci$pcadaptOUT = all_loci$ID %in% pcadapt_out$outliers

#add that column to the table with vcf information
write.table(all_loci, "loci_info_vcfStyle.csv", sep = ",")

#make a list with the real vcf ID of the pcadapt outliers
only_outs = all_loci[all_loci$pcadaptOUT==TRUE,]
dim(only_outs)

outs_id = only_outs$CHROM
length(outs_id)

write.table(outs_id, 'vcfID_pcadaptOutliers.csv', sep = ",")
```

Combining q-values of BayeScan and PCAdapt with the geometric mean

Code runs in R. It takes the geometric mean of q-values between methods used to identify outliers (BayeScan and PCAdapt) to combine the output of those methods instead of taking the intersection of outlier loci between them (but a way to take the intersection of loci is also provided). What is needed:

- Output from BayeScan with the qvalue of all loci (in csv format)
- Output from PCAdapt with the qvalue of all loci (in csv format)
- R packages tidyverse, tidyr

In [ ]:

```
library(tidyverse)
library(tidyr)

#Read datasets with q-values from PCAdapt and Bayescan
pca = read.csv("pcadapt_all_loci_qvalues.csv", header = T)
bay = read.csv("bayescan_all_loci_qvalues.csv", header = T)

#Combine the q-values of pcadapt and bayescan
head(pca)
head(bay)
#create a function to calculate the geometric mean of q-values
gmean = function(x, y) {
  sqrt(x*y)
}

#combine the pcadapt and bayescan outputs using the geometric mean of q-values (with the function obove)
combq = gmean (pca$qvalue, bay$qvalue)
head(combq)

#Make a dataframe with all results (combined and uncombined)
df = as.data.frame(cbind(pca$locus, pca$qvalue, bay$qvalue, combq))
head(df)
colnames(df) = c("locus", "PCAdapt_qval", "Bayescan_qval", "combined_qval")
summary(df)

#id outliers using the combined q-value
df$combined_out = ifelse(df$combined_qval <= 0.05, TRUE, FALSE)

#id outliers using the intersection of methods
#pcadapt outliers
df$pca_out = ifelse(df$PCAdapt_qval <= 0.05, TRUE, FALSE)
#bayescan outliers
df$Bay_out = ifelse (df$Bayescan_qval <= 0.05, TRUE, FALSE)
#instersection pcadapt and bayescan outliers
df$outIntersect = ifelse(df$pca_out == TRUE & df$Bay_out == TRUE, TRUE, FALSE)
summary(df)

write.table(df, "outliers_combined_qvals.csv", sep = ",", row.names = FALSE)
```

BayeScEnv

BayeScEnv was run with real environmental variables and with random variables. What is needed for both cases

- BayeScEnv installed locally
- The filtered VCF file converted to Bayescan format (which is also used by BayeScEnv). This was done with PGD spider, code at the end.

Real Environmental Variables: What is needed

- A file for each environmental variable (example giver for mean air temperature below, each column has the value for each population. Specifications can be found in BayeScEnv manual). More information on obtaining and extracting the environmental variables is also given below.
- The slurm script below to run BayeScEnv
- The R script below to plot results and identify outliers (the cut-off used was a q-value <= 0.05. The q-value is in one of BayescEnv's output file, the one ending with _fst.txt)
- The outputs from the BayescEnv runs (two for each environmental variable: _fst.txt, which will have the results and .sel, which can be used to check the run for convergence)

In [ ]:

```
less mean_air.txt
-0.187538249    -1.37885565     -1.140623751    -0.988321799    -1.678099623    0.291675506     0.428913641       0.432218721     -0.716593007    -0.187538249    1.199136564     1.180097937     1.498130364     -0.679646409     1.498130364     0.428913641
```

In [ ]:

```
#!/bin/bash -e
#SBATCH --job-name=bayescnev
#SBATCH --account=uoa00538
#SBATCH --time=48:00:00
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH --mem=24G
#SBATCH --output=slurm.out
#SBATCH --error=slurm.err

#./bayescenv allTObayescenv.txt -env env_file_bayescenv.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanAir.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanLat.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanLong.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanMslp.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanNswrs.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanOsalt.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanOtem.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanPrate.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanPres.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanSalt.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanSshg.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanSst.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanTcdc.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanUwnd.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
./bayescenv allTObayescenv.txt -env meanVwnd.txt -o all_out -nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 -out_pilot -out_freq
```

In [ ]:

```r
#Load packages
library(coda)
library(tibble)
library (ggplot2)


# 1. Analysing convergence and results for each environmental variable separately

#lat
filename = "meanLat_out.sel"
chain <- read.table(filename , header = TRUE)
chain <- mcmc(chain , thin = 10)   #Adapt thin to its actual value (10 is the default)
heidel.diag(chain)             #To test for convergence
effectiveSize(chain)           #To compute effective sample size (good above 1000)
autocorr.diag(chain)           #To look for auto-correlation (good below 0.7)
par(mar=c(1,1,1,1))
plot(chain)                    #To plot the "trace" and the posterior distribution

tablename = "meanLat_out_fst.txt"
dat = read.table(tablename, header = TRUE)
head(dat)
colnames(dat)
dat = add_column(dat, locus = 1:nrow(dat), .before = "PEP_g")
head(dat)
#fst
ggplot(dat, aes(x = locus, y=fst)) + geom_point() + labs(title = tablename)
#fst distribution
dist(dat$fst)
#g
ggplot(dat, aes(x = locus, y=g)) + geom_point() + labs(title = tablename)
#alpha
ggplot(dat, aes(x = locus, y=alpha)) + geom_point() + labs(title = tablename)
#-log10(PEP_g)
ggplot(dat, aes(x = locus, y=-log10(PEP_g))) + geom_point() + labs(title = tablename)
#-log10(qval_g)
ggplot(dat, aes(x = locus, y=-log10(qval_g))) + geom_point() + labs(title = tablename)
#-log10(PEP_alpha)
ggplot(dat, aes(x = locus, y=-log10(PEP_alpha))) + geom_point() + labs(title = tablename)
#-log10(qval_alpha)
ggplot(dat, aes(x = locus, y=-log10(qval_alpha))) + geom_point() + labs(title = tablename)
# Fst versus -log10(qval_g)
ggplot(dat, aes(x = -log10(qval_g), y=fst)) + geom_point() + labs(title = tablename)+
  geom_segment(aes(y=min(fst), yend=max(fst)+0.2, x=1.3, xend=1.3), color="black") + theme_classic()
#Fst versus -log10(qval_alpha)
ggplot(dat, aes(x = -log10(qval_alpha), y=fst)) + geom_point() + labs(title = tablename)+
  geom_segment(aes(y=min(fst), yend=max(fst)+0.2, x=1.3, xend=1.3), color="black") +theme_classic()
#Rinse and repeat. Change the variables below and re-run the code above
#long
filename = "meanLong_out.sel"
tablename = "meanLong_out_fst.txt"
#meanAir
filename = "meanAir_out.sel"
tablename = "meanAir_out_fst.txt"
#meanMslp
filename = "meanMslp_out.sel"
tablename = "meanMslp_out_fst.txt"
#meanNswrs
filename = "meanNswrs_out.sel"
tablename = "meanNswrs_out_fst.txt"
#meanOsalt
filename = "meanOsalt_out.sel"
tablename = "meanOsalt_out_fst.txt"
#meanOtem
filename = "meanOtem_out.sel"
tablename = "meanOtem_out_fst.txt"
#meanPrate
filename = "meanPrate_out.sel"
tablename = "meanPrate_out_fst.txt"
#meanPres
filename = "meanPres_out.sel"
tablename = "meanPres_out_fst.txt"
#meanSalt
filename = "meanSalt_out.sel"
tablename = "meanSalt_out_fst.txt"
#meanSshg
filename = "meanSshg_out.sel"
tablename = "meanSshg_out_fst.txt"
#meanSst
filename = "meanSst_out.sel"
tablename = "meanSst_out_fst.txt"
#meanTcdc
filename = "meanTcdc_out.sel"
tablename = "meanTcdc_out_fst.txt"
#meanUwnd
filename = "meanUwnd_out.sel"
tablename = "meanUwnd_out_fst.txt"
#meanVwnd
filename = "meanVwnd_out.sel"
tablename = "meanVwnd_out_fst.txt"

#2. Counting loci with sigifican alpha and g (-log10(qval) = 1.3)

env_variable = "meanLat_out_fst.txt"
df = read.table (env_variable, header= TRUE)
g = which (df$qval_g < 0.05)
length(g)
alpha = which (df$qval_alpha < 0.05)
length(alpha)
print(g)
print(alpha)
#Rinse and repeat. Change the variables below and re-run the code above
env_variable = "meanLong_out_fst.txt"
env_variable = "meanAir_out_fst.txt"
env_variable = "meanMslp_out_fst.txt"
env_variable = "meanNswrs_out_fst.txt"
env_variable = "meanOsalt_out_fst.txt"
env_variable = "meanOtem_out_fst.txt"
env_variable = "meanPrate_out_fst.txt"
env_variable = "meanPres_out_fst.txt"
env_variable = "meanSalt_out_fst.txt"
env_variable = "meanSshg_out_fst.txt"
env_variable = "meanSst_out_fst.txt"
env_variable = "meanTcdc_out_fst.txt"
env_variable = "meanUwnd_out_fst.txt"
env_variable = "meanVwnd_out_fst.txt"

#3. (optional) extract all outliers and make a data frame with it
na.pad <- function(x,len){
  x[1:len]
}
makePaddedDataFrame <- function(l,...){
  maxlen <- max(sapply(l,length))
  data.frame(lapply(l,na.pad,len=maxlen),...)
}

library(magicfor)

file_names=dir(pattern="*_out_fst.txt")
magic_for(print, silent = TRUE)

for (f in file_names) {
  df = read.table(f, header = TRUE)
  g = which (df$qval_g < 0.05)
  length(g)
  alpha = which (df$qval_alpha < 0.05)
  length(alpha)
  print(g)
  print(alpha)
}
results = magic_result_as_vector()
outliers_realVar = makePaddedDataFrame(results)
write.table(outliers_realVar, 'outliers_realVar.csv', sep = ",")

#
library(dplyr)
library(magicfor)

filelist=dir(pattern="*_out_fst.txt")
magic_for(print, silent = TRUE)

for (file in filelist) {
  df = read.table(file, header = TRUE)
  select (df, 2, 5)
  colnames(df) <- c(file, paste(file, 'qval_g', sep = ""), paste(file, 'qval_alpha', sep = ""))
}
results = magic_result_as_vector()
head(results)
results


#4. Store the qvalues for aplha and g in a different object for each environmental variable and make a dat
aframe with all results (a different env variable in each column for g and alpha)
#This will be the most helpful table to define outliers later on
data_all = data.frame(1:10987,1) #change the 10987 to reflect the number of loci in the dataset
head(data_all)
data_all = select(data_all,1)
colnames(data_all) = 'loci'
head(data_all)
#Lat data
lat = read.table("meanLat_out_fst.txt", header = T)
data_all$latG = lat$qval_g
data_all$latAlpha = lat$qval_alpha
#Long data
long = read.table("meanLong_out_fst.txt", header = T)
data_all$longG = long$qval_g
data_all$longAlpha = long$qval_alpha
#mslp data
mslp = read.table("meanMslp_out_fst.txt", header = T)
data_all$mslpG = mslp$qval_g
data_all$mslpAlpha = mslp$qval_alpha
#nswrs data
nswrs = read.table("meanNswrs_out_fst.txt", header = T)
data_all$nswrsG = nswrs$qval_g
data_all$nswrsAlpha = nswrs$qval_alpha
#osalt data
osalt = read.table("meanOsalt_out_fst.txt", header = T)
data_all$osaltG = osalt$qval_g
data_all$osaltAlpha = osalt$qval_alpha
#otemp data
otemp = read.table("meanOtem_out_fst.txt", header = T)
data_all$otempG = otemp$qval_g
data_all$otempAlpha = otemp$qval_alpha
#prate data
prate = read.table("meanPrate_out_fst.txt", header = T)
data_all$prateG = prate$qval_g
data_all$prateAlpha = prate$qval_alpha
#pres data
pres = read.table("meanPres_out_fst.txt", header = T)
data_all$presG = pres$qval_g
data_all$presAlpha = pres$qval_alpha
#salt data
salt = read.table("meanSalt_out_fst.txt", header = T)
data_all$saltG = salt$qval_g
data_all$saltAlpha = salt$qval_alpha
#sshg data
sshg = read.table("meanSshg_out_fst.txt", header = T)
data_all$sshgG = sshg$qval_g
data_all$sshgAlpha = sshg$qval_alpha
#sst data
sst = read.table("meanSst_out_fst.txt", header = T)
data_all$sstG = sst$qval_g
data_all$sstAlpha = sst$qval_alpha
#tcdc data
tcdc = read.table("meanTcdc_out_fst.txt", header = T)
data_all$tcdcG = tcdc$qval_g
data_all$tcdcAlpha = tcdc$qval_alpha
#uwnd data
uwnd = read.table("meanUwnd_out_fst.txt", header = T)
data_all$uwndG = uwnd$qval_g
data_all$uwndAlpha = uwnd$qval_alpha
#vwnd data
vwnd = read.table("meanVwnd_out_fst.txt", header = T)
data_all$vwndG = vwnd$qval_g
data_all$vwndAlpha = vwnd$qval_alpha
#Air data
air = read.table("meanAir_out_fst.txt", header = T)
data_all$airG = air$qval_g
data_all$airAlpha = air$qval_alpha

head(data_all)
dim(data_all)
getwd()
write.table(data_all, "summary_qVal_allRealEnv.csv", sep = ",", row.names = FALSE)

#5. Define outliers using the summary_qVal_allRealEnv.csv created above
data_all = read.csv("summary_qVal_allRealEnv.csv", header = T)
head(data_all)
str(data_all)
#Lat outs
length(which(data_all$latAlpha < 0.05)) #1391
length(which(data_all$latG < 0.05)) #0
#Long outs
length(which(data_all$longAlpha < 0.05)) #1437
length(which(data_all$longG < 0.05)) #0
#mslp
length(which(data_all$mslpAlpha < 0.05)) #1486
length(which(data_all$mslpG < 0.05)) #0
#nswrd
length(which(data_all$nswrsAlpha < 0.05)) #206
length(which(data_all$nswrsG < 0.05)) #2704
#osalt
length(which(data_all$osaltAlpha < 0.05)) #140
length(which(data_all$osaltG < 0.05)) #3064
#otemp
length(which(data_all$otempAlpha < 0.05)) #477
length(which(data_all$otempG < 0.05)) #485
#prate
length(which(data_all$prateAlpha < 0.05)) #1211
length(which(data_all$prateG < 0.05)) #0
#pres
length(which(data_all$presAlpha < 0.05)) #865
length(which(data_all$presG < 0.05)) #0
#salt
length(which(data_all$saltAlpha < 0.05)) #333
length(which(data_all$saltG < 0.05)) #1252
#sshg
length(which(data_all$sshgAlpha < 0.05)) #1533
length(which(data_all$sshgG < 0.05)) #0
#sst
length(which(data_all$sstAlpha < 0.05)) #1177
length(which(data_all$sstG < 0.05)) #0
#tcdc
length(which(data_all$tcdcAlpha < 0.05)) #1473
length(which(data_all$tcdcG < 0.05)) #0
#uwnd
length(which(data_all$uwndAlpha < 0.05)) #186
length(which(data_all$uwndG < 0.05)) #2872
#vwnd
length(which(data_all$vwndAlpha < 0.05)) #1736
length(which(data_all$vwndG < 0.05)) #0
#air
length(which(data_all$airAlpha < 0.05)) #474
length(which(data_all$airG < 0.05)) #12

#allAlpha
library(tidyverse)
allAlpha = data_all %>% select (1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31)
allAlpha$lociID = allAlpha$loci
allAlpha = allAlpha %>%
  gather(loci,'qval_alpha', 1:16) %>%
  filter (qval_alpha < 0.05)
head(allAlpha)
colnames(allAlpha) = c('loci', 'env_variable', 'qval_alpha')
head(allAlpha)
duplicated_alpha = (duplicated(allAlpha$loci)==TRUE)
length(which(duplicated_alpha==FALSE)) #1736 Unique outliers
summary(duplicated_alpha)
summary(allAlpha)

#allG
library(tidyverse)
allG = data_all %>% select (1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30)
allG$lociID = allG$loci
allG = allG %>%
  gather(loci,'qval_g', 1:16) %>%
  filter (qval_g < 0.05)
head(allG)
colnames(allG) = c('loci', 'env_variable', 'qval_g')
head(allG)
duplicated_g = (duplicated(allG$loci)==TRUE)
length(which(duplicated_g==FALSE)) #3093 Unique outliers
summary(duplicated_g)
summary(allG)
```

Random Variables: What is needed

- An R code to generate 100 random variables to be used as environmental files (will be reused in the RDA analyses)
- A population map (id_pop.csv)
- The slurm script below to run BayeScEnv, in a directory that has a subdirectory called env_all
- The R script below to identify outliers (the cut-off used was a q-value <= 0.05. The q-value is in one of BayescEnv's output file, the one ending with _fst.txt)
- The outputs from the BayeScEnv run on each random variable (focus on the _fst.txt, which will have the results. The .sel will also be generated, and can be used to check the run for convergence)

In [ ]:

```r
#generate a set of 16 random values to be the environmental variables, but do that 100 times
  nrSamples = 100  #how many times you want to generate the variables
  e <- list(mode="vector",length=nrSamples)  #"e" is a list to store the different variables
  for (i in 1:nrSamples) {
    e[[i]] <- rnorm(n = 16, mean = 0, sd = 1) #for loop generates 16 random variables normally distributed
 for nrSamples length (100 in this case)
  }
  e = as.data.frame(e) #convert the list e to a data frame

#For BayeScEnv --------------
#save each of the columns of e (each of the 100 sets of 16 random variables) to a different env file
for (i in 1:ncol(e)) {
  filename = paste(i,".env", sep = "") #define each of the 100 file names
  write.table(t(e[[i]]), filename , sep = "\t", row.names = FALSE, col.names = FALSE) #save each of the co
lumns of the dataframe e in a different file

#For RDA --------------
#save it as a dataframe with information on population ID
  dim(e)
  colnames(e) = c(1:100)
  colnames(e) = paste('X', colnames(e), sep = "")
  class(colnames(e))
  class(e)
  rownames(e) = c("AK", "AN", "AU", "BO", "CA", "CH", "CP", "CR", "CU", "DU", "EC", "NC", "RU", "ST", "TI"
, "WE")
  e$pops = as.character(rownames(e))
  head(e)
  dim(e)
  #read individual, population and group information
  pop_info= read.csv("id_pop.csv")
  #Organize environmental data based on population information
  env_all = merge(pop_info, e, by.x = 2, by.y = 0, all.x = TRUE) # The order of the data frames to merge i
s importat -> pop_info should come first, as e will be replicated for each pop
  dim(env_all)
  env_all$ind <- as.character(env_all$ind) # Make individual names characters (not factors)
  env_all$pop <- as.character(env_all$pop)
  env_all$group <- as.character(env_all$group)
  write.table(env_all, "all_arbitraryEnv.csv", sep = "")
```

In [ ]:

```bash
#!/bin/bash -e
#SBATCH --job-name=bayescenv
#SBATCH --account=uoa00538
#SBATCH --time=36:00:00
#SBATCH --cpus-per-task=8
#SBATCH --mem=1600
#SBATCH --array=1-100
#SBATCH --output=slurm%j-%a.out

./bayescenv allTObayescenv.txt \
-env ./env_all/${SLURM_ARRAY_TASK_ID}.env \
-o ./env_all/${SLURM_ARRAY_TASK_ID}.out \
-nbp 20 -pilot 5000 -thin 10 - n 5000 -burn 50000 \
-out_pilot -out_freq
```

In [ ]:

```r
file_names=dir(pattern="*.out.sel")

#Checking if the MCMC run converged
library(coda)
for (f in file_names) {
  chain <- read.table(f, header = TRUE)
  chain <- mcmc(chain , thin = 10)   #Adapt thin to its actual value (10 is the default)
  heidel.diag(chain)             #To test for convergence
  effectiveSize(chain)           #To compute effective sample size (good above 1000)
  autocorr.diag(chain)           #To look for auto-correlation (good below 0.7)
  par(mar=c(1,1,1,1))
  plot(chain)                    #To plot the "trace" and the posterior distribution
}
########################################################################
# SUMMARIZE DATA INCLUDING RANDOM LOCI, RANDOM ENV, AND q-value
#Try to read each data.frame and combine them into one big dataframe
library(tidyverse)
#setwd("env_all/")
#If you want to create a column in each of the data frames from the results with the locus ID, the loop be
low will do it and save each data frame as a new table in the working directory
#Create a common column in all dataframes for merging with rbind later (locus ID)
list.files()
file_names=dir(pattern="*.out_fst.txt")
loci=1:10987
for (f in file_names) {
  dfi = read.table(f, header = T)
  dfi$loci = 1:10987
  write.table(dfi, paste(f, '.MOD.csv', sep = ""), sep = ",")
}
allframes = lapply(1:100,function(x)read.csv(paste(x,'out_fst.txt.MOD.csv',sep='.')))
sapply(allframes,dim) #all data frames have all the loci 10987
bigdf = bind_rows(allframes) #if you bind them loke this, each run will take 13004 rows, so each variable
 will be organized under the correspondent column
#Now create a column called env that will contain the random env of each result@###
bigdf$env = rep(1:100, each = 10987)
summary(bigdf)
#rep(x, times = 1, length.out = NA, each = 1)

#select only the columns of qval, loci and env and spread dataframe (long to wide) across env variables, a
md transform q-values into 0 (larger than 0.05) and 1 (smaller than 0.05)
#g
gdf = bigdf %>%
  select("loci", "env", "qval_g")
gdf = gdf %>% spread(key = "env", value = "qval_g")
str(gdf)
head(gdf)
summary(gdf)
write.table (gdf, "../summary2021_qvaluesRandomEnv_Bayescenv.csv", sep = ",", row.names = F)
```

RDA

RDA implemented on the R package vegan was run with real environmental variables and with random variables. In addition, the test-statistic from the RDA is z-scores (comming from RDA loadings), and it was converted to q-values with a custom script. What is needed for running with both real environmental and random variables:

- The vegan and qvalue packages in R (in addition to other packages - check R script below)
- The filtered VCF file.
- A population map
- An environmental file (a csv file with individual, population and environmental data information for each individual sample. In the case of the random variables, the environmental variables were replaced by the same variables used in the random runs with BayeScEnv - code above) The RDA does not run with missing data, so imputation was done as follows
- NZ-wide was imputed using the most common genotype within clades
- Within-clade datasets were imputed using the most common genotype across all individuals

In [ ]:

```r
#Imputation
library(adegenet)
library(vcfR)

#----------NZ-all----------------
#Read the data
#Read vcf file with vcfR
vcf = read.vcfR("input.vcf")
vcf
#convert data to a genind pbject
gen = vcfR2genind(vcf)
gen = genind2df(gen)
dim(gen)
#trying to imput missing values using the most common genotype at each SNP within a population
sum(is.na(gen))
str(gen)
rownames(gen)
#AK - 1 - 9
#AN - 10 - 17
#AU - 18 - 26
#BO - 27 - 36
#CA - 37 - 46
#CH - 47 - 55
#CP - 56 - 65
#CR - 66 - 71
#CU - 72 - 76
#DU - 77 - 96
#EC - 97 - 105
#NC - 106 - 115
#RU - 116 - 129
#ST - 130 - 137
#TI - 138 - 149
#WE - 150 - 167

#Proportion of missing data per population (NA in population/total NA)
sum(is.na(gen[1:9,]))/sum(is.na(gen)) # AK 0.07272051
sum(is.na(gen[10:17,]))/sum(is.na(gen)) # AN 0.08951131
sum(is.na(gen[18:26,]))/sum(is.na(gen)) # AU 0.0369852
sum(is.na(gen[27:36,]))/sum(is.na(gen)) #BO 0.04048177
sum(is.na(gen[37:46,]))/sum(is.na(gen)) # CA 0.04503713
sum(is.na(gen[47:55,]))/sum(is.na(gen)) #CH 0.0548761
sum(is.na(gen[56:65,]))/sum(is.na(gen)) # CP 0.08970757
sum(is.na(gen[66:71,]))/sum(is.na(gen)) #CR 0.07930048
sum(is.na(gen[72:76,]))/sum(is.na(gen)) # CU 0.0358696
sum(is.na(gen[77:96,]))/sum(is.na(gen)) #DU 0.08585979
sum(is.na(gen[97:105,]))/sum(is.na(gen)) #EC 0.06887273
sum(is.na(gen[106:115,]))/sum(is.na(gen)) #NC 0.04338956
sum(is.na(gen[116:129,]))/sum(is.na(gen)) #RU 0.07376897
sum(is.na(gen[130:137,]))/sum(is.na(gen)) #ST 0.05992728
sum(is.na(gen[138:149,]))/sum(is.na(gen)) # TI 0.03850882
sum(is.na(gen[150:167,]))/sum(is.na(gen)) # WE 0.0851832

#having a look at AK population first
ak = gen[1:9,]
rownames(ak)
is.na(ak[,1])
aki =
for (i in 1:ncol(ak)) {
  replace (ak[,i], is.na(ak[,i]), as.numeric(names(which.max(table(ak[,i])))))
} #Error in x[list] <- values : replacement has length zero
aki = apply(ak, 2, function(x) replace (x, is.na(x), as.numeric(names(which.max(table(x)))))) #Error in x
[list] <- values : replacement has length zero

#This error means that within the AK population, there are columns (SNPs) that are completely missing, so
 it is impossible to find a value to imput them (there is no replacement, or the replacement has zero leng
th).
#Thus, instead of imputing by population for the RDA, I will try to imput by clade (N, C, S)
#The proportion of missing data within populations is not high, as can be seen above, so this should not h
ave a strong relevance to the conclusions.
#Northern
N = gen[c(97:129,138:149),]
dim(N)
Nimputed = apply(N, 2, function(x) replace (x, is.na(x), as.numeric(names(which.max(table(x))))))
sum(is.na(Nimputed)) #0 NA, imputation works!
#Central
C = gen[c(56:65,150:167),]
dim (C)
Cimputed = apply(C, 2, function(x) replace (x, is.na(x), as.numeric(names(which.max(table(x))))))
sum(is.na(Cimputed)) #0 NA, imputation works!
#Southern
S = gen[c(1:55,66:96, 130:137),]
dim(S)
Simputed = apply(S, 2, function(x) replace (x, is.na(x), as.numeric(names(which.max(table(x))))))
sum(is.na(Simputed)) #0 NA, imputation works!
#Combine the three imputed clades
gen.imp = rbind(Nimputed, Cimputed, Simputed)
dim(gen.imp)
sum(is.na(gen.imp))

#Checking how many loci required imputation and by how much
#Missing in the N
sum(is.na(gen[c(97:129,138:149),]))/sum(is.na(gen)) # 0.2245401
#Missing in the C
sum(is.na(gen[c(56:65,150:167),]))/sum(is.na(gen)) #0.1748908
#Missing in the S
sum(is.na(gen[c(1:55,66:96, 130:137),]))/sum(is.na(gen)) # 0.6005692

#checking the amount of imputation required by loci
gen[,1]
class(gen[,1])
gen.mis = ifelse(is.na(gen)==TRUE, 1 , 0)
gen.mis[,1]
dim(gen.mis)
sum.mis = apply(gen.mis, 2, sum )
summary(sum.mis)
hist(sum.mis) #3000 loci had to be imputed in less than 5 individuals; 1250 had to be imputed in less than
 10 individuals...

#per group
#Northern
N.mis = ifelse(is.na(N)==TRUE, 1 , 0)
N.mis[,1]
dim(N.mis)
sum.Nmis = apply(N.mis, 2, sum )
summary(sum.Nmis)
hist(sum.Nmis)
#Central
C.mis = ifelse(is.na(C)==TRUE, 1 , 0)
C.mis[,1]
dim(C.mis)
sum.Cmis = apply(C.mis, 2, sum )
summary(sum.Cmis)
hist(sum.Cmis)
#Southern
S.mis = ifelse(is.na(S)==TRUE, 1 , 0)
S.mis[,1]
dim(S.mis)
sum.Smis = apply(S.mis, 2, sum )
summary(sum.Smis)
hist(sum.Smis)
#
#In the gen dataset, 00 is homozygous ref allele, 01 is heterozygous and 11 is homozygous to the alternati
ve allele.
#In the gen.imp dataset, 00 is homozygous ref alelle, 1 is heterozygous, and 11 is homozygous to the alt a
llele.

#Read environmental data in
env_all <- read.csv("../envData_all.csv")
str(env_all)
env_all$ind <- as.character(env_all$ind) # Make individual names characters (not factors)
env_all$pop <- as.character(env_all$pop)
env_all$group <- as.character(env_all$group)

#Confirm genotypes and environmenta data are in the same order
identical(rownames(gen.imp), env_all[,1])
#Re-order to match
row.names(gen.imp)
gen.imp = gen.imp[order(row.names(gen.imp)),]
row.names(gen.imp)
identical(rownames(gen.imp), env_all[,1])
class(gen.imp)
gen.imp = as.data.frame(gen.imp)
str(gen.imp)
write.table(gen.imp, "gen.imp.csv", sep = ",") #this file will also be used to run RDA with the 100 random
 variables


#---------------Within-clade datasets-------------------------
###This was done previous to revision, for the first submission. The above was done for the second submiss
ion##
#RDA requires no missing data. Input missing values using the most common genotype at each SNP across all
 individuals
#Read the data
#Read vcf file with vcfR
vcf = read.vcfR("input.vcf")
vcf
#convert data to a genind pbject
gen = vcfR2genind(vcf)
gen = genind2df(gen)
dim(gen)

sum(is.na(gen))
gen.imp <- apply(gen, 2, function(x) replace(x, is.na(x), as.numeric(names(which.max(table(x))))))
sum(is.na(gen.imp)) # No NAs

#Read environmental data in
env_all <- read.csv("../envData_all.csv")
str(env_all)
env_all$ind <- as.character(env_all$ind) # Make individual names characters (not factors)
env_all$pop <- as.character(env_all$pop)
env_all$group <- as.character(env_all$group)

#Confirm genotypes and environmenta data are in the same order
identical(rownames(gen.imp), env_all[,1])
#Re-order to match
row.names(gen.imp)
gen.imp = gen.imp[order(row.names(gen.imp)),]
row.names(gen.imp)
identical(rownames(gen.imp), env_all[,1])
class(gen.imp)
gen.imp = as.data.frame(gen.imp)
str(gen.imp)
write.table(gen.imp, "gen.imp.csv", sep = ",") #this file will also be used to run RDA with the 100 random
 variables
```

RDA with real environmental variables

```r
# Load packages
# -------------
library(psych)    # Used to investigate correlations among predictors
library(vegan)    # Used to run RDA
library(adegenet)
library(vcfR)
library(usdm)       # calculates variance inflation factors
library(robust)     # for robust mahalanobis distance calculation
library(qvalue)     # FDR calculation

#Read datasets
#Environmental dataset
env_all <- read.csv("../envData_all.csv")
str(env_all)
env_all$ind <- as.character(env_all$ind) # Make individual names characters (not factors)
env_all$pop <- as.character(env_all$pop)
env_all$group <- as.character(env_all$group)
#Genomic dataset (imputed)
gen.imp = read.csv("gen.imp.csv", header = TRUE)
# Real predictors are often in different units (e.g. temp., forest cover, soil pH)
# In this case it is best to scale the predictors (which we'll do here as well):
env <- scale(env_all[4:18]) # default is center and scale = T; this subtracts the mean from each value, then divides by the st. dev.
env = as.data.frame(env)
env = cbind.data.frame(env_all$ind, env_all$pop, env_all$group, env)
colnames(env) <- c("ind", "pop", "group", "lat","lon","mean_mslp","mean_nswrs","mean_otemp","mean_prate", "mean_pres","mean_salt", "mean_osalt", "mean_sshg", "mean_sst", "mean_tcdc", "mean_uwnd", "mean_vwnd", "mean_air")
head(env)

#SUBSET THE DATA TO INCLUDE NON-CORRELATED ENV VARIABLES
pred <- subset(env, select=c(ind, pop, group, lat, lon, mean_prate, mean_pres, mean_sshg))
#RUN RDA_____
chiton_rda = rda(formula = gen.imp ~ lat + lon + mean_prate + mean_pres + mean_sshg, data = pred, scale = T)
#RDA SUMMARY OF RESULTS________________________________
chiton_rda
hists = screeplot(chiton_rda) #histogram of the loadings of the RDA axes
load.rda = scores(chiton_rda, choices=c(1:5), display="species")  # Species scores for the constrained axes
write.table (load.rda, "rda_loading.csv", sep = ",")
#SUMMARY OF PROPORTION EXPLAINED BY RDA AND PCA_________________
summary(eigenvals(chiton_rda, model = "constrained")) #Proportion explained by the constrained axes (RDA (canonical axis), or by the env variables)
summary(eigenvals(chiton_rda, model = "unconstrained")) #Proportion explained by the unscontrained axes (PCA, or the residual variance left from the RDA)
#PLOT RDA
plot(chiton_rda)

#EXTRACT DATA TO CUSTOMIZE PLOTS__________________________
#RDA 1 vs RDA 2
#extract the data to be ploted from the RDA object
smry = summary(chiton_rda)
df1 = data.frame(smry$sites[,1:5], pred$pop) #get RDA1 and RDA2 for all populations
df2 = data.frame(smry$species[,1:5]) #get the loadings of RDA1 and RDA2 for each locus
df3 = data.frame(smry$biplot)
rownames(df3)
row.names(df3) = c("lat","lon","mean_prate","mean_pres","mean_sshg" )

write.table(df1, "df1_plotRDA_ind.csv", sep = ',')
write.table(df2, "df2_plotRDA_loci.csv", sep = ',')
write.table(df3, "df3_plotRDA_env.csv", sep = ',')


#CUSTOMIZED RDA PLOTS________________________________
library(ggplot2)

#Set up colours
poplist.int = factor(env$pop, levels = c("RU", "TI", "NC", "EC", "WE", "CP","CR", "DU", "AK", "CU", "ST", "AU", "CA", "AN", "BO", "CH"))
#Create a custom color scale (called cols_pop_scale)
cols_pop = c("#cc0000", "#ff0000", "#990066", "#ff33cc", "#ff9900", "#ffcc00","#003300", "#006600", "#339933", "#33ff33", "#99cc66", "#3399ff", "#33ccff", "#99cccc", "#669999", "#000099")
names(cols_pop) = levels(poplist.int)
cols_pop_scale = scale_colour_manual(name = "poplist.int", values = cols_pop)

#make a base plot with the individuals
rda.plot <- ggplot(df1, aes(x=RDA1, y=RDA2, colour = poplist.int)) +
  geom_point(size = 4) +
  geom_hline(yintercept=0, linetype="dotted") +
  geom_vline(xintercept=0, linetype="dotted") +
  coord_fixed() +
  cols_pop_scale +
  theme_classic ()
rda.plot

#add the SNPs in the RDA space
rda.biplot = rda.plot + geom_point(data = df2, aes (x =RDA1, y = RDA2), colour = "grey", size = 1, shape=3)

#add the arrows indicating the direction of each environmental variable in the RDA space
rda.triplot <- rda.biplot +
  geom_segment(data=df3, aes(x=0, xend=RDA1, y=0, yend=RDA2),
               color="black", arrow=arrow(length=unit(0.01,"npc")), arrow.fill = NULL, lineend = "butt", linejoin = "round") +
  geom_text(data=df3,
            aes(x=RDA1,y=RDA2,label=rownames(df3),
                hjust=0.5*(1-sign(RDA1)),vjust=0.5*(1-sign(RDA2))),
            color="black", size=4)
rda.triplot +
  theme(text = element_text(size=20))

#have the arrows without labels
no_arrow_lables = rda.biplot +
  geom_segment(data=df3, aes(x=0, xend=2*(RDA1), y=0, yend=2*(RDA2)),
               color="black", size = 1, arrow=arrow(length=unit(0.02,"npc")), arrow.fill = NULL, lineend = "butt", linejoin = "round")  +
  theme(text = element_text(size=20))
no_arrow_lables


#DETERMINE OUTLIER LOCI (treating loadings as z-scores and converting them to p-values and q-values)
#Read the RDA loadings in
#load.rda = read.csv('rda_loading.csv', header = T)
head(load.rda)
class(load.rda)
load.rda = as.data.frame(load.rda)
load.rda$locus = rownames(load.rda)
colnames(load.rda)
head(rownames(load.rda))
dim(load.rda)

#group loadings from all axes together
library(tidyr)
mload.rda = gather(load.rda, axis, loading, -locus ) #same as using reshape2 function melt(): mload.rda = melt(load.rda, id = "locus")
tail(mload.rda)

#mean-center and scale loadings to variance = 1
mload.rda$scaled_loading = scale(mload.rda$loading)
head(mload.rda)
var(mload.rda$scaled_loading)
hist(mload.rda$loading)
hist(mload.rda$scaled_loading)

#transform loadings into p-values
mload.rda$pvalues =  (1-pnorm(abs(mload.rda$scaled_loading)))*2
head(mload.rda)
hist(mload.rda$pvalues)

#transform p-values into q-values
library(qvalue)
qvalue.rda = qvalue(mload.rda$pvalues)
summary(qvalue.rda)
hist(qvalue.rda$qvalues)
mload.rda$qvalues = qvalue.rda$qvalues
head(mload.rda)
summary(mload.rda$qvalues)
write.table(mload.rda, 'SNPs_loadings_pvalues_qvalues.csv', sep = ",")

#find outliers based on q-values (cut off = 0.05)
outlier.loci = function (x, y) { # x is a data frame with the q-values in a row named qvalues; y is the qvalue cutoff for outliers
  outs = x$qvalues <= y #find loci with qvalue smaller than cutoff
  outliers = x[outs, ] #subset the dataframe to include only outliers
  if (nrow(outliers) > 0) { #if there are outliers
  outliers_unique = outliers[!duplicated(outliers$locus),] # include only unique outliers in the dataframe
  print(dim(outliers_unique)) #print outliers to the console
  print(head(outliers_unique))
  hist(outliers_unique$qvalues, breaks = 10)#plot the distribution of the q-values of outliers
  write.table (outliers_unique, "outliers_qvalue_rda.csv", sep = ",")}# write a table to the working directory with the defined outliers (not duplicated)

  else { print ("No outlier locus was found with the selected q-value cutoff")}} #if there are no outliers

outlier.loci(mload.rda, 0.05)
#"No outlier locus was found with the selected q-value cutoff"

#ONLY RUN THE CODE BELOW IF OUTLIERS WERE IDENTIFIED ABOVE
##Add the correlation of each SNP with the env variables
outliers_unique = read.csv("outliers_qvalue_rda.csv")
head(outliers_unique)
foo = matrix (nrow = nrow(outliers_unique), ncol = 5)
colnames(foo) = c("A", "B", "C", "D", "E")
pred_1 = pred[,4:8]
for (i in 1:length(outliers_unique$locus)) {
  nam <- outliers_unique[i,2] #get the nameID of the outlier loci
  snp.gen <- gen.imp[,nam] #get its genotypes from the gen.imp dataset (genetic data imputed used in RDA)
  foo[i,] <- apply(pred_1,2,function(x) cor(x,snp.gen)) #apply a correlation function to the genetic and the env data for this loci
}
full = cbind.data.frame(outliers_unique, foo)
head(full)
#see which of the predictors each candidate SNP is most strongly correlated with
for (z in 1:length(full$locus)) {
  bar <- full[z,]
  full[z,12] <- names(which.max(abs(bar[7:11]))) # gives the variable
  full[z,13] <- max(abs(bar[7:11]))              # gives the correlation
}
colnames(full)[12] <- "predictor"
colnames(full)[13] <- "correlation"
head(full)
write.table (full, 'outliers_qvalue_correlation_fullTable.csv', sep = ",")

#THe code below tests the RDA model for significance (can be run when it did not identify outliers as well, but it takes time!)
#TESTS OF SIGNIFICANCE FOR THE RDA MODEL AND MULTICOLINEARITY AMONG ENV VARIABLES USED (run this last, it takes time)
#TEST THE RDA FULL MODEL FOR SIGNIFICANCE USING F-STATISTICS
signif.full = anova.cca(chiton_rda, parallel=getOption("mc.cores")) # default is permutation=999
signif.full
#TEST EACH CONSTRAINED AXIS (RDAs) FOR SIGNIFICANCE AS DONE WITH THE FULL MODEL, USING F-STATISTICS (THIS IS THE ONE THAT TAKES TIME)
signif.axis <- anova.cca(chiton_rda, by="axis", parallel=getOption("mc.cores")) #will take a couple of hours
signif.axis
```

RDA with 100 random variables

In hindsight, I do not think it makes any difference (maybe in the speed of the runs), but I broke the 100 variables into 20 RDA runs with 5 variables each. 20 folders for the output were needed. What is needed:

- the gen.imp.csv file (imputed file created from vcf, same used in the RDA run with real environmental variables)
- the csv file created with all random variables (same random variables as used in Bayescenv)
- the R script below, which was run in the background with nohup in Nesi using:

```
nohup Rscript script.R &
```

The output of the nohup run (nohup.out) will have the results of the RDA run. If the function to identify outliers return any outlier with q-value cut-off 0.05, it will be printed in this outfile.

```
nano nohup.out
```

```r
#script.R to run the RDA
# Load packages
# -------------
library(psych)    # Used to investigate correlations among predictors
library(vegan)    # Used to run RDA
library(adegenet)
library(vcfR)
library(usdm)       # calculates variance inflation factors
library(robust)     # for robust mahalanobis distance calculation
library(qvalue)     # FDR calculation

#Read genetic data (imputed from a previous run of the code)
gen.imp = read.csv("../gen.imp.csv", header = TRUE)
#Read environmental data
env <- read.csv("../all_arbitrary_env.csv")
colnames(env) = as.character(colnames(env))
#mean-center and scale environmental data
senv = scale(env[4:103])
env = data.frame(env[1:3], senv)

#Subset the env file with 100 environmental variables - this can be used for RDA without formula, but I am just keeping because all the code was developed with pred (after the RDA with formula)
pred <- subset(env, select=c(ind, pop, group, X1, X2, X3, X4, X5)) #CHANGE THE LINE FOR EACH OF THE 20 RUNS (run 2 should use the line below, run 3 the line below that, and so on)
#pred <- subset(env, select=c(ind, pop, group, X6, X7, X8, X9, X10))
#pred <- subset(env, select=c(ind, pop, group, X11, X12, X13, X14, X15))
#pred <- subset(env, select=c(ind, pop, group, X16, X17, X18, X19, X20))
#pred <- subset(env, select=c(ind, pop, group, X21, X22, X23, X24, X25))
#pred <- subset(env, select=c(ind, pop, group, X26, X27, X28, X29, X30))
#pred <- subset(env, select=c(ind, pop, group, X31, X32, X33, X34, X35))
#pred <- subset(env, select=c(ind, pop, group, X36, X37, X38, X39, X40))
#pred <- subset(env, select=c(ind, pop, group, X41, X42, X43, X44, X45))
#pred <- subset(env, select=c(ind, pop, group, X46, X47, X48, X49, X50))
#pred <- subset(env, select=c(ind, pop, group, X51, X52, X53, X54, X55))
#pred <- subset(env, select=c(ind, pop, group, X56, X57, X58, X59, X60))
#pred <- subset(env, select=c(ind, pop, group, X61, X62, X63, X64, X65))
#pred <- subset(env, select=c(ind, pop, group, X66, X67, X68, X69, X70))
#pred <- subset(env, select=c(ind, pop, group, X71, X72, X73, X74, X75))
#pred <- subset(env, select=c(ind, pop, group, X76, X77, X78, X79, X80))
#pred <- subset(env, select=c(ind, pop, group, X81, X82, X83, X84, X85))
#pred <- subset(env, select=c(ind, pop, group, X86, X87, X88, X89, X90))
#pred <- subset(env, select=c(ind, pop, group, X91, X92, X93, X94, X95))
#pred <- subset(env, select=c(ind, pop, group, X96, X97, X98, X99, X100))

#RDA with formula (allows calculating anova.cca per axis later on)
chiton_rda = rda(formula = gen.imp ~ X1 + X2 + X3 + X4 + X5, data = env, scale = T) #CHANGE THE LINE FOR EACH OF THE 20 RUNS (run 2 should use the line below, run 3 the line below that, and so on)
#chiton_rda = rda(formula = gen.imp ~ X6 + X7 + X8 + X9 + X10, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X11 + X12 + X13 + X14 + X15, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X16 + X17 + X18 + X19 + X20, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X21 + X22 + X23 + X24 + X25, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X26 + X27 + X28 + X29 + X30, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X31 + X32 + X33 + X34 + X35, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X36 + X37 + X38 + X39 + X40, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X41 + X42 + X43 + X44 + X45, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X46 + X47 + X48 + X49 + X50, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X51 + X52 + X53 + X54 + X55, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X56 + X57 + X58 + X59 + X60, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X61 + X62 + X63 + X64 + X65, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X66 + X67 + X68 + X69 + X70, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X71 + X72 + X73 + X74 + X75, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X76 + X77 + X78 + X79 + X80, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X81 + X82 + X83 + X84 + X85, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X86 + X87 + X88 + X89 + X90, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X91 + X92 + X93 + X94 + X95, data = env, scale = T)
#chiton_rda = rda(formula = gen.imp ~ X96 + X97 + X98 + X99 + X100, data = env, scale = T)

print(chiton_rda)
sum_constrained = print(summary(eigenvals(chiton_rda, model = "constrained"))) #Proportion explained by the constrained axes (RDA (canonical axis), or by the env variables)
sum_unconstrained = print(summary(eigenvals(chiton_rda, model = "unconstrained"))) #Proportion explained by the unscontrained axes (PCA, or the residual variance left from the RDA)
hists = screeplot(chiton_rda) #HISTOGRAM OF THE EIGENVALUES OF THE RDAs
plot(chiton_rda) #PLOT RDA
#make a plot with the loadings for each RDA axis and each SNP
load.rda = scores(chiton_rda, choices=c(1:5), display="species")  # Species scores for the constrained axes
write.table (load.rda, "rda_loading.csv", sep = ",")

#Read the RDA loadings in
head(load.rda)
class(load.rda)
load.rda = as.data.frame(load.rda)
load.rda$locus = rownames(load.rda)
colnames(load.rda)
head(rownames(load.rda))
dim(load.rda)

#group loadings from all axes together
library(tidyr)
mload.rda = gather(load.rda, axis, loading, -locus ) #same as using reshape2 function melt(): mload.rda = melt(load.rda, id = "locus")
tail(mload.rda)

#mean-center and scale loadings to variance = 1
mload.rda$scaled_loading = scale(mload.rda$loading)
head(mload.rda)
var(mload.rda$scaled_loading)
hist(mload.rda$loading)
hist(mload.rda$scaled_loading)

#transform loadings into p-values
mload.rda$pvalues =  (1-pnorm(abs(mload.rda$scaled_loading)))*2
head(mload.rda)
hist(mload.rda$pvalues)

#transform p-values into q-values
library(qvalue)
qvalue.rda = qvalue(mload.rda$pvalues)
summary(qvalue.rda)
hist(qvalue.rda$qvalues)
mload.rda$qvalues = qvalue.rda$qvalues
head(mload.rda)
summary(mload.rda$qvalues)
write.table(mload.rda, 'SNPs_loadings_pvalues_qvalues.csv', sep = ",")

#find outliers based on q-values (cut off = 0.05)
outlier.loci = function (x, y) { # x is a data frame with the q-values in a row named qvalues; y is the qvalue cutoff for outliers
  outs = x$qvalues <= y #find loci with qvalue smaller than cutoff
  outliers = x[outs, ] #subset the dataframe to include only outliers
  outliers_unique = outliers[!duplicated(outliers$locus),] # include only unique outliers in the dataframe
  print(dim(outliers_unique)) #print outliers to the console
  print(head(outliers_unique))
  hist(outliers_unique$qvalues)#plot the distribution of the q-values of outliers
  write.table (outliers_unique, "outliers_qvalue_rda.csv", sep = ",")} # write a table to the working directory with the defined outliers (not duplicated)

outlier.loci(mload.rda, 0.05)

#Add the correlation of each SNP with the env variables
outliers_unique = read.csv("outliers_qvalue_rda.csv")
head(outliers_unique)
foo = matrix (nrow = nrow(outliers_unique), ncol = 5)
colnames(foo) = c("A", "B", "C", "D", "E")
pred_1 = pred[,4:8]
for (i in 1:length(outliers_unique$locus)) {
  nam <- outliers_unique[i,2] #get the nameID of the outlier loci
  snp.gen <- gen.imp[,nam] #get its genotypes from the gen.imp dataset (genetic data imputed used in RDA)
  foo[i,] <- apply(pred_1,2,function(x) cor(x,snp.gen)) #apply a correlation function to the genetic and the env data for this loci
}

full = cbind.data.frame(outliers_unique, foo)
head(full)


#see which of the predictors each candidate SNP is most strongly correlated with
for (z in 1:length(full$locus)) {
  bar <- full[z,]
  full[z,12] <- names(which.max(abs(bar[7:11]))) # gives the variable
  full[z,13] <- max(abs(bar[7:11]))              # gives the correlation
}
colnames(full)[12] <- "predictor"
colnames(full)[13] <- "correlation"
head(full)
write.table (full, 'outliers_qvalue_correlation_fullTable.csv', sep = ",")

#TAKES TIME!!  #TESTS OF SIGNIFICANCE FOR THE RDA MODEL AND MULTICOLINEARITY AMONG ENV VARIABLES USED (run this last, it takes time)
#TEST THE RDA FULL MODEL FOR SIGNIFICANCE USING F-STATISTICS
signif.full = anova.cca(chiton_rda, parallel=getOption("mc.cores")) # default is permutation=999
signif.full
#TEST EACH CONSTRAINED AXIS (RDAs) FOR SIGNIFICANCE AS DONE WITH THE FULL MODEL, USING F-STATISTICS (THIS IS THE ONE THAT TAKES TIME)
#signif.axis <- anova.cca(chiton_rda, by="axis", parallel=getOption("mc.cores")) #will take a couple of hours
#signif.axis
```

For the NZ-wide dataset, the RDA results with Random variables showed two random variables correlated with the same six outliers (output from nohup, run the code below in R)

```r
#Running the new imputed dataset (imputed by clade) with the random environmental variables identified only six outliers
#2, 4, 12, 19, 36, and 44.
#All of these outliers are homozygous ref to N and S (00 in the plot and dataset), and homozygous alt to C (11 in the plot and dataset), for both not-imputed and imputed datasets

gen.imp = read.csv("../gen.imp.csv", header = TRUE)

plot(gen.imp[,2])
plot(gen.imp[,4])
plot(gen.imp[,12])
plot(gen.imp[,19])
plot(gen.imp[,36])
plot(gen.imp[,44])

plot(gen[,2])
plot(gen[,4])
plot(gen[12])
plot(gen[,19])
plot(gen[,36])
plot(gen[,44])
```

Combining q-values of BayeScEnv and RDA with the geometric mean

Code runs in R. It takes the geometric mean of q-values between methods used to identify loci in gene-environment association (BayeScEnv and RDA) to combine the output of those methods instead of taking the intersection of outlier loci between them (but a way to take the intersection of loci is also provided in another R script - "Checking gene-environment associations with real environmental variables and random variables".) What is needed:

- Output from BayeScEnv with the qvalue of all loci (in csv format)
- Output from RDA with the qvalue of all loci (in csv format)
- R packages tidyverse, tidyr

```r
#-----------Real environmental variables-------------
#Read files with RDA q-values and Bayescenv q-values for association with real environmental variables
rda = read.csv("SNPs_loadings_pvalues_qvalues.csv", header = T)
bay = read.csv("summary_qVal_allRealEnv.csv", header = T)

#working and tidying with the RDA data frame
head(rda)
library(tidyverse)
library(dplyr)
#select the columns of interest (locus id (which is CHROM in this case), qvalue and RDA/env) and transform table into wide format
rda = rda %>% select (1, 2, 6)
head(rda)
rda = rda %>% spread (axis, qvalues)
#remove uncecessary info of locus id (CHROM) and re-order table to increasing loci id/number (as to be able to use locus = 1:10987, matching rownumber and also bayescenv)
head(rda$locus)
library(stringr)
Cid = rda$locus
Cid = str_split_fixed(Cid, "_", n = 2)
Cid = Cid [,1]
Cid = str_split_fixed(Cid, "X", n = 2)
Cid = Cid[,2]
Cid
Cid = as.numeric(Cid)
Cid
#Add Cid (Chrom ID without X and position) as a column in the data frame rda
rda$CHROM = Cid
head(rda$CHROM)
#re-order the data frame so CHROM is in ascending order
rda = rda[order(rda$CHROM),]
head(rda$CHROM)
head(rda)
#re-write the 'locus' column with loci ID matching rownumber and bayescenv
rda$locus = 1:10987 #change this range to the number of loci in the dataset
head(rda)

#working and tidying the Bayescenv dataframe (we are only interested in g)
head(bay)
#select columns with q-values for g in each real env variable
bay = bay %>% select (1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30)
#We want to combine the q-values of Bayescenv and RDA for each env variable, but RDA does not include variables correlated with latitude (only latitude)
#So we will combine the q-value of bayescenv-latitude with the q-value of all the other variables correlated with latitude
#When interpreting the results, this will mean that gene-env correlations that are true for latitude, could be true for any of the latitude-correlated envs
#To combine the q-values of these, we will use the geometric mean ( the nth root of the product of n numbers)
#The variables used in RDA and Bayescenv were Lat, Long, Prate, Pres, and Sshg. All others are correlated with latitude

broad_lat = (bay$latG*bay$mslpG*bay$nswrsG*bay$osaltG*bay$otempG*bay$saltG*bay$sstG*bay$tcdcG*bay$uwndG*bay$vwndG*bay$airG)^(1/11)

#now remake the bayscenv (bay) dataframe to contain the latitude "broad sense" just calculated, and the other env variables also included in the RDA
bay$latG = broad_lat
head(bay)
bay = bay %>% select (1, 2, 3, 8, 9, 11)
head(bay)

#Combine the q-values of rda and bayescenv (don`t forget that lat is now "broad sense")
#create a function to do the geometric mean as above
gmean = function(x, y) {
  sqrt(x*y)
}
#Select the RDA columns to be used (leave loci ID out)
colnames(rda)
dim(rda)
rda = rda %>% select (2:6)
#Select the Bayescenv columns to be used (leave loci ID out)
colnames (bay)
dim(bay)
bay = bay %>% select (2:6)

#Create a variable from 1:5 to iterate from (the 5 columns, or env variables, of each dataset)
envs = 1:5
#create a list to store the results of each env variable
result_list = list()
#make a for loop to calculate the geometric mean
for (i in envs) {
  n = gmean (rda[,i], bay[,i])
  result_list [[i]] = n
}
summary(result_list) #check that the list stored the results
summary(result_list[[1]]) #check that there are values of geometric mean for env variable 1
#convert the list with all results to a data frame, containing a locus per row and an env variable per column
df = data.frame(matrix(unlist(result_list), nrow=10987, byrow=F)) #change the nrow to the number of loci in the dataset
df$locus = 1:10987 #add the column with locus id (which is just the row number because both datasets were ordered) -- change the nrow to the number of loci in the dataset
head(df)
summary(df)
#rename the cols with env variables (they were in the same order in the initial datasets: Lat, Long, Prate, Pres, and Sshg)
colnames(df) = c('lat', 'long', 'prate', 'press', 'sshg', 'loci')
head(df)
#save the table
write.table (df, "combined_qValues_real_RDA_Bayescenv.csv", sep = ",", row.names = F)

#----------------Random Variables----------------------------
#Read files with RDA q-values and Bayescenv q-values for the 100 random env variables
rda = read.csv("all_random_qValues2021.csv")
bay = read.csv("summary2021_qvaluesRandomEnv_Bayescenv.csv")
head(rda)
head (bay)
tail (rda)
tail(bay)
#calculate the geometric mean of q-values for random env 1 (combined q-value). The geometric mean is the the nth root of the product of n numbers
gmean1 = sqrt(rda$X1*bay$X1)
gmean1
#create a function to do the geometric mean as above
gmean = function(x, y) {
  sqrt(x*y)
}
#test the function and compare with the result of gmean1
gmean(rda$X1, bay$X1)
#now automate to calculate the geometric mean between RDA and Bayescenv q-values of each random env
library(tidyverse)
#Select the RDA dataframe to be used (leave loci ID out)
colnames(rda)
dim(rda)
rda = rda %>% select (2:101)
#Select the Bayescenv dataframe to be used (leave loci ID out)
colnames (bay)
dim(bay)
bay = bay %>% select (2:101)
#Create a variable from 1:100 to iterate from (the 100 columns, or random env variables, of each dataset)
envs = 1:100
#create a list to store the results of each random variable
result_list = list()
#make a for loop to calculate the geometric mean
for (i in envs) {
  n = gmean (rda[,i], bay[,i])
  result_list [[i]] = n
}
summary(result_list) #check that the list stored the results
summary(result_list[[1]]) #check that there are values of geometric mean for random variable 1
#convert the list with all results to a data frame, containing a locus per row and a random env variable per column
df = data.frame(matrix(unlist(result_list), nrow=10987, byrow=F)) #change nrow to the number of loci in the dataset
df$locus = 1:10987 #add the column with locus id (which is just the row number because both datasets were ordered)   -- change this range to the number of loci in the dataset
head(df)
#save the table
write.table (df, "combined_qValues_random_RDA_Bayescenv.csv", sep = ",", row.names = F)

#How many 5% random associations, combined qvalue?
df_bin = df[,1:100]
df_bin = ifelse(df_bin <= 0.05, 1, 0)
df_bin = as.data.frame(df_bin)
head(df_bin)
df_bin$total = apply(df_bin, 1, sum)
head(df_bin)
length(which(df_bin$total>=5))#0 RDA-Bayescenv combined 5% associations
summary(df_bin)

###Looking into RDA and Bayescenv results separately
rda_bin = ifelse (rda <= 0.05, 1, 0)
head(rda_bin)
rda_bin = as.data.frame(rda_bin)
rda_bin$total = apply (rda_bin, 1, sum)
head(rda_bin)
length(which(rda_bin$total>=5)) #0 RDA 5% random associations
summary(rda_bin)

bay_bin = ifelse(bay <= 0.05, 1 , 0)
bay_bin = as.data.frame(bay_bin)
head(bay_bin)
bay_bin$total =apply(bay_bin, 1, sum)
head(bay_bin)
length(which(bay_bin$total>= 5)) #722 Bayescenv 5% random associations
summary(bay_bin)


#Checking back with the runs with real environmental variables to see if any association was found after combining q-values
real = read.csv("../real/combined_qValues_real_RDA_Bayescenv.csv")
head(real)
real = real[,1:5]
real_bin = ifelse(real <= 0.05, 1, 0)
real_bin = as.data.frame(real_bin)
head(real_bin)
real_bin$total = apply(real_bin, 1, sum)
length(which(real_bin$total>=1))
summary(real_bin)
```

Checking gene-environment associations with real environmental variables and random variables (loci associated with at least 5% of the random variables will no longer be considered true associations)

Waht is needed

- The file with the combined q-value of BayeScEnv and RDA for the real environmental variables (code for combining q-values provided above)
- The file with the combined q-values of BayeScEnv and RDA for the random variables (code for combining q-values provided above)
- A csv file with ID info from the vcf file (loci_info_vcfStyle.csv), for matching the locus ID of the loci associated with real environmental variables with the locus ID of the vcf file

In [ ]:

```
#Southern (this example is provided with the Southern dataset, since the NZ-wide did not have any associations in the RDA)
library(tidyr)

#Read file with combined q-values of Bayescenv and RDA for real and random variables
real = read.csv("real/combined_qValues_real_RDA_Bayescenv.csv")
random = read.csv("arbitrary/Random_combinedGIF_QvalsRDA-Bayescenv.csv")
head(real)
head(random)
summary(real)
summary(random)

#Define which loci are outliers (qvalue < 0.05) in real and random datasets using T and F
#Real envs
envs = 5 #number of environmental variables
loci = nrow(real) #number of loci

real_out = matrix(ncol=envs, nrow=loci) #create a matrix to store the results of a for loop

for (col in 1:envs) { #to each column (each environmental variable)
  real_out[,col] = ifelse (real[,col] < 0.05, TRUE, FALSE) #(call qvalues < 0.05 of TRUE and larger of FALSE)
}

head(real_out)
class(real_out)
real_out = as.data.frame(real_out) #Transform the matrix of results into a dataframe
real_out$loci = 1:loci #add the locus ID into the results dataframe
colnames(real_out) = colnames(real) #rename the columns using the env variable names
summary(real_out) # There are 111 outliers in LAT*, 1 in LONG, 5 in PRATE, 135 in PRESS and 23 in SSHG.


#Find the total number of outliers and which ones they are for vcf_style table
total_real = ifelse(real_out$lat== TRUE, TRUE,
       ifelse(real_out$long ==TRUE , TRUE,
              ifelse(real_out$prate == TRUE, TRUE,
                     ifelse(real_out$press== TRUE, TRUE,
                            ifelse(real_out$sshg==TRUE, TRUE, FALSE)))))
real_out$total_real = total_real
head(real_out)
summary(real_out$total)
write.table (real_out, "real/combinedQvals_vcfStyleSummary.csv", sep = ",", row.names = FALSE)

#Random variables
envs = 100 #number of random variables
loci = nrow(random) #number of loci

random_out = matrix(ncol=envs, nrow=loci) #create a matrix to store the results of a for loop

for (col in 1:envs) { #to each column (each environmental variable)
  random_out[,col] = ifelse (random[,col] < 0.05, TRUE, FALSE) #(call qvalues < 0.05 of TRUE and larger of FALSE)
}

head(random_out)
class(random_out)
random_out = as.data.frame(random_out) #Transform the matrix of results into a dataframe
random_out$loci = 1:loci #add the locus ID into the results dataframe
summary(random_out) #There are some associations between random env variables and some of the loci when the q-values of Bayescenv and RDA are combined
class(random_out)
write.table (random_out, "arbitrary/combinedQvals_vcfStyleSummary.csv", sep = ",", row.names = FALSE)

#Find the total number of random outliers and which ones they are for vcf_style table (associated with only one random variable)
random_out = ifelse(random_out[,1:100]==TRUE,1,0)
total_random = apply(random_out,1,sum)
class(total_random)
length(total_random)
class(random_out)
as.data.frame(random_out)
total_random = (ifelse(total_random==0, FALSE, TRUE))
class(total_random)
class(random_out)
random_out = as.data.frame(random_out)
random_out$total_random = total_random
dim(random_out)
length(total_random)
head(random_out)
summary(random_out)#2088 random associations (with at least one loci, not the 5%)

#High confidence outliers (associated only with real envs and less than 5 random envs)
# Transform random dataset from T anf F to 0 and 1
#random
random_out = ifelse(random_out[,1:100]==TRUE,1,0)
sum(random_out[,1:100]) #1193 outliers in total (regardless of random variable - that means one locus can be associated with more than one environmental variable)
random_out = as.data.frame(random_out)
head(random_out)
#sum to find out which loci have sum > 5 (outliers in at least 5 out of the 100 runs with random variables)
random_out$sum = apply(random_out, 1, sum)
summary(random_out)
random_out$loci = 1:nrow(random_out)#add loci ID
#define which loci were associated with at least 5 random variables (TRUE if sum column is equal to or larger than 5)
random_out$out = ifelse(random_out$sum >= 5, TRUE, FALSE) ###THIS IS THE LINE TO COPY TO THE VCF STYLE SUMMARY!!!
head(random_out)
summary(random_out) #260 of the loci were associated with at least five random environmental variables (the 5%)
write.table (random_out, "arbitrary/combinedQvals_vcfStyleSummary.csv", sep = ",", row.names = FALSE)


#Match which of the real variables is high confidence (not associated with five or more random env variables)
head(real_out)
class(real_out)
head(random_out)
class(random_out)
library(tidyverse)
'%ni%' <- Negate('%in%')
random_out_loci = random_out %>% filter( out == TRUE) %>% select (loci)
random_out_loci = random_out_loci[,1] #random_out_loci is now a vector including loci id of loci associated with at least 5 random env variables

#Latitude* (broad sense)
lat = real_out %>% filter (lat == TRUE) %>% select (loci)
lat = lat[,1]#lat is now a vector containing loci ID of loci associated with latitude
high_confidence_lat = lat %ni% random_out_loci #which loci associated with lat are not associated with random_out_loci?
length(which(high_confidence_lat == TRUE)) #73
#get a list of those loci IDs
df_lat = data.frame(lat, high_confidence_lat)
high_confidence_lat = df_lat %>% filter (high_confidence_lat==TRUE) %>% select (lat) #the 81 loci are only associated with latitude broad sense (no duplication with other real env variables!)
high_confidence_lat = as.vector(high_confidence_lat$lat)

#Longitude
long = real_out %>% filter (long == TRUE) %>% select (loci)
long = long[,1]#long is now a vector containing loci ID of loci associated with latitude
high_confidence_long = long %ni% random_out_loci #which loci associated with long are not associated with random_out_loci?
length(which(high_confidence_long == TRUE)) #0

#prate
prate = real_out %>% filter (prate == TRUE) %>% select (loci)
prate = prate[,1]#prate is now a vector containing loci ID of loci associated with latitude
high_confidence_prate = prate %ni% random_out_loci #which loci associated with prate are not associated with random_out_loci?
length(which(high_confidence_prate == TRUE)) #4
#get a list of those loci IDs
df_prate = data.frame(prate, high_confidence_prate)
high_confidence_prate = df_prate %>% filter (high_confidence_prate==TRUE) %>% select (prate) #Locus ID 3228
high_confidence_prate = as.vector(high_confidence_prate$prate)

#press
press = real_out %>% filter (press == TRUE) %>% select (loci)
press = press[,1]#press is now a vector containing loci ID of loci associated with latitude
high_confidence_press = press %ni% random_out_loci #which loci associated with press are not associated with random_out_loci?
length(which(high_confidence_press == TRUE)) #102
#get a list of those loci IDs
df_press = data.frame(press, high_confidence_press)
high_confidence_press = df_press %>% filter (high_confidence_press==TRUE) %>% select (press) #Locus ID 3228
high_confidence_press = as.vector(high_confidence_press$press)


#sshg
sshg = real_out %>% filter (sshg == TRUE) %>% select (loci)
sshg = sshg[,1]#sshg is now a vector containing loci ID of loci associated with latitude
high_confidence_sshg = sshg %ni% random_out_loci #which loci associated with sshg are not associated with random_out_loci?
length(which(high_confidence_sshg == TRUE)) #21
#get a list of those loci IDs
df_sshg = data.frame(sshg, high_confidence_sshg)
high_confidence_sshg = df_sshg %>% filter (high_confidence_sshg==TRUE) %>% select (sshg) #Locus ID 5134 - same as Long
high_confidence_sshg = as.vector(high_confidence_sshg$sshg)

#make a vector of T and F to add to the vcf_style file
lat_high = ifelse(real_out$loci %in% high_confidence_lat, TRUE, FALSE)
summary(lat_high)
long_high = ifelse(real_out$loci %in% high_confidence_long, TRUE, FALSE)
prate_high = ifelse(real_out$loci %in% high_confidence_prate, TRUE, FALSE)
press_high = ifelse(real_out$loci %in% high_confidence_press, TRUE, FALSE)
sshg_high = ifelse(real_out$loci %in% high_confidence_sshg, TRUE, FALSE)
#combine the vectors above in a table
highConfidenceSummaryTable = data.frame(lat_high,long_high,prate_high,press_high,sshg_high, real_out$loci)
head(highConfidenceSummaryTable)
colnames(highConfidenceSummaryTable) = c('lat_high','long_high','prate_high','press_high','sshg_high', 'loci')
#make a vector with the combined result (total high confidence associations)
total_high = ifelse(highConfidenceSummaryTable$lat_high== TRUE, TRUE,
                    ifelse(highConfidenceSummaryTable$long_high ==TRUE , TRUE,
                           ifelse(highConfidenceSummaryTable$prate_high == TRUE, TRUE,
                                  ifelse(highConfidenceSummaryTable$press_high== TRUE, TRUE,
                                         ifelse(highConfidenceSummaryTable$sshg_high==TRUE, TRUE, FALSE
)))))
highConfidenceSummaryTable$total_high = total_high
head(highConfidenceSummaryTable)
summary(highConfidenceSummaryTable)
write.table(highConfidenceSummaryTable, "highConf_geneEnvAssociations.csv", sep = ",", row.names = FALSE)

#Which are the high-confidence loci id?
lat_id = highConfidenceSummaryTable %>% filter (lat_high==TRUE) %>% select (loci)
lat_id = lat_id$loci
lat_id
#
prate_id = highConfidenceSummaryTable %>% filter (prate_high==TRUE) %>% select (loci)
prate_id = prate_id$loci
prate_id
#
press_id = highConfidenceSummaryTable %>% filter (press_high==TRUE) %>% select (loci)
press_id = press_id$loci
press_id
#
sshg_id = highConfidenceSummaryTable %>% filter (sshg_high==TRUE) %>% select (loci)
sshg_id = sshg_id$loci
sshg_id

#Check for duplicates -> are the high-confidence loci associated with more than one environmental variable?
head(highConfidenceSummaryTable)
#lat_high and prate_high
highConfidenceSummaryTable %>% filter (lat_high == TRUE & prate_high == TRUE) %>% select (loci) #0
#lat_high and press_high
highConfidenceSummaryTable %>% filter (lat_high == TRUE & press_high == TRUE) %>% select (loci) #0
#lat_high and sshg_high
highConfidenceSummaryTable %>% filter (lat_high == TRUE & sshg_high == TRUE) %>% select (loci) #0
#prate_high and press_high
highConfidenceSummaryTable %>% filter (prate_high == TRUE & press_high == TRUE) %>% select (loci) #0
#prate_high and sshg_high
highConfidenceSummaryTable %>% filter (prate_high == TRUE & sshg_high == TRUE) %>% select (loci) #0
#press_high and sshg_high
highConfidenceSummaryTable %>% filter (press_high == TRUE & sshg_high == TRUE) %>% select (loci) #0


#Find CHROM id of high-confidence associations
chromid = read.csv("../../CHROM_LOCI_ID_Sonly.csv", header = T)
head(chromid)
lat_chrom = chromid[lat_id, 1]
prate_chrom = chromid[prate_id, 1]
press_chrom = chromid[press_id, 1]
sshg_chrom = chromid[sshg_id, 1]


#Make a table with all high-confidence loci IDs to use for grepping the fasta sequence and blast it against Acanthopleura granulata genome
hicdf = cbind(highConfidenceSummaryTable, chromid)
hic = hicdf %>% filter (total_high == TRUE) %>% select (CHROM)
head(hic)
chrom = hic$CHROM

for (i in 1:200) { #change 200 by nrow(chrom), that is, the number of CHROM ids to search the fasta sequence for
  text = paste ( "grep -A 1 -w CLocus_", chrom[i], " populations.loci.fa > loci", chrom[i], ".fasta" , sep = "")
  print(text)
} #copy and paste output in notepad, than save as .sh to run in bash
```

Extract fasta sequeces from loci associated with the environment (high-confidence) and Blast against the Acanthopleura granulata genome (custom Blast database)

What is needed

- the .sh file from the R code above, which contain grep commands to find the CHROM id and fasta sequence using bash
- the fasta output from the populations module on Stacks (it is unfiltered, but it is fine, since it will be used to search for specific loci IDs)
- Acantopleura granulata assembly and transcriptoms (downloaded fasta from Dryad https://doi.org/10.5061/dryad.wstqjq2k9)
- Blast installed locally (or load the module in NeSI)

In [ ]:

```
cd Stacks/ #Go to the Stacks directory that contains the fasta output from the populations module
srun sequencesToGrep.sh #Run the file with the fasta sequences to find, which was generated with the R script above

cd Agranulata/ #Go to the directory with A. granulata .fasta assembly and transcriptomes
cp ../Sonly_outlocus_RDA-Bayescenv_HighConfidence.fasta #copy the .fasta file with the target loci sequences (output from previous step)
#load Blast module
Module spider/load Blast
#Run Blast (can also be done in a slurm script) with A. granulata scaffolds (example below) and transcriptomes (blastp instead of blastn)
blastn -query Sonly_outlocus_RDA-Bayescenv_HighConfidence.fasta -subject Acanthopleura_granulata-scaffolds.fas -outfmt '7 qseqid sseqid length qlen slen qstart qend sstart send evaule' -out SonlyHighConf_AgranScaffold_BlastResult.txt -export_search_strategy SonlyHighConf_AgranScaffold_BlastSearchStrategy.txt
#check the results of Blasting against A. granulata sequences - did it hit something?
less SonlyHighConf_AgranScaffold_BlastResult.txt
#Get A. granulata flanking sequeences from the hits and blast in the NCBI website. If it hits something, look for the GO term and ProtDB
```

Demographic Analyses

These analyses were done comparing datasets with all loci, only outliers and only neutral. The dataset with all loci can be split into the other two datasets using VCFtools

1. Make a list of loci to be kept in the outliers (all with combined q-value of BayeScan and PCAdapt < 0.05) and the neutral datasets (all with combined q-valye of BayeScan and PCAdapt > 0.05). The .csv file with vcf ID and TRUE/FALSE for each method and combination will be helpful for that, and maybe the outliers_combined_qvals.csv (from combining q-values of BayeScan and PCAdapt) will be helpful too.
2. Copy the VCF file with all loci in a directory containing the lists of loci to be kept in each subset (the ID column in the VCF must be present, I used line numbers and made it match the IDs I was using to identify outliers. Another option is to copy the CHROM ID under this column in the vcf and use the CHROM ID to identify outliers)
3. Load VCFtools module and run the VCFtools fork below

In [ ]:

```
module load VCFtools/0.1.15-GCC-9.2.0-Perl-5.30.1
#NZ-wide
vcftools --vcf MOD_input.vcf --snps outliers.txt --recode --recode-INFO-all --out outliers.out
vcftools --vcf MOD_input.vcf --snps neutral.txt --recode --recode-INFO-all --out neutral.out
#Southern
vcftools --vcf SMOD_input.vcf --snps Soutliers.txt --recode --recode-INFO-all --out Soutliers.out
vcftools --vcf SMOD_input.vcf --snps Sneutral.txt --recode --recode-INFO-all --out Sneutral.out
#NI
vcftools --vcf NIMOD_input.vcf --snps NIoutliers.txt --recode --recode-INFO-all --out NIoutliers.out
vcftools --vcf NIMOD_input.vcf --snps NIneutral.txt --recode --recode-INFO-all --out NIneutral.out
#SI
vcftools --vcf SIMOD_input.vcf --snps SIoutliers.txt --recode --recode-INFO-all --out SIoutliers.out
vcftools --vcf SIMOD_input.vcf --snps SIneutral.txt --recode --recode-INFO-all --out SIneutral.out
```

PCA and allele frequencies

Using LEA in R to calculate PCA and convert vcf to a lfmm file format used to calculate allele frequencies with a custom script (thanks Pierre!)

In [ ]:

```
#Compute allele frequency by pop using lfmm file format, but not imputed

#Load libraries
library(tidyverse)
library(tidyr)
library(naniar)
library(ggplot2)
library(viridis)
library(LEA)

####-------ALLELE FREQUENCIES---------------
#Define function (Pierre`s)
compute_freq <- function(mat) {
  +     colMeans(mat, na.rm = TRUE) / 2}

vcf2lfmm("../conserved_MOD.vcf")

#- number of detected individuals:      167
#- number of detected loci:             10987
#The lfmm format has one row for each individual. Each row contains one value at each loci (separated by spaces or tabulations)
#corresponding to the number of alleles. The number of alleles corresponds to the number of reference alleles or the number of
#derived alleles. Missing genotypes are encoded by the value -9 or the value 9.

#Get allele frequencies by population
list.files()
data = read.lfmm("conserved_MOD.lfmm")
dim(data)
data[data==9] <- NA
pops = list(AK = 1:9, AN = 10:16, AU = 17:26, BO = 27:36, CA = 37:46, CH = 47:55, CP = 56:65, CR = 66:71, CU = 72:76, DU = 77:96, EC = 97:105, NC = 106:115, RU = 116:129, ST = 130:137, TI = 138:149, WE = 150:167)

freq <- sapply(pops, function(indices) {
  +   +     compute_freq(data[indices, ])})

head(freq)
summary(freq)
write.table(freq, "alleleFreq_byPop.csv", sep = ",")

#Checking missing data
length(print(which(is.na(freq)==TRUE))) #346
hist(freq)
dim(freq)
class(freq)
head(freq)
df = as.data.frame(freq)
hist(df$AK)
#visualizing missing data (naniar package)
which(is.na(df$AK))
for (i in df[,1:16]) {missing = which(is.na(i))
print(missing)}
vis_miss(df) #populations on X axis
df = data.frame('RU' = df$RU, 'TI' = df$TI, 'NC' = df$NC, 'EC' = df$EC, 'WE' = df$WE, 'CP' = df$CP, 'CR' = df$CR, 'DU' = df$DU,'AK' = df$AK, 'CU' =  df$CU, 'ST' = df$ST, 'AU' = df$AU, 'CA' = df$CA, 'AN' = df$AN, 'BO' =  df$BO,'CH' = df$CH)
vis_miss(df)
#distribution of allele frequencies for each population
for (i in df[,1:16]) {
  print(ggplot(df, aes(x=i)) + geom_histogram(aes(y=..density..), binwidth=.05, colour="black", fill="white") +
          geom_density(alpha=.8, fill="grey") +
          xlab ("allele frequency") +
          theme_classic())
}

#prepare freq dataset to plot allele frequencies
graph_data = gather(data = freq, key = "pop", value = "allele_frequency")
locusID = read.table("locusID.txt", header = F) #file used in VCFtools, with one locud ID per row and no header, to specify the locus ID in the mfreq dataframe, later used for plotting. Can also be line number in the dataset with all loci (rep (1:nrow(freq),16))
graph_data$locus = as.factor(rep(locusID$V1, 16)) #the graph_data dataframe is in long format, with all loci ID for each 16 populations
head(graph_data)
#graph_data$group = factor(graph_data$group, levels = c("North", "Central", "Central"))
graph_data$pop = factor(graph_data$pop, levels = c("RU", "TI", "NC", "EC", "WE", "CP", "CR", "DU", "AK", "CU", "ST", "AU", "CA", "AN", "BO", "CH"))
str(graph_data)
#plot allele frequencies
ggplot(graph_data, aes(x=pop, y = fct_reorder2(graph_data$locus, graph_data$allele_frequency, graph_data$pop))) +
  geom_tile(aes(fill=allele_frequency)) +
  labs(x = "Populations",
       y = "All loci - Conserved dataset",
       title = "Allele frequency of all loci per group, conserved dataset") +
  scale_color_viridis(discrete = FALSE, option = "D") +
  scale_fill_viridis(discrete = FALSE, option = "D") +
  theme(axis.text.y=element_blank()) +
  theme(panel.grid=element_blank()) +
  theme(axis.text.y=element_blank()) +
  theme(strip.background=element_blank()) +
  theme(strip.text=element_text(size=12)) +
  theme(legend.key=element_rect(colour="grey25")) +
  guides(fill=guide_legend(override.aes=list(colour=NULL)))


####-------PCA---------------
#read file with information (ID, pop, groups)
info = read.table("id_pop_info.tsv", header = TRUE, sep = "\t") #population map
pc = pca ("input.lfmm", scale = TRUE)
# Summarize the PCA analysis.
show(pc)
pca_summary = summary(pc)
write.table(pca_summary, file = "pca_summary.tsv", sep = "\t")
# Plot eigenvalues.
eigen_plot = plot(pc, lwd=5, col="black", cex=.7, xlab=("Factors"),ylab="Eigenvalues")
coord <- pc$"projections" # subset the PC projections (scores) from the PCA project
ncol= ncol(coord) #check how many columns in coord (which contains the pc projections)
colnames(coord) <- paste0(rep("PC",ncol), seq(1:ncol))# rename the column names in coord to PC#
graph_data <- as.data.frame(cbind(info$pop, coord))

#Customized colours for plots with corrrect population colours
Populations = factor(info$pop, levels = c("RU", "TI", "NC", "EC", "WE", "CP", "CR", "DU", "AK", "CU", "ST", "AU", "CA", "AN", "BO", "CH"))
#Create a custom color scale (called cols_pop_scale)
cols_pop = c( "#cc0000","#ff0000",      "#990066",      "#ff33cc",      "#ff9900",      "#ffcc00", "#003300",      "#006600",      "#339933",      "#33ff33",      "#99cc66",      "#3399ff",      "#33ccff",      "#99cccc",       "#669999",      "#000099")
names(cols_pop) = levels(Populations)
cols_pop_scale = scale_colour_manual(name = "Populations", values = cols_pop)
fills_pop_scale = scale_fill_manual(name = "Populations", values = cols_pop)

#Plotting PCA
ggplot(data = graph_data) + geom_point(aes(x = PC1, y = PC2, colour = info$pop), size = 6) +
    labs(title = "PC1 versus PC2 for SNPs") +
    cols_pop_scale +
    theme_classic()
```

Heterozygosity per locus and basic statistics (Fst, Ho, Hs and Fis overall)

Using Hiesfstat in R to calculate basic statistics and Observed Heterozygosity per locus and population

In [ ]:

```
# Fst and heterozygosity
#Load Packages
library(vcfR)
library(poppr)
library(ape)
library(RColorBrewer)
library(igraph)
library(ggplot2)
library(reshape2)
library(phangorn)
library(hierfstat)
library(fsthet)
library(graph4lg)

#Read vcf file with vcfR
dataset = 'NZ-wide_all_loci'
vcf = read.vcfR("../files_vcf-env-popmap/revision_man3/NZ-wide_conserved.vcf")
vcf
#Read table with population information
pop.data = read.table("../files_vcf-env-popmap/revision_man3/NZ_wide.popmap.txt", header = FALSE, sep = "\t")

# check that all the samples in the VCF and info file are included
all(colnames(vcf@gt)[-1] == pop.data$ind)

#convert vcf into geneind data
gi_vcf = vcfR2genind(vcf)

# specify the ploidy of the organism in order to calculate some population genetic metrics
ploidy(gi_vcf) = 2
#add population information to the genind object
other(gi_vcf)$pop = pop.data$V2
strata(gi_vcf) = data.frame(pop = other(gi_vcf)$pop)
setPop(gi_vcf) = ~pop
gi_vcf

#calculate basic statistics for the dataset (estimates individual counts, allelic frequencies, observed heterozygosity and genetic diversitites per locus and per population. Also estimates mean observed heterozygosities, mean gene diversities withinb population Hs, Gene diversities oveall Ht and corrected Htp, and Dst, Dstp. Finally, estimates Fst and Fstp, as well as Fis following Nei (1987) per locus and oveall loci.
bs = basic.stats(gi_vcf, diploid = TRUE)
class(bs)
bs$n.ind.samp #how many individuals have each locus in each population
bs$pop.freq #allele frequency of each locus within each population
bs$Ho #observed heterozygosity per locus per populatiom
bs$Hs #gene diversity (Hs) whithin populations for each locus
bs$Fis #Fis (Nei 1987) per locus within each population (have many NaN)
bs$perloc
bs$overall #overall estimates of observed heterozygosity (Ho), gene diversity (Hs), Ht, Dst, Htp, Dsto, Fst, Fstp, Fis, Dest

#write table with overall basic statistics
overall = as.data.frame(bs$overall)
write.table(overall, paste(dataset, "basic_stats.tsv", sep = "_"), sep = "\t", row.names = FALSE)

#make a table with expected heterozygosity
hist(bs$Ho)
class(bs$Ho)
dat_ho = as.data.frame(bs$Ho)


hist(x$pvalues, xlab = "p-values", main = NULL, breaks = 50, col = "orange")
poplist.int = factor(info$pop, levels = c("RU", "TI", "NC", "EC", "WE", "CP", "CR", "DU", "AK", "CU", "ST", "AU", "CA", "AN", "BO", "CH"))
#Create a custom color scale (called cols_pop_scale)
cols_pop = c("#cc0000", "#ff0000", "#990066", "#ff33cc",
             "#ff9900", "#ffcc00",
             "#003300", "#006600", "#339933", "#33ff33", "#99cc66",
             "#3399ff", "#33ccff", "#99cccc", "#669999", "#000099")

#par(mfrow = c(4,4))
hist(dat_ho$RU, xlab = "Ho", ylab = "", main = "RU", breaks = 10, col = "#cc0000" )#, cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$TI, xlab = "Ho", ylab = "", main = "TI", breaks = 10, col = "#ff0000") #,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$NC, xlab = "Ho", ylab = "", main = "NC", breaks = 10, col = "#990066")# ,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$EC, xlab = "Ho", ylab = "", main = "EC", breaks = 10, col = "#ff33cc")# ,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)

hist(dat_ho$WE, xlab = "Ho", ylab = "", main = "WE", breaks = 10, col = "#ff9900")# , cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$CP, xlab = "Ho", ylab = "", main = "CP", breaks = 10, col = "#ffcc00")# ,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)

hist(dat_ho$CR, xlab = "Ho", ylab = "", main = "CR", breaks = 10, col = "#003300")# ,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$DU, xlab = "Ho", ylab = "", main = "DU", breaks = 10, col = "#006600")#, cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$AK, xlab = "Ho", ylab = "", main = "AK", breaks = 10, col = "#339933")#,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$CU, xlab = "Ho", ylab = "", main = "CU", breaks = 10, col = "#33ff33")#,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$ST, xlab = "Ho", ylab = "", main = "ST", breaks = 10, col = "#99cc66")#,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)

hist(dat_ho$AU, xlab = "Ho", ylab = "", main = "AU", breaks = 10, col = "#3399ff")#,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$CA, xlab = "Ho", ylab = "", main = "CA", breaks = 10, col = "#33ccff")#,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$AN, xlab = "Ho", ylab = "", main = "AN", breaks = 10, col = "#99cccc")#,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$BO, xlab = "Ho", ylab = "", main = "BO", breaks = 10, col = "#669999")#,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)
hist(dat_ho$CH, xlab = "Ho", ylab = "", main = "CH", breaks = 10, col = "#000099")#,  cex.lab=2, cex.axis=2, cex.main=2, cex.sub=2)


# Check the Fis of specific loci (outliers and GEA)
#fis of all loci #too many NAN`s, comming from using 0 heterozygosity to calculate it (1-(Ho/He)= fis)
fis = bs$Fis #Too many NAN
dat_ho
dat_hs = bs$Hs
my_fis = 1-(dat_ho/dat_hs) #too many NAN
summary(fis)
summary(my_fis)
#Check the Ho, He and Fis of specific loci
write.table(dat_ho, "NZ-wide_ho-per-locus.tsv", sep = "\t")
write.table(dat_hs, "NZ-wide_hs-per-locus.tsv", sep = "\t")
write.table(fis, "NZ-wide_fis-per-locus.tsv", sep = "\t")
```

Fst per locus with vcf tools

What is needed:

- A .txt file with individual IDs (matching the vcf file) for each clade or population (one ID per row, no population definition in the file, use one file per population)
- The vcf file containing all loci and all populations (in this case NZ-wide) #The result will give the overall Fst between clades in the log file, and the fst per clade in the .fst file names in --out. This file can then be used in R to plot a histogram (distribution) of fst per loci clade-pairwise

In [ ]:

```
module load VCFtools/0.1.15-GCC-9.2.0-Perl-5.30.1

vcftools --vcf NZ-wide_conserved.vcf --weir-fst-pop N_clade.txt --weir-fst-pop C_clade.txt --out N-C_fst.out
vcftools --vcf NZ-wide_conserved.vcf --weir-fst-pop N_clade.txt --weir-fst-pop S_clade.txt --out N-S_fst.out
vcftools --vcf NZ-wide_conserved.vcf --weir-fst-pop C_clade.txt --weir-fst-pop S_clade.txt --out C-S_fst.out
```

Allele (or site) Frequency Spectrum

Using vcf2sfs in R to calculate allele frequency spectrum (https://github.com/shenglin-liu/vcf2sfs/blob/master/vcf2sfs.r)

In [ ]:
```
#https://github.com/shenglin-liu/vcf2sfs/blob/master/vcf2sfs.r

# Functions for creating site frequency spectra (SFS) from a VCF file.
# Functions for manipulating and plotting SFS.
# By Shenglin Liu, Feb 12, 2020.

## Requirements:
# Only for diploids.
# Convention for file name extension (not mandatory):
#       fs (dadi), txt (dadi SNP data), fsc (fastSimCoal).
# Format of dadi:
#       strictly two lines;
#       dimension vector for the first line;
#       SFS for the second line;
#       no information for fold, mask or population names.
# Format of dadi SNP data:
#       refIn   refOut  Allele1 Aan     Aro     Allele2 Aan     Aro
#       AAA     AAA     A       49      56      T       7       0

## Frequent variables in the script:
# gt: a genotype object created from a VCF file and a popmap file.
# sfs: a table or array object representing SFS.

## Function list:
#vcf2gt(f.vcf,f.popmap)
#choose.pops(gt,pops)
#samSize(gt)
#minSamSize(gt)
#viewMissing(gt)
#filter.gt(gt,filter.indi=NA,filter.snp=NA)
#gt2sfs.raw(gt,pops)
#gt2sfs.impute(gt,pops,sampleSizes)
#project.sfs(sfs,sampleSizes)
#write.sfs.dadi(sfs,f.output)
#read.sfs.dadi(f.dadi,popnames=NULL)
#fold.sfs(sfs)
#sample.sfs(sfs)
#gt2snp(gt,pops)
#snp2sfs.raw(snp,pops,sampleSizes=NA)
#plot.sfs(sfs,cr=c(1,20000),colScheme=function(x)rainbow(x,s=0.8),...)
#plot.2D.pairwise(pops,sep="&",ext.o="fs",ext.e=NA,fold=T,mask=NA,
#       cr=c(1,20000),colScheme=function(x)rainbow(x,s=0.8))
#write.1D.fsc(sfs,f.output)
#write.2D.fsc(sfs,f.output)
#read.1D.fsc(f.fsc,popname=NULL)
#read.2D.fsc(f.fsc,popnames=NULL)

## Read VCF file and popmap file, and create a gt (genotype) object.
# gt: a list with two elements, i.e., a popmap vector and a genotype matrix.
# The FORMAT column of the VCF file should start with GT tag;
#       only biallelic, encoded with 0 and 1; "." as missing.
# The popmap file lists the population IDs of individuals;
#       two columns; tab delimited; individual ID as 1st column, pop ID as 2nd column;
#       1st column can be arbitrary; 2nd column should be an integer or string vector;
#       minimum sample size per population is 2.
vcf2gt<-function(f.vcf,f.popmap)
{
  oldw<-getOption("warn")
  options(warn=-1)

  # Read VCF file and popmap file.
  vcf.gt<-as.matrix(read.table(f.vcf,sep="\t",stringsAsFactors=F))[,-c(1:9)]
  popmap<-read.table(f.popmap,sep="\t",stringsAsFactors=F)[,2]

  nrow.vcf<-nrow(vcf.gt)
  ncol.vcf<-ncol(vcf.gt)

  # Parse genotypes.
  chrom1<-substring(vcf.gt,1,1)
  chrom2<-substring(vcf.gt,3,3)
  chrom<-matrix(as.integer(chrom1)+as.integer(chrom2),nrow.vcf,ncol.vcf)

  options(warn=oldw)

  list(popmap=popmap,genotype=chrom)
}

## Choose populations (subsetting the gt object by populations).
# pops: a character or integer vector; IDs of the chosen populations.
choose.pops<-function(gt,pops)
{
  popmap<-gt$popmap
  chrom<-gt$genotype

  # Choose populations.
  index<-sapply(pops,function(x){which(popmap==x)})
  index<-sort(unlist(index))
  popmap<-popmap[index]
  chrom<-chrom[,index]

  list(popmap=popmap,genotype=chrom)
}

## Calculate sample sizes of the populations.
samSize<-function(gt)
{
  popmap<-gt$popmap
  chrom<-gt$genotype

  pops<-unique(popmap)
  sampleSizes<-sapply(pops,function(x){sum(popmap==x)})
  names(sampleSizes)<-pops
  sampleSizes
}

## Calculate minimum sample sizes of the populations accounting for the missing values.
minSamSize<-function(gt)
{
  popmap<-gt$popmap
  chrom<-gt$genotype

  pops<-unique(popmap)
  sampleSizes<-sapply(pops,function(x){min(rowSums(!is.na(chrom[,popmap==x])))})
  names(sampleSizes)<-pops
  sampleSizes
}

## View the distribution of the missing values.
# Help decide whether some individuals or SNPs should be filtered out.
viewMissing<-function(gt)
{
  popmap<-gt$popmap
  chrom<-gt$genotype

  missing<-is.na(chrom)
  layout(matrix(c(1,2,1,3),2,2))
  image(1:nrow(missing),1:ncol(missing),missing,
        xlab=paste("SNPs: ",dim(missing)[1],sep=""),
        ylab=paste("Individuals: ",dim(missing)[2],sep=""),main="Missing values (red)",
        col=c("olivedrab2","firebrick3"))
  plot(sort(colSums(missing)),main="By individual",
       xlab="Individuals",ylab="Number of missing values")
  plot(table(rowSums(missing)),main="By SNP",
       xlab="Number of missing values",ylab="Number of SNPs")
}

## Filter the gt object for the missing values.
# filter.indi: integer; when the number of missing values is lager than this,
#       the individual will be filtered out; when NA, no filtering.
# filter.snp: integer; when the number of missing values is lager than this,
#       the SNP will be filtered out; when NA, no filtering.
filter.gt<-function(gt,filter.indi=NA,filter.snp=NA)
{
  popmap<-gt$popmap
  chrom<-gt$genotype

  # Filter individuals according to missing values.
  if(!is.na(filter.indi))
  {
    missingVal<-is.na(chrom)
    delete.indi<-which(colSums(missingVal)>filter.indi)
    if(length(delete.indi)>0)
    {
      chrom<-chrom[,-delete.indi]
      popmap<-popmap[-delete.indi]
    }
    rm(missingVal)
  }

  # Filter SNPs according to missing values.
  if(!is.na(filter.snp))
  {
    missingVal<-is.na(chrom)
    delete.snp<-which(rowSums(missingVal)>filter.snp)
    if(length(delete.snp)>0)
    {
      chrom<-chrom[-delete.snp,]
    }
    rm(missingVal)
  }

  list(popmap=popmap,genotype=chrom)
}

## Generate a SFS (table object) from the gt object.
# It will output a SFS based on raw count without accounting for the missing values.
# pops: a character or integer vector; IDs of populations to be included in the SFS.
gt2sfs.raw<-function(gt,pops)
{
  popmap<-gt$popmap
  chrom<-gt$genotype

  nrow.vcf<-nrow(chrom)
  ncol.vcf<-ncol(chrom)

  n.pop<-length(pops)

  # Number of chromosomes.
  ns.chr<-sapply(pops,function(x){sum(popmap==x)})*2

  # SFS based on raw count.
  cnt<-matrix(0,nrow.vcf,n.pop)
  ext<-list()
  for(i in 1:n.pop)
  {
    index<-which(popmap==pops[i])
    cnt[,i]<-rowSums(chrom[,index],na.rm=T)
    ext<-c(ext,list(0:ns.chr[i]))
  }
  ext<-as.matrix(expand.grid(ext))
  cnt<-data.frame(rbind(cnt,ext))
  sfs.raw<-table(cnt)-1
  names(dimnames(sfs.raw))<-pops

  sfs.raw
}

## Generate a SFS (table object) from the gt object.
# It will output an imputed SFS accounting for the missing values through binomial distribution.
# pops: a character or integer vector; IDs of populations to be included in the SFS.
# sampleSizes: an integer vector; downsized number of individuals in each population.
gt2sfs.impute<-function(gt,pops,sampleSizes)
{
  popmap<-gt$popmap
  chrom<-gt$genotype/2

  nrow.vcf<-nrow(chrom)
  ncol.vcf<-ncol(chrom)

  n.pop<-length(pops)

  # Number of chromosomes.
  ns.chr<-sampleSizes*2

  # SFS based on count imputed through binomial distribution.
  cnt<-matrix(0,nrow.vcf,n.pop)
  ext<-list()
  for(i in 1:n.pop)
  {
    index<-which(popmap==pops[i])
    p<-rowMeans(chrom[,index],na.rm=T)
    cnt[,i]<-rbinom(nrow.vcf,ns.chr[i],p)
    ext<-c(ext,list(0:ns.chr[i]))
  }
  ext<-as.matrix(expand.grid(ext))
  cnt<-data.frame(rbind(cnt,ext))
  sfs.imp<-table(cnt)-1
  names(dimnames(sfs.imp))<-pops

  sfs.imp
}

## Project a SFS to smaller sample sizes according to hypergeometric distribution.
# sampleSizes: an integer vector; downsized number of individuals in each population.
project.sfs<-function(sfs,sampleSizes)
{
  project<-function(a,m,n)
  {
    b<-numeric(m+1)
    for(i in 0:m)for(j in i:(n-m+i))
    {
      b[i+1]<-b[i+1]+choose(m,i)*choose(n-m,j-i)/choose(n,j)*a[j+1]
    }
    b
  }
  dims<-dim(sfs)
  l.dim<-length(dims)
  sfs.pro<-sfs
  if(l.dim==1)
  {
    m<-sampleSizes[1]*2
    n<-dims[1]-1
    sfs.pro<-array(project(sfs.pro,m,n))
  }else
  {
    for(k in 1:l.dim)
    {
      m<-sampleSizes[k]*2
      n<-dims[k]-1
      sfs.pro<-apply(sfs.pro,c(1:l.dim)[-k],function(x)project(x,m,n))
      perm<-integer(l.dim)
      perm[k]<-1
      perm[-k]<-2:l.dim
      sfs.pro<-aperm(sfs.pro,perm)
    }
  }
  dimnames(sfs.pro)<-lapply(sampleSizes*2,function(x)0:x)
  names(dimnames(sfs.pro))<-names(dimnames(sfs))
  sfs.pro<-as.table(sfs.pro)
  sfs.pro
}

## Write a SFS to a file in dadi format.
write.sfs.dadi<-function(sfs,f.output)
{
  sink(f.output)
  cat(dim(sfs))
  cat("\n")
  cat(aperm(sfs))
  sink()
}

## Read a SFS from a dadi format file.
read.sfs.dadi<-function(f.dadi,popnames=NULL)
{
  dims<-scan(f.dadi,nlines=1,comment.char="#",quiet=T)
  sfs<-scan(f.dadi,skip=1,nlines=1,comment.char="#",quiet=T)
  sfs<-aperm(array(sfs,rev(dims)))
  sfs<-as.table(sfs)
  dimnames(sfs)<-lapply(dims-1,function(x)0:x)
  names(dimnames(sfs))<-popnames
  sfs
}

## Fold a SFS.
fold.sfs<-function(sfs)
{
  sfs[]<-sfs+rev(sfs)
  dims<-dim(sfs)
  cnt.pool<-rowSums(expand.grid(lapply(dims-1,function(x)0:x)))
  index<-cnt.pool>(sum(dims-1)/2)
  sfs[index]<-0
  index<-cnt.pool==(sum(dims-1)/2)
  sfs[index]<-sfs[index]/2
  sfs
}

## Resampling from a SFS for the purpose of bootstrapping.
sample.sfs<-function(sfs)
{
  n.site<-sum(sfs)
  sfs[]<-0
  sfs[]<-rpois(length(sfs),sfs)
  sfs[1]<-n.site-sum(sfs)
  sfs
}

## Transform the gt object to a data.frame of dadi SNP data format.
# The function will output a dataframe as such:
#       refIn   refOut  Allele1 Aan     Aro     Allele2 Aan     Aro
#       AAA     AAA     A       49      56      T       7       0
# pops: a character or integer vector; IDs of populations to be included in the SFS.
gt2snp<-function(gt,pops)
{
  popmap<-gt$popmap
  chrom<-gt$genotype

  nrow.vcf<-nrow(chrom)
  ncol.vcf<-ncol(chrom)

  n.pop<-length(pops)

  ref<-integer(0)
  alt<-integer(0)
  for(i in 1:n.pop)
  {
    index<-which(popmap==pops[i])
    temp<-rowSums(chrom[,index],na.rm=T)
    ref<-cbind(ref,rowSums(!is.na(chrom[,index]))*2-temp)
    alt<-cbind(alt,temp)
  }

  snp<-data.frame("AAA","AAA","A",ref,"T",alt)
  names(snp)<-c("refIn","refOut","Allele1",pops,"Allele2",pops)

  snp
}

## Transform dadi SNP data format to SFS (table object).
# It will output a SFS based on raw count.
# snp: a headered data.frame of dadi SNP data as such:
#       refIn   refOut  Allele1 Aan     Aro     Allele2 Aan     Aro
#       AAA     AAA     A       49      56      T       7       0
# pops: a character vector; IDs of populations to be included in the SFS.
# sampleSizes: an integer vector of the same length as pops;
#       actual sample sizes (number of diploid individuals) of populations;
#       if NA, calculated from the SNP data.
snp2sfs.raw<-function(snp,pops,sampleSizes=NA)
{
  i.ref<-sapply(pops,function(x)which(names(snp)==x)[1])
  i.alt<-sapply(pops,function(x)which(names(snp)==x)[2])
  names(i.ref)<-names(i.alt)<-pops

  # Number of chromosomes.
  if(is.na(sampleSizes[1]))
  {
    ns.chr<-sapply(pops,function(x)max(snp[,i.ref[x]]+snp[,i.alt[x]]))
  }else
  {
    ns.chr<-sampleSizes*2
  }

  # SFS based on raw count.
  ext<-list()
  for(i in 1:length(pops))ext<-c(ext,list(0:ns.chr[i]))
  ext<-as.matrix(expand.grid(ext))
  cnt<-data.frame(rbind(snp[,i.alt],ext))
  sfs.raw<-table(cnt)-1
  names(dimnames(sfs.raw))<-pops

  sfs.raw
}

## Plot a SFS (barplot for 1D, image for 2D).
# cr: color range.
plot.sfs<-function(sfs,cr=c(1,20000),colScheme=function(x)rainbow(x,s=0.8),...)
{
  pops<-names(dimnames(sfs))
  dims<-dim(sfs)
  n.pop<-length(dims)
  sfs[1]<-sfs[length(sfs)]<-0

  # Color coding for number of SNPs.
  cr<-log10(cr)                       ## Color range
  nb<-101                             ## Number of breaks
  breaks<-10^seq(cr[1],cr[2],length.out=nb)

  if(n.pop==1)
  {
    barplot(sfs,names.arg=c(0,rep(NA,dims-2),dims-1),main=pops,...)
  }
  if(n.pop==2)
  {
    image(sfs,axes=F,col=colScheme(nb-1),breaks=breaks,
          xlab=pops[1],ylab=pops[2],...)
    axis(1,at=c(0,1),labels=c(0,dims[1]-1))
    axis(2,at=c(0,1),labels=c(0,dims[2]-1))
    box()
  }
  if(n.pop>2)stop("Cannot plot SFS with dimension higher than 2!")
}

## Plot pairwise 2D-SFSs among populations.
# Each file should have observed SFS of population pairs separated by certain symbol.
# pops: names of the populations as in the file names; order matters.
# sep: the separation symbol in the file names.
# ext.o: extension name of the files for the observed SFSs.
# ext.e: extension name of the files for the expected SFSs; if NA, not plotted.
# mask: a two-column matrix; coordinates of the bins to be masked; if NA, only mask fixed.
# cr: color range.
# colScheme: color scheme function.
plot.2D.pairwise<-function(pops,sep="&",ext.o="fs",ext.e=NA,fold=T,mask=NA,
                           cr=c(1,20000),colScheme=function(x)rainbow(x,s=0.8))
{
  n<-length(pops)

  # Color coding for number of SNPs.
  cr<-log10(cr)                       ## Color range
  nb<-101                             ## Number of breaks
  breaks<-10^seq(cr[1],cr[2],length.out=nb)

  # Layout of the plotting region.
  a<-matrix(0,n,n);k<-1;for(i in 1:(n-1))for(j in (i+1):n){a[i,j]<-k;k<-k+1}
  temp<-a^2-1;temp[temp[,]==-1]<-0;a<-t(temp)+a*2;a[a[,]==0]<-n^(n-1)+1:n
  layout(a,widths=c(rep(8/9/n,n),1/9))

  # Plot SFSs.
  par(mar=c(0,0.5,0.5,0),oma=c(1,1,1,1))          ##
  for(i in 1:(n-1))
  {
    for(j in (i+1):n)
    {
      # Plot observed.
      f.dadi<-sprintf("%s%s%s.%s",pops[i],sep,pops[j],ext.o)
      sfs<-read.sfs.dadi(f.dadi)
      if(fold)sfs<-fold.sfs(sfs)
      sfs[1]<-sfs[length(sfs)]<-0
      if(!is.na(mask[1]))sfs[mask]<-0
      image(sfs,col=colScheme(nb-1),axes=F,breaks=breaks)
      temp<-sum(sfs)

      # Plot expected.
      if(is.na(ext.e))
      {
        plot(0,type="n",axes=F,xlab=NA,ylab=NA)
      }else
      {
        f.dadi<-sprintf("%s%s%s.%s",pops[i],sep,pops[j],ext.e)
        sfs<-read.sfs.dadi(f.dadi)
        if(fold)sfs<-fold.sfs(sfs)
        sfs[1]<-sfs[length(sfs)]<-0
        if(!is.na(mask[1]))sfs[mask]<-0
        sfs<-round(sfs/sum(sfs)*temp)
        image(sfs,col=colScheme(nb-1),axes=F,breaks=breaks)
      }
    }
  }

  # Plot diagonal.
  for(i in 1:n)
  {
    plot(0,0,type="n",axes=F,xlab=NA,ylab=NA)
    text(0,0,pops[i],cex=1.5,font=2)
  }

  # Color bar.
  h<-dev.size()[2]
  par(mar=c(12,2,12,3)/9*h,cex.axis=1.5)          ##
  image(1,breaks[-nb],matrix(1:(nb-1),1),
        col=colScheme(nb-1),log="y",axes=F,xlab=NA,ylab=NA)
  axis(4)
}

## Write a 1D-SFS to a file in fastSimCoal format.
write.1D.fsc<-function(sfs,f.output)
{
  fsc<-matrix(sfs,1)
  pops<-names(dimnames(sfs))
  # add column names
  colnames(fsc)<-paste("d",pops,"_",1:ncol(fsc)-1,sep="")
  # output
  cat("1 observations\n",file=f.output)
  oldw<-getOption("warn")
  options(warn=-1)
  write.table(fsc,file=f.output,append=T,sep="\t",col.names=T,row.names=F,quote=F)
  options(warn=oldw)
}

## Write a 2D-SFS to a file in fastSimCoal format.
write.2D.fsc<-function(sfs,f.output)
{
  fsc<-as.matrix(sfs)
  pops<-names(dimnames(sfs))
  # add row and column names
  colnames(fsc)<-paste("d",pops[1],"_",1:nrow(fsc)-1,sep="")
  rownames(fsc)<-paste("d",pops[2],"_",1:ncol(fsc)-1,sep="")
  # output
  cat("1 observations\n",file=f.output)
  oldw<-getOption("warn")
  options(warn=-1)
  write.table(fsc,file=f.output,append=T,sep="\t",col.names=T,row.names=T,quote=F)
  options(warn=oldw)
}

## Read a 1D-SFS from a fastSimCoal format file.
read.1D.fsc<-function(f.fsc,popname=NULL)
{
  fsc<-read.table(f.fsc,header=T,sep="\t",skip=1)
  sfs<-array(unlist(fsc),ncol(fsc))
  sfs<-as.table(sfs)
  dimnames(sfs)<-lapply(ncol(fsc)-1,function(x)0:x)
  names(dimnames(sfs))<-popname
  sfs
}

## Read a 2D-SFS from a fastSimCoal format file.
read.2D.fsc<-function(f.fsc,popnames=NULL)
{
  fsc<-read.table(f.fsc,header=T,sep="\t",skip=1,row.names=1)
  sfs<-array(unlist(fsc),dim(fsc))
  sfs<-as.table(sfs)
  dimnames(sfs)<-lapply(dim(fsc)-1,function(x)0:x)
  names(dimnames(sfs))<-popnames
  sfs
}

##--------------Running the functions on my dataset ----
#Input files:
# - vcf
# - popmap (same format used in Stacks)

#Read the vcf and popmap files and create a gt object
mygt<-vcf2gt("conserved_MOD.vcf", "popmap.txt")
class(mygt)
str(mygt)

#samSize: Calculate sample sizes of the populations.
samSize(mygt)
#minSamSize: Calculate minimum sample sizes of the populations accounting for the missing values.
minSamSize(mygt)
#viewMissing: View the distribution of the missing values to help decide whether some individuals or SNPs
 should be filtered out.
viewMissing(mygt)

#generate SFS
ak = gt2sfs.raw(mygt, 'AK')
ak_an = gt2sfs.raw(mygt, c('AK','AN'))
ak_an_au = gt2sfs.raw(mygt, c('AK', 'AN', 'AU'))
# DOES NOT WORK - OBJECT WOULD NEED TO MUCH MEMORY! - mysfs<-gt2sfs.raw(mygt, c('AK', 'AN', 'AU', 'BO',
 'CA', 'CH', 'CR', 'CU', 'DU', 'ST')) # 2-dimensional

#plot.sfs: Plot a SFS (barplot for 1D, image for 2D)
plot.sfs(ak)
ak
plot.sfs(ak_an)
plot.sfs(ak_an_au) #Error in plot.sfs(ak_an_au) :  Cannot plot SFS with dimension higher than 2!

#SFS for each population
pop = read.table("popmap.txt", header = F, sep = "\t")
pops = popSV2
pops = levels(pops)

#par(mfrow=c(16,2))

for (i in pops) {
  sfs = gt2sfs.raw(mygt, i)
  plot.sfs(sfs)
  imp.sfs = gt2sfs.impute (mygt, i, length(which(popSV2==i)))
  plot.sfs(imp.sfs)
}


#SFS without population subdivision
#Read the vcf and popmap files and create a gt object
allgt<-vcf2gt("conserved_MOD.vcf", "popmap_noPopDivision.txt")
str(allgt)
samSize(allgt)
minSamSize(allgt)
viewMissing(allgt) #same pattern as when popmap divided populations, as expected
allsfs<-gt2sfs.raw(allgt, 'NZ')
plot.sfs(allsfs)

imp.allgt = gt2sfs.impute(allgt, 'NZ', 167) #this will generate an sfs with missing data imputed using a b
inomial distribution
plot.sfs(imp.allgt)
#dadi.allgt = project.sfs (allgt, 167) #Did not work for only one pop (might need two?) - This will genera
te an sfs with missing data imputed using a method similar to dadi`s imputation method (hypergeometric dis
tribution)


#SFS subdivided per clade
#Read the vcf and popmap files and create a gt object
cgt<-vcf2gt("conserved_MOD.vcf", "popmap_byClade.txt")
str(cgt)
samSize(cgt)
minSamSize(cgt)
viewMissing(cgt) #same pattern as when popmap divided populations, as expected
#sfs for North clade
Nsfs<-gt2sfs.raw(cgt, 'N')
plot.sfs(Nsfs)
imp.Nsfs = gt2sfs.impute(cgt, 'N', 45) #this will generate an sfs with missing data imputed using a binomi
al distribution
plot.sfs(imp.Nsfs)
#sfs for central clade
Csfs<-gt2sfs.raw(cgt, 'C')
plot.sfs(Csfs)
imp.Csfs = gt2sfs.impute(cgt, 'C', 28) #this will generate an sfs with missing data imputed using a binomi
al distribution
plot.sfs(imp.Csfs)
#sfs for southern clae
Ssfs<-gt2sfs.raw(cgt, 'S')
plot.sfs(Ssfs)
imp.Ssfs = gt2sfs.impute(cgt, 'S', 94) #this will generate an sfs with missing data imputed using a binomi
al distribution
plot.sfs(imp.Ssfs)
#sfs for N-C clades
NCsfs<-gt2sfs.raw(cgt,c('N', 'C'))
plot.sfs(NCsfs)
plot.2D.pairwise(NCsfs)

#sfs for N-S clades
NSsfs<-gt2sfs.raw(cgt, c('N', 'S'))
plot.sfs(NSsfs)
plot.2D.pairwise(NSsfs)

#sfs for C-S clades
CSsfs<-gt2sfs.raw(cgt, c('C', 'S'))
plot.sfs(CSsfs)
plot.2D.pairwise (CSsfs)
```

Current migration (DivMigrate)

Using divMigrate in R to calculate current migration rates and draw migration networks

In [ ]:
```
#install.packages("diveRsity")
library(diveRsity)

#diveRsity has a function to estimates contemporary migration rate. The imput can be in genepop format (po
ssible to convert vcf to genepop using PGDspider)

#read dataset in genepop format
x = readGenepop("allConservedman3/AllconservedTOgenepop.txt")
summary(x)

#running divMigrate (the function to estimate migration rates)
?divMigrate()
#man3 NZ-wide all loci
migration = divMigrate("allConservedman3/AllconservedTOgenepop.txt", outfile = "allConservedMigration", bo
ots = 100, stat = "all", filter_threshold = 0.05, plot_network = TRUE)

#Weir and Cockerham`s 1984 Fst, theta and F for loci, global and pairwise
fst =diffCalc(Test_data, fst = TRUE, bs_pairwise = T, bs_locus = T, boots = 100)


#test on my data (all loci used in man 2)
dat = readGenepop("test_myNeutralDataMan2/allInputTOgenepopGenalex.txt")
summary(dat)
class(dat) #list

corPlot("test_myNeutralDataMan2/allInputTOgenepopGenalex.txt")

#man3 NZ-wide all loci
migration = divMigrate("allConservedman3/AllconservedTOgenepop.txt", outfile = "allConservedMigration", bo
ots = 100, stat = "all", filter_threshold = 0.05, plot_network = TRUE)
```

dadi

Generate 2d site frequency spectrum (folded) with dadi for the NZ-wide scale, comapring the three clades Note: Maximum number of populations in dadi is 3, and 2 for plotting.

In [ ]:
```
#2. Work on dadi
#On the terminal
module purge && module load Miniconda3
source $(conda info --base)/etc/profile.d/conda.sh
export PYTHONNOUSERSITE=1
conda activate /nesi/project/uoa00538/dadi
python
import dadi
import matplotlib.pyplot as pyplot

#With Projections = 20 for each clade
dd = dadi.Misc.make_data_dict_vcf("conserved.vcf", "clade_def_conserved.txt") #import vcf file and populat
ion map specifying the three clades
fs = dadi.Spectrum.from_data_dict(dd, ['N', 'C', 'S'], projections = [20, 20, 20], polarized = False) #mak
e sfs with projections down  to 20/clade
fs.to_file('dadi_N20-C20-S20.Spectrum') #write spectrum to file
S = fs.S() #estimate number of segregating sites when projectios = 20 (will be smaller than the total numb
er of SNPs, which is 10987 in this dataset)
S # 10936.490549657357
Fst = fs.Fst() #estimate Fst when projectios = 20
Fst # 0.8527991593051584

#reduncing the 3D spectrum to 2D (comparing two clades at a time) with marginalize
fsNC = fs.marginalize([2])
fsNC.Fst() # Fst N-C = 0.8612213708348196
fsNC.S() # Number of segregating sites N-C = 7368.361432315463
fsNS = fs.marginalize([1])
fsNS.Fst() # Fst N-S = 0.8408142810575718
fsNS.S() # Number of segregating sites N-S = 8570.890473794867
fsCS = fs.marginalize([0])
fsCS.Fst() # Fst C-S = 0.8576458131009106
fsCS.S() # Number of segregating sites C-S = 6800.9246979136515

#plot the 2D sfs between clades, with projections = 20
#N-C
pyplot.figure()
dadi.Plotting.plot_single_2d_sfs (fsNC, vmin = 1)
pyplot.savefig('fs_N20-C20.pdf')
#N-S
pyplot.figure()
dadi.Plotting.plot_single_2d_sfs (fsNS, vmin = 1)
pyplot.savefig('fs_N20-S20.pdf')
#C-S
pyplot.figure()
dadi.Plotting.plot_single_2d_sfs (fsCS, vmin = 1)
pyplot.savefig('fs_C20-S20.pdf')
```

In [ ]:
```
#Splitting the dataset by individuals with VCFtools (keep.txt is a list of IDs to keep, matching the indiv
idual`s ID in the vcf file)
module load VCFtools/0.1.15-GCC-9.2.0-Perl-5.30.1
vcftools --vcf populations.snps.vcf --keep keep.txt --recode --recode-INFO-all --out central.out
```

Stairway plot 2

https://github.com/xiaoming-liu/stairway-plot-v2/blob/master/READMEv2.1.pdf (Installed with git clone in the nobackup folder from which it will be run)

1. Make blueprint file for each run/site frequency spectrum to be used (note: all samples are grouped into one single 'NZ' population in this sfs - generated by code above); the easiest way is to copy the example and modify it (nseq = 2*n, so for 61 numbers in the sfs generated with dadi, nseq = 61x2=122. This corresponds only to the first line of the spectrum, the second line is dadi's information on what to mask (1) and what not to mask (0), and also has 61 entries. Only copy the first line to the blueprint file.)
2. Run the first command to generate an .sh file
3. Modify the .sh file to transform it into a slurm script (can use sbatch to run)
4. sbatch the .sh file

In [ ]:
```
less proj60_NZ-neutral_fold.blueprint
#example blueprint file
#input setting
popid: NZ-neutral # id of the population (no white space)
nseq: 122 # number of sequences
L: 7008 # total number of observed nucleic sites, including polymorphic and monomorphic
whether_folded: true # whethr the SFS is folded (true or false)
SFS:19.08261783586516 94.84488732053092 227.834479729133 359.630662774561 432.8893767222623 442.6745834930
766 419.9138659371357 391.7353476507188 366.4184719713792 341.1548972667326 312.9607693061484 283.09218712
41347 255.6518461336769 234.2806221672121 219.9241804338117 210.6545896619242 203.0189528372821 193.858425
6353255 181.6976309531104 167.2194512213139 152.7878302784482 141.3528085617176 135.26227766805 135.454979
8630526 141.2741392436306 150.8566962965817 161.8443167726917 172.0893707139001 180.0927670996564 185.0722
659577284 93.37470136961919 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 #
 Snp frequency spectrum: number of singleton, number of doubleton, etc. (separated by white space)
smallest_size_of_SFS_bin_used_for_estimation: 0 # default is 1; to ignore singletons, uncomment this line
 and change this number to 2
largest_size_of_SFS_bin_used_for_estimation: 61 # default is nseq/2 for folded SFS
pct_training: 0.67 # percentage of sites for training
nrand:  30      60      90      122 # number of random break points for each try (separated by white spac
e)
project_dir: NZ-neutral_fold # project directory
stairway_plot_dir: stairway_plot_es # directory to the stairway plot files
ninput: 200 # number of input files to be created for each estimation
random_seed: 6
#output setting
mu: 2.2e-8 # assumed mutation rate per site per generation (considering lower bound used for COI in Sallou
m et al. 2020)
year_per_generation: 2 # assumed generation time (in years) (considering generation time in Ni et al., 202
0)
#plot setting
plot_title: NZ-neutral_fold # title of the plot
xrange: 0.1,10000 # Time (1k year) range; format: xmin,xmax; "0,0" for default
yrange: 0,0 # Ne (1k individual) range; format: xmin,xmax; "0,0" for default
xspacing: 2 # X axis spacing
yspacing: 2 # Y axis spacing
fontsize: 12 # Font size
```

In [ ]:
```
java -cp stairway_plot_es Stairbuilder proj60_NZ-neutral_fold.blueprint #generate the .sh file (run this f
rom the folder where stairway plot was unzipped)
```

In [ ]:
```
#Add the following to the top of the .sh file created with the above command, and then just run it with: s
batch prof60_NZ-neutral_fold.blueprint.sh
less proj60_NZ-neutral_fold.blueprint.sh

#!/bin/bash -e
#SBATCH --job-name=stairway
#SBATCH --output=stairway_%j.out
#SBATCH --error=stairway_%j.err
#SBATCH --mail-user=psal591@aucklanduni.ac.nz
#SBATCH --mail-type=END
#SBATCH --time=24:00:00
#SBATCH --mem=10G
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --account=uoa00538

cd /nesi/nobackup/uoa00538/programs/revisions_man3/migrationNe/stairway_plot2/stairway_plot_v2.1.1
```

PGDspider for converting vcf to Bayescan/BayeScEnv format

Install PGDspider locally. Files required for converting vcf to Bayescan/BayeScEnv format are:

- vcf file to be converted
- a file defining populations
- spid file (example below)

```
less example_vcf2bayescan.spid
# spid-file generated: Tue Sep 17 16:13:54 NZST 2019

# VCF Parser questions
PARSER_FORMAT=VCF

# Only output SNPs with a phred-scaled quality of at least:
VCF_PARSER_QUAL_QUESTION=
# Select population definition file:
VCF_PARSER_POP_FILE_QUESTION=./pop_def_conserved.tsv
# What is the ploidy of the data?
VCF_PARSER_PLOIDY_QUESTION=
# Do you want to include a file with population definitions?
VCF_PARSER_POP_QUESTION=TRUE
# Output genotypes as missing if the phred-scale genotype quality is below:
VCF_PARSER_GTQUAL_QUESTION=
# Do you want to include non-polymorphic SNPs?
VCF_PARSER_MONOMORPHIC_QUESTION=
# Only output following individuals (ind1, ind2, ind4, ...):
VCF_PARSER_IND_QUESTION=
# Only input following regions (refSeqName:start:end, multiple regions: whitespace separated):
VCF_PARSER_REGION_QUESTION=
# Output genotypes as missing if the read depth of a position for the sample is below:
VCF_PARSER_READ_QUESTION=
# Take most likely genotype if "PL" or "GL" is given in the genotype field?
VCF_PARSER_PL_QUESTION=
# Do you want to exclude loci with only missing data?
VCF_PARSER_EXC_MISSING_LOCI_QUESTION=

# GESTE / BayeScan Writer questions
WRITER_FORMAT=GESTE_BAYE_SCAN

# Specify which data type should be included in the GESTE / BayeScan file  (GESTE / BayeScan can only analyze one data type per file):
GESTE_BAYE_SCAN_WRITER_DATA_TYPE_QUESTION=SNP
```

```
#population definition file (one individual per line, no header, second column containing population definition)
head NZ-wide.popmap.txt

OnegAK10        AK
OnegAK11        AK
OnegAK12        AK
OnegAK13        AK
OnegAK14        AK
OnegAK15        AK
OnegAK16        AK
OnegAK17        AK
OnegAK18        AK
OnegAN01        AN
```

```
#Comand to run PGDspider (make sure to change the -inputfile to the name of the vcf input to be converted,
#the -outputfile to the desired name of the converted output file (with .txt at the end) and
#the -spid to the corresponding spid file for this dataset (the spid file contains infomation on the population definition file, so it is important that it matched the correct vcf file)

less southern_split_vcf2bayescan.sh
#!/bin/bash -e
#SBATCH --job-name=PGDSpider
#SBATCH --account=uoa00538
#SBATCH --time=00:15:00
#SBATCH --mem=12G
#SBATCH --output=PS%j.out
#SBATCH --error=PS%j.err

module load Java/1.8.0_144
module load SAMtools/1.9-GCC-7.4.0
module load BamTools/2.5.1-gimkl-2018b

java -Xmx2024m -Xms512m -jar PGDSpider2-cli.jar -inputfile ./southern_split.vcf -inputformat VCF -outputfile ./southern_splitBayescan.txt -outputformat GESTE_BAYE_SCAN -spid southern_split_vcf2bayescan.spid
```

Environmental files with ncdf4 in R

Note: the imput are downloaded netCDF version 4 files from the database (in this case NOOA)

```r
#Extracting environmental information from netCDF version 4 files


#load packages
library(chron)
library(RColorBrewer)
library(lattice)
library(ncdf4)

#check working directory
getwd()


#Read a NetCDF file using the ncdf package.
#Also, set the values for some temporary variables: ncname is the name of the neCDF file, while dname is the name of the variable that will be read in
ncname = "FILE_NAME.nc"
dname = "sst" #sea surface temperature

#Open the netcdf file
ncin = nc_open(ncname)
#print basic info about the file
print(ncin)

#Read the longitude and latitude
lon = ncvar_get(ncin, "lon")
nlon = dim(lon)
head(lon)

lat = ncvar_get(ncin, "lat")
nlat = dim(lat)
head(lat)

#confirm the dimensions od the data
print(c(nlon, nlat))

t = ncvar_get(ncin, "time")
tunits = ncatt_get(ncin, "time", "units")
t
# get the number of values and list them
nt = dim(t)
nt

#Print the time units string.
tunits

#Read the variable and its longname, units and fill value (_FillValue) attributes
tmp.array = ncvar_get(ncin, dname)
dlname = ncatt_get(ncin, dname, "long_name")
dunits = ncatt_get(ncin, dname, "units")
fillvalue = ncatt_get(ncin, dname, "_Fillvalue")

#the dimension of the array containing the values of the variables can be confirmed
dim(tmp.array)

#Read a set of global attributes (metadata)
title = ncatt_get(ncin, 0, "title")
institution = ncatt_get(ncin, 0, "institution")
datasource = ncatt_get(ncin, 0, "source")
references = ncatt_get(ncin, 0, "references")
history = ncatt_get(ncin, 0, "history")
Conventions = ncatt_get(ncin, 0, "Conventions")
#List the attribute values
title$value
institution$value
datasource$value
references$value
history$value
Conventions$value

#We are now done with the input dataset, so close it
nc_close(ncin)

#Convert the time variable from "time-since" units into "real" time values
# split the time units string into fields
tustr = strsplit(tunits$value, " ")
tdstr = strsplit(unlist(tustr)[3], "-")
tmonth = as.integer(unlist(tdstr)[2])
tday = as.integer(unlist(tdstr)[3])
tyear = as.integer(unlist(tdstr)[1])
chron(t, origin = c(tmonth, tday, tyear))

#Replcade Netcdf fillvalues (missing data) with R NAs
tmp.array[tmp.array == fillvalue$value] = NA

#Get the total number of non-missing grid cells on a slice (one slice, or layer) of the data (or brick)
length(na.omit(as.vector(tmp.array[, , 1])))


#Get a single time slice of the data, create an R data frame, and write a .csv file
m = 1
tmp.slice = tmp.array[, , m]
dim(tmp.slice)
lonlat = expand.grid(lon, lat)
tmp.vec = as.vector(tmp.slice)
length(tmp.vec)
tmp.df01 = data.frame(cbind(lonlat, tmp.vec))
names(tmp.df01) = c("lon", "lat", paste(dname, as.character(m), sep = "_"))
head(na.omit(tmp.df01), 20)
#Save the data frame as a .csv file, using na.omit() to drop the observations with missing data
csvfile = "FILENAME_TO_SAVE.csv"
write.table(na.omit(tmp.df01), csvfile, row.names = FALSE, sep = ",")


#Convert the whole array to a data frame, and calculate minimum, maximum and the annual mean
tmp.vec.long = as.vector(tmp.array)
length(tmp.vec.long)
tmp.mat = matrix(tmp.vec.long, nrow = nlon * nlat, ncol = nt)
dim(tmp.mat)
head(na.omit(tmp.mat))
lonlat = expand.grid(lon, lat)
tmp.df02 = data.frame(cbind(lonlat, tmp.mat))
#CHECK WHAT ARE EACH COLUMNS OF MY DATAFRAME BEFRE USING THE NAMES AS BELOW, WHICH WERE FOR ONLY 12 COLUMNS AND NOT 365
names(tmp.df02) = c("lon", "lat", 1:365)
options(width = 110)
head(na.omit(tmp.df02, 365))

#Get annual mean, mtwa and mtco values and add them the second data frame
tmp.df02$mtwa <- apply(tmp.df02[3:367], 1, max)  # mtwa
tmp.df02$mtco <- apply(tmp.df02[3:367], 1, min)  # mtco
tmp.df02$mat <- apply(tmp.df02[3:367], 1, mean)  # annual (i.e. row) means
head(na.omit(tmp.df02))

dim(na.omit(tmp.df02))

#Write the second data frame out as a .csv file, dropping NAs.
csvfile <- "FILENAME_TO_SAVE.csv"
write.table(na.omit(tmp.df02), csvfile, row.names = FALSE, sep = ",")

#Create a third data frame, with only non-missing values
tmp.df03 <- na.omit(tmp.df02)
head(tmp.df03)

write.table(na.omit(tmp.df03), "FILENAME_TO_SAVE.csv", row.names = FALSE, sep = ",")
```