Reconstructing global invasion pathways of the Asian topmouth gudgeon (*Pseudorasbora parva*): Inferences of multiple introductions and admixture

Thomas Brazier

Rennes 1 University - 2nd year MSc internship

Contact: brazier.thomas@gmail.com

Supervisor: Scott McCairns - INRA UMR ESE

Description

With the increase of global trade exchanges, many species are introduced outside their native distribution and become invasive, hence taking part in biotic homogenization and biodiversity loss. The Topmouth Gudgeon or Pseudorabora parva, a small Asian cyprinid, colonized with success the entire European river network. First documented introductions seem linked to the development of aquaculture in the Eastern Europe during 1960s, probably few individuals hitchhiking with aquaculture Chinese carps. An extensive sampling in the Asian native zone, with a genetic dataset of 3,000 SNPs, permitted the reconstruction of invasion scenario and estimation of invasion parameters (i.e. number of founders, time of introduction). A genetic structure analysis demonstrated the existence of three invasive demes clearly distinct, and a complex genetic structure in the native range, with high admixture rates between populations. Then, an Approximate Bayesian Computation method associated to Machine-Learning were used to infer the most probable demographic model to describe invasion pathways. Three independent introductions events were assessed, originating from different highly admixed populations. Thus, origins of invasive demes were uncovered in Chinese regions with the highest admixture rates (Central China, Wuhan region) corresponding to regions of intensive aquaculture. Introduced exotic populations often suffer of a genetic diversity loss due to the strong bottleneck during foundation. Yet P. parva introduced populations probably overcame the negative effects of bottlenecks thanks to genetic consequences of admixture and multiple introductions, hence contributing to their invasive success.

This Rmarkdown script summarize the data analyses done during the MSc internship. Only analyses kept for the report were reported here (see data analysis workflow for further details). When the scripts were runned multiple times (e.g. many different subsets in Europe and Asia, different datasets, replicates), procedures were displayed only one time, as example. Additionally, in order to stay under a reasonable number of pages, only a selection of analyses' outputs was displayed in the current report.

Loading environment

```
# clear global environment: remove all variables
rm(list=ls(all=TRUE))

#----------------------
# Loading packages
# SYSTEM
library(rstudioapi)
library(devtools)
library(data.table)
# GENETICS
library(ade4)
library(adegenet) # DAPC and population genetics statistics
library(hierfstat) # population genetics statistics
library(ape) # Phylogenetic trees
library(pegas) # population genetics statistics
library(genepop) # population genetics statistics
library(assignPOP) # population assignment
library(diveRsity) #population genetics statistics
library(poppr) # for function 'private_alleles'
# devtools::install_github("thierrygosselin/radiator")
library(radiator) # Genomic converter among different genomic data formats
# devtools::install_github("rystanley/genepopedit",dependencies=TRUE)
library(genepopedit)
# devtools::install_github('wrengels/HWxtest', subdir='pkg')
library(HWxtest)
# CARTOGRAPHY
library(rgdal)
library(raster)
library(gdistance)
library(maps) # draw maps for figures
library(mapdata)
library(mapplots)
library(rnaturalearth) # For finer cartography of river network
# Install gstudio, for spatial genetic analysis
# install.packages(
c("RgoogleMaps","geosphere","proto","sampling","seqinr","spacetime","spdep"),
dependencies=TRUE )
# devtools::install_github("dyerlab/gstudio")
# library(gstudio)
# GRAPHIC LIBRARIES
library(ggplot2)
library(ggpubr)
library(reshape2)
library(RColorBrewer)
library(ggplotify)
# STATS
```

```
library(MCMCglmm)
library(coda) # Tests of convergence for MCMC sampling chains
library(abc) # ABC model selection and parameters esimates with Neural
Network and Local Linear regression
library(abcrf) # ABC model selection and parameters esimates with Random
Forest
library(car)
# TREEMIX dependencies
# Dependencies of 'dartR'
# if (!requireNamespace("BiocManager", quietly = TRUE))
{install.packages("BiocManager")}
# BiocManager::install("SNPRelate", version = "3.8")
# BiocManager::install("qvalue")
library(dartR) # For converting genind to treemix input
library("ape")
# BiocManager::install("Biostrings")
library("Biostrings")
library("ggplot2")
# BiocManager::install("ggtree")
library("ggtree")

#----------------------
# Loading variables & objects
# Get the directory of the file & set working directory automatically with
rstudioapi::getSourceEditorContext()
# filedir=dirname(rstudioapi::getSourceEditorContext()$path)
# But this solution does not work in Rmarkdown, hence returning to classical
hand-written pathway
filedir="~/Google Drive/INRA PSEUDORASBORA/Analyses/R"
wd=filedir
setwd(wd)
# Define data directory
datadir=paste(filedir,"/Data",sep="")
# Define DIYABC directory
DIYABCdir=paste(gsub("/R","",filedir),"/DIYABC",sep="")
# Define STRUCTURE directory
STRdir=paste(gsub("/R","",filedir),"/STRUCTURE",sep="")
# Define the directory where to save figures
figuresdir=paste(filedir,"/Figures",sep="")
# Define the directory where to save tables in .csv format
tablesdir=paste(filedir,"/Tables",sep="")

# Import sources of functions
# Only functions developped by myself will be printed in appendix,
# some other functions were offered to scientific community by their owner
(e.g. Github)
source("Sources/map_nomenclature.R") # Function to draw scalebar and north
arrow
source("Sources/structureLauncher.R") # A function to generate a bash or a
qsub files for computation servers
```

```
#--------------------------------
# Import data and objects from Scott McCairns, INRA ESE Epix
#--------------------------------
load(paste(datadir,"/Pparva.3000.Rda",sep=""))
#--------------------------------
# Import coordinates of sampled locations
Pparva.sites=read.table(paste(datadir,"/Geographic/Pparva.sites.csv",sep=""),
sep=";",h=T)
labelPops=read.table("Data/STRUCTURE/labelPops.txt",header = TRUE)
# Lists of sampled sites names with their associated coordinates
native.names=c("Jap", "S1", "S2", "S3", "S4", "S6","S9",
               "S10", "S11", "S13", "S14", "S15", "S16", "S17", "S18", "S19",
"S20", "Tib")
native.coords=Pparva.sites[which(Pparva.sites$Pop %in% native.names),]
invasive.names=c("Aus", "Bel", "Bul1", "Bul2", "Hun", "Ira", "Ita", "Pol",
"Spa", "Tur", "UK")
invasive.coords=Pparva.sites[which(Pparva.sites$Pop %in% invasive.names),]

#--------------------------------
# GENIND FOR NATIVE & INVASIVE
# Create native and invasise genind
Native=Pparva[Pparva$pop %in% native.names]
Invasive=Pparva[Pparva$pop %in% invasive.names]
```

Cleaning & trimming the dataset

Description

This chapter performed basic descriptive analyses and computes populations summary statistics. All different analyses were performed in separate chapters.

Explore and describe the dataset

Populations:

18 populations in Chinese native area, 3 additional sites in Tibet, Japan and Taiwan, 13 populations in European invasive area, 34 populations in the data set before trimming…

16 populations in Chinese native area, 2 additional sites in Tibet, Japan, 11 populations in European invasive area, 29 populations in the data set after trimming…

```
# Number of populations
length(levels(Pparva@pop))

## [1] 29

# Populations size (number of individuals)
table(Pparva@pop)
```

```
##
##  Aus  Bel Bul1 Bul2  Hun  Ira  Ita  Jap  Pol   S1  S10  S11  S13  S14  S15
##   17   17   10   10   22   10   18   18   12   22   21   15   14   10   18
##  S16  S17  S18  S19   S2  S20   S3   S4   S6   S9  Spa  Tib  Tur   UK
##   13   22   10   16   17   20    8   19   15   16   19   26   19   14
```

Individuals

Description of individuals in the dataset

```
head(Pparva)

## /// GENIND OBJECT /////////
##
##  // 1 individual; 3,000 loci; 5,987 alleles; size: 1.6 Mb
##
##  // Basic content
##    @tab:  1 x 5987 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 1-2)
##    @loc.fac: locus factor for the 5987 columns of @tab
##    @all.names: list of allele names for each locus
##    @ploidy: ploidy of each individual  (range: 2-2)
##    @type:  codom
##    @call: .local(x = x, i = i, j = j, drop = drop)
##
##  // Optional content
##    @pop: population of each individual (group size range: 1-1)

# summary(Pparva)

# We need another data format, also from adegenet
# Instead of individuals, the new unit is populations
Pparva.pop=genind2genpop(Pparva)

##
##  Converting data from a genind to a genpop object...
##
## ...done.

# To which we add population coordinates
Pparva.pop@other$xy=Pparva.sites[,c(2,3)]

# Description of data set of populations
# summary(Pparva.pop)
```

Missing data

Load the full dataset, untrimmed, to evaluate missing data proportions

```
load(paste(datadir,"/Pparva.Rda",sep=""))
Pparva.hier=genind2hierfstat(Pparva)
```

```r
# Missing data per indiviudal
## First, compute a vector of missing data % per individual
missing.ind=c()
for (i in 1:nrow(Pparva.hier)) {
  missing.ind[i]=sum(is.na(Pparva.hier[i,]))/length(Pparva.hier[i,])
}
sum(missing.ind<0.1) # 112 indiv (15%) with less than 10% of missing data
```

```
## [1] 114
```

```r
# Arbitrary threshold of 45% that is the percentage of missing data on the
global data set
sum(missing.ind<0.45) # 412 indiv (56%) with less than 45% of missing data
(percentage of missing data in the global data set)
```

```
## [1] 414
```

```r
# Missing data per population
missing.pop=data.frame(unique(Pparva@pop),rep(NA,length(unique(Pparva@pop))))
for (i in unique(Pparva@pop)) {

missing.pop[which(missing.pop[,1]==i),2]=sum(is.na(genind2df(Pparva[Pparva@po
p==i,])))/(sum(Pparva@pop==i)*3999)
}
colnames(missing.pop)=c("Population","Missing data")
missing.pop
```

```
##    Population Missing data
## 1         Aus    0.3198180
## 2         Bel    0.4531459
## 3        Bul1    0.1318330
## 4        Bul2    0.1919730
## 5         Hun    0.1768594
## 6         Ira    0.6418126
## 7         Ita    0.2662689
## 8         Jap    0.3739522
## 9         Mor    0.8946125
## 10        Pol    0.5468492
## 11         S1    0.5205399
## 12        S10    0.2522131
## 13        S11    0.4336813
## 14        S13    0.5137697
## 15        S14    0.5573325
## 16        S15    0.2974948
## 17        S16    0.5411244
## 18        S17    0.2949383
## 19        S18    0.5205775
## 20        S19    0.4646616
## 21         S2    0.3902339
## 22        S20    0.2922731
## 23         S3    0.5147441
```

```
## 24         S4    0.4632908
## 25         S5    0.7258620
## 26         S6    0.4497077
## 27         S7    0.7760008
## 28         S9    0.5354491
## 29        Slo    0.8566460
## 30        Spa    0.2736252
## 31        Tai    0.7672085
## 32        Tib    0.3302232
## 33        Tur    0.1916729
## 34         UK    0.4598292
```

```
# Choose an arbitrary threshold of 0.7 for missing data in a population
missing.pop[missing.pop[,2]>0.7,]
```

```
##    Population Missing data
## 9         Mor    0.8946125
## 25         S5    0.7258620
## 27         S7    0.7760008
## 29        Slo    0.8566460
## 31        Tai    0.7672085
```

```
# Some populations have very high amounts of missing data, over 70%: Morocco
(89%), Slovenia (85%), S7 (77%), S5 (73%), Tai (76%)
# Zoom on details of the Moroccan population
Mor.data=genind2df(Pparva[Pparva@pop=="Mor",])

# Missing data per allele
## First, compute a vector of missing data % per allele
missing.all=c()
for (i in 1:ncol(Pparva@tab)) {
  missing.all[i]=sum(is.na(Pparva@tab[,i]))/length(Pparva@tab[,i])
}
sum(missing.all<0.1) # There is no allele with less than 10% of missing data
```

```
## [1] 0
```

```
sum(missing.all<0.45) # 4682 of 9146 alleles (52%) have less than 45% of
missing data
```

```
## [1] 4210
```

```
# Missing data per locus
## First, compute a vector of missing data % per locus
missing.locus=c()
for (i in 1:ncol(Pparva.hier)) {
  missing.locus[i]=sum(is.na(Pparva.hier[,i]))/length(Pparva.hier[,i])
}
sum(missing.locus<0.1) # There is only 1 locus with less than 10% of missing
data
```

```
## [1] 1
```

```
sum(missing.locus<0.45) # 2112 of 3999 locus (53%) have less than 45% of
missing data
```

```
## [1] 2112
```

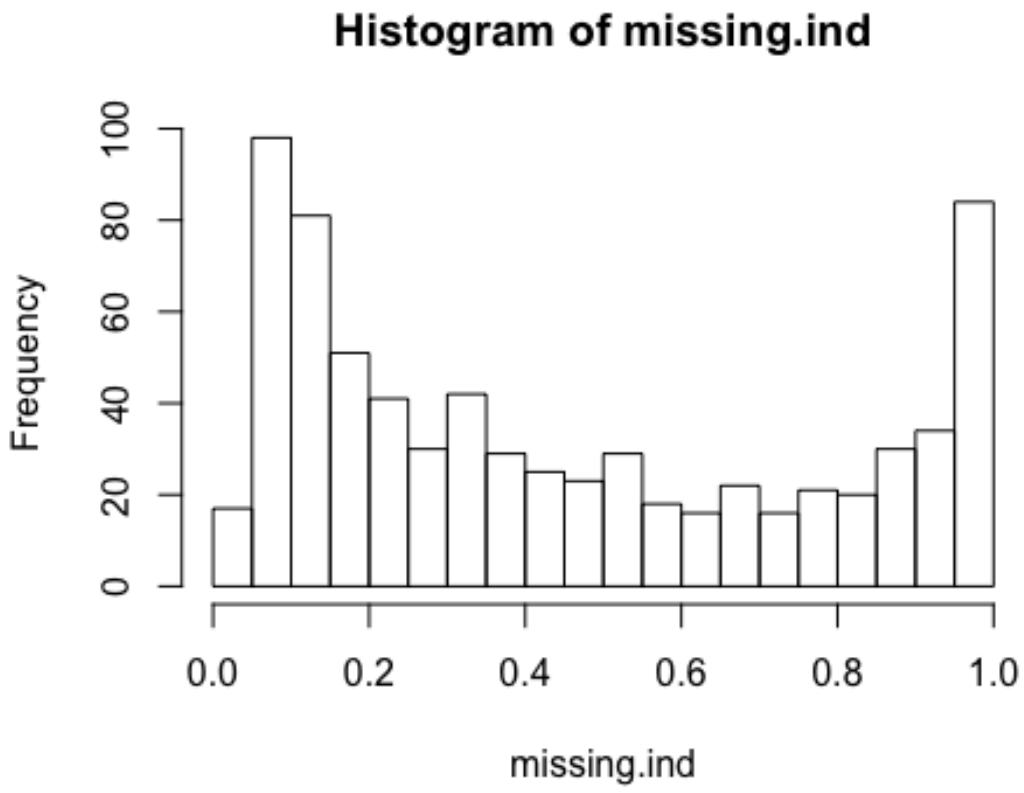
```
sum(missing.locus>0.5) #
```

```
## [1] 1076
```

Strategy for missing data

In this part, we described further the proportion of missing data to design a selection procedure improving our dataset qualtiy. We tried to assess which trimming criterion would enhance the quality. Was it better to filter individuals or loci in order to remove high proportions of missing data?

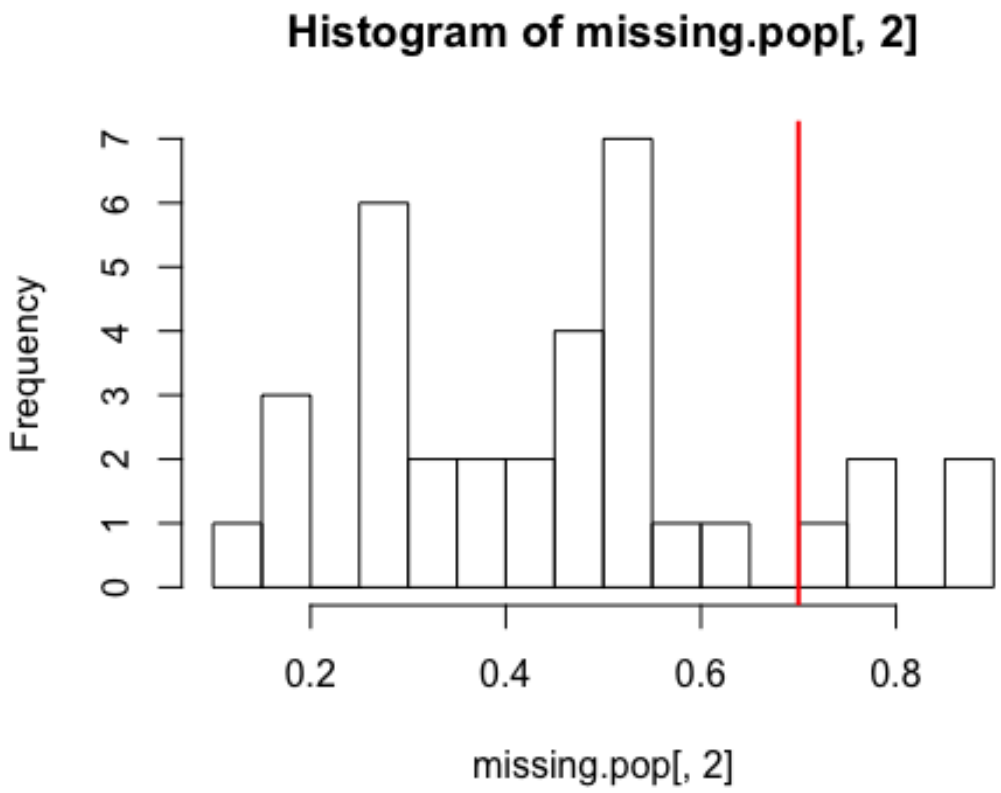
```
# Global missing data
hist(missing.ind,breaks=20)
```



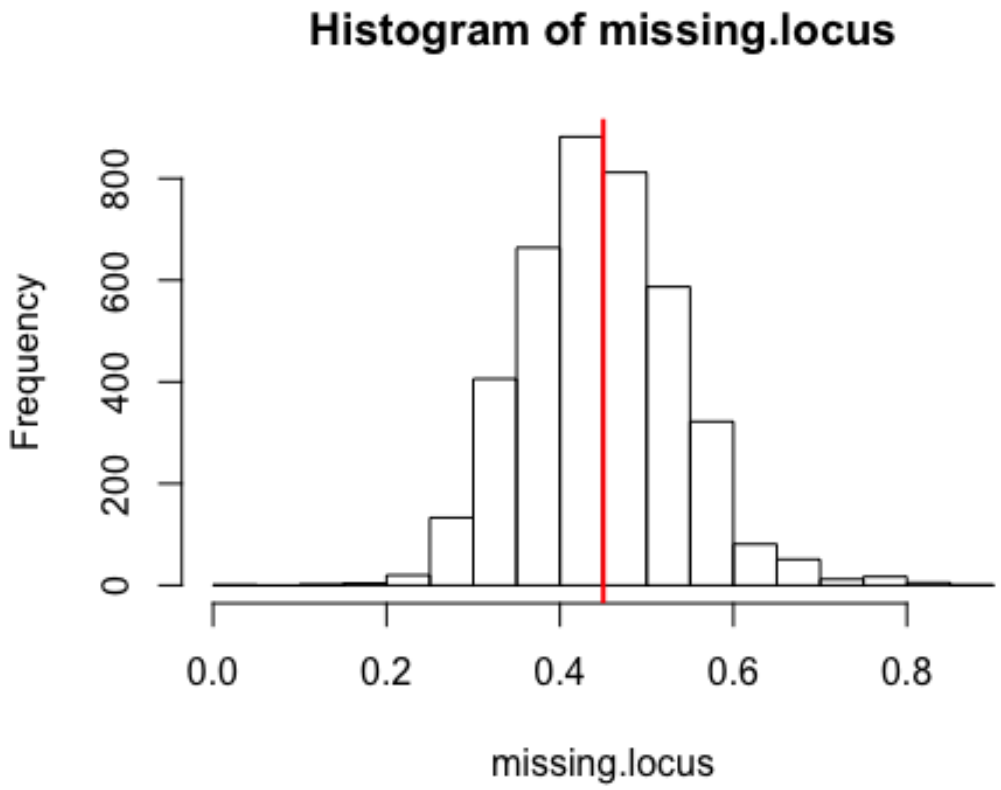

```
# Inflection point around 0.5-0.6 of missing data per individual
# We removed individuals with more than 60% of missing data
sum(missing.ind<0.6)
```

```
## [1] 484
```

```r
sum(missing.ind<0.45)

## [1] 414

# Missing data per population
# missing.pop
hist(missing.pop[,2],breaks=20)
abline(v=0.7,lwd=2,col="red")
```




```r
# % of missing data in pop. dropped after 0.6, and then there was a peak
after 0.7
# Cutting pop. above 0.7 of missing data removed these outliers

# Missing data per alleles
# hist(missing.all,breaks=20)
# abline(v=0.45,lwd=2,col="red")

# Missing data per locus
hist(missing.locus,breaks=20)
abline(v=0.45,lwd=2,col="red")
```




```
# By retaining only loci with less than 45% of missing data, we kept only
half of original markers
# that were the most informative markers.
# We assumed that markers with more than 45% of missing data did not carried
more information than markers with less than 45% of missing,
# but this threshold was somewhat arbitrary

# N=468 (64%) if we removed all individuals with more than 60% of missing
data
# & populations with more than 70% of missing data (i.e. 5 pops leading to
too few individuals per pop. to retain)
(Pparva.clean=Pparva[!(Pparva@pop %in% c("Mor","Tai","Slo","S5","S7")) &
missing.ind<0.6])

## /// GENIND OBJECT /////////
##
##  // 468 individuals; 3,999 loci; 7,980 alleles; size: 16.4 Mb
##
##  // Basic content
##    @tab:  468 x 7980 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 1-2)
##    @loc.fac: locus factor for the 7980 columns of @tab
##    @all.names: list of allele names for each locus
```

```
##    @ploidy: ploidy of each individual  (range: 2-2)
##    @type:  codom
##    @call: .local(x = x, i = i, j = j, drop = drop)
##
##  // Optional content
##    @pop: population of each individual (group size range: 8-26)
```

```
table(Pparva.clean@pop)
```

```
##
##  Aus  Bel Bul1 Bul2  Hun  Ira  Ita  Jap  Pol   S1  S10  S11  S13  S14  S15
##   17   17   10   10   22   10   18   18   12   22   21   15   14   10   18
##  S16  S17  S18  S19   S2  S20   S3   S4   S6   S9  Spa  Tib  Tur   UK
##   13   22   10   16   17   20    8   19   15   16   19   26   19   14
```

```
#-----------------------------------
# Keep only SNPs with less than 45% of missing data
# Recompute missing data per locus on the new data set
missing.locus2=c()
for (i in 1:ncol(Pparva.hier)) {
  missing.locus2[i]=sum(is.na(Pparva.hier[,i]))/length(Pparva.hier[,i])
}
sum(missing.locus2<0.1) # There is only 1 locus with less than 10% of missing
data
```

```
## [1] 1
```

```
sum(missing.locus2<0.45) # 2112 of 3999 locus (53%) have less than 45% of
missing data
```

```
## [1] 2112
```

```
# Percentage of missing data is reduced to 22% instead of 45% by eliminating
the half of markers that were the less informative

#-----------------------------------
# Finally, we needed to retain more loci, to gain genetic information for DIY
ABC
# We chose to retain the 3,000 best loci instead of 2,112 previously
missing.locus3=c()
for (i in 1:ncol(Pparva.hier)) {
  missing.locus3[i]=sum(is.na(Pparva.hier[,i]))/length(Pparva.hier[,i])
}
sum(missing.locus3<0.1) # There is only 1 locus with less than 10% of missing
data
```

```
## [1] 1
```

```
sum(missing.locus3<0.45) # 2112 of 3999 locus (53%) have less than 45% of
missing data
```

```
## [1] 2112
```

```
(Pparva.3000=Pparva.clean[loc=order(missing.locus3,decreasing=FALSE)[1:3000]]
)

## /// GENIND OBJECT /////////
##
##  // 468 individuals; 3,000 loci; 5,987 alleles; size: 12.3 Mb
##
##  // Basic content
##    @tab:  468 x 5987 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 1-2)
##    @loc.fac: locus factor for the 5987 columns of @tab
##    @all.names: list of allele names for each locus
##    @ploidy: ploidy of each individual  (range: 2-2)
##    @type:  codom
##    @call: .local(x = x, i = i, j = j, loc = ..1, drop = drop)
##
##  // Optional content
##    @pop: population of each individual (group size range: 8-26)

# From now, Pparva.clean became the new Pparva
# Pparva=Pparva.clean
# save(Pparva,file=file(paste(datadir,"/Pparva.clean.Rda",sep="")))
# After running STRUCTURE on Pparva.clean (2,112 loci), we reconsidered the
number of loci to retain, in order to improve genetic information for DIY ABC
# Hence, we took 3,000 loci
# Pparva=Pparva.3000
# save(Pparva,file=file(paste(datadir,"/Pparva.3000.Rda",sep="")))
load(paste(datadir,"/Pparva.3000.Rda",sep=""))
# Recompute genpop
Pparva.pop=genind2genpop(Pparva)

##
##  Converting data from a genind to a genpop object...
##
## ...done.
```

Sensitivity analysis to missing data

Huge large-scale data set of SNP are computationnaly demanding, as well as Bayesian approaches. Thus, a reduction of the original data set is often recommended and performed before Bayesian analysis. Hence, we needed first to assess the sensitivity of ABC approaches to our data set and look at which extent SuSt of ABC were affected by subsetting inside the data set. There exist different paradigms of subsetting to evaluate:

- Random sampling among the full dataset
- Selective sampling, based on a quality criterion (e.g. most informative markers, with a threshold of missing data proportion)

Sensitivity analysis is the study of how the uncertainty in the output of a mathematical model or system (numerical or otherwise) can be apportioned to different sources of uncertainty in its inputs. Here, we tested sensitivity to missing data and data reduction (subsetting). Sensitivity analysis was measured on independent population summary statistics (SuSt).

SAMPLING A RANDOM SET OF MARKERS

Basic idea here was to study the evolution of SuSt (summary statistics) as a function of a number of markers randomly sampled inside the global data set.

Usual SuSt for ABC are (Jeffries et al. 2016):

- mean genetic diversity across all polymorphic loci
- variance of genetic diversity across all polymorphic loci
- mean genetic diversity across all loci
- variance of genetic diversity across all loci
- mean/variance of Fst (for Fst > 0 for 2 loci & for all loci)
- mean/variance of Nei's distance (for Nei's distance > 0 for 2 loci & for all loci)
- max-likelihood estimates of admixture proportions

We did it for SuSt presented in the user manual of DIYABC for more detailed information:

- proportion of loci with null gene diversity (= proportion of monomorphic loci)
- mean gene diversity across polymorphic loci (Nei, 1987)
- variance of gene diversity across polymorphic loci
- mean gene diversity across all loci

MEAN GENETIC DIVERSITY ACROSS ALL LOCI: Hst

We chose to investigate first: mean genetic diversity across all loci, the Hst stat. of hierfstat's function basic.stats() (Nei, 1987)

In subsequent chunks of code, computation can be very demanding, hence codelines were not evaluated, and figures and results were printed with saved R objects.

```
mean(basic.stats(Pparva.hier)$overall[3])
# Confidence interval (95%) of the "true" Hst on the global data set was
computed with 1,000 bootstraps (resampling of individuals)
# BOOTSTRAP MEAN Hst
boots=numeric(1000)
for (i in 1:1000){
  print(i)
```

```
boots[i]=basic.stats(Pparva.hier[,c(1,sample(2:(ncol(Pparva.hier)),replace=T)
)])$overall[3]
}
save(boots,file="Data/bootstrap.mean.Hst")
load(file="Data/bootstrap.mean.Hst")
hist(boots)
limBoot=quantile(boots,c(.025,.975))
(IC.Hst=limBoot)
(mean.Hst=mean(boots))
(sd.Hst=sd(boots))

# Build a matrix with n iterations for each value of N (nb of randomly
sampled SNPs), with iterations in columns and SuSt in rows
niter=100
samp.n=seq(100,(ncol(Pparva.hier)-101),100) # Number of markers to sample
SuSt.Hst=matrix(NA,nrow=niter,ncol=length(samp.n)) # Index of each row
correspond to one randomised replicate of the SuSt
# Index of the column give the number of markers sampled: (N -101)/100
markers sampled (-1 for first column of pop & -100 for setting lowest number
of markers to 100)
# Indices are a sequence from 100 to N markers by step of 100 markers

# From 1 sampled SNP only to all SNPs sampled
c=0
for (N in samp.n) {
  c=c+1
  # The SuSt is computed from a random subset of N markers for each iteration
  for (i in 1:niter) {
    cat("Iteration:", i,"of",N,"markers sampled
(",c,"/",length(samp.n),")\n")
    # Subset N marker randomly at each iteration (subset N columns in
genind@tab)
    subset=Pparva.hier[,c(1,sample(2:ncol(Pparva.hier),N,replace=FALSE))]
    # Compute the SuSt, and add it to the matrix 'SuSt'
    SuSt.Hst[i,c]=basic.stats(subset)$overall[3]
  }
}
rm(c)
save(SuSt.Hst,file="Data/SuSt_Hst.Rda")
load("Data/SuSt_Hst.Rda")

# !!! #  Computing time was very long and time increased exponentially with
number of markers,
# !!! #  so we needed to implement parallel programmming to achieve a larger
number of random bootstraps

load(paste(datadir,file="/Pparva.Rda",sep=""))
Pparva.hier=genind2hierfstat(Pparva)
samp.n=seq(100,(ncol(Pparva.hier)-101),100) # Number of markers to sample
```

```r
load("Data/bootstrap.mean.Hst")
load("Data/SuSt_Hst.Rda")
limBoot=quantile(boots,c(.025,.975))
(IC.Hst=limBoot)
```

```
##   2.5%  97.5%
## 0.0334 0.0381
```

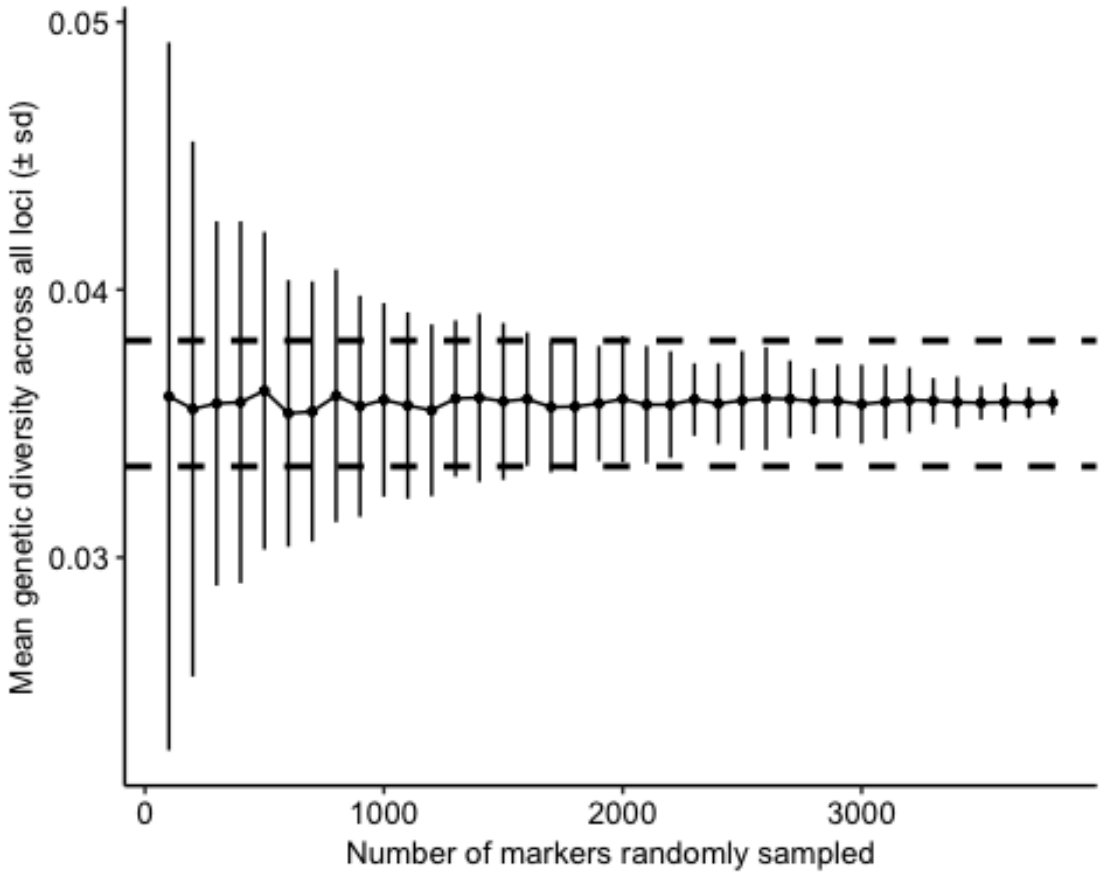
```r
(mean.Hst=mean(boots))
```

```
## [1] 0.0357758
```

```r
(sd.Hst=sd(boots))
```

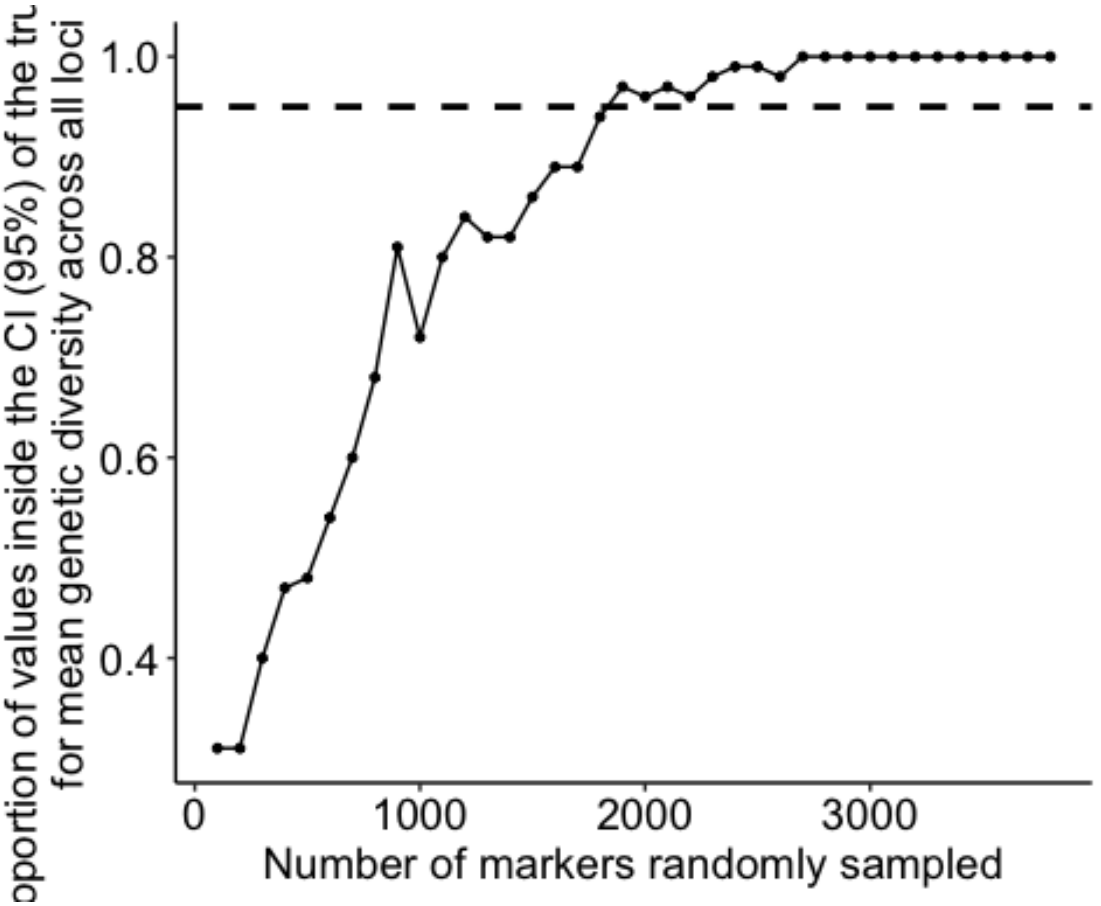
```
## [1] 0.001158143
```

```r
df=data.frame(N=samp.n, Mean=apply(SuSt.Hst, 2, mean), sd=apply(SuSt.Hst,
2,function(x) quantile(x,0.975))-apply(SuSt.Hst, 2, mean)) # Plot the CI
instead of sd of the resampled means

plot.Hst=ggplot(data=df, aes(x=N, y=Mean)) +
    geom_point(colour="Black",size=1)+
    geom_line() +
    geom_errorbar(aes(ymin=Mean-sd, ymax=Mean+sd), width=.2) +
    geom_hline(yintercept=c(IC.Hst[1], IC.Hst[2]), linetype="dashed", size=1)
+
    xlab("Number of markers randomly sampled") + ylab("Mean genetic diversity
across all loci (± sd)") +
    theme(axis.line = element_line(colour = "black"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank(),
          plot.title = element_text(color="black", size=10,
face="bold.italic",hjust = 0.5),
          plot.subtitle = element_text(color="black",size=10,hjust = 0.5),
          axis.title.x = element_text(color="black", size=10),
          axis.title.y = element_text(color="black", size=10),
          axis.text=element_text(size=10, colour="black"),
          legend.key = element_rect(fill = "white", size = 1),
          legend.text=element_text(size=10),
          legend.title=element_text(size=10))
plot.Hst
```




```
# ggsave(paste(figuresdir,"/Sensitivity/plot.Hst.png",sep=""),
#        device="png",dpi=320,units="cm",width=20,height=16)

############
# PROPORTION OF REPLICATES INSIDE THE CONFIDENCE INTERVAL OF THE TRUE VALUE
OF THE SU. STATISTIC
# TYPE I ERROR: rejection of a true null hypothesis (false positive), i.e.
seing a difference between the original data set and a subset

# For each number of marker, compute the proportion of values inside the CI
of the true value
# A matrix of 2 columns with 1/ number of markers and 2/ proportion of exact
values
true.Hst=matrix(NA,ncol=2,nrow=length(samp.n))
true.Hst[,1]=samp.n
true.Hst[,2]=apply(SuSt.Hst, 2, function(x) (sum(x>IC.Hst[1] &
x<IC.Hst[2])/length(x)))
colnames(true.Hst)=c("N.Markers","Prop.Exact.Values")
# true.Hst
# Plotting the proportion of values inside the CI of the true value
# Add a horizontal line at 95%
plot.prop.Hst=ggplot(data=as.data.frame(true.Hst), aes(x=N.Markers,
y=Prop.Exact.Values)) +
```

```
  geom_point(colour="Black",size=1)+
  geom_line() +
  geom_hline(yintercept=0.95, linetype="dashed", size=1) +
  xlab("Number of markers randomly sampled") + ylab("Proportion of values
inside the CI (95%) of the true value\nfor mean genetic diversity across all
loci") +
  theme(axis.line = element_line(colour = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        plot.title = element_text(color="black", size=14,
face="bold.italic",hjust = 0.5),
        plot.subtitle = element_text(color="black",size=14,hjust = 0.5),
        axis.title.x = element_text(color="black", size=14),
        axis.title.y = element_text(color="black", size=14),
        axis.text=element_text(size=14, colour="black"),
        legend.key = element_rect(fill = "white", size = 1),
        legend.text=element_text(size=14),
        legend.title=element_text(size=14))
plot.prop.Hst
```



```
# ggsave(paste(figuresdir,"/Sensitivity/plot.trueValues.Hst.png",sep=""),
#        device="png",dpi=320,units="cm",width=20,height=16)
```

MEAN GENETIC DIVERSITY ACROSS POLYMORPHIC LOCI: Hst_poly

```
# Confidence interval (95%) of the "true" Hst on the global data set was
computed with 1,000 bootstraps
# BOOTSTRAP MEAN Hst
boots=numeric(100)
for (i in 1:100){
  print(i)
  temp=c()

temp=basic.stats(Pparva.hier[,c(1,sample(2:(ncol(Pparva.hier)),replace=T))])$
Hs
  boots[i]=mean(apply(temp,2,function(x) mean(x[x>0],na.rm=TRUE)))
  rm(temp)
}
save(boots,file="Data/bootstrap.mean.Hst_poly")
load(file="Data/bootstrap.mean.Hst_poly")
hist(boots)
limBoot=quantile(boots,c(.025,.975))
(IC.Hst=limBoot)
(mean.Hst=mean(boots))
(sd.Hst=sd(boots))

# Build a matrix with n iterations for each value of N (nb of randomly
sampled SNPs), with iterations in columns and SuSt in rows
niter=100
samp.n=seq(100,(ncol(Pparva.hier)-101),100) # Number of markers to sample
SuSt.Hst.poly=matrix(NA,nrow=niter,ncol=length(samp.n)) # Index of each row
correspond to one randomised replicate of the SuSt
# Index of the column give the number of markers sampled: (N -101)/100
markers sampled (-1 for first column of pop & -100 for setting lowest number
of markers to 100)
# Indices are a sequence from 100 to N markers by step of 100 markers

# From 1 sampled SNP only to all SNPs sampled
c=0
for (N in samp.n) {
  c=c+1
  # The SuSt is computed from a random subset of N markers for each iteration
  for (i in 1:niter) {
    cat("Iteration:", i,"of",N,"markers sampled
(",c,"/",length(samp.n),")\n")
    temp=c()
    # Subset N marker randomly at each iteration (subset N columns in
genind@tab)
    subset=Pparva.hier[,c(1,sample(2:ncol(Pparva.hier),N,replace=FALSE))]
    # Compute the SuSt on all loci
    temp=basic.stats(subset)$Hs
```

```
    # remove monomorphic loci (0 values)
    SuSt.Hst.poly[i,c]=mean(apply(temp,2,function(x)
mean(x[x>0],na.rm=TRUE)))
    rm(temp)
  }
}
rm(c)
save(SuSt.Hst.poly,file="Data/SuSt_Hst_poly.Rda")
load("Data/SuSt_Hst_poly.Rda")

###########
# PROPORTION OF REPLICATES INSIDE THE CONFIDENCE INTERVAL OF THE TRUE VALUE
OF THE SU. STATISTIC
# TYPE I ERROR: rejection of a true null hypothesis (false positive), i.e.
seing a difference between the original data set and a subset

# For each number of marker, compute the proportion of values inside the CI
of the true value
# A matrix of 2 columns with 1/ number of markers and 2/ proportion of exact
values
true.Hst.poly=matrix(NA,ncol=2,nrow=length(samp.n))
true.Hst.poly[,1]=samp.n
true.Hst.poly[,2]=apply(SuSt.Hst.poly, 2, function(x) (sum(x>IC.Hst.poly[1] &
x<IC.Hs[2])/length(x)))
colnames(true.Hst.poly)=c("N.Markers","Prop.Exact.Values")
true.Hst.poly
```

PROPORTION OF LOCI WITH NULL GENE DIVERSITY (I.E. MONOMORPHIC LOCI)

Then we chose to investigate proportion of loci with null gene diversity. We computed the mean proportion.

```
# Confidence interval (95%) of the "true" proportion on the global data set
was computed with 1,000 bootstraps
# BOOTSTRAP PROPORTION OF LOCI WITH NULL GENE DIVERSITY
(gene.div=basic.stats(Pparva.hier)$Hs) # bootstrap on populations
boots=numeric(1000)
for (i in 1:1000){
  print(i)
  # We first computed the bootstrap by resampling Hs values
  #boots[i]=mean(sample(apply(gene.div, 2, mean, na.rm=TRUE),replace=TRUE))
  # BUT it was a mistake and we computed next by resampling markers randomly
  # This way, it corresponded to the mean value at N=3999

boots[i]=mean(basic.stats(Pparva.hier[,c(1,sample(2:(ncol(Pparva.hier)),repla
ce=T))])$Hs,na.rm=TRUE)
}
save(boots,file="Data/bootstrap.mean.Hs")
load(file="Data/bootstrap.mean.Hs")
hist(boots)
```

```
(IC.Hs=quantile(boots,c(.025,.975)))
(mean.Hs=mean(boots))
(sd.Hs=sd(boots))

# Build a matrix with n iterations for each value of N (nb of randomly
sampled SNPs), with iterations in columns and SuSt in rows
niter=100
samp.n=seq(100,(ncol(Pparva.hier)-101),100) # Number of markers to sample
SuSt.Hs=matrix(NA,nrow=niter,ncol=length(samp.n)) # Index of each row
correspond to one randomised replicate of the SuSt
# Index of the column give the number of markers sampled: (N -101)/100
markers sampled (-1 for first column of pop & -100 for setting lowest number
of markers to 100)
# Indices are a sequence from 100 to N markers by step of 100 markers


# From 1 sampled SNP only to all SNPs sampled
c=0
for (N in samp.n) {
  c=c+1
  # The SuSt is computed from a random subset of N markers for each iteration
  for (i in 1:niter) {
    cat("Iteration:", i,"of",N,"markers sampled
(",c,"/",length(samp.n),")\n")
    # Subset N marker randomly at each iteration (subset N columns in
genind@tab)
    subset=Pparva.hier[,c(1,sample(2:ncol(Pparva.hier),N,replace=FALSE))]
    # Compute the SuSt, and add it to the matrix 'SuSt'
    SuSt.Hs[i,c]=mean(basic.stats(subset)$Hs,na.rm=TRUE)
  }
}
rm(c)
save(SuSt.Hs,file="Data/SuSt_GeneDiversityHs.Rda")
load("Data/SuSt_GeneDiversityHs.Rda")

###########
# PROPORTION OF REPLICATES INSIDE THE CONFIDENCE INTERVAL OF THE TRUE VALUE
OF THE SU. STATISTIC
# TYPE I ERROR: rejection of a true null hypothesis (false positive), i.e.
seing a difference between the original data set and a subset

# For each number of marker, compute the proportion of values inside the CI
of the true value
# A matrix of 2 columns with 1/ number of markers and 2/ proportion of exact
values
true.Hs=matrix(NA,ncol=2,nrow=length(samp.n))
true.Hs[,1]=samp.n
true.Hs[,2]=apply(SuSt.Hs, 2, function(x) (sum(x>IC.Hs[1] &
x<IC.Hs[2])/length(x)))
```

```
colnames(true.Hs)=c("N.Markers","Prop.Exact.Values")
true.Hs
```

SUBSETTING ONLY THE MOST INFORMATIVE MARKERS

The subsetting was made after sorting loci by proportion of missing data. For each iteration, we retained the N loci with the lower proportion of missing data. Then, a bootstrap with resampling was made on these loci to estimate the mean value of the statistic.

```
# MAKE AN INDEX OF LOCI SORTED PER PROPORTION OF MISSING DATA
missing.locus=c() # Proportion of missing data per locus
for (i in 2:ncol(Pparva.hier)) {
  missing.locus[i]=sum(is.na(Pparva.hier[,i]))/length(Pparva.hier[,i])
}
# Sort loci per increasing proportion of missing data and create an ordered
index
(order.idx=order(missing.locus))
missing.locus[order(missing.locus)]
# Hence, by sampling the N first loci of this index, we'll be assured to
sample only the N most informative markers (in terms of missing data)

#-------------------------------------------------------------------------
------------------------
# MEAN GENETIC DIVERSITY ACROSS ALL LOCI: Hst
#-------------------------------------------------------------------------
------------------------
# We chose to investigate first: mean genetic diversity across all loci, the
Hst stat. of hierfstat's function basic.stats() (Nei, 1987)

# Confidence interval (95%) of the "true" Hst on the global data set was
computed with 1,000 bootstraps
# BOOTSTRAP MEAN Hst
boots=numeric(1000)
for (i in 1:1000){
  print(i)

boots[i]=basic.stats(Pparva.hier[,c(1,sample(2:(ncol(Pparva.hier)),replace=T)
)])$overall[3]
}
save(boots,file="Data/bootstrap.mean.Hst.ordered")
load(file="Data/bootstrap.mean.Hst.ordered")
hist(boots)
(IC.Hst.ordered=quantile(boots,c(.025,.975)))
(mean.Hst.ordered=mean(boots))
(sd.Hst.ordered=sd(boots))

# We suspected a bias in the global data set that could be imputed to missing
data (see subsequent figures)
# So we also computed the:
# Confidence interval (95%) of the "true" Hst on the reduced data set
(N=2112) with 1,000 bootstraps
```

```
# BOOTSTRAP MEAN Hst
boots=numeric(1000)
for (i in 1:1000){
  print(i)
  # Random resampling within the 2112 best markers

boots[i]=basic.stats(Pparva.hier[,c(1,sample(order.idx[1:2112],replace=T))])$
overall[3] # random sampling within the 2112 best loci, those taht have been
sampled
}
save(boots,file="Data/bootstrap.mean.Hst.reduced")
load(file="Data/bootstrap.mean.Hst.reduced")
hist(boots)
(IC.Hst.reduced=quantile(boots,c(.025,.975)))
(mean.Hst.reduced=mean(boots))
(sd.Hst.reduced=sd(boots))

# Build a vector for each value of N (nb of randomly sampled SNPs), with
iterations in columns and SuSt in rows
samp.n=c(seq(100,3900,100),3999) # Number of markers to sample
SuSt.Hst.ordered=rep(NA,length(samp.n)) # Index of each row correspond to one
randomised replicate of the SuSt
SuSt.Hst.upper=rep(NA,length(samp.n))
SuSt.Hst.lower=rep(NA,length(samp.n))
# Index of the column give the number of markers sampled: (N -101)/100
markers sampled (-1 for first column of pop & -100 for setting lowest number
of markers to 100)
# Indices are a sequence from 100 to N markers by step of 100 markers

# From 1 sampled SNP only to all SNPs sampled
c=0
for (N in samp.n) {
  c=c+1
  # The SuSt is computed from a random subset of N markers
  cat("\n", N,"markers sampled\n")
  # Subset N most informative markers at each iteration (subset N columns in
genind@tab given the index 'order.idx')
  subset=Pparva.hier[,c(1,order.idx[1:N])]
  # Compute the SuSt, and add it to the matrix 'SuSt'
  SuSt.Hst.ordered[c]=basic.stats(subset)$overall[3]
  # Compute the CI of the SuSt by resampling loci (variance of loci chosen)
  boots=numeric(1000)
  for (i in 1:1000){
    cat("Bootstrap #", i, "... ")
    # Random resampling of loci within the N best markers

boots[i]=basic.stats(subset[,c(1,sample(2:(ncol(subset)),replace=T))])$overal
l[3] # random sampling within the N best loci, those that have been sampled
  }
  SuSt.Hst.upper[c] = quantile(boots,c(.975))
```

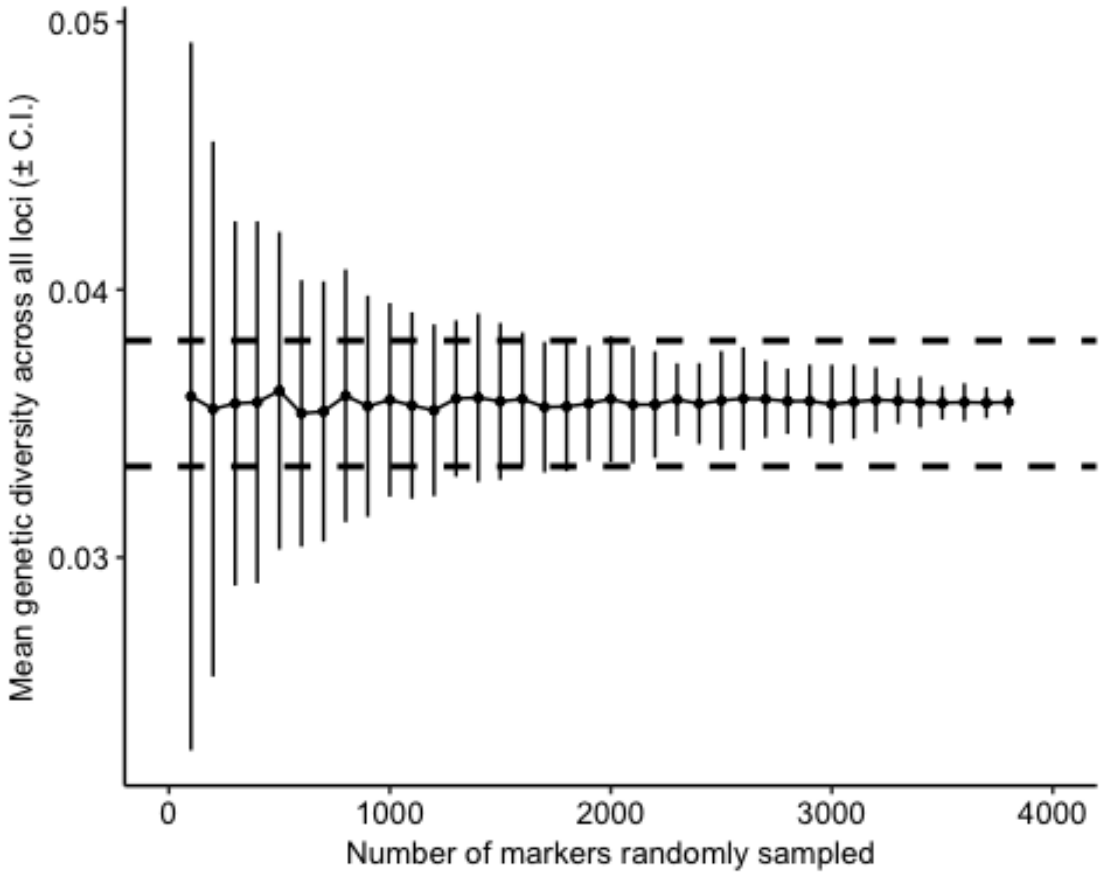
```
  SuSt.Hst.lower[c] = quantile(boots,c(.025))
}
SuSt.Hst.ordered = data.frame(SuSt.Hst.ordered, SuSt.Hst.upper,
SuSt.Hst.lower)
save(SuSt.Hst.ordered,file="Data/SuSt_Hst_ordered.boot.Rda")
load("Data/SuSt_Hst_ordered.boot.Rda")

# Find the value where missing data overcome 0.55, correspond to the
inflection point of the SuSt around locus n°3500
length(missing.locus)-
length(order.idx[which(missing.locus[order(missing.locus)]>0.55)]) # Locus
3511
idx=length(missing.locus)-
length(order.idx[which(missing.locus[order(missing.locus)]>0.55)])

# Plotting the mean of the stat for each number of sampled markers by order
# Add a grey ribbon for the confidence interval (95%) of the "true" Hst on
the global data set, computed with 1,000 bootstraps
df=data.frame(N=samp.n, Mean=SuSt.Hst.ordered$SuSt.Hst.ordered, lower =
SuSt.Hst.lower, upper = SuSt.Hst.upper)
plot.Hst.ordered=ggplot(data=df, aes(x=N, y=Mean)) +
  geom_point(colour="Black",size=1)+
  geom_line() +
  geom_errorbar(aes(ymin=lower, ymax=upper), width=.2) +
  geom_hline(yintercept=c(IC.Hst.ordered[1], IC.Hst.ordered[2]),
linetype="dashed", size=1) +
  geom_hline(yintercept=c(IC.Hst.reduced[1], IC.Hst.reduced[2]),
linetype="dashed", col="DarkGrey", size=1) +
  geom_vline(xintercept=idx,col="Red") +
  xlab("Number of selected markers") + ylab("Mean genetic diversity across
all loci ") +
  theme(axis.line = element_line(colour = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        plot.title = element_text(color="black", size=14,
face="bold.italic",hjust = 0.5),
        plot.subtitle = element_text(color="black",size=14,hjust = 0.5),
        axis.title.x = element_text(color="black", size=14),
        axis.title.y = element_text(color="black", size=14),
        axis.text=element_text(size=14, colour="black"),
        legend.key = element_rect(fill = "white", size = 1),
        legend.text=element_text(size=14),
        legend.title=element_text(size=14))
plot.Hst.ordered
# Error bars are the CI by resampling loci (variance of loci sampling)
ggsave(paste(figuresdir,"/Sensitivity/plot.Hst.ordered.png",sep=""),
       device="png",dpi=320,units="cm",width=20,height=16)
```

```
#-------------------------------------------------------------
# PROPORTION OF LOCI WITH NULL GENE DIVERSITY (I.E. MONOMORPHIC LOCI) --> Hs
#-------------------------------------------------------------
# Then we chose to investigate proportion of loci with null gene diversity
(Hs)
# We computed the mean proportion

# Confidence interval (95%) of the "true" Hs on the global data set was
computed with 1,000 bootstraps
# BOOTSTRAP MEAN Hs
boots=numeric(1000)
for (i in 1:1000){
  print(i)

boots[i]=mean(basic.stats(Pparva.hier[,c(1,sample(2:(ncol(Pparva.hier)),repla
ce=T))])$Hs,na.rm=TRUE)
}
save(boots,file="Data/bootstrap.mean.Hs.ordered")
load(file="Data/bootstrap.mean.Hs.ordered")
hist(boots)
(IC.Hs.ordered=quantile(boots,c(.025,.975)))
(mean.Hs.ordered=mean(boots))
(sd.Hs.ordered=sd(boots))

# We suspected a bias in the global data set that could be imputed to missing
data (see subsequent figures)
# So we also computed the:
# Confidence interval (95%) of the "true" Hst on the reduced data set
(N=2112) with 1,000 bootstraps
# BOOTSTRAP MEAN Hst

# We suspected a bias in the global data set that could be imputed to missing
data (see subsequent figures)
# So we also computed the:
# Confidence interval (95%) of the "true" Hst on the reduced data set
(N=2112) with 1,000 bootstraps
# BOOTSTRAP MEAN Hst
boots=numeric(1000)
for (i in 1:1000){
  print(i)
  # Random resampling within the 2112 best markers

boots[i]=mean(basic.stats(Pparva.hier[,c(1,sample(order.idx[1:2112],replace=T
))])$Hs,na.rm=TRUE) # random sampling within the 2112 best loci, those that
have been sampled for analysis
}
boots[i]=mean(basic.stats(Pparva.hier[,c(1,sample(order.idx[1:2112],replace=T
))])$Hs,na.rm=TRUE) # random sampling within the 2112 best loci, those that
have been sampled for analysis
```

```r
save(boots,file="Data/bootstrap.mean.Hs.reduced")
load(file="Data/bootstrap.mean.Hs.reduced")
hist(boots)
(IC.Hs.reduced=quantile(boots,c(.025,.975)))
(mean.Hs.reduced=mean(boots))
(sd.Hs.reduced=sd(boots))

# Build a vector for each value of N (nb of randomly sampled SNPs), with
iterations in columns and SuSt in rows
samp.n=c(seq(100,3900,100),3999) # Number of markers to sample
SuSt.Hst.ordered=rep(NA,length(samp.n)) # Index of each row correspond to one
randomised replicate of the SuSt
# Index of the column give the number of markers sampled: (N -101)/100
markers sampled (-1 for first column of pop & -100 for setting lowest number
of markers to 100)
# Indices are a sequence from 100 to N markers by step of 100 markers

# From 1 sampled SNP only to all SNPs sampled
c=0
for (N in samp.n) {
  c=c+1
  # The SuSt is computed from a random subset of N markers
  cat(N,"markers sampled\n")
  # Subset N most informative markers at each iteration (subset N columns in
genind@tab given the index 'order.idx')
  subset=Pparva.hier[,c(1,order.idx[1:N])]
  # Compute the SuSt, and add it to the matrix 'SuSt'
  SuSt.Hs.ordered[c]=mean(basic.stats(subset)$Hs,na.rm=TRUE)
}
save(SuSt.Hs.ordered,file="Data/SuSt.Hs.ordered.Rda")
load("Data/SuSt.Hs.ordered.Rda")

# Find the value where missing data overcome 0.55, correspond to the
inflection point of the SuSt around locus n°3500
length(missing.locus)-
length(order.idx[which(missing.locus[order(missing.locus)]>0.55)])
idx=length(missing.locus)-
length(order.idx[which(missing.locus[order(missing.locus)]>0.55)])

# Plotting the mean of the stat for each number of sampled markers by order
# Add a grey ribbon for the confidence interval (95%) of the "true" Hst on
the global data set, computed with 1,000 bootstraps
df=data.frame(N=samp.n, Mean=SuSt.Hs.ordered)
plot.Hs.ordered=ggplot(data=df, aes(x=N, y=Mean)) +
  geom_point(colour="Black",size=1)+
  geom_line() +
  geom_hline(yintercept=c(IC.Hs.ordered[1], IC.Hs.ordered[2]),
linetype="dashed", size=1) +
  geom_hline(yintercept=c(IC.Hs.reduced[1], IC.Hs.reduced[2]),
linetype="dashed", col="DarkGrey", size=1) +
```

```
  geom_vline(xintercept=idx,col="Red") +
  xlab("Number of selected markers") + ylab("Proportion of monomorphic loci")
+
  theme(axis.line = element_line(colour = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        plot.title = element_text(color="black", size=14,
face="bold.italic",hjust = 0.5),
        plot.subtitle = element_text(color="black",size=14,hjust = 0.5),
        axis.title.x = element_text(color="black", size=14),
        axis.title.y = element_text(color="black", size=14),
        axis.text=element_text(size=14, colour="black"),
        legend.key = element_rect(fill = "white", size = 1),
        legend.text=element_text(size=14),
        legend.title=element_text(size=14))
plot.Hs.ordered
ggsave(paste(figuresdir,"/Sensitivity/plot.GeneDiversityHs.ordered.png",sep="
"),
       device="png",dpi=320,units="cm",width=20,height=16)

##### Figures illustrating results

# MEAN GENETIC DIVERSITY ACROSS ALL LOCI: Hst

#```{r, message = FALSE}
# Manipulation of a hierfstat object is easier than a genind
load(paste(datadir,"/Pparva.Rda",sep=""))
Pparva.hier=genind2hierfstat(Pparva)
#------------------------------------------------------------------------
# Markers randomly sampled
#------------------------------------------------------------------------
# We chose to investigate first: mean genetic diversity across all loci, the
Hst stat. of hierfstat's function basic.stats() (Nei, 1987)
load(file="Data/bootstrap.mean.Hst")
limBoot=quantile(boots,c(.025,.975))
(IC.Hst=limBoot)

##   2.5%  97.5%
## 0.0334 0.0381

(mean.Hst=mean(boots))

## [1] 0.0357758

(sd.Hst=sd(boots))

## [1] 0.001158143
```

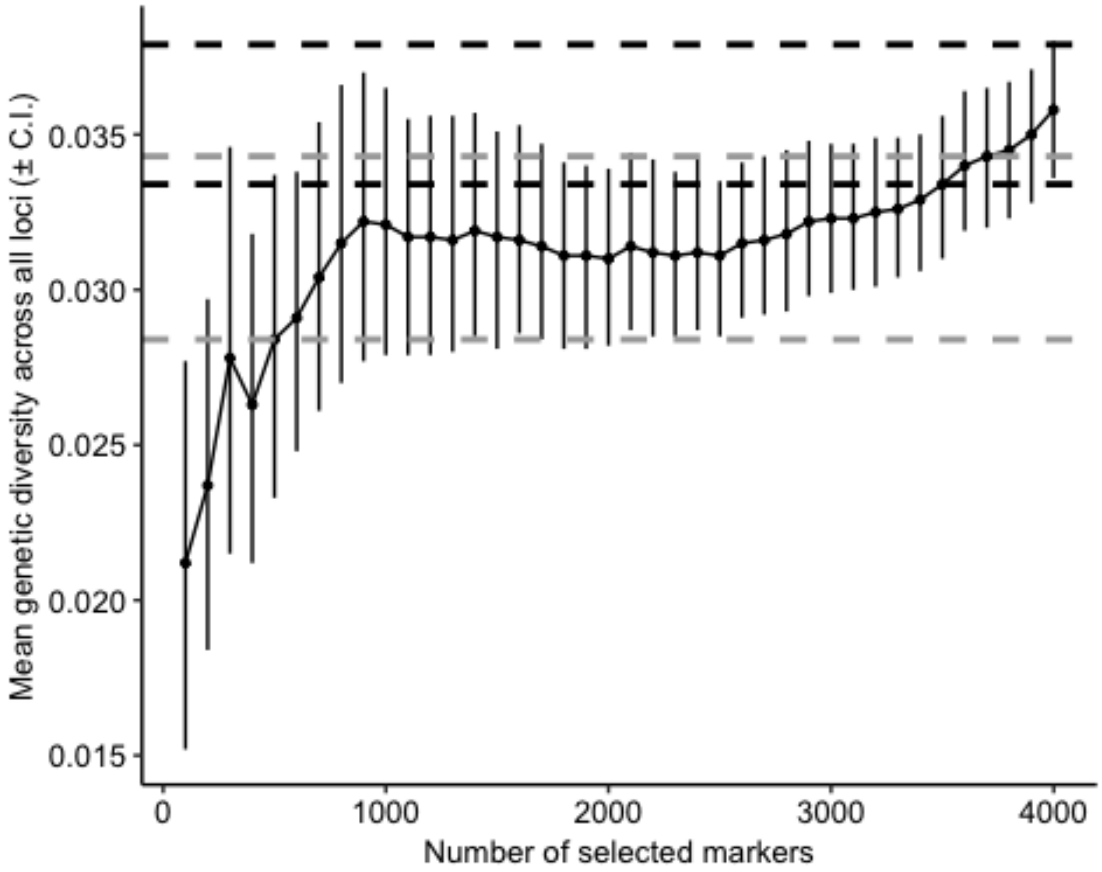
```r
load("Data/SuSt_Hst.Rda")
samp.n=seq(100,(ncol(Pparva.hier)-101),100) # Number of markers to sample

df=data.frame(N=samp.n, Mean=apply(SuSt.Hst, 2, mean), sd=apply(SuSt.Hst,
2,function(x) quantile(x,0.975))-apply(SuSt.Hst, 2, mean)) # Plot the CI
instead of sd of the resampled means
plot.Hst=ggplot(data=df, aes(x=N, y=Mean)) +
  geom_point(colour="Black",size=1)+
  geom_line() +
  geom_errorbar(aes(ymin=Mean-sd, ymax=Mean+sd), width=.2) +
  geom_hline(yintercept=c(IC.Hst[1], IC.Hst[2]), linetype="dashed", size=1) +
  xlab("Number of markers randomly sampled") + ylab("Mean genetic diversity
across all loci (± C.I.)") +
  xlim(0, 4000) +
  theme(axis.line = element_line(colour = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        plot.title = element_text(color="black", size=10,
face="bold.italic",hjust = 0.5),
        plot.subtitle = element_text(color="black",size=10,hjust = 0.5),
        axis.title.x = element_text(color="black", size=10),
        axis.title.y = element_text(color="black", size=10),
        axis.text=element_text(size=10, colour="black"),
        legend.key = element_rect(fill = "white", size = 1),
        legend.text=element_text(size=10),
        legend.title=element_text(size=10))
plot.Hst
```




```
#-----------------------------------------------------------------------
# Markers ordered by proportion of missing data
#-----------------------------------------------------------------------
# MAKE AN INDEX OF LOCI SORTED PER PROPORTION OF MISSING DATA
missing.locus=c() # Proportion of missing data per locus
for (i in 2:ncol(Pparva.hier)) {
  missing.locus[i]=sum(is.na(Pparva.hier[,i]))/length(Pparva.hier[,i])
}

# Sort loci per increasing proportion of missing data and create an ordered
index
order.idx=order(missing.locus)
missing.locus[order(missing.locus)]

load(file="Data/bootstrap.mean.Hst.ordered")
(IC.Hst.ordered=quantile(boots,c(.025,.975)))

##   2.5%  97.5%
## 0.0334 0.0379

(mean.Hst.ordered=mean(boots))

## [1] 0.0357434

(sd.Hst.ordered=sd(boots))
```

```
## [1] 0.001125586

load(file="Data/bootstrap.mean.Hst.reduced")
(IC.Hst.reduced=quantile(boots,c(.025,.975)))

##   2.5%  97.5%
## 0.0284 0.0343

(mean.Hst.reduced=mean(boots))

## [1] 0.0313704

(sd.Hst.reduced=sd(boots))

## [1] 0.001478117

load("Data/SuSt_Hst_ordered.boot.Rda")
samp.n=c(seq(100,3900,100),3999) # Number of markers to sample

# Find the value where missing data overcome 0.55, correspond to the
inflection point of the SuSt around locus n°3500
length(missing.locus)-
length(order.idx[which(missing.locus[order(missing.locus)]>0.55)]) # Locus
3511

## [1] 3511

idx=length(missing.locus)-
length(order.idx[which(missing.locus[order(missing.locus)]>0.55)])

# Plotting the mean of the stat for each number of sampled markers by order
# Add a grey ribbon for the confidence interval (95%) of the "true" Hst on
the global data set, computed with 1,000 bootstraps
df=data.frame(N=samp.n, Mean=SuSt.Hst.ordered$SuSt.Hst.ordered, lower =
SuSt.Hst.ordered$SuSt.Hst.lower,
              upper = SuSt.Hst.ordered$SuSt.Hst.upper)
plot.Hst.ordered=ggplot(data=df, aes(x=N, y=Mean)) +
  geom_point(colour="Black",size=1)+
  geom_line() +
  geom_errorbar(aes(ymin=lower, ymax=upper), width=.2) +
  geom_hline(yintercept=c(IC.Hst.ordered[1], IC.Hst.ordered[2]),
linetype="dashed", size=1) +
  geom_hline(yintercept=c(IC.Hst.reduced[1], IC.Hst.reduced[2]),
linetype="dashed", col="DarkGrey", size=1) +
  # geom_vline(xintercept=idx,col="Red", size = 0.8) +
  xlab("Number of selected markers") + ylab("Mean genetic diversity across
all loci (± C.I.)") +
  theme(axis.line = element_line(colour = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
```

```
        plot.title = element_text(color="black", size=10,
face="bold.italic",hjust = 0.5),
        plot.subtitle = element_text(color="black",size=10,hjust = 0.5),
        axis.title.x = element_text(color="black", size=10),
        axis.title.y = element_text(color="black", size=10),
        axis.text=element_text(size=10, colour="black"),
        legend.key = element_rect(fill = "white", size = 1),
        legend.text=element_text(size=10),
        legend.title=element_text(size=10))
plot.Hst.ordered
```




```
# Error bars are the CI by resampling loci (variance of loci sampling)

# ggarrange(plot.Hst,plot.Hst.ordered,widths=1:1,heights=1:1,labels="auto")
# ggsave(paste(figuresdir,"/Sensitivity/PlotArrange.Hst.ordered.png",sep=""),
       # device="png",dpi=320,units="cm",width=40,height=16)
```

Sampled sites - Geography

We imported the coordinates of sampled locations. Geographic coordinates were important for figures and the Isolation-by-Distance (IBD) analysis.

```
#--------------------------------
# Import coordinates of sampled locations
Pparva.sites=read.table(paste(datadir,"/Geographic/Pparva.sites.csv",sep=""),
sep=";",h=T)
labelPops=read.table("Data/STRUCTURE/labelPops.txt",header = TRUE)
# Lists of sampled sites names with their associated coordinates
native.names=c("Jap", "S1", "S2", "S3", "S4", "S6","S9",
               "S10", "S11", "S13", "S14", "S15", "S16", "S17", "S18", "S19",
"S20", "Tib")
native.coords=Pparva.sites[which(Pparva.sites$Pop %in% native.names),]
invasive.names=c("Aus", "Bel", "Bul1", "Bul2", "Hun", "Ira", "Ita", "Pol",
"Spa", "Tur", "UK")
invasive.coords=Pparva.sites[which(Pparva.sites$Pop %in% invasive.names),]

#--------------------------------
# GENIND FOR NATIVE & INVASIVE
# Create native and invasise genind
Native=Pparva[Pparva$pop %in% native.names]
Invasive=Pparva[Pparva$pop %in% invasive.names]

#============================================================
# SAMPLED SITES
#------------------------------------------------------------
# WORLDWIDE MAP OF SAMPLED SITES FOR M&M (ALL IN ONE MAP)
# Download database of river network at scale 50
# rivers50=ne_download(scale = 50, type = 'rivers_lake_centerlines', category
= 'physical')
# save(rivers50,file="Data/rivers50.Rda")
load("Data/rivers50.Rda")

map("world", xlim=c(-15,150), ylim=c(20,59), col="gray90",  fill=TRUE) # Plot
the map of European area
# Nomenclature
north.arrow(149,56,1, lab="N", lab.pos="above")
scalebar(c(135,20),1000, bg=NA, border=NA, division.cex=2)
points(invasive.coords$X, invasive.coords$Y, pch = 16, cex = 1,col="grey49")
# Add points to the map
# Labels of sampled sites
# Jitter the 4rth label
text(invasive.coords$X, y = invasive.coords$Y, invasive.coords$Pop,
     pos = c(rep(4,3), 1, rep(4,7)), offset = 1, cex = 1, font = 2)
points(native.coords$X, native.coords$Y, pch = 16, cex = 1,col="black") # Add
points to the map
points(native.coords$X[c(11,13,18)], native.coords$Y[c(11,13,18)], pch = 16,
cex = 1,col="grey49") #\ Add points to the map
# Labels of sampled sites
# Jitter the 13rd label
text(native.coords$X, y = native.coords$Y, native.coords$Pop,
     pos = c(rep(4,12), 1, rep(4,5)), offset = 1, cex = 1, font = 2)
```

Populations summary statistics

It was important to describe populations/locations and estimate summary statistics. We assessed that genetic diversity was similar in the native and invasive ranges, despite a loss of private alleles in the invasive range.

```
load(paste(datadir,"/Pparva.3000.Rda",sep=""))
# convert the genind to a hierfstat object
Pparva.hier=genind2hierfstat(Pparva)

#============================================================
# Basic population genetics statistics
#------------------------------------------------------------
# Compute a table with 6 predictors (columns) on 28 populations (rows):
# N – number of sampled individuals
# AR –allelic richness
# HE – expected heterozygosity
# HO – observed heterozygosity
# Fst
# Fis

# Barker et al. 2017 described its RAD seq data set like that:
# N, number of individuals analysed; P, proportion of variable loci; number
of private alleles (Priv.); NAR, mean allelic richness corrected
# for uneven sample size; NPAR, mean private allelic richness corrected for
uneven sample size; HO, observed heterozygosity for
# polymorphic loci; p, nucleotide diversity. Standard errors (SE) for NAR,
NPAR, HO and p are indicated in parentheses.

# Hohenlohe et al. 2011 described He and Hobs for SNP (p.3):
# At each putative
# locus, we calculated expected heterozygosity as Hexp =
#   1)Pni(ni ) 1) / n(n ) 1), where ni is the count of allele i in
# the sample and n = Pni, and observed heterozygosity
# Hobs as the proportion of individuals that appear
# heterozygous at the locus. We also calculated FIS = 1 )
# (Hobs /Hexp), which provides a measure of the deviation
# of genotype frequencies (i.e. observed heterozygosity)
# from Hardy–Weinberg proportions.
basic.genestats=as.data.frame(matrix(nrow = 29, ncol = 14))
colnames(basic.genestats)=c("Population","Range","N","Allelic_Richness","Priv
ate","Monomorphic_Loci","HE","HE.sem","HO","HO.sem","Fst","Fst.sem","Fis","Fi
s.sem")
# Get population names
basic.genestats$Population=unique(Pparva$pop)

#------------------------------------------------------------
# Range – Asia or Europe
basic.genestats$Range=c(rep("Europe",7),"Asia","Europe",rep("Asia",16),"Europ
```

```
e","Asia",rep("Europe",2))
#-------------------------------------------------------
# N – number of sampled individuals
basic.genestats$N=table(Pparva.hier$pop)
#-------------------------------------------------------
# AR - mean Allelic Richness per population, corrected for sample size
for (i in 1:29) {
  print(i)

basic.genestats$Allelic_Richness[i]=round(mean(allelic.richness(subset(Pparva
.hier, pop==basic.genestats$Population[i]))$Ar,na.rm=TRUE),digits=3)
}
#-------------------------------------------------------
# NUMBER OF PRIVATE ALLELES PER POPULATION
# basic.genestats$Private=apply(private_alleles(Pparva),1,sum)
basic.genestats$Private=apply(private_alleles(Pparva,form = alleles ~ .,
report = "table",
                                              level = "population"),1,sum)
#-------------------------------------------------------
# PROPORTION OF LOCI WITH NULL GENE DIVERSITY (I.E. MONOMORPHIC LOCI) --> Hs
basic.genestats$Monomorphic_Loci=round(apply(basic.stats(Pparva.hier)$Hs,2,fu
nction(x) sum(x==0,na.rm=TRUE)/length(x)),digits=3)
#-------------------------------------------------------
# HE – mean expected heterozygosity across polymorphic loci +- s.e.m.
#basic.genestats$HE=round(Hs(Pparva),digits=3)
# Bootstrap of HE - random resampling of individuals within populations
n.pop=seppop(Pparva)
for (i in 1:29) {
  cat("-------------------------\npop",i,"\n-------------------------\n")
  boots=numeric(100)
  for (j in 1:100){
    cat("pop",i,"boot",j,"\n")
    temp=c()
    subs=n.pop[[i]]
    temp=summary(subs[sample(1:nrow(n.pop[[i]]@tab),replace=TRUE),])$He
    boots[j]=mean(temp[temp>0],na.rm=TRUE)
    #
boots[j]=mean(summary(Pparva[which(Pparva$pop==basic.genestats$Population[i])
,sample(1:4216,replace=TRUE)])$He)
    rm(temp)
  }
  basic.genestats$HE[i]=round(mean(boots),digits=3)
  basic.genestats$HE.sem[i]=round(sd(boots),digits=3)
  print(round(mean(boots),digits=3))
  print(round(sd(boots),digits=3))
}

barplot(basic.genestats$HE)
#-------------------------------------------------------
# HO – mean observed individual heterozygosity across polymorphic loci +-
```

```
s.e.m.
n.pop=seppop(Pparva)
# (basic.genestats$HO=round(do.call("c", lapply(n.pop, function(x)
mean(summary(x)$Hobs,na.rm=TRUE))),digits=3)) # Compute mean Hobs for each
pop

# Bootstrap of Ho - random resampling individuals within populations
for (i in 1:29) {
  cat("-------------------------\npop",i,"\n-------------------------\n")
  boots=numeric(100)
  for (j in 1:100){
    cat("pop",i,"boot",j,"\n")
    temp=c()
    subs=n.pop[[i]]
    temp=summary(subs[sample(1:nrow(n.pop[[i]]@tab),replace=TRUE),])$Hobs
    boots[j]=mean(temp[temp>0],na.rm=TRUE)
    #
boots[j]=mean(summary(Pparva[which(Pparva$pop==basic.genestats$Population[i])
,sample(1:4216,replace=TRUE)])$He)
    rm(temp)
  }
  basic.genestats$HO[i]=round(mean(boots),digits=3)
  basic.genestats$HO.sem[i]=round(sd(boots),digits=3)
}

barplot(basic.genestats$HO)

#-----------------------------------------------------------
# Weir and Cockerham Fst per population  +- s.e.m.
# Compute Weir and Cockerham Fstat
# As the mean Fst per population in:
# - intercontinental range, to compare native and invasive populations in a
single metapopulation
# - within continental range (Asia OR Europe) for estimating differentiation
with geographical neighbours
(Pparva.pairwiseWC=pairwise.WCfst(Pparva.hier))
basic.genestats$Fst=round(apply(Pparva.pairwiseWC,2, function(x) mean(x,
na.rm=TRUE)),digits=3) # Mean Fst
# Bootstrap of Fst s.e.m. with 1,000 bootstraps, by randomly resampling Fst
from pairwise Fst list for a given population
sem=c()
for (j in 1:29) {
  print(j)
  boots=numeric(1000)
  for (i in 1:1000){
    boots[i]=mean(sample(Pparva.pairwiseWC,replace=TRUE),na.rm=TRUE)
  }
  sem[j]=sd(boots)# Fst s.e.m
}
basic.genestats$Fst.sem=round(sem,digits=3)
```

```
# # Asia
# Pparva.hier.native=genind2hierfstat(native)
# (Pparva.pairwiseWC.native=pairwise.WCfst(Pparva.hier.native))
#
# # Europe
# Pparva.hier.invasive=genind2hierfstat(invasive)
# (Pparva.pairwiseWC.invasive=pairwise.WCfst(Pparva.hier.invasive))

#-------------------------------------------------------
# Fis per population +- s.e.m.
# with 1000 bootstraps
boot=boot.ppfis(dat=Pparva.hier,nboot=1000,quant=c(0.025,0.975),diploid=TRUE)
basic.genestats$Fis=round(apply(boot$fis.ci,1, mean),digits=3) # Mean
basic.genestats$Fis.sem=round((as.numeric(unlist(boot$fis.ci[2]))-
as.numeric(unlist(boot$fis.ci[1])))/3.92,digits=3) # Standard error

########
# Sort populations by range (first Asia, the Europe)
basic.genestats=basic.genestats[c(8,10,20,22:25,11:19,21,27,1:7,9,26,28,29),]
# And then export table:
save(basic.genestats,file="Data/basic.genestats.Rda")
write.csv(basic.genestats,file="Tables/basic.genestats.csv",row.names =
FALSE)
load(basic.genestats,file="Data/basic.genestats.Rda")

#-------------------------------------------------------------
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# HIERARCHICAL FST
# Fst among sampled sites, among demes (DAPC and STRUCTURE results), among
continents and isolated populations (Jap, Ira)
(Pparva.wc=wc(Pparva.hier)) # Mean Fst and Fis
fstat(Pparva)
Fst(as.loci(Pparva))
(Gtest=gstat.randtest(Pparva,nsim=999))
plot(Gtest)
# Compute Weir & Cockerham Fst global pairwise
(Pparva.pairwiseWC=pairwise.WCfst(Pparva.hier))
# Compute Weir & Cockerham Fst native pairwise
(Pparva.pairwiseWC.native=pairwise.WCfst(Pparva.hier.native))
# Compute Weir & Cockerham Fst invasive pairwise
(Pparva.pairwiseWC.invasive=pairwise.WCfst(Pparva.hier.invasive))

#============================================================
# Pairwise Fst among demes:
# Pairwise Fst per demes (Western Europe, Easter Europe, Iran, Japan, North
China, South China, North Central China, South Central China, Continental
China, Tibet)
# Weir & Cockerham pairwise Fst, significant ones in bold
```

```
# Putative demes were:
#   in ASIA
# - North China: S13 S14 S15 S17
# - North Central China: S1 S2 S3 S16
# - Admixed coastal zone: S3 S4 S6
# - Admixed continental zone: S10 S11
# - South China: S9 S18 S19 S20
# - Japan
# - Tibet
#   in EUROPE
# - Western Europe: AUS BEL HUN ITA POL SPA UK
# - Eastern Europe: BUL1 BUL2 TUR
# - Iran

Pparva.hier=genind2hierfstat(Pparva)
Pparva.demes=Pparva.hier
# Remove the undesirable populations (outlier, admixed zone)
# Pparva.demes=Pparva.demes[-which(Pparva.demes$pop %in%
c("Mor","Slo","Tai","S5","S7")),]
# Assign populations to demes in the genind object (genind object of 3,000
loci)
Pparva.demes$pop=as.character(Pparva.demes$pop)
Pparva.demes$pop[Pparva.demes$pop %in%
c("Aus","Bel","Hun","Ita","Spa","Pol","UK")]="WesternEurope"
Pparva.demes$pop[Pparva.demes$pop %in%
c("Bul1","Bul2","Tur")]="EasternEurope"
Pparva.demes$pop[Pparva.demes$pop %in%
c("S1","S2","S3","S16")]="NorthCentralChina"
Pparva.demes$pop[Pparva.demes$pop %in%
c("S13","S14","S15","S17")]="NorthChina"
Pparva.demes$pop[Pparva.demes$pop %in% c("S4","S6")]="AdmixedCoastalChina"
Pparva.demes$pop[Pparva.demes$pop %in%
c("S9","S18","S19","S20")]="SouthChina"
Pparva.demes$pop[Pparva.demes$pop %in%
c("S10","S11")]="AdmixedContinentalChina"

# Get pop names (as factors) in the right order
Pparva.demes$pop=as.factor(Pparva.demes$pop)
Pparva.demes$pop=factor(Pparva.demes$pop,levels=c("WesternEurope","EasternEur
ope","Ira","Jap","AdmixedContinentalChina","SouthChina","AdmixedCoastalChina"
,"NorthChina","NorthCentralChina","Tib"))
# Due to a bug in boot.ppfst, populations need to be encoded as numeric and
the data frame ordered by population number
v.pop=as.data.frame(Pparva.demes$pop) # save a data frame of population names
for the significance matrix
v.pop$num=as.numeric(Pparva.demes$pop)
v.pop=unique(v.pop)
write.table(v.pop,file="Tables/Pairwise.Fst.labels.csv") # Save labels of pop
names with corresponding numeric label
```

```
Pparva.demes$pop=as.numeric(Pparva.demes$pop)
setorder(Pparva.demes,pop)
# Compute pairwise Fst (Weir & Cockerham's Fst) between demes
(Pparva.pairwiseWC=pairwise.WCfst(Pparva.demes))
(Pparva.pairwiseWC=round(Pparva.pairwiseWC,digits=3))
# Keep only the upper triangular, lower triangular will be filled with
pairwise Nei's distances
Pparva.pairwiseWC[lower.tri(Pparva.pairwiseWC, diag = TRUE)]=0
# And now compute the Nei's distances
(Pparva.pairwiseNei=pairwise.neifst(Pparva.demes))
(Pparva.pairwiseNei=round(Pparva.pairwiseNei,digits=3))
Pparva.pairwiseNei[upper.tri(Pparva.pairwiseNei, diag = TRUE)]=0
# Combine the two triangular matrices of WC Fst and Nei's Fst
Pparva.pairwiseWC[is.na(Pparva.pairwiseWC)]=0
Pparva.pairwiseNei[is.na(Pparva.pairwiseNei)]=0
(mat=Pparva.pairwiseWC + Pparva.pairwiseNei)
write.table(mat,file="Tables/Pairwise.Fst.csv")
rm(mat)

# Compute significance of the Fst by bootstrap
bootppfst=boot.ppfst(dat=Pparva.demes,nboot=1000,quant=c(0.005,0.995),dig=3)
# CI at 1%
# make a matrix with upper limit in top triangular matrix and lower limit in
bottom triangular matrix
# Fst were not significantly different from 0 when the lower CI was negative
or equal to 0
(mat.ll=t(bootppfst$ll))
(mat.ul=bootppfst$ul)
mat.ll[is.na(mat.ll)]=0
mat.ul[is.na(mat.ul)]=0
(mat=mat.ll + mat.ul)
write.table(mat,file="Tables/Pairwise.Fst.significance.csv")
rm(mat)
# All pairwise Fst were significant (higher than expected just by chance).
Putative demes seemed well defined
# All pairwise comparisons are significantly differentiated (P < 0.01)

#-------------------------------------------------------------
# Neighbour-Joining tree estimated on pairwise distances
# with 'ape' package
?nj() # require a distance matrix
# NJ tree with Weir & Cockerham distances
# and populations in order:
#
"WesternEurope","EasternEurope","Ira","Jap","AdmixedContinentalChina","SouthC
hina",
#    "AdmixedCoastalChina","NorthChina","NorthCentralChina","Tib"
(Pparva.pairwiseWC=pairwise.WCfst(Pparva.demes))
colnames(Pparva.pairwiseWC)=c("WesternEurope","EasternEurope","Ira","Jap","Ad
mixedContinentalChina","SouthChina","AdmixedCoastalChina","NorthChina","North
```

```
CentralChina","Tib")
rownames(Pparva.pairwiseWC)=c("WesternEurope","EasternEurope","Ira","Jap","Ad
mixedContinentalChina","SouthChina","AdmixedCoastalChina","NorthChina","North
CentralChina","Tib")
nj.fst=nj(as.dist(Pparva.pairwiseWC))
plot(nj.fst, main="NJ tree for Weir & Cockerham's distances",
type="phylogram")
plot(nj.fst, main="NJ tree for Weir & Cockerham's distances",
type="unrooted")

# NJ tree with Nei's distances
(Pparva.pairwiseNei=pairwise.neifst(Pparva.demes))
colnames(Pparva.pairwiseNei)=c("WesternEurope","EasternEurope","Ira","Jap","A
dmixedContinentalChina","SouthChina","AdmixedCoastalChina","NorthChina","Nort
hCentralChina","Tib")
rownames(Pparva.pairwiseNei)=c("WesternEurope","EasternEurope","Ira","Jap","A
dmixedContinentalChina","SouthChina","AdmixedCoastalChina","NorthChina","Nort
hCentralChina","Tib")
nj.Nei=nj(as.dist(Pparva.pairwiseNei))
plot(nj.Nei, main="NJ tree for Nei's distances", type="phylogram")
plot(nj.Nei, main="NJ tree for Nei's distances", type="unrooted")

# Heterozygosities weighted by group sizes
(Pparva.pairwiseNei=pairwise.fst(Pparva,res.type="matrix"))
save(Pparva.pairwiseNei,file=file(paste(datadir,"/Pparva.pairwiseNei.Rda",sep
="")))
load(paste(datadir,"/Pparva.pairwiseNei.Rda",sep=""))
# Summary statistics of pairwise Nei's distance
mean(Pparva.pairwiseNei) # Mean
sd(Pparva.pairwiseNei)/sqrt(nrow(Pparva.pairwiseNei)) # standard error of the
mean
max(Pparva.pairwiseNei)
min(Pparva.pairwiseNei)
# Mean pairwise Fst (Nei's Fst) was of  (S.E.M = ), in range -

pairNei.tree=nj(Pparva.pairwiseNei)
# Draw a tree plot of genetic distances (Fst) between populations
plot(pairNei.tree, type="unr", tip.col=funky(17)[-17], font=2)
annot  =  round(pairNei.tree$edge.length,2)
edgelabels(annot[annot>0], which(annot>0), frame="n")
add.scale.bar()
# Draw a boxplot of Fst per population
temp=Pparva.pairwiseNei
diag(temp)=NA
boxplot(temp, col=funky(17)[-17], xlab="Population", ylab="Fst")

# basic statistics (hierfstat) per loci and overall
(Pparva.stat=basic.stats(Pparva.hier))
# other basic statistics (allelic richness) not included in the basic.stats
function (hierfstat)
```

```
# Allelic richness per loci: number of alleles corrected for sample size
#(Pparva.allrich=(as.data.frame(allelic.richness(Pparva.hier))))
```

Isolation-by-distance

Isolation by distance at a continental scale could indicate a dispersal limited by distance

First, IBD was run on native and invasive area separately…

```
#######################################
#         ASIA
#######################################

#-------------------------------------
## IBD in Asia (native area)
#-------------------------------------
# Mantel test between genetic distances Fst/1-Fst (based on pairwise Fst
values (hierfstat)) and log of Euclidean distances (raster)
# First, we do a matrix of pairwise Fst values
(Pparva.pfst.Native=pairwise.fst(Pparva[which(Pparva@pop %in%
labelPops[c(8,10:25,27),1])],res.type="matrix"))
# Correlation plot between pairwise Fst values (hierfstat)
# lower Fst are darker
heatmap(Pparva.pfst.Native)
# Matrix of pairwise correlations converted to distance matrix type (same
numerical values)
Pparva.pfst.Native=as.dist(Pparva.pfst.Native)
# Transform negative Fst in null values
for (i in 1:length(Pparva.pfst.Native)) {
  if (Pparva.pfst.Native[i]<0) {
    Pparva.pfst.Native[i]=0
  }
}
# Then, we compute the matrix of geographical distance
(Dgeo.Native=as.dist(pointDistance(Pparva.sites[c(8,11:24,26,28,32),2:3],lonl
at=T)))
# To finally do Mantel test on native populations (Asian populations)
(IBD.Native.geo=mantel.randtest((Pparva.pfst.Native/(1-
Pparva.pfst.Native)),log(Dgeo.Native),nrepet = 9999))
# P = 0.037, there was a significant isolation by distance in Asia
# as illustrated by a simple plot
df=data.frame(Distance=as.vector(log(Dgeo.Native)),GeneDistance=as.vector(Ppa
rva.pfst.Native/(1-Pparva.pfst.Native)))
plot(df, pch=19, cex=1, col="black", xlab="Geographic distances (logarithmic
scale))",
     ylab="Genetic distance (FST/(1-FST))", cex.lab=1.5, main="2-dimensional
IBD in Asian populations")
mod=lm(GeneDistance~Distance,data=df)
abline(lm(GeneDistance~Distance,data=df))
```

```r
mod$coefficients[1]
mod$coefficients[2]

########################################
#         EUROPE
########################################

#-------------------------------------
## IBD in Europe (invasive area)
#-------------------------------------
# Mantel test between pairwise Fst values (hierfstat) and Euclidean distances
(raster)
# First, we do a matrix of pairwise Fst values
(Pparva.pfst.Invasive=pairwise.fst(Pparva[which(Pparva@pop %in%
labelPops[c(1:7,9,26,28,29),1])],res.type="matrix"))
# Correlation plot between pairwise Fst values (hierfstat)
# lower Fst are darker
heatmap(Pparva.pfst.Invasive)
# Matrix of pairwise correlations converted to distance matrix type (same
numerical values)
Pparva.pfst.Invasive=as.dist(Pparva.pfst.Invasive)
# Transform negative Fst in null values
for (i in 1:length(Pparva.pfst.Invasive)) {
  if (Pparva.pfst.Invasive[i]<0) {
    Pparva.pfst.Invasive[i]=0
  }
}
# Then, we compute the matrix of geographical distance 8,11:24,26,28,32
(Dgeo.Invasive=as.dist(pointDistance(Pparva.sites[c(1:7,10,30,33,34),2:3],lon
lat=T)))
# To finally do Mantel test on invasive populations (European populations)
(IBD.Invasive.geo=mantel.randtest((Pparva.pfst.Invasive/(1-
Pparva.pfst.Invasive)),log(Dgeo.Invasive),nrepet = 9999))
# P = 0.12, there is no significant isolation by distance
# There is no sign of isolation by distance, as illustrated by a simple plot
df2=data.frame(Distance=as.vector(log(Dgeo.Invasive)),GeneDistance=as.vector(
Pparva.pfst.Invasive/(1-Pparva.pfst.Invasive)))
plot(df2, pch=19, cex=1, col="black", xlab="Geographic distances (logarithmic
scale))",
     ylab="Genetic distance (FST/(1-FST))", cex.lab=1.5, main="2-dimensional
IBD in European populations")
mod=lm(GeneDistance~Distance,data=df2)
abline(lm(GeneDistance~Distance,data=df2))
mod$coefficients[1]
mod$coefficients[2]
# The plot revealed 2 clouds of points, one strange pattern of 0 genetic
distance and
# a cloud that looked like a linear relationship of non-zero values for
genetic distance,
# suggesting a cluster of IBD and a cluster of "similar" populations not
```

```
influenced by geographic distance

#-------------------------------------
# IBD per demes in invasive area: separate analysis of Western and Eastern
Europe
#-------------------------------------
############## Western Europe (UK, Bel, Spa, Aus, Ita, Pol, Hun)
# First, we do a matrix of pairwise Fst values
(Pparva.pfst.WestEurope=pairwise.fst(Pparva[which(Pparva@pop %in%
labelPops[c(1,2,5,7,9,26,29),1])],res.type="matrix"))
# Correlation plot between pairwise Fst values (hierfstat)
# lower Fst are darker
heatmap(Pparva.pfst.WestEurope)
# Matrix of pairwise correlations converted to distance matrix type (same
numerical values)
Pparva.pfst.WestEurope=as.dist(Pparva.pfst.WestEurope)
# Transform negative Fst in null values
for (i in 1:length(Pparva.pfst.WestEurope)) {
  if (Pparva.pfst.WestEurope[i]<0) {
    Pparva.pfst.WestEurope[i]=0
  }
}
# Then, we compute the matrix of geographical distance
(Dgeo.WestEurope=as.dist(pointDistance(Pparva.sites[c(1,2,5,7,9,26,29),2:3],l
onlat=T)))
# To finally do Mantel test on invasive populations (European populations)
(IBD.WestEurope.geo=mantel.randtest((Pparva.pfst.WestEurope/(1-
Pparva.pfst.WestEurope)),log(Dgeo.WestEurope),nrepet = 9999))
# P = 0.47, there is no significant isolation by distance
# There is no sign of isolation by distance, as illustrated by a simple plot
df2=data.frame(Distance=as.vector(log(Dgeo.WestEurope)),GeneDistance=as.vecto
r(Pparva.pfst.WestEurope/(1-Pparva.pfst.WestEurope)))
plot(df2, pch=19, cex=1, col="black", xlab="Geographic distances (logarithmic
scale))",
     ylab="Genetic distance (FST/(1-FST))", cex.lab=1.5, main="2-dimensional
IBD in Western Europe populations")
mod=lm(GeneDistance~Distance,data=df2)
abline(lm(GeneDistance~Distance,data=df2))
mod$coefficients[1]
mod$coefficients[2]
##### Eastern Europe
# Not enough populations (N = 3): Bul1, Bul2, Turkey
```

Nonetheless, IBD was assessed on euclidian distances between sites, not on distances through the river network. Hence we should assessed IBD for least-cost paths within/between river basins, but it was impossible with our sampling (continental scale imposed very large SIG datasets) and we didn't find any estimates of dispersal cost between river basins (i.e. human-mediated dispersal rates). Moreover, this approach is more suitable in a landscape genetic framework than in a global phylogeography study.

GENETIC CLUSTERING - Discriminant Analyis of Principal Component (DAPC)

DAPC is a multivariate analysis implemented in 'adegenet'. It is a K-mean clustering method, performing a discriminant analysis of the results of a first principal component analysi on genetic data. The main interest of DAPC in clustering purposes is that it increases the inter-group variance while minimizing the intra-group variance, hence increasing its power to separate groups.

```
#=========================================================
#                          Clustering of Native Populations
#=========================================================
x = c("Jap", "S1", "S2", "S3", "S4", "S6", "S9", "S10", "S11", "S13", "S14",
"S15", "S16", "S17", "S18", "S19", "S20", "Tib")
i = 1
y = which(Pparva$pop==x[i])
# create an index y of all individuals of Asian populations
for (i in 2:length(x)){ y = c(y, which(Pparva$pop==x[i])) }
rm(i)
rm(x)
native = Pparva
# Retain only individuals from Asian populations
native$tab = native$tab[y,]
native$pop = native$pop[y]
native$ploidy = native$ploidy[y]
rm(y)
save(native,file="Data/DAPC/native.Rda")
load(file="Data/DAPC/native.Rda")

#=========================================================
#  Preliminary Clustering of Data -- k Means
#---------------------------------------------------------
# run interactively to select the number of clusters (based on minimizing
BIC) and to estimate variance explained
# number of retained PCs will be set to maximum allowed for model "stability"
(< N/3; 100< 300/3)
# 100 retained PCs appears to exlain 75-80% of cumulative variance
# BIC minimum occurs at k=7
#!# HOWEVER, true inflection point may be between 6 and 8 based on graphics
k.native = find.clusters(native, max.n=19)
# Compare if clusters corresponded to observed groups (i.e. sampled
locations)
table.value(table(pop(native), k.native$grp), col.lab=paste("inf", 1:19),
            row.lab=paste("ori", 1:19))
# Observed groups globally corresponded to inferred groups

#---------------------------------------------------------
#  Initial Discriminant Analysis of Principal Components (DAPC)
#---------------------------------------------------------
```

```
# again set to retain maximum numbers of PCs & DA's
# again run interactively to examine scree-plot of F-statistics
dapc.native = dapc(native, pop=k.native$grp, n.pca=100,n.da=7)
#-----------------------------------------------------------
compoplot(dapc.native, posi="top", col=c(rainbow(6,s=1,v=1),
rainbow(6,s=0.5,v=0.75)))
dapc.native$grp
compoplot(dapc.native, posi="top", col=c(rainbow(6,s=1,v=1),
rainbow(6,s=0.5,v=0.75)),show.lab=TRUE,lab=dapc.native$grp)
# no individual show evidence of admixture
# Japanese indiv. globally clustered together
# Tibetan and some S10-S11 indiv. clustered together
# Clusters seemed to correspond to existing sampled populations

#============================================================
# Alternate Strategy for Clustering of Data:  No. PCs Retained to Achieve Set
Cumulative Variance Explained
#-----------------------------------------------------------
#!# over-fitting?
# perhaps retain fewer PCs initially
# use cumulative variance as a threshold
#-----------------------------------------------------------
# 75% of variance
#-----------------------------------------------------------
k.native = find.clusters(native, max.n=19, pca.select="percVar", perc.pca=75)
# run interactively to select the number of clusters
# BIC minimum occurs at k=9
#!# HOWEVER, suspect this is also an over-fitting given previous attempts,
plus:
# 1 "uptick" at 10
# near zero slope between 6 & 8

#-----------------------------------------------------------
# 70% of variance
#-----------------------------------------------------------
k.native = find.clusters(native, max.n=19, pca.select="percVar", perc.pca=70)
# run interactively to select the number of clusters
# BIC minimum occurs at k=8
# 1 "downtick" at K=8
# checking k=8
# same pattern as before re: scree plot of F-stats
dapc.native = dapc(native, pop=k.native$grp, pca.select="percVar",
perc.pca=70, n.da=7)
compoplot(dapc.native, posi="top", col=c(rainbow(6,s=1,v=1),
rainbow(6,s=0.5,v=0.75)))

#!#!# NB! there is NO consistency between clusters if multiples iterations
are compared
# how then does one evaluate which model is truly the best representation of
the data?!?
```

```
# perhaps I need to do so iteratively

#=========================================================
#  Alternate Strategy for Clustering of Data:  Using a "best fit" Selection
Criterion (in addition to cum. var.)
#---------------------------------------------------------
test1.k = list()
for (i in 1:100){ print(paste("Iteration No.", i, "of 100"))
  k.native = find.clusters(native, max.n=19, pca.select="percVar",
perc.pca=70, choose.n.clust=F, criterion="goodfit", n.iter=1000,
n.start=sample(10:21,1))
  test1.k[[i]] = k.native
  rm(k.native) }
rm(i)

i=1
x = length(test1.k[[i]]$size)
for(i in 2:100){ x = c(x, length(test1.k[[i]]$size))}
rm(i)
table(x)
hist(x)
# 100 iterations is enough
# Clear score of 97% for K=5
# BUT "Goodfit" criterion tended to favor a small K, that was not informative
in our case
# Lack of resolution in native populations

# Same iterative test but with the criterion "min" of BIC
# As BIC~Nb of clusters decreases up to a plateau and stop decreasing,
# in an exponential negative shape like an island model (see details of
?find.clusters())
test1b.k = list()
for (i in 1:100){ print(paste("Iteration No.", i, "of 100"))
  k.native = find.clusters(native, max.n=19, pca.select="percVar",
perc.pca=70, choose.n.clust=F, criterion="min", n.iter=1000,
n.start=sample(10:21,1))
  test1b.k[[i]] = k.native
  rm(k.native) }
rm(i)

i=1
x = length(test1b.k[[i]]$size)
for(i in 2:100){ x = c(x, length(test1b.k[[i]]$size))}
rm(i)
table(x)
hist(x)
# "min" criterion tended to over-estimate K
# Same iterative test but with the criterion "diffNgroup" of BIC
test1.k = list()
for (i in 1:100){ print(paste("Iteration No.", i, "of 100"))
```

```
  k.native = find.clusters(native, max.n=19, pca.select="percVar",
perc.pca=70, choose.n.clust=F, criterion="diffNgroup", n.iter=1000,
n.start=sample(10:21,1))
  test1.k[[i]] = k.native
  rm(k.native) }
rm(i)

i=1
x = length(test1.k[[i]]$size)
for(i in 2:100){ x = c(x, length(test1.k[[i]]$size))}
rm(i)
table(x)
hist(x)
# This criterion gave K=4 as the most probable K (100/100 iterations)

#-----------------------------------------------------------
# probably best to run at least 1,000 total iterations
#-----------------------------------------------------------
for (i in 1:1000){ print(paste("Iteration No.", i, "of 1000"))
  k.native = find.clusters(native, max.n=19, pca.select="percVar",
perc.pca=70, choose.n.clust=F, criterion="goodfit", n.iter=1000,
n.start=sample(10:21,1))
  test1.k[[i]] = k.native
  rm(k.native) }
rm(i)

i=1
x = length(test1.k[[i]]$size)
for(i in 1:1000){ x = c(x, length(test1.k[[i]]$size))}
rm(i)
table(x)
hist(x)
# 5 clusters (n=958/1000)
for (i in 1:1000){ print(paste("Iteration No.", i, "of 1000"))
  k.native = find.clusters(native, max.n=19, pca.select="percVar",
perc.pca=70, choose.n.clust=F, criterion="min", n.iter=1000,
n.start=sample(10:21,1))
  test1b.k[[i]] = k.native
  rm(k.native) }
rm(i)

i=1
x = length(test1b.k[[i]]$size)
for(i in 1:1000){ x = c(x, length(test1b.k[[i]]$size))}
rm(i)
table(x)
hist(x)
# K=9-10 were favored for minimum criterion (n=303 for K=9 & n=390 for K=10 ;
n=251 for K=11)
# BUT, perhaps should test for over-fitting
```

```
#-------------------------------------------------------
###  Step 2 in Strategy:  Resolving the Over-fitting Problem
#-------------------------------------------------------
# cross-validation to estimate the optimal number of retained PCs
# given size of dataset, consider 100 as an upper limit
# maximum "allowed" for model stability (< N/3; 100< 259/3)
mat = tab(native, NA.method="mean")
# 80 PCs to retain to discriminate between the 19 sampling sites, although it
seemed that at least 20 is sufficient
xval = xvalDapc(mat, grp=native@pop, n.pca=seq(5,85,5))
# 20 was enough to discriminate amongst k=5 clusters
xval = xvalDapc(mat, grp=test1.k[[10]]$grp, n.pca=seq(5,85,5))
#!# NB
# NONE of the plots displayed a nice peaked pattern
#-------------------------------------------------------
# a-score optimization
# Linear decline of a-score from 0 to 85
# suggests far fewer (<10) PCs to avoid overfitting
dapc.1 = dapc(native, pop=test1.k[[10]]$grp, pca.select="percVar",
perc.pca=70, n.da=6)
scatter(dapc.1)
optim.a.score(dapc.1, n.pca=seq(5,85,5))
dapc.2 = dapc(native, pop=test1.k[[10]]$grp, pca.select="percVar", n.pca=85,
n.da=6)
scatter(dapc.2)
dapc.3 = dapc(native, pop=test1.k[[10]]$grp, pca.select="percVar", n.pca=10,
n.da=6)
scatter(dapc.3)
# scatterplot shows greater degree of separation with 85 PCs than with 10 PCs

#=======================================================
#  Working Clusters
#-------------------------------------------------------
# K=5 was a parcimonious number of K and the lowest BIC value
# BIC for K=6 was very close to the BIC value of K=5, even if rare, and
clusters were more consistent with geography
# But clusters with K=5 did not gave us useful information about source
populations
# We looked for more clusters, for a finer structuring resolution
# First "naive" graphical approach tended to favor K=8-9; Good, as we
searched for a fine-scale genetic structure of source populations
# Selection by the "min" criterion favored K=9-10, but tended to produce
overclustering with frequent admixture in a large pan-asian region
# So we chose to use K=7, producing more clusters and more consistent ones
with the geography (Tib, Jap)
nclust=9
k.native = find.clusters(native, max.n=19,n.pca=100,n.clust=nclust)
k.native$size
# save(k.native,file="Data/DAPC/k.native.rda")
```

```
xval = xvalDapc(mat, grp=k.native$grp, n.pca=seq(5,100,5))
dapc.native = dapc(native, pop=k.native$grp, n.pca=40,n.da=(nclust-1))
# save(dapc.native,file="Data/DAPC/dapc.native.rda")
# load("Data/DAPC/dapc.native.rda")
scatter(dapc.native)
compoplot(dapc.native, posi="top", col=c(rainbow(6,s=1,v=1),
rainbow(6,s=0.5,v=0.75)))

#-----------------------------------------------------------
# Study of admixture between inferred clusters
#-----------------------------------------------------------
### Distribution of posterior probabilities across clusters
hist(dapc.native$posterior,breaks=50)
# Posterior probabilities distribution show two extreme values, close to 0
and 1
# Suggesting no admixture between clusters
assignplot(dapc.native, subset=1:300)
# No evidence of admixed individuals with DAPC

#-----------------------------------------------------------
# Identifying Members of Each Cluster
#-----------------------------------------------------------
native.cluster.df = data.frame(Cluster=factor(), Ind.Site=factor(),
Ind=factor())
for(g in 1:7){ x = sort(names(k.native$grp)[which(k.native$grp==g)])
for(i in 1:length(x)){ temp.df = data.frame(Cluster=paste("Cluster", g,
sep="_"), Ind.Site=strsplit(x[i], "_")[[1]][1], Ind=x[i])
native.cluster.df = rbind(native.cluster.df, temp.df)
rm(i)
rm(temp.df)}
rm(x) }
rm(g)

#-----------------------------------------------------------
# Create a data frame with all populations allocated to each cluster
unique(native.cluster.df[,1:2])
write.csv(unique(native.cluster.df[,1:2]), "Tables/DAPC/native_cluster.csv",
row.names=F,quote=F)

#-----------------------------------------------------------
# To look for a given cluster
subset((unique(native.cluster.df[,1:2])), Cluster=="Cluster_3")
barplot(table(subset(native.cluster.df, Cluster=="Cluster_3")$Ind.Site),
las=1)
barplot(sort(table(subset(native.cluster.df,
Cluster=="Cluster_3")$Ind.Site)), las=1)

#-----------------------------------------------------------
# Ou bien pour identifier chaque cluster ou se trouvent des echantillons
d'une site en particuliere
```

```
subset((unique(native.cluster.df[,1:2])), Ind.Site=="Jap")
x = c("Jap", "S1", "S2", "S3", "S4", "S6", "S9", "S10", "S11", "S13", "S14",
"S15", "S16", "S17", "S18", "S19", "S20", "Tib")
for (i in 1:length(x)){ print(i)
  print(subset((unique(native.cluster.df[,1:2])), Ind.Site==x[i])) }
rm(i)
rm(x)

#=========================================================
#  Using what we've learned from STRUCTURE (K=6), we investigated furthermore
K=6 with DAPC...
#---------------------------------------------------------
# Running 1,000 models of DAPC with the 'a priori' number of clusters
(estimated from STRUCTURE): K=6
test1c.k = list()
for (i in 1:1000){ print(paste("Iteration No.", i, "of 1000"))
  k.native = find.clusters(native, max.n=19, pca.select="percVar",
perc.pca=70, choose.n.clust=F, criterion="goodfit", n.iter=1000,
n.start=sample(10:21,1))
  test1c.k[[i]] = k.native
  rm(k.native) }
rm(i)
# Selection of the most probable K
i=1
x = length(test1c.k[[i]]$size)
for(i in 1:1000){ x = c(x, length(test1c.k[[i]]$size))}
rm(i)
table(x)
hist(x)
# Selection of the best model among 1,000 models
i=1
x=test1c.k[[i]]$stat
for(i in 2:1000){ x=c(x, test1c.k[[i]]$stat)}
rm(i)
table(x)
hist(x)
# Models with K=5
index.K5=(which(names(x)=="K=5"))
test1c.k[[index.K5]]
# Get a vector containing the BIC values for K=5
x=c()
for(i in index.K5){ x=c(x, test1c.k[[i]]$stat)}
x
# And then select the best model (lowest BIC value)
which(x==min(x)) # Identify the clusters with the lowest BIC for K=6 -> 2
models:
# Models with the lowest BIC value for K=5 were explored
# and model 383 was one of the best explicative model
test1c.k[[383]]$stat # lowest BIC for K=6 was here 832.6823
# Model 383 was our chosen model for DAPC in native populations assuming K=5
```

```
as the most probable number of genetic clusters
# Don't forget to save our 1,000 iterations for best model research
save(test1c.k,file="Data/DAPC/test1c.k")
load(file="Data/DAPC/test1c.k")
# BUT, perhaps should test for over-fitting

#-----------------------------------------------------------
#  Step 2 in Strategy:  Resolving the Over-fitting Problem
#-----------------------------------------------------------
# cross-validation to estimate the optimal number of retained PCs
# given size of dataset, consider 100 as an upper limit
# maximum "allowed" for model stability (< N/3; 100< 300/3)
mat=tab(native, NA.method="mean")
# 80 PCs to retain to discriminate between the 19 sampling sites, although it
seemed that at least 20 is sufficient
xval=xvalDapc(mat, grp=native@pop, n.pca=seq(5,100,5))
# Flat pattern of cross-validation with a slight maximum around 40 PCAs
# 20 was enough to discriminate amongst k=6 clusters for the chosen model
n?57, but no evidence of overfitting
xval=xvalDapc(mat, grp=test1c.k[[383]]$grp, n.pca=seq(5,100,5))
#!# NB
# NONE of the plots displayed a nice peaked pattern
#-----------------------------------------------------------
# a-score optimization
# Linear decline of a-score from 0 to 100 (300/3)
# suggests far fewer (<10) PCs to avoid overfitting
dapc.1 = dapc(native, pop=test1c.k[[383]]$grp, pca.select="percVar",
perc.pca=70, n.da=5)
scatter(dapc.1)
optim.a.score(dapc.1, n.pca=seq(5,100,5))
dapc.2 = dapc(native, pop=test1c.k[[383]]$grp, pca.select="percVar",
n.pca=85, n.da=5)
scatter(dapc.2)
dapc.3 = dapc(native, pop=test1c.k[[383]]$grp, pca.select="percVar",
n.pca=10, n.da=5)
scatter(dapc.3)
# scatterplot shows greater degree of separation with 85 PCs than with 10 PCs

#===========================================================
#  Working Clusters -- Last one, and definitive value of K=6
#-----------------------------------------------------------
# Given the a priori from STRUCTURE (clear assessment of K=6) and the rage of
probable values of K within 4-9 from DAPC under different criterion
# We retained K=6 and the best model was n?57
k.native=test1c.k[[383]]
k.native$size
save(k.native,file="Data/DAPC/k.native.rda")
xval = xvalDapc(mat, grp=k.native$grp, n.pca=seq(5,100,5))
dapc.native = dapc(native, pop=k.native$grp, n.pca=85,n.da=6)
save(dapc.native,file="Data/DAPC/dapc.native.rda")
```

```
load("Data/DAPC/dapc.native.rda")
scatter(dapc.native)
compoplot(dapc.native, posi="top", col=c(rainbow(6,s=1,v=1),
rainbow(6,s=0.5,v=0.75)))

#----------------------------------------------------------
# Study of admixture between inferred clusters
#----------------------------------------------------------
### Distribution of posterior probabilities across clusters
hist(dapc.native$posterior,breaks=50)
# Posterior probabilities distribution show two extreme values, close to 0
and 1
# Suggesting no admixture between clusters
assignplot(dapc.native, subset=1:300)
# No evidence of admixed individuals with DAPC

#----------------------------------------------------------
# Identifying Members of Each Cluster
#----------------------------------------------------------
native.cluster.df=data.frame(Cluster=factor(), Ind.Site=factor(),
Ind=factor())
for(g in 1:6){ x=sort(names(k.native$grp)[which(k.native$grp==g)])
for(i in 1:length(x)){ temp.df=data.frame(Cluster=paste("Cluster", g,
sep="_"), Ind.Site=strsplit(x[i], "_")[[1]][1], Ind=x[i])
native.cluster.df = rbind(native.cluster.df, temp.df)
rm(i)
rm(temp.df)}
rm(x) }
rm(g)
#----------------------------------------------------------
# Create a data frame with all populations allocated to each cluster
unique(native.cluster.df[,1:2])
write.csv(unique(native.cluster.df[,1:2]), "Tables/DAPC/native_cluster.csv",
row.names=F,quote=F)

#==========================================================
#                          Invasive Populations
#==========================================================
# will take multiple approaches:
# 1/ First look at genetic structure in invasive populations, to see if there
is genetic divergence
# and/or multiple genetic pools (e.g. suggesting multiple introductions, or
stepping-stone dispersal for colonization)
# perform a round of k-means clustering to look at relative divergence since
colonization
# divergence from ancestral groups
# divergence w/in the invasive range
# 2/ WORLDWIDE ANALYSIS
# individuals from invasive populations have been assigned to the native
clusters
```

```
# perhaps will provide some insight into hypotheses to be tested via ABC ->
source population(s)
x=c("Aus", "Bel", "Bul1", "Bul2", "Hun", "Ira", "Ita", "Pol", "Spa", "Tur",
"UK")
i=1
y=which(Pparva$pop==x[i])
# create an index y of all individuals of Asian populations
for (i in 2:length(x)){ y=c(y, which(Pparva$pop==x[i])) }
rm(i)
rm(x)
invasive = Pparva
# Retain only individuals from Asian populations
invasive$tab=invasive$tab[y,]
invasive$pop=invasive$pop[y]
invasive$ploidy=invasive$ploidy[y]
rm(y)
save(invasive,file="Data/DAPC/invasive.Rda")
load(file="Data/DAPC/invasive.Rda")

#============================================================
#  Preliminary Clustering of Data -- k Means
#------------------------------------------------------------
# run interactively to select the number of clusters (based on minimizing
BIC) and to estimate variance explained
# number of retained PCs will be set to maximum allowed for model "stability"
(< N/3; 56< 168/3)
# 56 retained PCs appears to exlain 75-80% of cumulative variance
# BIC minimum occurs clearly at K=6, then aplateau at K=7 and increases above
K=7
k.invasive = find.clusters(invasive, max.n=11)
# Compare if clusters corresponded to observed groups (i.e. sampled
locations)
table.value(table(pop(invasive), k.invasive$grp), col.lab=paste("inf", 1:11),
            row.lab=paste("ori", 1:11))
# Observed groups globally corresponded to inferred groups, with some
outliers

#------------------------------------------------------------
#  Initial Discriminant Analysis of Principal Components (DAPC)
#------------------------------------------------------------
# again set to retain maximum numbers of PCs & DA's
# again run interactively to examine scree-plot of F-statistics
# number of discriminant axis is set at (nb of clusters - 1)
dapc.invasive = dapc(invasive, pop=k.invasive$grp, n.pca=56,n.da=5)
#------------------------------------------------------------
compoplot(dapc.invasive, posi="top", col=c(rainbow(6,s=1,v=1),
rainbow(6,s=0.5,v=0.75)))
# Clear clusters, without shared posterior membership probability between
clusters (no 'admixture' evidence within individuals)
dapc.invasive$grp
```

```
# no individual show evidence of admixture
# Large pan-european cluster of Aus., Bel., UK, Hun. and Pol.
# Hun. showed signs of membership to 2 clusters (large pan-european and
isolated cluster) -> sign of 2 populations? divergence?
# Clusters globally seemed to correspond to existing sampled populations

#============================================================
#  Alternate Strategy for Clustering of Data:  No. PCs Retained to Achieve
Set Cumulative Variance Explained
#------------------------------------------------------------
#!# over-fitting?
# perhaps retain fewer PCs initially
# use cumulative variance as a threshold
#------------------------------------------------------------
# 75% of variance
#------------------------------------------------------------
k.invasive = find.clusters(invasive, max.n=11, pca.select="percVar",
perc.pca=75)
# run interactively to select the number of clusters
# BIC minimum occured at k=6
# Same pattern as before, no evidence of overfitting
# plateau with K=7

#------------------------------------------------------------
# 70% of variance
#------------------------------------------------------------
k.invasive = find.clusters(invasive, max.n=11, pca.select="percVar",
perc.pca=70)
# run interactively to select the number of clusters
# BIC minimum still occured clearly at k=6
# plateau with K=7

#============================================================
#  Alternate Strategy for Clustering of Data:  Using a "best fit" Selection
Criterion with multiple iterations (in addition to cum. var.)
#------------------------------------------------------------
test1.k = list()
for (i in 1:100){ print(paste("Iteration No.", i, "of 100"))
  k.invasive = find.clusters(invasive, max.n=11, pca.select="percVar",
perc.pca=70, choose.n.clust=F, criterion="goodfit", n.iter=1000)
  test1.k[[i]] = k.invasive
  rm(k.invasive) }
rm(i)
i=1
x = length(test1.k[[i]]$size)
for(i in 2:100){ x = c(x, length(test1.k[[i]]$size))}
rm(i)
table(x)
hist(x)
# 100 iterations is not enough
```

```
# Mixed scores of 58% for K=4 and 41% for K=5
# BUT "Goodfit" criterion tended to favor a small K, that might not be
informative in our case
# Lack of resolution in invasive populations (weak genetic structure in data)
or no real clustering?

# Same iterative test but with the criterion "min" of BIC
# As BIC~Nb of clusters decreases up to a plateau and stop decreasing,
# in an exponential negative shape like an island model (see details of
?find.clusters())
test1.k = list()
for (i in 1:100){ print(paste("Iteration No.", i, "of 100"))
  k.invasive = find.clusters(invasive, max.n=11, pca.select="percVar",
perc.pca=70, choose.n.clust=F, criterion="min", n.iter=1000)
  test1.k[[i]] = k.invasive
  rm(k.invasive) }
rm(i)

i=1
x = length(test1.k[[i]]$size)
for(i in 2:100){ x = c(x, length(test1.k[[i]]$size))}
rm(i)
table(x)
hist(x)
# K=6 for 17 iterations, and K=7 for 83 iterations
# Once again, there is a range between 4 and 7 by considering different
criterion

#-----------------------------------------------------------
# probably best to run at least 1,000 total iterations
#-----------------------------------------------------------
for (i in 1:1000){ print(paste("Iteration No.", i, "of 1000"))
  k.invasive=find.clusters(invasive, max.n=19, pca.select="percVar",
perc.pca=70, choose.n.clust=F, criterion="goodfit", n.iter=1000,
n.start=sample(10:21,1))
  test1.k[[i]]=k.invasive
  rm(k.invasive) }
rm(i)
i=1
x=length(test1.k[[i]]$size)
for(i in 1:1000){ x=c(x, length(test1.k[[i]]$size))}
rm(i)
table(x)
hist(x)
# 4 clusters (n=794/1000) or 5 clusters (n=205/1000)
for (i in 1:1000){ print(paste("Iteration No.", i, "of 1000"))
  k.invasive=find.clusters(invasive, max.n=19, pca.select="percVar",
perc.pca=70, choose.n.clust=F, criterion="min", n.iter=1000,
n.start=sample(10:21,1))
  test1.k[[i]]=k.invasive
```

```
  rm(k.invasive) }
rm(i)

i=1
x=length(test1.k[[i]]$size)
for(i in 1:1000){ x=c(x, length(test1.k[[i]]$size))}
rm(i)
table(x)
hist(x)
# K=6-7 were favored for minimum criterion (n=119 for K=6 & n=882 for K=7)
# But might be overfitting
# SO, we tested for over-fitting

#-----------------------------------------------------------
###  Step 2 in Strategy:  Resolving the Over-fitting Problem
#-----------------------------------------------------------
# cross-validation to estimate the optimal number of retained PCs
# given size of dataset, consider 56 as an upper limit
# maximum "allowed" for model stability (< N/3; 56< 168/3)
mat=tab(invasive, NA.method="mean")
# It seemed that at least 20 PCA was sufficient to discriminate amongst
clusters
xval=xvalDapc(mat, grp=invasive@pop, n.pca=seq(5,55,5))
# 20 seemed enough to discriminate amongst k=7 clusters
xval=xvalDapc(mat, grp=test1.k[[10]]$grp, n.pca=seq(5,55,5))

#!# NB
# NONE of the plots displayed a nice peaked pattern
#-----------------------------------------------------------
# a-score optimization for K=7 -> 'min' criterion that increased the number
of clusters compared to 'goodfit' criterion
# Linear decline of a-score from 0 to 55
# suggested far fewer (<10) PCs to avoid overfitting
dapc.1=dapc(invasive, pop=test1.k[[10]]$grp, pca.select="percVar",
perc.pca=70, n.da=6)
scatter(dapc.1)
optim.a.score(dapc.1, n.pca=seq(5,55,5))
dapc.2=dapc(invasive, pop=test1.k[[10]]$grp, pca.select="percVar", n.pca=56,
n.da=6)
scatter(dapc.2)
dapc.3=dapc(invasive, pop=test1.k[[10]]$grp, pca.select="percVar", n.pca=10,
n.da=6)
scatter(dapc.3)
# scatterplot showed greater degree of separation with 55 PCs than with 10
PCs -> we should decide to keep the max. of PCs (i.e. 56)

###=========================================================
###  Working Clusters
#-----------------------------------------------------------
# K=4 was a parcimonious number of K and the lowest BIC value (goodfit
```

```
criterion)
# BIC for K=5 was close to the BIC value of K=4
k.invasive = find.clusters(invasive, max.n=11,n.pca=56,n.clust=4)
k.invasive$size
save(k.invasive,file="Data/DAPC/k.invasive.rda")
load(file="Data/DAPC/k.invasive.rda")
xval = xvalDapc(mat, grp=k.invasive$grp, n.pca=seq(5,55,5))
dapc.invasive = dapc(invasive, pop=k.invasive$grp, n.pca=55,n.da=3)
save(dapc.invasive,file="Data/DAPC/dapc.invasive.rda")
load("Data/DAPC/dapc.invasive.rda")
scatter(dapc.invasive)
compoplot(dapc.invasive, posi="top", col=c(rainbow(6,s=1,v=1),
rainbow(6,s=0.5,v=0.75)))

#-----------------------------------------------------------
### Identifying Members of Each Cluster
#-----------------------------------------------------------
invasive.cluster.df = data.frame(Cluster=factor(), Ind.Site=factor(),
Ind=factor())
for(g in 1:4){ x = sort(names(k.invasive$grp)[which(k.invasive$grp==g)])
for(i in 1:length(x)){ temp.df = data.frame(Cluster=paste("Cluster", g,
sep="_"), Ind.Site=strsplit(x[i], "_")[[1]][1], Ind=x[i])
invasive.cluster.df = rbind(invasive.cluster.df, temp.df)
rm(i)
rm(temp.df)}
rm(x) }
rm(g)
#-----------------------------------------------------------
# Create a data frame with all populations allocated to each cluster
unique(invasive.cluster.df[,1:2])
write.csv(unique(invasive.cluster.df[,1:2]),
"Tables/DAPC/invasive_cluster.csv", row.names=F,quote=F)
#-----------------------------------------------------------
# To look for a given cluster
subset((unique(invasive.cluster.df[,1:2])), Cluster=="Cluster_3")
barplot(table(subset(invasive.cluster.df, Cluster=="Cluster_3")$Ind.Site),
las=1)
barplot(sort(table(subset(invasive.cluster.df,
Cluster=="Cluster_3")$Ind.Site)), las=1)
#-----------------------------------------------------------
# Ou bien pour identifier chaque cluster ou se trouvent des echantillons
d'une site en particulier
subset((unique(invasive.cluster.df[,1:2])), Ind.Site=="Aus")
x=c("Aus", "Bel", "Bul1", "Bul2", "Hun", "Ira", "Ita", "Pol", "Spa", "Tur",
"UK")
for (i in 1:length(x)){ print(i)
  print(subset((unique(invasive.cluster.df[,1:2])), Ind.Site==x[i])) }
rm(i)
rm(x)
```

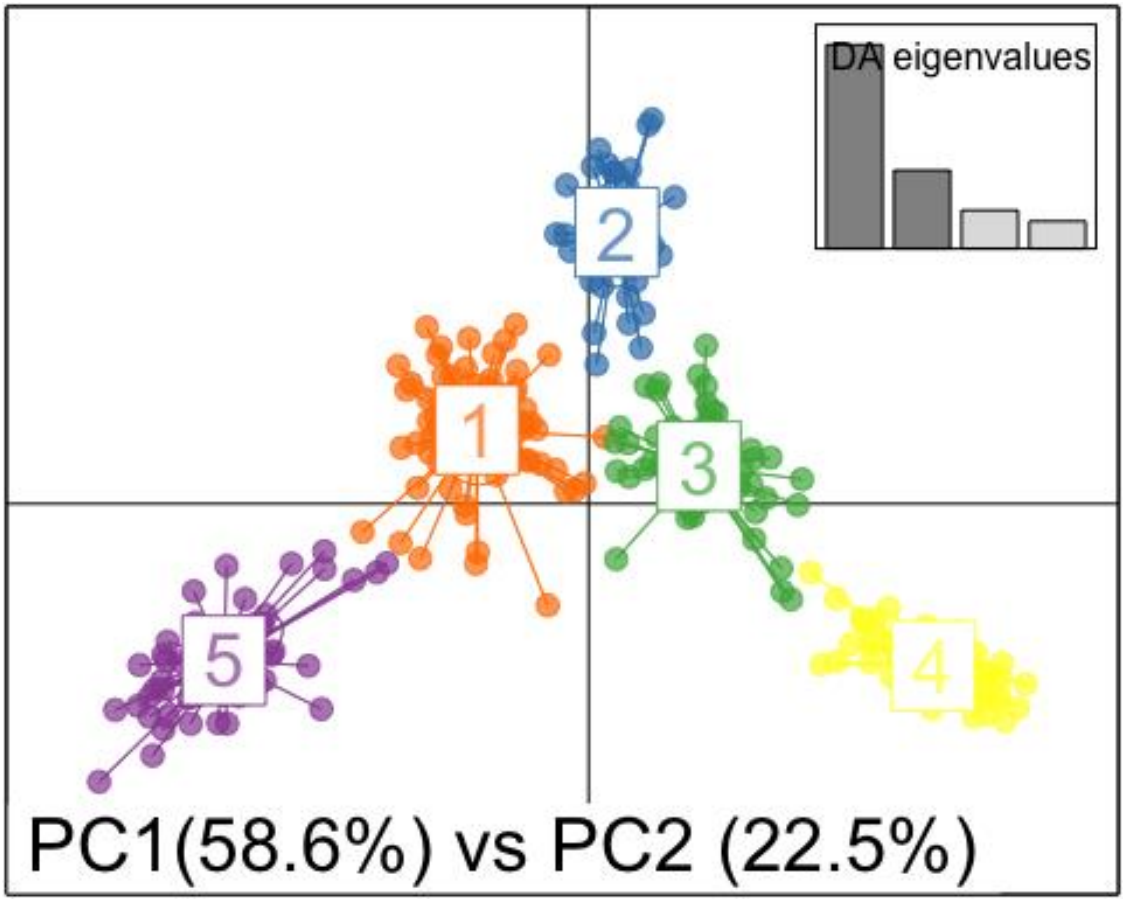
```
#=========================================================
#         WORLDWIDE ANALYSIS
#
# 2/     Assignments of invasive populations to native Sampling Site
#=========================================================
#---------------------------------------------------------
#  Assignments by Sampling Site
#---------------------------------------------------------
# Compute assignment probability to native clusters for any invasive
population in the vector 'invasive.names'
for (p in invasive.names) {
  # cat("Population assignment for", p,"\n")
  temp.genind=Pparva[pop=p]
  pred=predict.dapc(dapc.native, newdata=temp.genind)
  table(tail(pred$assign, length(which(Pparva$pop==p))))
  barplot(t((tail(pred$posterior, length(which(Pparva$pop==p))))), legend=T,
args.legend=list(x="right", horiz=F),
          col=c(rainbow(6,s=1,v=1),
rainbow(3,s=0.5,v=0.75)),main=paste("Assignment of",p,"individuals to native
clusters",sep=" "))
  rm(p)
  rm(temp.genind)
}

#---------------------------------------------------------
### All in a Single Plot
#---------------------------------------------------------
png("Figures/Pop.assignment.InvasiveToNative.png",width=1600,height = 1050)
par(mfrow=c(4,3))
par(oma=c(1,1,2,1))
par(mar=c(2,3,2,0))
for (p in invasive.names) {
  cat("Population assignment for", p,"\n")
  temp.genind=Pparva[pop=p]

  pred=predict.dapc(dapc.native, newdata=temp.genind)
  table(tail(pred$assign, length(which(Pparva$pop==p))))
  barplot(t((tail(pred.Aus$posterior, length(which(Pparva$pop==p))))),
legend=F,
          col=c(rainbow(6,s=1,v=1),
rainbow(3,s=0.5,v=0.75)),main=paste("Individuals of",p,sep=" "))
  rm(p)
  rm(temp.genind)
}
plot(NULL ,xaxt='n',yaxt='n',bty='n',ylab='',xlab='', xlim=0:1, ylim=0:1)
legend("left", legend=(1:7), fill=c(rainbow(6,s=1,v=1),
rainbow(3,s=0.5,v=0.75)),cex=2)
mtext("Posterior Probability of Assignment", side=3, outer=T, line=0, font=2)
par(mfrow=c(1,1))
par(oma=c(1,1,1,1))
```

```
par(mar=c(1,1,1,1))
dev.off()
```

OUTPUTS

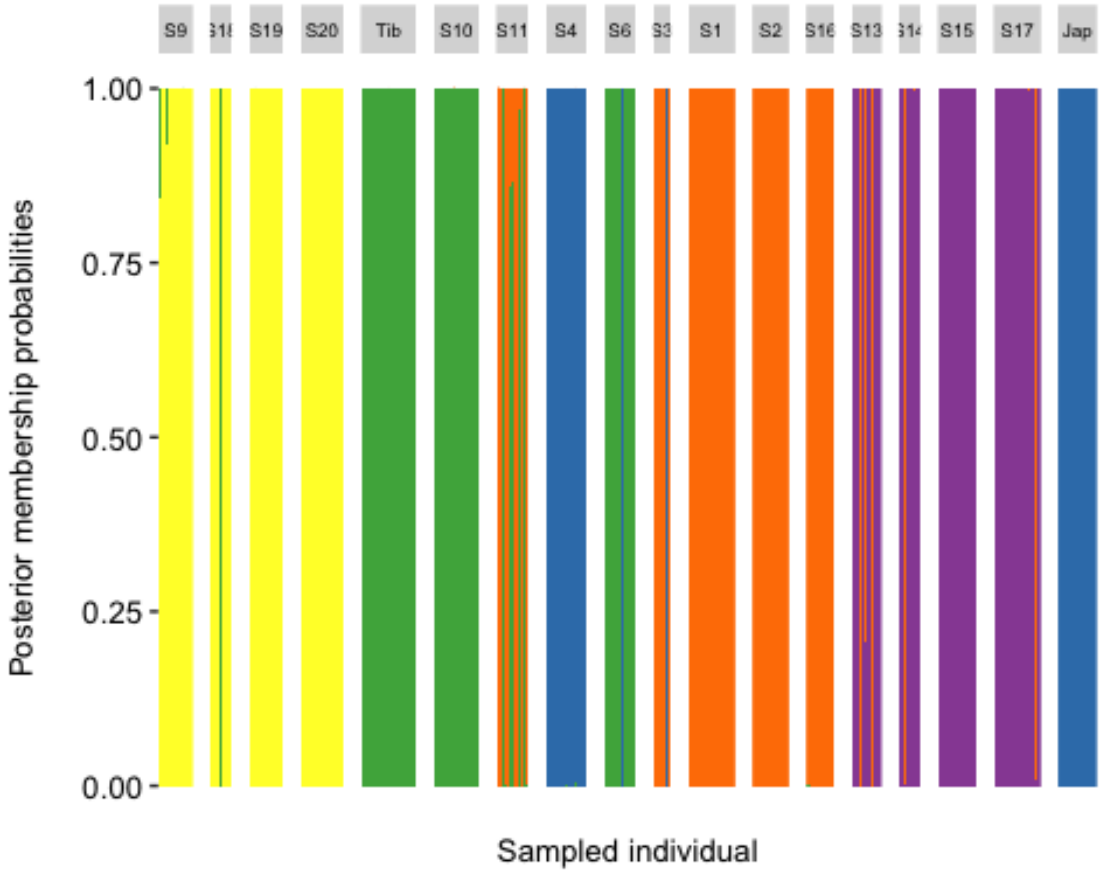
```
#--------------------------------
# Import coordinates of sampled locations
Pparva.sites=read.table("Data/Geographic/Pparva.sites.csv",sep=";",h=T)
#
write.table(genind2df(Pparva),paste(datadir,"/Pparva.alleles.txt",sep=""),sep
=" ",quote=FALSE,row.names=F,col.names=F)
labelPops=read.table("Data/STRUCTURE/labelPops.txt",header = TRUE)
# Lists of sampled sites names with their associated coordinates
native.names=c("Jap", "S1", "S2", "S3", "S4", "S6","S9",
               "S10", "S11", "S13", "S14", "S15", "S16", "S17", "S18", "S19",
"S20", "Tib")
native.coords=Pparva.sites[which(Pparva.sites$Pop %in% native.names),]
invasive.names=c("Aus", "Bel", "Bul1", "Bul2", "Hun", "Ira", "Ita", "Pol",
"Spa", "Tur", "UK")
invasive.coords=Pparva.sites[which(Pparva.sites$Pop %in% invasive.names),]
# For maps, Bul1 and Bul2 pie charts are displayed on the same point
# S19 and S20 too
# Jitter these populations coordinates
native.coords[native.coords$Pop=="S19",3]=native.coords[native.coords$Pop=="S
19",3]-2
native.coords[native.coords$Pop=="S20",3]=native.coords[native.coords$Pop=="S
19",3]+2

invasive.coords[invasive.coords$Pop=="Bul1",3]=invasive.coords[invasive.coord
s$Pop=="Bul1",3]-1
invasive.coords[invasive.coords$Pop=="Bul2",3]=invasive.coords[invasive.coord
s$Pop=="Bul2",3]+1
#--------------------------------
# GENIND FOR NATIVE & INVASIVE
# Create native and invasise genind
Native=Pparva[Pparva$pop %in% native.names]
Invasive=Pparva[Pparva$pop %in% invasive.names]
#-----------------------------------------------------------
# NATIVE AREA
#-----------------------------------------------------------
# Load the K value selected in DAPC.R
load("Data/DAPC/native.rda") # Data set genind for DAPC
load("Data/DAPC/k.native.rda") # Clusters
load("Data/DAPC/dapc.native.rda") # Results of the dapc for 6 clusters in
native populations
# Set a vector of colors for clusters
cols.native=c("#FF7F00",  "#377EB8", "#4DAF4A", "#FFFF33", "#984EA3")
#------------
# Fig. Scatter plot of 2 first axis of DAPC with clusters
# With barplot of clusters
scatter.dapc(dapc.native,xax=1,yax=2,addaxes=TRUE,xlab="Axe1",ylab="Axe
```

```
2",grp=dapc.native$grp,col=cols.native, cex=2,
bg="white",cstar=1,cellipse=0.95,
             csub =2, clabel = 2, posi.da="topright",

sub=paste("PC1(",round(dapc.native$eig[1]/sum(dapc.native$eig)*100,digits=1),
"%) vs PC2
(",round(dapc.native$eig[2]/sum(dapc.native$eig)*100,digits=1),"%)",sep=""))
```

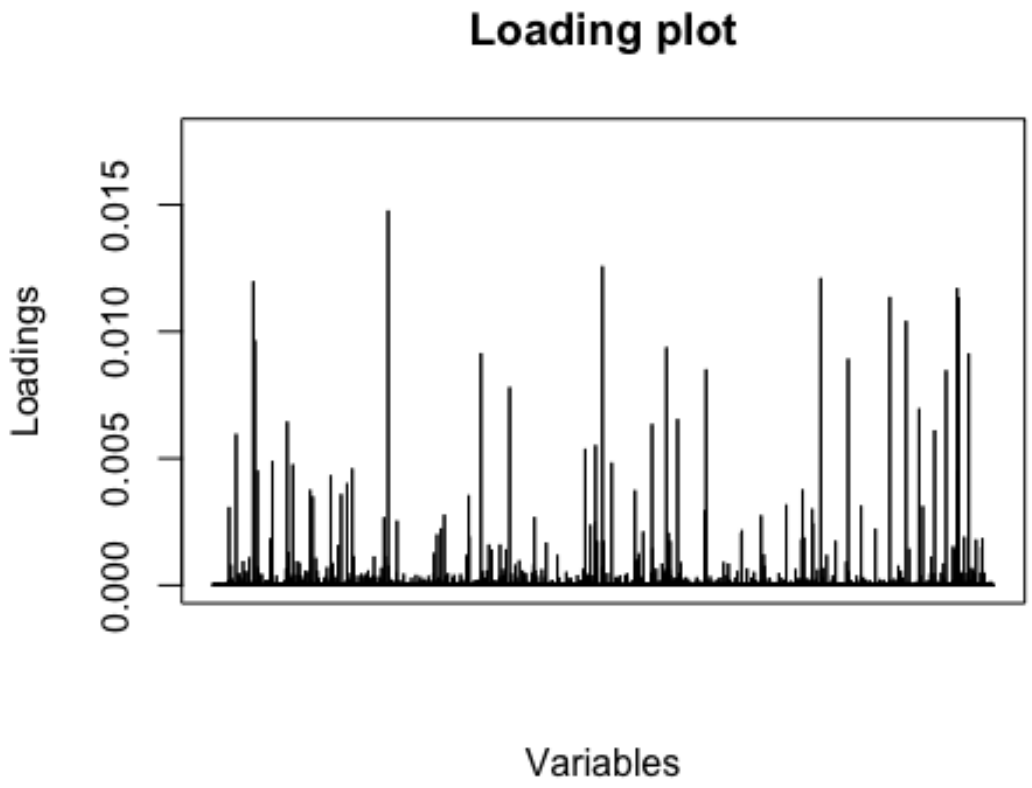



```
# Custom barplot (Posterior membership probability as a function of
individuals)
# Format a data frame with 4 columns, and as many rows per individual as
there is clusters
dapc.results=as.data.frame(dapc.native$posterior)
dapc.results$pop=pop(native)
dapc.results$indNames=rownames(dapc.results)
dapc.results=melt(dapc.results)
colnames(dapc.results)=c("Original_Population","Sample","Assigned_Population"
,"Posterior_membership_probability")

######### GGPLOT
# Cluster populations together in reference demes before plotting...
dapc.results$Original_Population=factor(dapc.results$Original_Population,
                                        levels =
```

```
c("S9","S18","S19","S20","Tib","S10","S11","S4","S6","S3","S1","S2","S16","S1
3","S14","S15","S17", "Jap"))# Plotting the barplot
dapc_barplot=ggplot(data=dapc.results, aes(x=Sample,
y=Posterior_membership_probability, fill=Assigned_Population))+
  geom_bar(stat='identity',width=1) +
  scale_fill_manual(values = cols.native) +
  facet_grid(~Original_Population, scales = "free",space="free_x") +
  labs(x="Sampled individual", y="Posterior membership probabilities\n",
fill="Assigned\npopulation") +
  theme(axis.line = element_blank(),
        # axis.line.x = element_blank(), # No x axis
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        plot.title = element_text(color="black", size=10,
face="bold.italic",hjust = 0.5),
        plot.subtitle = element_text(color="black",size=10,hjust = 0.5),
        axis.title.x = element_text(color="black", size=10),
        axis.title.y = element_text(color="black", size=10),
        axis.text=element_text(size=10, colour="black"),
        axis.text.x=element_blank(), # No samples names
        axis.ticks.x=element_blank(), # No x axis
        strip.text=element_text(size=6, colour="black"),
        # legend.key = element_rect(fill = "white", size = 1),
        # legend.text=element_text(size=22),
        # legend.title=element_text(size=22),
        legend.position = "none")
dapc_barplot
```




```
#
ggsave(paste(figuresdir,"/DAPC/Final/Native_Posterior_membership_probability_
barplot_K5.png",sep=""),
#        device="png",dpi=320,units="cm",width=60,height=9)

# Contribution of alleles to the variance in the data set
loadingplot(dapc.native$var.contr, axis = 2, thres = 0.07, lab.jitter = 1)
```

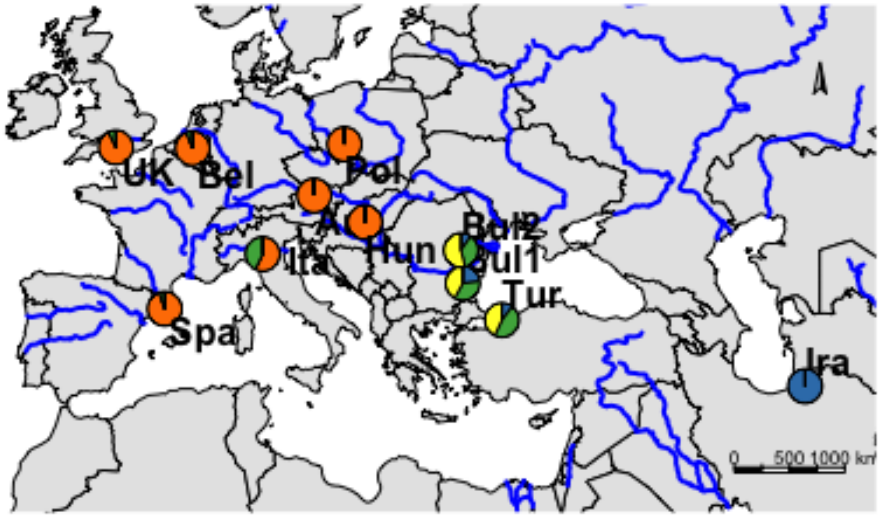



```
## NULL

# Pattern of contribution seemed neutral (random pattern) with multiple
contributors to the variance
sum(dapc.native$var.contr>0.005)

## [1] 178

# 236 alleles contributed to more than 0.005 of the explained variance
1/ncol(native$tab)

## [1] 0.0002371917

# null hypothesis of equal contribution was 1/nb of alleles = 0.00024
sum(dapc.native$var.contr>0.00024)

## [1] 2116

sum(dapc.native$var.contr>0.00024)/ncol(native$tab)

## [1] 0.5018975

# 2456 alleles/4216 (58%) contributed more than expected by chance
```

```
#-----------------------------------------------------------
# INVASIVE AREA
#-----------------------------------------------------------

#### Plots of invasive area independent clustering (> Supplementary material)
# Load the K value selected in DAPC.R
load("Data/DAPC/invasive.rda") # Data set genind for DAPC
load("Data/DAPC/k.invasive.rda") # Clusters
load("Data/DAPC/dapc.invasive.rda") # Results of the dapc for 6 clusters in
native populations
# dapc.invasive
# summary(dapc.invasive)
# Contribution of alleles to the variance in the data set
loadingplot(dapc.invasive$var.contr, axis = 2, thres = 0.07, lab.jitter = 1)
# Pattern of contribution seemed neutral (random pattern) with multiple
contributors to the variance
sum(dapc.invasive$var.contr>0.005)
# 144 alleles contributed to more than 0.005 of the explained variance
1/ncol(invasive$tab)
# null hypothesis of equal contribution was 1/nb of alleles = 0.00024
sum(dapc.invasive$var.contr>0.00024)
sum(dapc.invasive$var.contr>0.00024)/ncol(invasive$tab)
# 1434 alleles/4216 (34%) contributed more than expected by chance
####### Figures #######
# Set a vector of colors for the number of clusters
cols.invasive=brewer.pal(n = length(levels(dapc.invasive$grp)), name =
"Set2")
# Fig 1. Bar plot of admixture proportions per individual, sorted by
populations
compoplot(dapc.invasive,col=cols.invasive)
# Fig 2. Scatter plot of 2 first axis of DAPC with clusters
scatter.dapc(dapc.invasive,xax=1,yax=2,grp=dapc.invasive$grp,col=cols.invasiv
e, cex=2, bg="white",cstar=1,cellipse=0.95,
             sub=paste("PC1
(",round(dapc.invasive$eig[1]/sum(dapc.invasive$eig)*100,digits=1),"%) vs PC2
(",round(dapc.invasive$eig[2]/sum(dapc.invasive$eig)*100,digits=1),"%)",sep="
"))
# Look at other axes of DAPC vs PC1
par(mfrow=c(2,1))
scatter.dapc(dapc.invasive,xax=1,yax=2,grp=dapc.invasive$grp,col=cols.invasiv
e, cex=2,csub=2, clab=2, bg="white",cstar=1,cellipse=0.95,
             sub=paste("PC1
(",round(dapc.invasive$eig[1]/sum(dapc.invasive$eig)*100,digits=1),"%) vs PC2
(",round(dapc.invasive$eig[2]/sum(dapc.invasive$eig)*100,digits=1),"%)",sep="
"))
scatter.dapc(dapc.invasive,xax=1,yax=3,grp=dapc.invasive$grp,col=cols.invasiv
e, cex=2,csub=2, clab=2, bg="white",cstar=1,cellipse=0.95,
             sub=paste("PC1
(",round(dapc.invasive$eig[1]/sum(dapc.invasive$eig)*100,digits=1),"%) vs PC3
(",round(dapc.invasive$eig[3]/sum(dapc.invasive$eig)*100,digits=1),"%)",sep="
```

```
"))
par(mfrow=c(1,1))

#-----------------------------------------------------------
# WORLDWIDE CLUSTERING
#-----------------------------------------------------------
# Fig. Map of the invasive area with pies chart of cluster membership on each
sampling location
load("Data/DAPC/invasive.rda") # Data set genind for DAPC
load("Data/DAPC/k.invasive.rda") # Clusters
load("Data/DAPC/dapc.invasive.rda") # Results of the dapc for 6 clusters in
native populations
load("Data/Pparva.clean.Rda")
dapc.posterior.membership=data.frame()
for (p in invasive.names) {
  # cat("Population assignment for", p,"\n")
  temp.genind=Pparva[pop=p]
  pred=predict.dapc(dapc.native, newdata=temp.genind)

dapc.posterior.membership=rbind(dapc.posterior.membership,cbind(rep(as.charac
ter(p), nrow(pred$posterior)),pred$posterior))
  rm(p)
  rm(temp.genind)
}
# Prepare the data set for mapping
names(dapc.posterior.membership)=c("Population", "Cluster 1", "Cluster 2",
"Cluster 3", "Cluster 4", "Cluster 5")
for (i in 2:6) {

dapc.posterior.membership[,i]=as.numeric(as.character(dapc.posterior.membersh
ip[,i]))
}
dapc.posterior.membership_means=sapply(split(dapc.posterior.membership[2:6],d
apc.posterior.membership$Population), colMeans) ## This just makes population
cluster averages for each cluster
# Download database of river network at scale 50
load("Data/rivers50.Rda")
# Mapping
map("world", xlim=c(-15,60), ylim=c(30,59), col="gray90",  fill=TRUE) # Plot
the map of
sp::plot(rivers50,xlim=c(-15,30), ylim=c(30,59),lwd=2, col = 'blue',
add=TRUE)
# Nomenclature
north.arrow(56,56,1, lab="", lab.pos="above")
scalebar(c(49,32),1000, bg=NA, border=NA, division.cex=0.5)
text(x=105, y=34,"",cex=3,font=2)
text(x=139, y=39,"",cex=3,font=2)
points(invasive.coords$X, invasive.coords$Y, pch = 16, cex = 0.7,col="red")
#\ Add points to the map
# map.scale(x=50,y=55,relwidth=0.1,cex=2)
```

```
## Pies ##
# Draw a pie with proportions of membership for each population
for (i in 1:ncol(dapc.posterior.membership_means)) {
  add.pie(dapc.posterior.membership_means[,i],x=invasive.coords$X[i],
y=invasive.coords$Y[i],labels=invasive.coords[i,1],label.dist=1.5,
          radius=1,edges=200,clockwise=T,   col =   cols.native,cex=1,font=2)
  # Add labels to pies
}
```



GENETIC CLUSTERING - STRUCTURE

Convert GENIND to STRUCTURE

First, we had to format the GENIND to a STRUCTURE file. There was no export function doing this in 'adegenet' (R package managing genind objects). Hence, we implemented a function genind2structure.R.

```
#=============================================================
# FUNCTION genind2structure
#=============================================================
# Available at github.com/ThomasBrazier
#-------------------------------------------------------------
# DESCRIPTION
# Function to export to STRUCTURE format from genind object.
# genind objects are created in the R package adegenet. Function below is an
R function.

# STRUCTURE input files are described as following by the user manual:
# Each row of individual data contains the following elements. These form
columns in the data file.
# 1. Label (Optional; string) A string of integers or characters used to
designate each individual in the sample.
# 2. PopData (Optional; integer) An integer designating a user-defined
population
# 3. PopFlag (Optional; 0 or 1) A Boolean flag which indicates whether to use
```

```
the PopData
#       when using learning samples (Note: A Boolean variable (flag) is a
variable which takes the values TRUE or FALSE, which are designated here by
the integers
#       1 (use PopData) and 0 (don't use PopData), respectively.)
# 4. LocData (Optional; integer) An integer designating a user-defined
sampling location
# 5. Phenotype (Optional; integer) An integer designating the value of a
phenotype of interest
# 6. Extra Columns (Optional; string) It may be convenient for the user to
include additional data in the input file which are ignored by the program.
# 7. Genotype Data (Required; integer) Each allele at a given locus should be
coded by a unique integer (eg microsatellite repeat score).

#-------------------------------------------------------
# USAGE
# genind2structure(obj, file="", pops=FALSE, missingdata.label=-9,
popflag=NA, locdata=NA, phenotype=NA, extra=NA)

#-------------------------------------------------------
# ARGUMENTS
# obj: a genind object to convert
# file: file name to write
# pops: whether to include population info in the file, population names are
taken in the genind object
# missingdata.label: value is the expected value for missing data
# popflag: if popflag is not NA, then it is a vector of popflags (numeric, 0
or 1) of the same size as the number of individuals
# locdata: if locdata is not NA, then it is a a numeric vector of user-
defined sampling locations
# phenotype: if phenotype is not NA, then it is a numeric vector of
phenotypes
# extra: if extra is not NA, then it is a character vector with extra
informations

# example use:
# data(nancycats)
# genind2structure(nancycats, file="nancy_structure.txt", pops=TRUE)

#-------------------------------------------------------
# VALUE
# Return either a data.frame if file="", or write the data.frame into the
file given in argument.
#-------------------------------------------------------
genind2structure = function(obj, file="", pops=TRUE, missingdata.label=-9,
popflag=NA, locdata=NA,phenotype=NA, extra=NA){
  # Dependencies
  if ('adegenet' %in% installed.packages()) {
    library('adegenet')
  } else {
```

```
    warning("adegenet is not installed.")
    ans = readline("Do you want to install it? (y/n)")
    if (ans=="y" | ans=="yes") {
      install.packages('adegenet')
    } else {
      stop("Function ended because adegenet is required and not installed.")
    }
  }

  # Check first that the object is a genind object; if not, stop the function
  if(!"genind" %in% class(obj)){
    stop("Function was designed for genind objects.")
  }
  # get the max ploidy of the dataset
  ploidy = max(obj@ploidy)
  # get the number of individuals
  Nind = nInd(obj)
  #------------------------
  # 1. ADD IND. LABEL
  # column of individual names to write; set up data.frame
  tab = data.frame(ind=rep(indNames(obj), each=ploidy))
  # 2. ADD POP. LABEL
  # column of pop ids to write
  if(pops){
    popnums = 1:nPop(obj)
    names(popnums) = as.character(unique(pop(obj)))
    popcol = rep(popnums[as.character(pop(obj))], each=ploidy)
    tab = cbind(tab, data.frame(pop=popcol))
  }
  # 3. Add POPFLAG column if needed
  # Consider the 2 cases where popflag is the same length as structure data
frame (replicated values for ploidy)
  # or same length as number of individuals (just one value per individual
that we need to replicate given the ploidy level)
  if (all(!is.na(popflag))) {
    if (is.numeric(popflag)) {
      if (length(popflag)==Nind) {
        tab=cbind(tab, data.frame(popflag=rep(popflag, each=ploidy))) #
popflags need to be replicated given the ploidy level
      } else {
        if (length(popflag)==nrow(tab)) { # popflags do not need to be
replicated given the ploidy
          tab=cbind(tab, data.frame(popflag=popflag))
        }else {
          warning("Not the correct number of popflags in the list")
        }
      }
    } else {
      warning("Popflag is not a numeric vector")
    }
```

```
  }
  # 4. ADD LOCDATA
  if (all(!is.na(locdata))) {
    if (is.numeric(locdata)) {
      if (length(locdata)==Nind) {
        tab=cbind(tab, data.frame(locdata=rep(locdata, each=ploidy))) #
locdatas need to be replicated given the ploidy level
      } else {
        if (length(locdata)==nrow(tab)) { # locdatas do not need to be
replicated given the ploidy
          tab=cbind(tab, data.frame(locdata=locdata))
        }else {
          warning("Not the correct number of locdatas in the list")
        }
      }
    } else {
      warning("Locdata is not a numeric vector")
    }
  }
  # 5. ADD PHENOTYPE
  if (all(!is.na(phenotype))) {
    if (is.numeric(phenotype)) {
      if (length(phenotype)==Nind) {
        tab=cbind(tab, data.frame(phenotype=rep(phenotype, each=ploidy))) #
phenotypes need to be replicated given the ploidy level
      } else {
        if (length(phenotype)==nrow(tab)) { # phenotypes do not need to be
replicated given the ploidy
          tab=cbind(tab, data.frame(phenotype=phenotype))
        }else {
          warning("Not the correct number of phenotypes in the list")
        }
      }
    } else {
      warning("Phenotype is not a numeric vector")
    }
  }
  # 6. ADD EXTRA COLUMN
  if (all(!is.na(extra))) {
    if (is.character(extra)) {
      if (length(extra)==nrow(tab)) {
        tab=cbind(tab, data.frame(extra=rep(extra, each=ploidy)))
      } else {
        warning("Not as many extra informations as individuals")
      }
    } else {
      warning("Extra information is not a character vector")
    }
  }
  if (all(!is.na(extra))) {
```

```
    if (is.character(extra)) {
      if (length(extra)==Nind) {
        tab=cbind(tab, data.frame(extra=rep(extra, each=ploidy))) # extras
need to be replicated given the ploidy level
      } else {
        if (length(extra)==nrow(tab)) { # extras do not need to be replicated
given the ploidy
          tab=cbind(tab, data.frame(extra=extra))
        }else {
          warning("Not the correct number of extras in the list")
        }
      }
    } else {
      warning("Extra is not a character vector")
    }
  }
  # 7. ADD GENOTYPES
  # get loci names
  loci = locNames(obj)
  # add columns for genotypes
  tab = cbind(tab, matrix(missingdata.label, nrow=dim(tab)[1],
ncol=nLoc(obj), dimnames=list(NULL,loci)))
  # begin going through loci
  for(L in loci){
    gen = obj@tab[,grep(paste("^", L, "\\.", sep=""),
                                dimnames(obj@tab)[[2]]),
                           drop = FALSE] # genotypes by locus
    al = 1:dim(gen)[2] # numbered alleles
    for(s in 1:Nind){
      if(all(!is.na(gen[s,]))){
        tabrows = (1:dim(tab)[1])[tab[[1]] == indNames(obj)[s]] # index of
rows in output to write to
        tabrows = tabrows[1:sum(gen[s,])] # subset if this is lower ploidy
than max ploidy
        tab[tabrows,L] = rep(al, times = gen[s,])
      }
    }
  }
  #------------------------------
  # export table as a file (if no filename specified, then tab is returned at
the end of the function)
  if (file != "") {
    write.table(tab, file=file, sep="\t", quote=FALSE, row.names=FALSE,
col.names=FALSE)
  } else {
    return(tab)
  }
}
```

Make STRUCTURE input files

Dataset was splitted to get separate input files for invasive populations, native populations and sub-structure in the Western European deme.

```
#=========================================================
# STRUCTURE input files
#=========================================================
# Pparva.structure is formatted to produce STRUCTURE data set
# See STRUCTURE.R for the complete procedure...
Pparva.structure=read.table("Data/STRUCTURE/Pparva.structure.txt",sep="\t")
#-------------------------------
# WRITE DATA SET FOR STRUCTURE
#-------------------------------
# Rewrite data file
# 468 individuals, so 936 lines in the file for verification
write.table(Pparva.structure,"Data/STRUCTURE/1a_Allpops/100a_Pparva_structure
_allpops.txt",sep="\t",quote=FALSE,row.names=FALSE,col.names=FALSE)
# Add POPFLAG
# Asian populations are popflagged 1 whereas european populations (ancestry
to predict) are popflagged 0
popflag=Pparva.structure[,2] # popflag contain population number
popflag[which(popflag %in% c(1:7,9,26,28,29))]=0
popflag[which(popflag != 0)]=1
Pparva.structure.popflag=cbind(Pparva.structure[,1:2],
                               popflag,
                               Pparva.structure[,3:ncol(Pparva.structure)])
write.table(Pparva.structure.popflag,"Data/STRUCTURE/1b_Allpops/100b_Pparva_s
tructure_allpops.txt",sep="\t",quote=FALSE,row.names=FALSE,col.names=FALSE)

###### LABELS for Distruct
# Labels below figure were simply population number in data set
# Generate a file of labels atop figure (Population names)
# A first column of population's number (number in dataset) and a second
column with the corresponding population name
labels_atop=data.frame(seq(1,length(levels(Pparva@pop))),levels(Pparva@pop))
write.table(labels_atop,"Data/STRUCTURE/labels_atop.txt",sep="\t",quote=FALSE
,row.names=FALSE,col.names=FALSE)

#-------------------------------
# Only populations of native range
# i.e. individuals in 'temp' whom populations (col. 2) were included in
labelPops[c(8,11:28,31:32),2]
## Write data file
# 300 individuals, so 600 lines in the file for verification
write.table(as.data.frame(Pparva.structure[which(Pparva.structure[,2] %in%
c(8,10:25,27)),]),"Data/STRUCTURE/2_Native/200_Pparva_structure_native.txt",s
ep="\t",quote=FALSE,row.names=FALSE,col.names=FALSE)
write.table(as.data.frame(Pparva.structure[which(Pparva.structure[,2] %in%
c(8,10:25,27)),2]),"Data/STRUCTURE/2_Native/labelPops_native.txt",sep="\t",qu
ote=FALSE,row.names=FALSE,col.names=FALSE)
```

```
# 20 replicates for K=1-21 for 100,000 BURNIN + 100,000 iterations:
structureLauncher(project="2_Native",path="/home/users/tbrazier/STRUCTURE/2_Native",niter=c(1,20),Kmin=1,Kmax=21,
                  popsize=nrow(Pparva.structure[which(Pparva.structure[,2] %in% c(8,10:25,27)),])/2)

# HIERARCHICAL PROCEDURE
# STRUCTURE on a subset of populations in native area: explain genetic structure of the large Pan-asian admixed zone
write.table(as.data.frame(Pparva.structure[which(Pparva.structure[,2] %in% c(10:17,20,22:24,27)),]),"Data/STRUCTURE/2b1_Native/200b_Pparva_structure_native.txt",sep="\t",quote=FALSE,row.names=FALSE,col.names=FALSE)
structureLauncher(project="2b1_Native",path="/home/users/tbrazier/STRUCTURE/2b1_Native",niter=c(1,20),Kmin=1,Kmax=16,
                  popsize=nrow(Pparva.structure[which(Pparva.structure[,2] %in% c(10:17,20,22:24,27)),])/2)

##################7
# There was a lack of convergence for 100,000 iterations, but it was suspected that the K value was between 3, 6 and 7
# So, we decided to relaunch STRUCTURE for K=6 and 500,000 BURNIN + 500,000 iterations (previously, no sign of convergence after 100,000 iterations)
structureLauncher(project="21_Native_1MillionIterations",path="/home/users/tbrazier/STRUCTURE/21_Native_1MillionIterations",niter=c(1,16),Kmin=4,Kmax=8,
                  popsize=nrow(Pparva.structure[which(Pparva.structure[,2] %in% c(8,10:25,27)),])/2) # 8 cores on local server --> 2*8=16 replicates
structureLauncher(project="21_Native_1MillionIterations",path="/home/users/tbrazier/STRUCTURE/21_Native_1MillionIterations",niter=c(1,16),Kmin=1,Kmax=3,
                  popsize=nrow(Pparva.structure[which(Pparva.structure[,2] %in% c(8,10:25,27)),])/2) # 8 cores on local server --> 2*8=16 replicates

#-----------------------------------
# Only populations of invasive range
## Write data file
# 168 individuals, so 336 lines in the file for verification
write.table(as.data.frame(Pparva.structure[which(Pparva.structure[,2] %in% c(1:7,9,26,28:29)),]),"Data/STRUCTURE/3_Invasive/300_Pparva_structure_invasive.txt",sep="\t",quote=FALSE,row.names=FALSE,col.names=FALSE)
write.table(as.data.frame(Pparva.structure[which(Pparva.structure[,2] %in% c(1:7,9,26,28:29)),2]),"Data/STRUCTURE/3_Invasive/labelPops_invasive.txt",sep="\t",quote=FALSE,row.names=FALSE,col.names=FALSE)

structureLauncher(project="3_Invasive",path="/home/users/tbrazier/STRUCTURE/3_Invasive",niter=c(1,20),Kmin=1,Kmax=14,
                  popsize=nrow(Pparva.structure[which(Pparva.structure[,2] %in% c(1:7,9,26,28:29)),2])/2)

# HIERARCHICAL PROCEDURE
```

```
# Western Europe  showed a pattern of strong admixture (spurious clusters
suspected?) with 2 blended clusters
# STRUCTURE on a subset of populations in invasive area: explain genetic
structure of the Western Europe admixed zone
# n=119
write.table(as.data.frame(Pparva.structure[which(Pparva.structure[,2] %in%
c(1,2,5,7,9,26,29)),]),"Data/STRUCTURE/3a1_Invasive/300a1_Pparva_structure_in
vasive.txt",sep="\t",quote=FALSE,row.names=FALSE,col.names=FALSE)
structureLauncher(project="3a1_Invasive",path="/home/users/tbrazier/STRUCTURE
/3a1_Invasive",niter=c(1,20),Kmin=1,Kmax=10,
                  popsize=nrow(Pparva.structure[which(Pparva.structure[,2]
%in% c(1,2,5,7,9,26,29)),])/2)

#----------------------------------
# WORLDWIDE ANALYSES
# We ran this intercontinental cluster analysis to see if some source
populations were closest to invasive populations than their own neighbors in
the native area
# It was also interesting to estimates admixture within populations in order
to parameterize, DIY ABC
# 20 replicates for K=1-32 (all sampled sites + 3) for 100,000 BURNIN +
100,000 iterations:
structureLauncher(project="1a_AllPops",path="/home/users/tbrazier/STRUCTURE/1
a_AllPops",niter=c(1,20),Kmin=1,Kmax=32,
                  popsize=nrow(Pparva.structure)/2)
# 20 replicates for K=1-21 (all sampled sites in Asia (reference) + 3) for
100,000 BURNIN + 100,000 iterations:
structureLauncher(project="1b_AllPops",path="/home/users/tbrazier/STRUCTURE/1
b_AllPops",niter=c(1,20),Kmin=1,Kmax=21,
                  popsize=nrow(Pparva.structure)/2)
# It may be smarter to define populations in Asia as demes and not as sampled
populations.
# 20 replicates for K=1-13 (10 putative demes in Asia (reference) + 3) for
100,000 BURNIN + 100,000 iterations:
#-------------------------- NATIVE
# POP1 S4 S6
# POP2 Jap
# POP3 S10
# POP4 S13 S14 S15 S17
# POP5 S1 S2 S16
# POP6 S9 S18
#----------------------- INVASIVE
# POP7 Tib
# POP8 S19 S20
#---------------------- ADMIXTURE
# POP9 S3
# POP10 S11
Pparva.structure.demes=Pparva.structure.popflag
# Replace sampled sites in Asia by putative deme (putative source
population): the deme number is the pop number of the first population in the
```

```
vector
Pparva.structure.demes[which(Pparva.structure.demes$V2 %in%
c("S4","S6")),3]="23"
Pparva.structure.demes[which(Pparva.structure.demes$V2 %in% c("S10")),3]="11"
Pparva.structure.demes[which(Pparva.structure.demes$V2 %in% c("S13", "S14"
,"S15", "S17")),3]="13"
Pparva.structure.demes[which(Pparva.structure.demes$V2 %in% c("S1", "S2",
"S16")),3]="10"
Pparva.structure.demes[which(Pparva.structure.demes$V2 %in% c("S9",
"S18")),3]="25"
Pparva.structure.demes[which(Pparva.structure.demes$V2 %in% c("S19",
"S20")),3]="19"
# Write the data set for STRUCTURE
write.table(Pparva.structure.demes,"Data/STRUCTURE/1c_Allpops/100c_Pparva_str
ucture_demes.txt",sep="\t",quote=FALSE,row.names=FALSE,col.names=FALSE)
structureLauncher(project="1c_AllPops",path="/home/users/tbrazier/STRUCTURE/1
c_AllPops",niter=c(1,20),Kmin=1,Kmax=13,
                  popsize=nrow(Pparva.structure.demes)/2)
```

Analyses of STRUCTURE outputs

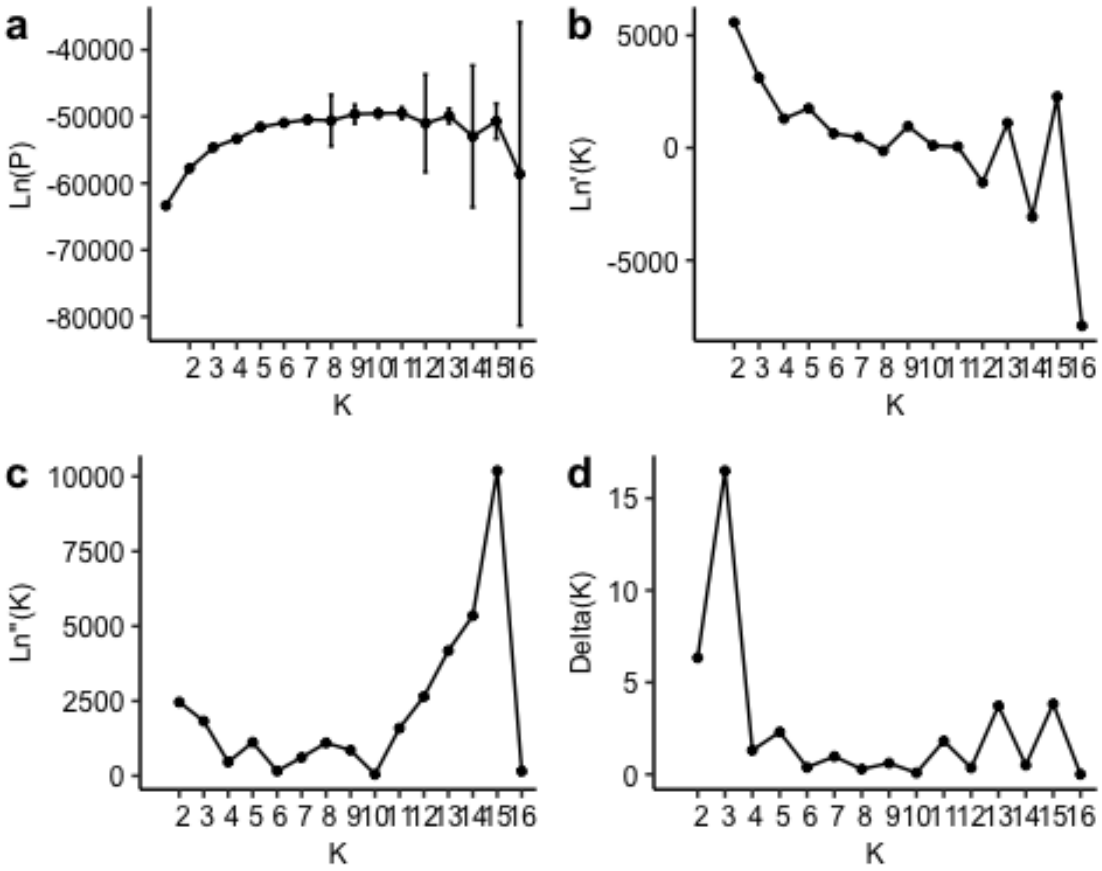
STRUCTURE was run indepently on the INRA facilities' cluster, since it was very demanding in computation time. Outputs were then analyzed back in R.

All values of K (number of genetic clusters) were equally investigated, but we display here the code only for a small subset of figures. All values of K were graphically compared, as the assessment of a K can be very practical and need a lot of attention. There were few adaptations in code given the K value.

```
#============================================================
# Part 3. STRUCTURE outputs
#============================================================
#---------------------------------
# NATIVE POPULATIONS
#########################
# All populations at once
# 1/ Graphic evaluation with the four plots of Evanno's method (Evanno et al.
2005)
# Most important plot is DeltaK = mean(|L''(K)|)/sd(L(K)) of L(K) as a
function of K
### Choose the most probable K value (number of cluster inferred), K value
the most appropriate for the data
# Load results from Harvester: 'evanno.txt'
evanno=read.table(paste(STRdir,"/2b_Native/Harvester/evanno.txt",sep=""),head
er=F,sep="\t")
# With K's assessment on convergent replciates only, Delta K suggested K=17
for 18 populations --> may be overclustering
# Look for more reasonable values under K=17
evanno=evanno[1:16,]
colnames(evanno)=c("K","Reps","Mean_LnP","sd_LnP","LnK","Ln2K","DeltaK")
######## Mean Ln(P) as a function of K
```

```
pLnP=ggplot(data=evanno, aes(x=K, y=Mean_LnP)) +
    geom_point(colour="Black",size=1)+
    geom_line() +
    geom_errorbar(aes(ymin=Mean_LnP-sd_LnP, ymax=Mean_LnP+sd_LnP), width=.2)
+
    xlab("K") + ylab("Ln(P)") +
    scale_x_discrete(limits=c(2:nrow(evanno))) +
    theme(axis.line = element_line(colour = "black"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank(),
          plot.title = element_text(color="black", size=9,
face="bold.italic",hjust = 0.5),
          plot.subtitle = element_text(color="black",size=9,hjust = 0.5),
          axis.title.x = element_text(color="black", size=9),
          axis.title.y = element_text(color="black", size=9),
          axis.text=element_text(size=9, colour="black"),
          legend.key = element_rect(fill = "white", size = 1),
          legend.text=element_text(size=9),
          legend.title=element_text(size=9))
######## Mean Ln(K) as a function of K
pLnK=ggplot(data=evanno, aes(x=K, y=LnK)) +
    geom_point(colour="Black",size=1)+
    geom_line() +
    xlab("K") + ylab("Ln'(K)") +
    scale_x_discrete(limits=c(2:nrow(evanno))) +
    theme(axis.line = element_line(colour = "black"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank(),
          plot.title = element_text(color="black", size=9,
face="bold.italic",hjust = 0.5),
          plot.subtitle = element_text(color="black",size=9,hjust = 0.5),
          axis.title.x = element_text(color="black", size=9),
          axis.title.y = element_text(color="black", size=9),
          axis.text=element_text(size=9, colour="black"),
          legend.key = element_rect(fill = "white", size = 1),
          legend.text=element_text(size=9),
          legend.title=element_text(size=9))
######## Mean Ln''(K) as a function of K
pLn2K=ggplot(data=evanno, aes(x=K, y=Ln2K)) +
    geom_point(colour="Black",size=1)+
    geom_line() +
    xlab("K") + ylab("Ln''(K)") +
    scale_x_discrete(limits=c(2:nrow(evanno))) +
    theme(axis.line = element_line(colour = "black"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
```

```
          panel.border = element_blank(),
          panel.background = element_blank(),
          plot.title = element_text(color="black", size=9,
face="bold.italic",hjust = 0.5),
          plot.subtitle = element_text(color="black",size=9,hjust = 0.5),
          axis.title.x = element_text(color="black", size=9),
          axis.title.y = element_text(color="black", size=9),
          axis.text=element_text(size=9, colour="black"),
          legend.key = element_rect(fill = "white", size = 1),

          legend.text=element_text(size=9),
          legend.title=element_text(size=9))
######## Mean Ln''(K) as a function of K
pDeltaK=ggplot(data=evanno, aes(x=K, y=DeltaK)) +
    geom_point(colour="Black",size=1)+
    geom_line() +
    xlab("K") + ylab("Delta(K)") +
    scale_x_discrete(limits=c(2:nrow(evanno))) +
    theme(axis.line = element_line(colour = "black"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank(),
          plot.title = element_text(color="black", size=9,
face="bold.italic",hjust = 0.5),
          plot.subtitle = element_text(color="black",size=9,hjust = 0.5),
          axis.title.x = element_text(color="black", size=9),
          axis.title.y = element_text(color="black", size=9),
          axis.text=element_text(size=9, colour="black"),
          legend.key = element_rect(fill = "white", size = 1),

          legend.text=element_text(size=9),
          legend.title=element_text(size=9))
#### SAVE TO FIGURE
ggarrange(pLnP,pLnK,pLn2K,pDeltaK,widths=1:1,heights=1:1,labels="auto")
```



This figure present the plateau method **(a)** and the Evanno method **(e)** to choose the value of K. On **fig. a**, the most probable K is when the difference in likelihood is lower and just before the strong increase of the variance. On **fig.e**, the most probable K is assessed by the modal value(s) of the distribution of DeltaK as a function of K.

The plateau method suggest a number of genetic clusters between 5 and 7. Before 5, increase in Ln(P) seemed significant. Over K = 7, the variance in Ln(P) increased strongly, suggesting a reduced convergence among replicates. Nonetheless, the DeltaK method gave a K = 3, 13, 15, with K=3 being the most probable of all.

```
#-------------------------------------
### Bar plot representation
# Representation of admixture (Q) for each individual FOR K=6 (chosen K)
# Format a data frame with 4 columns, and as many rows per individual as
there is clusters
# strK6=read.table(paste(STRdir,"/2a_Native/CLUMPP/K6.outfile",sep=""))
strK6=read.table(paste(STRdir,"/2b_Native/CLUMPP/K6.outfile",sep=""))
# Set a vector of colors for clusters
cols.native=c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00",
"#FFFF33", "#A65628", "#F781BF",
              "#8DD3C7", "#FFFFB3", "#BEBADA", "#FB8072", "#80B1D3",
"#FDB462", "#B3DE69", "#FCCDE5", "#D9D9D9")
# Get the same colors and cluster number as DAPC
strK6.results=as.data.frame(strK6[,(c(1,2,3,5,4,6)+5)]) # population clusters
in the good order for colors
colnames(strK6.results)=c(1,2,3,4,5,6)
strK6.results$pop=as.character(strK6[,4])
```

```
for (i in 1:nrow(strK6.results)) {
  strK6.results$pop[i]=as.character(labelPops[which(strK6.results$pop[i] ==
labelPops$Location_index),1])
}
strK6.results$indNames=names(Native$tab[,1])
strK6.results=melt(strK6.results)
colnames(strK6.results)=c("Original_Population","Sample","Assigned_Population
","Posterior_membership_probability")
######### GGPLOT
# Cluster populations together in reference demes before plotting...
strK6.results$Original_Population=factor(strK6.results$Original_Population,
                                         levels =
c("S9","S18","S19","S20","Tib","S10","S11","S4","S6","S3","S1","S2","S16","S1
3","S14","S15","S17", "Jap"))
# Plotting the barplot
strK6_barplot=ggplot(data=strK6.results, aes(x=Sample,
y=Posterior_membership_probability, fill=Assigned_Population))+
  geom_bar(stat='identity',width=1) +
  scale_fill_manual(values = cols.native) +
  facet_grid(~Original_Population, scales = "free",space="free_x") +
  labs(x="Sampled individual", y="Posterior\nadmixture\nproportions\n",
fill="Assigned\npopulation") +
  theme(axis.line = element_blank(),
        # axis.line.x = element_blank(), # No x axis
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        plot.title = element_text(color="black", size=28,
face="bold.italic",hjust = 0.5),
        plot.subtitle = element_text(color="black",size=28,hjust = 0.5),
        axis.title.x = element_text(color="black", size=28),
        axis.title.y = element_text(color="black", size=28),
        axis.text=element_text(size=28, colour="black"),
        axis.text.x=element_blank(), # No samples names
        axis.ticks.x=element_blank(), # No x axis
        strip.text=element_text(size=22, colour="black"),
        legend.key = element_rect(fill = "white", size = 1),
        legend.text=element_text(size=28),
        legend.title=element_text(size=28))
strK6_barplot
#
ggsave(paste(figuresdir,"/STRUCTURE/2b_Native/Native_barplot_K6b.png",sep="")
,
#        device="png",dpi=320,units="cm",width=60,height=9)
#-------------------------------------------------------
# Fig 3. Map of the native area with pies chart of admixture proportions on
each sampling location
# Prepare the data set for mapping
Native.pies.str=cbind(as.character(labelPops$Location_name[strK6[,4]]),as.dat
```

```
a.frame(strK6[,(c(1,2,3,5,4,6)+5)])) # Clusters are selected to correspodn to
DAPC colors
names(Native.pies.str)=c("Population", "Cluster 1", "Cluster 2", "Cluster 3",
"Cluster 4", "Cluster 5", "Cluster 6")
Native.pie.str_means=sapply(split(Native.pies.str[2:7],Native.pies.str$Popula
tion), colMeans) ## This just makes population cluster averages for each
cluster
# Mapping
# Download database of river network at scale 50
# rivers50=ne_download(scale = 50, type = 'rivers_lake_centerlines', category
= 'physical')
#
png("Figures/STRUCTURE/2a_Native/STR_Native_pies_map.png",width=2000,height=1
100)
png("Figures/STRUCTURE/2b_Native/STR_Native_pies_map_K6.png",width=2000,heigh
t=1100)
map("worldHires", xlim=c(92, 160), ylim=c(20,48), col="gray90",  fill=TRUE) #
Plot the map of Asian area
sp::plot(rivers50,lwd=2, col = 'blue',add=TRUE)
# Nomenclature
north.arrow(155,45,1, lab="N", lab.pos="above")
scalebar(c(150,20),1000, bg=NA, border=NA, division.cex=2)

text(x=105, y=34,"China",cex=3,font=2)
text(x=139, y=39,"Japan",cex=3,font=2)
points(native.coords$X, native.coords$Y, pch = 16, cex = 0.7,col="red") #\
Add points to the map
# map.scale(x=140,y=25,relwidth=0.1,cex=2)
## Pies ##
# Draw a pie with admixture proportions for each population
for (i in 1:ncol(Native.pie.str_means)) {
  add.pie(Native.pie.str_means[,i],x=native.coords$X[i],
y=native.coords$Y[i],labels=native.coords[i,1],label.dist=1.5,
          radius=1,edges=200,clockwise=T,   col =   cols.native,cex=2,font=2)
  # Add labels to pies
}
dev.off()

# Other values of K were tested (from K=2 to K=12), but we only displayed her
the chunk code for K=6 (the chosen K), in order to save paper.

#-------------------------------------------------------------
# Admixed populations: assess which population have less than 0.7 of mean Q
(ancestry coefficient)
#-------------------------------------------------------------
# A population where the mean of the max Q (the major ancestry coeff) over
all individuals is inferior to 0.7 should be considered as an admixed
population
# Get max Q for each individuals, and then mean(max Q) for each population,
for the chosen K: here K=6
```

```
# Qk is the proportion of individual's ancestry from population k
ancestry=strK6[,c(1,4,6:11)]
colnames(ancestry)=c("ind","pop","cluster1","cluster2","cluster3","cluster4",
"cluster5","cluster6")
ancestry$Qmax=apply(ancestry[,3:8],1,max)
# Get the mean (max Q)
pop.admixed=data.frame(pop=unique(ancestry$pop))
i=0
for (p in unique(ancestry$pop)) {
  print(p)
  i=i+1
  pop.admixed$Q[i]=mean(ancestry$Qmax[which(ancestry$pop==p)])

pop.admixed$pop[i]=as.character(labelPops$Location_name[which(labelPops$Locat
ion_index==p)])
}
pop.admixed$Q=round(pop.admixed$Q,digits = 3)
write.table(x=pop.admixed,file="Tables/Major ancestry coeff Native
K6.txt",sep="\t",quote = FALSE,row.names = FALSE)
```

The STRUCTURE analysis in the native range revealed a complex admixture-like pattern. Moreover, a hierarchical structure could be involved. We tested for it, but none hierarchical structure was assessed. We rejected all possible reasons that may have bias or reduced power in the current analysis. Hence, we must consider that this structure has a biological meaning.

We ran the same analysis on the invasive range and searched for hierarchical structure too (see results in report).

Checking convergence of multiple replicated sampling chains of STRUCTURE

In this part, the workflow for convergence assessment was computed on native populations' STRUCTURE analysis. It was the same workflow for all other analyses.

```
#=====================================================================
#                        CONVERGENCE ASSESSMENT OF STRUCTURE IN R
#=====================================================================

#=====================================================================
#                        CONVERGENCE ASSESSMENT FOR NATIVE POPULATIONS
#          & 100,000 ITERATIONS AFTER A BURNIN OF 100,000
#=====================================================================
# STRUCTURE was preliminary run with 100,000 ITERATIONS AFTER A BURNIN OF
100,000 in order to identify a close range of most probable K
# at a reasonable computational time
# However, we didn't expected to reach convergence because of the reduced
iterations and burnin
# We needed to import the trace of Alpha and posterior estimates of Ln P(D)
(parameters to estimate and the sampling chain)
# One file per value of K
for (k in 1:9) {
  assign(paste("sampling.chain.K",k,sep=""),
read.table(paste(STRdir,"/2b_Native/Results/sampling_chain_K",k,sep=""),heade
r=T,sep=" "))
```

```
  # assign(paste("sampling.chain.K",k,sep=""),
read.table(paste(STRdir,"/21_Native_1MillionIterations/Results/sampling_chain
_K",k,sep=""),header=T,sep=" "))
}
nrep=20 # Number of replicates of the sampling chain

#=============================================================================
#!# Checking Convergence Visually
#=============================================================================
#=============================================================================
# Alpha -- Sampling chain after burnin
#-----------------------------------------------------------------------------
# Trace plot of multiple replicates
nrep=10 # Reduce the number of replicates to increase clarity of trace plots
cols=c(brewer.pal(n = 12, name = "Paired"),brewer.pal(n = 8, name = "Set3"))
# Set a vector of 20 colors for 20 replicates
#-----------------------------------------------------------------------------
# Assessing convergence for K=2
k=2 # K value to test
for (i in 1:k) {
  a=paste("Alpha",i,"_K",k,sep="") # Legend of Y axis
  sampling.chain=get(paste("sampling.chain.K",k,sep=""))
  test.list = list()
  for (r in 1:nrep){
test.list[[r]]=as.mcmc(sampling.chain[sampling.chain$Run==r &
!is.na(sampling.chain$Ln.Like),(3+i)]) }
  rm(r)
  # just the sampling chain, after burnin
  traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
  rm(a)
  rm(test.list)
}
# Most sampling chains didn't converge for parameter Alpha in K=2 and 20
replicates
# High variance within and between chains
#-----------------------------------------------------------------------------
# Assessing convergence for K=3
k=3 # K value to test
for (i in 1:k) {
  a=paste("Alpha",i,"_K",k,sep="") # Legend of Y axis
  sampling.chain=get(paste("sampling.chain.K",k,sep=""))
  test.list = list()
  for (r in 1:nrep){
test.list[[r]]=as.mcmc(sampling.chain[sampling.chain$Run==r &
!is.na(sampling.chain$Ln.Like),(3+i)]) }
  rm(r)
  # just the sampling chain, after burnin
  traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
  rm(a)
  rm(test.list)
```

```
}
# Most sampling chains didn't converge for parameter Alpha in K=3 and 20
replicates
# High variance within and between chains
#-----------------------------------------------------------------------
# Assessing convergence for K=5
k=5 # K value to test
for (i in 1:k) {
  a=paste("Alpha",i,"_K",k,sep="") # Legend of Y axis
  sampling.chain=get(paste("sampling.chain.K",k,sep=""))
  test.list = list()
  for (r in 1:nrep){
test.list[[r]]=as.mcmc(sampling.chain[sampling.chain$Run==r &
!is.na(sampling.chain$Ln.Like),(3+i)]) }
  rm(r)
  # just the sampling chain, after burnin
  traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
  rm(a)
  rm(test.list)
}
# Some sampling chains seemed close to convergence for parameter Alpha in K=5
and 20 replicates
# But there were still high variance and different means inferred.
# Parallel chains didn't converged on the same value.
#-----------------------------------------------------------------------
# Assessing convergence for K=6
k=6 # K value to test
for (i in 1:k) {
  a=paste("Alpha",i,"_K",k,sep="") # Legend of Y axis
  sampling.chain=get(paste("sampling.chain.K",k,sep=""))
  test.list = list()
  for (r in 1:nrep){
test.list[[r]]=as.mcmc(sampling.chain[sampling.chain$Run==r &
!is.na(sampling.chain$Ln.Like),(3+i)]) }
  rm(r)
  # just the sampling chain, after burnin
  traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
  rm(a)
  rm(test.list)
}
# Some sampling chains seemed close to convergence for parameter Alpha in K=6
and 20 replicates
# But there were still high variance and different means inferred.
# Parallel chains didn't converged on the same value.
#-----------------------------------------------------------------------
# Assessing convergence for K=7
k=7 # K value to test
for (i in 1:k) {
  a=paste("Alpha",i,"_K",k,sep="") # Legend of Y axis
  sampling.chain=get(paste("sampling.chain.K",k,sep=""))
```

```
  test.list = list()
  for (r in 1:nrep){
test.list[[r]]=as.mcmc(sampling.chain[sampling.chain$Run==r &
!is.na(sampling.chain$Ln.Like),(3+i)]) }
  rm(r)
  # just the sampling chain, after burnin
  traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
  rm(a)
  rm(test.list)
}
# Some sampling chains seemed close to convergence for parameter Alpha in K=7
and 20 replicates
# But there were still high variance and different means inferred.
# Parallel chains didn't converged on the same value.
#!!!!!!!!!!!!!!!!#
# Alpha parameter globally converged in some samplign chains, while most
others chains didn't showed signs of convergence
# High variance whithin and between chains, and no clear graphical assessment
of convergence
# A pattern was distinctive: there was no convergence at all for K=2-3, but
convergence seemed to appear with higher K values
# There were stationarity of most sampling chains for K=6-7, despite some
unsteadiness and outliers chains
# We need to go further with more quantitative diagnostics...

#=========================================================================
# Ln Likelihood - Sampling chain after burnin
#-------------------------------------------------------------------------
par(mfrow=c(2,3))
cols=c(brewer.pal(n = 12, name = "Paired"),brewer.pal(n = 8, name = "Set3"))
# Set a vector of 20 colors for 20 replicated sampling chains
#-------------------------------------------------------------------------
# Assessing convergence for K=2
a="Ln Likelihood (K2)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K2[sampling.chain.K2$Run==r &
!is.na(sampling.chain.K2$Ln.Like),]$Ln.Lik) }
rm(r)
# just the sampling chain, after burnin
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a,cex=4)
rm(a)
rm(test.list)
#-------------------------------------------------------------------------#
# Assessing convergence for K=3
a="Ln Likelihood (K3)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K3[sampling.chain.K3$Run==r &
!is.na(sampling.chain.K3$Ln.Like),]$Ln.Lik) }
```

```
rm(r)
# just the sampling chain, after burnin
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
#-------------------------------------------------------------------------#
# Assessing convergence for K=4
a="Ln Likelihood (K4)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K4[sampling.chain.K4$Run==r &
!is.na(sampling.chain.K4$Ln.Like),]$Ln.Lik) }
rm(r)
# just the sampling chain, after burnin
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
#-------------------------------------------------------------------------#
# Assessing convergence for K=5
a="Ln Likelihood (K5)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K5[sampling.chain.K5$Run==r &
!is.na(sampling.chain.K5$Ln.Like),]$Ln.Lik) }
rm(r)
# just the sampling chain, after burnin
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
#-------------------------------------------------------------------------#
# Assessing convergence for K=6
a="Ln Likelihood (K6)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K6[sampling.chain.K6$Run==r &
!is.na(sampling.chain.K6$Ln.Like),]$Ln.Lik) }
rm(r)
# just the sampling chain, after burnin
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
#-------------------------------------------------------------------------#
# Assessing convergence for K=7
a="Ln Likelihood (K7)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K7[sampling.chain.K7$Run==r &
!is.na(sampling.chain.K7$Ln.Like),]$Ln.Lik) }
rm(r)
# just the sampling chain, after burnin
```

```
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
par(mfrow=c(1,1))

# Likelihood sampling chains seemed all convergent, graphically
# But only K=3 (range -53400;-53100) and K=7 (range -48200;-47600) showed
convergence of all replicates to a single mode
# Others K were multimodal, with different means between replicates
# 2 modes for K=2 (range -57200;-56200), 3 modes for K=4-5 (range -52200;-
50800 & -50700;-49200 respectively)
# and one mode with one outlier for K=6 (range -49200;-48400)

#-------------------------------------------------------------------------#
# r
#-------------------------------------------------------------------------#
# Assessing convergence for K=6
par(mfrow=c(2,3))
cols=c(brewer.pal(n = 12, name = "Paired"),brewer.pal(n = 8, name = "Set3"))
# Set a vector of colors
#----------------------------------
a="r (K2)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K2[sampling.chain.K2$Run==r &
!is.na(sampling.chain.K2$Ln.Like),]$r) }
rm(r)
# just the sampling chain, after burnin
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
#----------------------------------
a="r (K3)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K3[sampling.chain.K3$Run==r &
!is.na(sampling.chain.K3$Ln.Like),]$r) }
rm(r)
# just the sampling chain, after burnin
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
#----------------------------------
a="r (K4)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K4[sampling.chain.K4$Run==r &
!is.na(sampling.chain.K4$Ln.Like),]$r) }
rm(r)
# just the sampling chain, after burnin
```

```
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
#---------------------------------
a="r (K5)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K5[sampling.chain.K5$Run==r &
!is.na(sampling.chain.K5$Ln.Like),]$r) }
rm(r)
# just the sampling chain, after burnin
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
#---------------------------------
a="r (K6)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K6[sampling.chain.K2$Run==r &
!is.na(sampling.chain.K6$Ln.Like),]$r) }
rm(r)
# just the sampling chain, after burnin
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
#---------------------------------
a="r (K7)" # Legend of Y axis
test.list = list()
for (r in 1:nrep){ test.list[[r]] =
as.mcmc(sampling.chain.K7[sampling.chain.K7$Run==r &
!is.na(sampling.chain.K7$Ln.Like),]$r) }
rm(r)
# just the sampling chain, after burnin
traceplot(test.list,lty=1, col=cols,xlab="Iterations (x100)", ylab=a)
rm(a)
rm(test.list)
par(mfrow=c(1,1))
# The r parameter didn't seemed convergent, with a high variance within and
between chains

#=============================================================================
# Testing Within Chain Convergence & Stability
#=============================================================================
#=============================================================================
# Basic diagnostics of MCMC sampling chain & convergence assessment
#-----------------------------------------------------------------------------
# Basic statistics computed on the sampling chains
nrep=20
k=3
sampling.chain=get(paste("sampling.chain.K",k,sep=""))
```

```
# Alpha chains
for (rep in 1:nrep) { # With rep the index of the replicate
  cat("-----------------------------------------\nReplicate
number",rep,"\n")
  print(summary(as.mcmc(sampling.chain.K6[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),c(4:9)])))
}
# Ln Likelihood chain
for (rep in 1:nrep) { # With rep the index of the replicate
  cat("-----------------------------------------\nReplicate
number",rep,"\n")
  print(summary(as.mcmc(sampling.chain.K6[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),ncol(sampling.chain)-1])))
}# r parameter chain
for (rep in 1:nrep) { # With rep the index of the replicate
  cat("-----------------------------------------\nReplicate
number",rep,"\n")
  print(summary(as.mcmc(sampling.chain.K6[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),(3+2*k+1)])))
}
#-------------------------------------------------------------------------#
# Trace & density
#-------------------------------------------------------------------------#
rep=1 # Index of replicate to plot and summarize
########### Alphas
par(mfrow=c(3,2))
a.draws=as.mcmc(sampling.chain.K6[which(sampling.chain.K6$Run==rep &
!(is.na(sampling.chain.K6$Ln.Like))),4])
summary(a.draws)
plot(a.draws)
a.draws=as.mcmc(sampling.chain.K6[which(sampling.chain.K6$Run==rep &
!(is.na(sampling.chain.K6$Ln.Like))),5])
summary(a.draws)
plot(a.draws)
a.draws=as.mcmc(sampling.chain.K6[which(sampling.chain.K6$Run==rep &
!(is.na(sampling.chain.K6$Ln.Like))),6])
summary(a.draws)
plot(a.draws)
a.draws=as.mcmc(sampling.chain.K6[which(sampling.chain.K6$Run==rep &
!(is.na(sampling.chain.K6$Ln.Like))),7])
summary(a.draws)
plot(a.draws)
a.draws=as.mcmc(sampling.chain.K6[which(sampling.chain.K6$Run==rep &
!(is.na(sampling.chain.K6$Ln.Like))),8])
summary(a.draws)
plot(a.draws)
a.draws=as.mcmc(sampling.chain.K6[which(sampling.chain.K6$Run==rep &
!(is.na(sampling.chain.K6$Ln.Like))),9])
summary(a.draws)
plot(a.draws)
```

```
par(mfrow=c(1,1))
########## Ln Likelihood
LnLk.draws=as.mcmc(sampling.chain.K6[which(sampling.chain.K6$Run==rep &
!(is.na(sampling.chain.K6$Ln.Like))),ncol(sampling.chain)-1])
summary(LnLk.draws)
plot(LnLk.draws)
########## Parameter r
r.draws=as.mcmc(sampling.chain.K6[which(sampling.chain.K6$Run=rep &
!(is.na(sampling.chain.K6$Ln.Like))),(3+2*k+1)])
summary(r.draws)
plot(r.draws)
rm(rep)

#=========================================================================
# TESTS OF CONVERGENCE
#=========================================================================
# Subsequently, we ran a battery of diagnostic tests of convergence on the
sampling chains of Alpha, r and Ln Likelihood of K
#=========================================================================
# Geweke Diagnostic of MC Chain Stability
#-------------------------------------------------------------------------
# if the whole chain is stationary, the means of the values early and late in
the sequence should be similar
# convergence diagnostic 'Z' is the difference between the 2 means divided by
the asymptotic standard error of their difference
# values of 'Z' near the extreme tails of the N(0,1) indicates lack of
convergence
# can also estimate p-value of 'Z' from the normal distribution
# yields the probability that the divided chain means are different
?geweke.diag
#-------------------------------------------------------------------------
# Alpha
#-------------------------------------------------------------------------
# We investigated K=2-3-5-6-7 for 1 replicate and all alpha values at a time
# Remind that it were 100,000 burnin + 100,000 iterations
# We had to discard the burnin by not considering iterations with NA values
in Ln.likelihood
# First step: assess convergence
nrep=20
k=5
sampling.chain=get(paste("sampling.chain.K",k,sep=""))
convergent=c()
for (rep in 1:nrep) {
  cat("-----------------------------------------\nReplicate
number",rep,"\n")
  print(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]), frac1=0.1, frac2=0.5))
  # A p-value can be computed for this Z-score
  print(2*pnorm(-
abs(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
```

```
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]), frac1=0.1, frac2=0.5)$z)))
  # count the number of convergent chains in the replicate (i.e. p-value >
0.05)
  print(sum(2*pnorm(-
abs(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]), frac1=0.1,
frac2=0.5)$z))>0.05))
  convergent[rep]=sum(2*pnorm(-
abs(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]), frac1=0.1,
frac2=0.5)$z))>0.05)
  # As well as a diagnostic plot
  geweke.plot(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]), frac1=0.1, frac2=0.5,
nbins=100)
}
print(convergent)
sum(convergent==k) # number of totally convergent replicates
sum(convergent>=(k/2) & convergent!=k) # number of partially convergent
replicates (at least half of parameters)
rm(convergent)

#---------------------------------------------------------------------
# Ln Likelihood
#---------------------------------------------------------------------
# First step: assess convergence
nrep=20
k=7
sampling.chain=get(paste("sampling.chain.K",k,sep=""))
convergent=c()
for (rep in 1:nrep) {
  cat("-----------------------------------------\nReplicate
number",rep,"\n")
  print(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),ncol(sampling.chain)-1]), frac1=0.1,
frac2=0.5))
  # A p-value can be computed for this Z-score
  print(2*pnorm(-
abs(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),ncol(sampling.chain)-1]), frac1=0.1,
frac2=0.5)$z)))
  # count the number of convergent chains in the replicate (i.e. p-value >
0.05)
  print(sum(2*pnorm(-
abs(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),ncol(sampling.chain)-1]), frac1=0.1,
frac2=0.5)$z))>0.05))
  convergent[rep]=sum(2*pnorm(-
abs(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),ncol(sampling.chain)-1]), frac1=0.1,
```

```
frac2=0.5)$z))>0.05)
  # As well as a diagnostic plot
  geweke.plot(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),ncol(sampling.chain)-1]), frac1=0.1,
frac2=0.5)
}
print(convergent)
sum(convergent==1) # number of convergent replicates
rm(convergent)

#---------------------------------------------------------------------
# Parameter r
#---------------------------------------------------------------------
nrep=20
k=7
sampling.chain=get(paste("sampling.chain.K",k,sep=""))
convergent=c()
for (rep in 1:nrep) {
  cat("-----------------------------------------\nReplicate
number",rep,"\n")
  print(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),(3+2*k+1)]), frac1=0.1, frac2=0.5))
  # A p-value can be computed for this Z-score
  print(2*pnorm(-
abs(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),(3+2*k+1)]), frac1=0.1, frac2=0.5)$z)))
  # count the number of convergent chains in the replicate (i.e. p-value >
0.05)
  print(sum(2*pnorm(-
abs(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),(3+2*k+1)]), frac1=0.1,
frac2=0.5)$z))>0.05))
  convergent[rep]=sum(2*pnorm(-
abs(geweke.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),(3+2*k+1)]), frac1=0.1,
frac2=0.5)$z))>0.05)
  # As well as a diagnostic plot
  geweke.plot(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),(3+2*k+1)]), frac1=0.1, frac2=0.5)
}
print(convergent)
sum(convergent==1) # number of convergent replicates
rm(convergent)

#=====================================================================
# Heidelberger and Welch's Convergence Diagnostic
#---------------------------------------------------------------------
# probability of rejecting hypothesis that Markov Chain is a
stable/stationary distribution
# if Halfwidth test fails, chain should be extended
```

```
?heidel.diag
#-------------------------------------------------------------------------
# Alpha
#-------------------------------------------------------------------------
nrep=20
k=2
sampling.chain=get(paste("sampling.chain.K",k,sep=""))
for (rep in 1:nrep) {
  cat("-----------------------------------------------\nReplicate
number",rep,"\n")
  print(heidel.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))])))
}
# Convergence of Alpha was unclear, with high variability between replicates
#-------------------------------------------------------------------------
# Ln Likelihood
#-------------------------------------------------------------------------
nrep=20
k=7
sampling.chain=get(paste("sampling.chain.K",k,sep=""))
for (rep in 1:nrep) {
  cat("-----------------------------------------------\nReplicate
number",rep,"\n")
  print(heidel.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),ncol(sampling.chain)-1])))
}

#-------------------------------------------------------------------------
# Parameter r
#-------------------------------------------------------------------------
nrep=20
k=2
sampling.chain=get(paste("sampling.chain.K",k,sep=""))
for (rep in 1:nrep) {
  cat("-----------------------------------------------\nReplicate
number",rep,"\n")
  print(heidel.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep &
!(is.na(sampling.chain$Ln.Like))),(3+2*k+1)])))
}

#=========================================================================
# Raftery and Lewis' Diagnostic
#-------------------------------------------------------------------------
# run length control diagnostic
# intended for use on a short pilot run of a Markov chain
# to estimate number of iterations required to estimate the quantile 'q'
within an accuracy of +/- 'r' with probability 'p'
# only tests marginal convergence on each parameter
#!# high dependence factors (i.e. >5) are worrisome
# may indicate influential starting values, high correlations between
```

```
coefficients, or poor mixing
?raftery.diag
#-------------------------------------------------------------------------
# Alpha
#-------------------------------------------------------------------------
nrep=20
k=2
sampling.chain=get(paste("sampling.chain.K",k,sep=""))
for (rep in 1:nrep) {
  cat("-----------------------------------------\nReplicate
number",rep,"\n")
  #!!!!!!!! q=0.05 (default 0.025) and precision at 5% (0.05) to get results:
instead sample size was not enough

print(raftery.diag(as.mcmc(sampling.chain[which(sampling.chain$Run==rep),c(4:
(3+k))]), q=0.025, r=0.05, s=0.95))
}

#-------------------------------------------------------------------------
# Summarizing Run Length Inference
#-------------------------------------------------------------------------
r=1
chain.length=data.frame(k=g, Rep=r,
                        Alpha.N=raftery.diag(as.mcmc(subset(trace.files,
k==g & Rep==r)$Alpha), q=0.025, r=0.005, s=0.95)[[2]][,"N"],

Alpha.DepFactor=raftery.diag(as.mcmc(subset(trace.files, k==g &
Rep==r)$Alpha), q=0.025, r=0.005, s=0.95)[[2]][,"I"])
row.names(chain.length) = NULL
chain.length
for (r in 2:5)
{ temp.df = data.frame(k=g, Rep=r,
                       Alpha.N=raftery.diag(as.mcmc(subset(trace.files, k==g &
Rep==r)$Alpha), q=0.025, r=0.005, s=0.95)[[2]][,"N"],

Alpha.DepFactor=raftery.diag(as.mcmc(subset(trace.files, k==g &
Rep==r)$Alpha), q=0.025, r=0.005, s=0.95)[[2]][,"I"])
row.names(temp.df) = NULL
chain.length = rbind(chain.length, temp.df)
rm(temp.df)
}
chain.length
rm(g)
rm(r)
for (g in 2:7)
{ for (r in 1:5)
{ temp.df = data.frame(k=g, Rep=r,
                       Alpha.N=raftery.diag(as.mcmc(subset(trace.files, k==g &
Rep==r)$Alpha), q=0.025, r=0.005, s=0.95)[[2]][,"N"],
```

```
Alpha.DepFactor=raftery.diag(as.mcmc(subset(trace.files, k==g &
Rep==r)$Alpha), q=0.025, r=0.005, s=0.95)[[2]][,"I"])
row.names(temp.df) = NULL
chain.length = rbind(chain.length, temp.df)
rm(temp.df)
}
  rm(r)
}
rm(g)
chain.length
summary(chain.length$Alpha.N)
boxplot(Alpha.N~k, data=chain.length, xlab="k", ylab="Min. Chain Length
Required for Convergence", log="y", ylim=c(20000,500000), cex.lab=1.5)

#=============================================================================
# Testing Convergence Among Chains
#=============================================================================
#=============================================================================
# Gelman diagnostic: 'potential scale reduction factor'
#-----------------------------------------------------------------------------
?gelman.diag
#-----------------------------------------------------------------------------
# Alphas
#-----------------------------------------------------------------------------
# testing convergence among chains
#!# scaling factor should be less than 1.2
# values substantially greater than 1 indicate lack of convergence
# No autoburnin, as burnin had already been removed
nrep=20
k=2
sampling.chain=get(paste("sampling.chain.K",k,sep=""))
gelman.diag(mcmc.list(list(
  as.mcmc(sampling.chain[which(sampling.chain$Run==1 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==2 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==3 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==4 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==5 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==6 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==7 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==8 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==9 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
```

```
  as.mcmc(sampling.chain[which(sampling.chain$Run==10 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==11 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==12 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==13 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==14 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==15 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==16 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==17 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==18 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==19 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==20 &
!(is.na(sampling.chain$Ln.Like))),c(4:(3+k))])
)),autoburnin=FALSE)

#-----------------------------------------------------------------------
# Ln Likelihood
#-----------------------------------------------------------------------
k=7
sampling.chain=get(paste("sampling.chain.K",k,sep=""))
gelman.diag(mcmc.list(list(
  as.mcmc(sampling.chain[which(sampling.chain$Run==1 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==2 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==3 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==4 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==5 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==6 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==7 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==8 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==9 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==10 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
```

```
  as.mcmc(sampling.chain[which(sampling.chain$Run==11 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==12 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==13 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==14 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==15 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==16 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==17 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==18 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==19 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)]),
  as.mcmc(sampling.chain[which(sampling.chain$Run==20 &
!(is.na(sampling.chain$Ln.Like))),(ncol(sampling.chain)-1)])
)),autoburnin=FALSE)
```

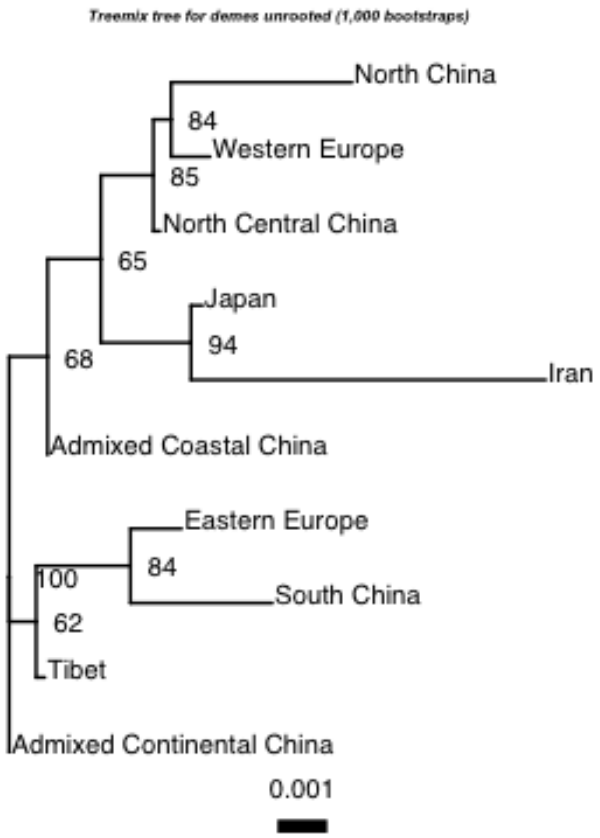
Treemix Bayesian phylogenetic tree

Treemix is a command-line tool computing Bayesian phylogenetic trees that could incorporate migration events between populations. First, data had to be formatted in a compatible input file. Then, Treemix was run in a UNIX cluster with command-lines provided in this script. Lastly, outputs were analyzed in R.

```
# loading dataset
load("Data/Pparva.3000.Rda")
#=========================================================
# TREEMIX INPUT
#=========================================================
# Produce the input dataset for TreeMix
# Take a genlight object in input
############
# Per population approach
Pparva.gl=gi2gl(Pparva)
gl2treemix(Pparva.gl, outfile = "Data/treemix_input_pop.gz", outpath = wd, v
= 3)
############
# Per demes approach
Pparva.demes=Pparva
v.pop=pop(Pparva.demes)
v.pop=as.character(v.pop)
v.pop[v.pop %in% c("Aus","Bel","Hun","Ita","Spa","Pol","UK")]="WesternEurope"
v.pop[v.pop %in% c("Bul1","Bul2","Tur")]="EasternEurope"
v.pop[v.pop %in% c("S1","S2","S3","S16")]="NorthCentralChina"
v.pop[v.pop %in% c("S13","S14","S15","S17")]="NorthChina"
```

```
v.pop[v.pop %in% c("S4","S6")]="AdmixedCoastalChina"
v.pop[v.pop %in% c("S9","S18","S19","S20")]="SouthChina"
v.pop[v.pop %in% c("S10","S11")]="AdmixedContinentalChina"
v.pop
pop(Pparva.demes)=v.pop
pop(Pparva.demes)
rm(v.pop)
Pparva.gl.demes=gi2gl(Pparva.demes)
gl2treemix(Pparva.gl.demes, outfile = "Data/treemix_input_demes.gz", outpath
= wd, v = 3)
# Then we need to gzip the input files, with the command in terminal

#============================================================
# TREEMIX ANALYSES, PERFORMED IN TERMINAL
#============================================================
# Command in terminal, inside the treemix data directory
# treemix -i input treemix_input_pop.gz -o out_pop
# To root the tree
# treemix -i input treemix_input_pop.gz -root Outgroup -o out_pop

# To generate a bootstrap replicate
# treemix -i treemix_input_pop.gz -bootstrap -k 500 -o replicate1
#============================================================
# TREEMIX RESULTS
#============================================================
setwd("~/Google Drive/INRA PSEUDORASBORA/Analyses/TreeMix/Analyses")
source("Sources/plotting_funcs.R") # Plotting functions for TreeMix
#============================================================
# PLOTS
#============================================================
# TREES
#
https://bioconductor.org/packages/release/bioc/vignettes/ggtree/inst/doc/tree
Annotation.html
# Trees from TreeMix are Newick
# Trees from bootstrap procedure of Treemix are NEXUS
# tree = read.tree("Demes/out_demes.treeout.gz") # TreeMix
tree = read.nexus("Demes/consensustree_demes") # Bootstrapped Treemix
tree$tip.label = c("Admixed Continental China", "Admixed Coastal China",
"Iran" ,"Japan", "Western Europe","North China", "North Central China",
"South China", "Eastern Europe", "Tibet")
treeplot = ggtree(tree) +
  geom_treescale(fontsize=3, linesize=2, width=0.001) +
  geom_tiplab(size=3,vjust="bottom") +
  geom_nodelab(aes(), hjust=-0.5, size=3) +
  scale_x_continuous(expand = c(.5, 0)) +
  ggtitle("Treemix tree for demes unrooted (1,000 bootstraps)") +
  theme(axis.line = element_line(colour = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
```

```
        panel.border = element_blank(),
        panel.background = element_blank(),
        plot.title = element_text(color="black", size=5,
face="bold.italic",hjust = 0.5),
        plot.subtitle = element_text(color="black",size=5,hjust = 0.5),
        axis.title.x = element_text(color="black", size=5),
        axis.title.y = element_text(color="black", size=5),
        axis.text=element_text(size=5, colour="black"),
        legend.key = element_rect(fill = "white", size = 1),
        legend.text=element_text(size=5),
        legend.title=element_text(size=5))
treeplot
```




```
# ggsave("Figures/ggtree_demes.png",device =
"png",dpi=150,width=35,height=15)
setwd("~/Google Drive/INRA PSEUDORASBORA/Analyses/R")
```

Population assignment with 'AssignPOP'

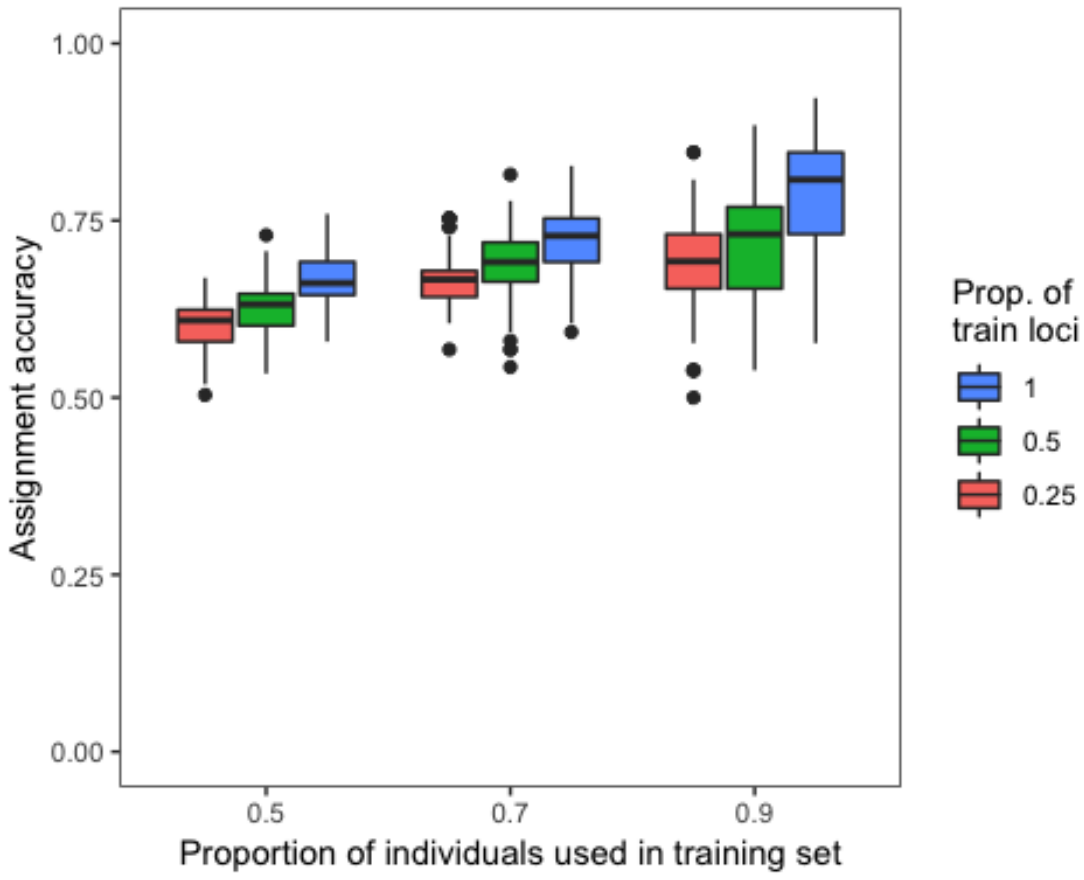
AssignPOP is a Machine-Learning framework designed to assign individuals of unknown origin to known populations. The first phase is a training with individuals of known origin (source popualtions), i.e. individuals of the native. Then, origins of individuals in the invasive range can be predicted…

```
load(paste(datadir,"/Pparva.3000.Rda",sep=""))
#=============================================================
#  DATA
#=============================================================
# Known populations names
native.names=c("Jap", "S1", "S2", "S3", "S4", "S6","S9",
               "S10", "S11", "S13", "S14", "S15", "S16", "S17", "S18", "S19",
"S20", "Tib")
invasive.names=c("Aus", "Bel", "Bul1", "Bul2", "Hun", "Ira", "Ita", "Pol",
"Spa", "Tur", "UK")
#-------------------------------
# Format in genepop file
training.pop = native.names[-c(4,18)]
training.demes = c("Japan", "Coastal_Admixed_China",
"Continental_Admixed_China", 'North_Central_China', "North_China",
"South_China")

genetic.training = read.Genepop("AssignPOP/Data/genepop_training_demes.txt",
pop.names = training.demes) # Inferred putative demes in Asia to train the
model

#=============================================================
#  TRAINING
#=============================================================
# https://alexkychen.github.io/assignPOP/analyze.html

# Remove loci with low variance
# genetic.training = reduce.allele(genetic.training, p = 0.95)

#=============================================================
# Perform resampling cross-validation (training of the model)
#-------------------------------------------------------------
# 2 resampling methods to evaluate baseline data

# Monte-Carlo cross-validation helps estimate the mean and variance of
assignment accuracy
# through resampling random training individuals

# K-fold cross-validation helps determine membership probability across all
individuals
# through using one group as test individuals and the remaining K-1 groups as
training individuals
# (hence every individual is tested once).

#-------------------------------------------------------------
# Monte-Carlo cross-validation by Random Forest
#-------------------------------------------------------------
# A total of 360 assignment tests was performed
# (3 levels of training individuals by 4 levels of training loci by 30
iterations)
```

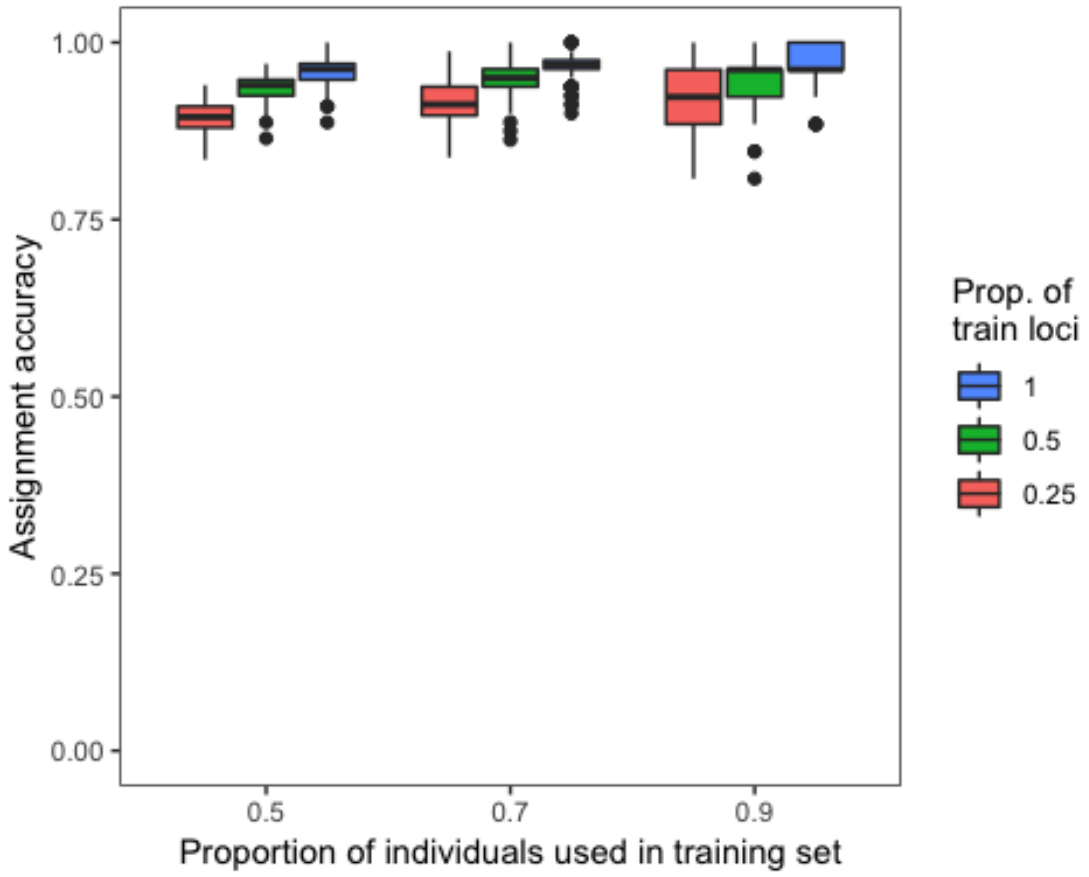
```r
# assign.MC(genetic.training, train.inds=c(0.5, 0.7, 0.9), train.loci=c(0.1,
0.25, 0.5, 1),
#           loci.sample="fst", iterations=30, model="randomForest",
dir="AssignPOP/assignPOP.MC/",
#           multiprocess = TRUE) # Beware of overheating when multiprocessing
!

# To improve assignment quality:
# - add more iterations
# - do not use training loci level of 0.1 (less efficient than others, high
error rate)
# (3 levels of training individuals by 3 levels of training loci by 100
iterations) = 900 assignment tests
assign.MC(genetic.training, train.inds=c(0.5, 0.7, 0.9), train.loci=c(0.25,
0.5, 1),
          loci.sample="fst", iterations=100, model="randomForest",
dir="AssignPOP/assignPOP.MC.randomForest/",
          multiprocess = TRUE) # Beware of overheating when multiprocessing !
# Try the Support Vector Machine (SVM) algorithm
assign.MC(genetic.training, train.inds=c(0.5, 0.7, 0.9), train.loci=c(0.25,
0.5, 1),
          loci.sample="fst", iterations=100, model="svm",
dir="AssignPOP/assignPOP.MC.svm/",
          multiprocess = TRUE) # Beware of overheating when multiprocessing !
# Try the naive Bayes algorithm
assign.MC(genetic.training, train.inds=c(0.5, 0.7, 0.9), train.loci=c(0.25,
0.5, 1),
          loci.sample="fst", iterations=100, model="naiveBayes",
dir="AssignPOP/assignPOP.MC.naiveBayes/",
          multiprocess = TRUE) # Beware of overheating when multiprocessing !

############################
# Assignment on demes
# randomForest
assign.MC(genetic.training, train.inds=c(0.5, 0.7, 0.9), train.loci=c(0.25,
0.5, 1),
          loci.sample="fst", iterations=100, model="randomForest",
dir="AssignPOP/assignPOP.MC.randomForest.demes/",
          multiprocess = TRUE) # Beware of overheating when multiprocessing !
# Support Vector Machine (SVM) algorithm
assign.MC(genetic.training, train.inds=c(0.5, 0.7, 0.9), train.loci=c(0.25,
0.5, 1),
          loci.sample="fst", iterations=100, model="svm",
dir="AssignPOP/assignPOP.MC.svm.demes/",
          multiprocess = TRUE) # Beware of overheating when multiprocessing !
# naive Bayes algorithm
assign.MC(genetic.training, train.inds=c(0.5, 0.7, 0.9), train.loci=c(0.25,
0.5, 1),
          loci.sample="fst", iterations=100, model="naiveBayes",
dir="AssignPOP/assignPOP.MC.naiveBayes.demes/",
```

```
          multiprocess = TRUE) # Beware of overheating when multiprocessing !

#-------------------------------------------------------------
# K-fold cross validation
#-------------------------------------------------------------
# K-fold cross-validation in which individuals from each population are
divided into K groups. One of the K groups is tested by the predictive model
built based on the remaining K-1 groups. Such an assignment test repeats
until every group was tested.
#  K-fold cross-validation performed a total of 99 assignment tests (99 =
(3+4+5+6+7+8) folds * 3 levels of training loci)
assign.kfold(genetic.training, k.fold=c(3,4,5,6,7,8), train.loci=c(0.25, 0.5,
1),
              loci.sample="random", model="lda",
dir="AssignPOP/assignPOP.Kfold/",
             multiprocess = FALSE)
# Trainings are stored in files, in their directory
# They don't have to be run again, once correctly parameterized

#=============================================================
# Visualize results
#-------------------------------------------------------------

#-------------------------------------------------------------
# Calculate assignment accuracy
#-------------------------------------------------------------
# Assignment matrix heatmap
# Self assignment to the known population
# At this step, assignment accuracies can help to choose a training algorithm
and training parameters
# to achieve the best predictive model
# When resampling cross-validations are done, calculate assignment accuracies

accu.MC.randomForest = accuracy.MC(dir =
"AssignPOP/assignPOP.MC.randomForest/")

accu.MC.svm = accuracy.MC(dir = "AssignPOP/assignPOP.MC.svm/")

##
##   Correct assignment rates were estimated!!
##   A total of 901 assignment tests for 16 pops.
##   Results were also saved in a 'Rate_of_901_tests_16_pops.txt' file in the
directory.

accu.MC.naiveBayes = accuracy.MC(dir = "AssignPOP/assignPOP.MC.naiveBayes/")

# For demes:
accu.MC.randomForest.demes = accuracy.MC(dir =
"AssignPOP/assignPOP.MC.randomForest.demes/")
```

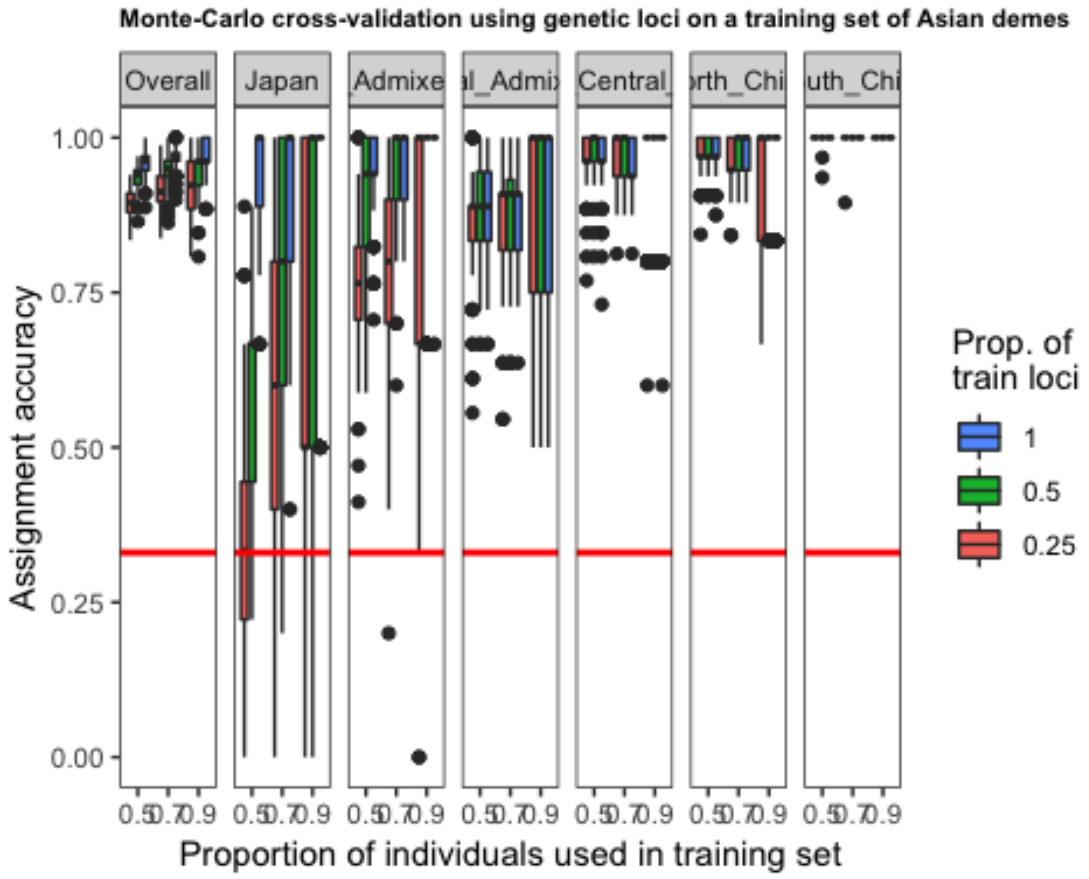
```
accu.MC.svm.demes = accuracy.MC(dir = "AssignPOP/assignPOP.MC.svm.demes/")

##
##   Correct assignment rates were estimated!!
##   A total of 900 assignment tests for 6 pops.
##   Results were also saved in a 'Rate_of_900_tests_6_pops.txt' file in the
directory.

accu.MC.naiveBayes.demes = accuracy.MC(dir =
"AssignPOP/assignPOP.MC.naiveBayes.demes/")
# K-fold cross validation procedure
accu.Kfold = accuracy.kfold(dir = "AssignPOP/assignPOP.Kfold/")
# or read directly the results if already computed
# accu.MC = read.table("AssignPOP/assignPOP.MC/Rate_of....txt", header=T)
# accu.Kfold = read.table("AssignPOP/assignPOP.Kfold/Rate_of....txt",
header=T)

#-----------------------------------------
# Assignment accuracy plots
# On populations:
accuracy.plot(accu.MC.svm, pop = "all") + ylim(0,1)
```




```
# On demes:
accuracy.plot(accu.MC.svm.demes, pop = "all") + ylim(0,1)
```




```
# Demes had better performance for assignment accuracy in training set, ranging from 0.85 to 1
accuracy.plot(accu.MC.svm.demes, pop=c("all", training.demes)) +
  ylim(0, 1) + #Set y limit between 0 and 1
  annotate("segment",x=0.4,xend=3.6,y=0.33,yend=0.33,colour="red",size=1) + #Add a red horizontal line at y = 0.33 (null assignment rate for 3 populations)
  ggtitle("Monte-Carlo cross-validation using genetic loci on a training set of Asian demes")+ #Add a plot title
  theme(plot.title = element_text(size=8, face="bold")) #Edit plot title text size
```



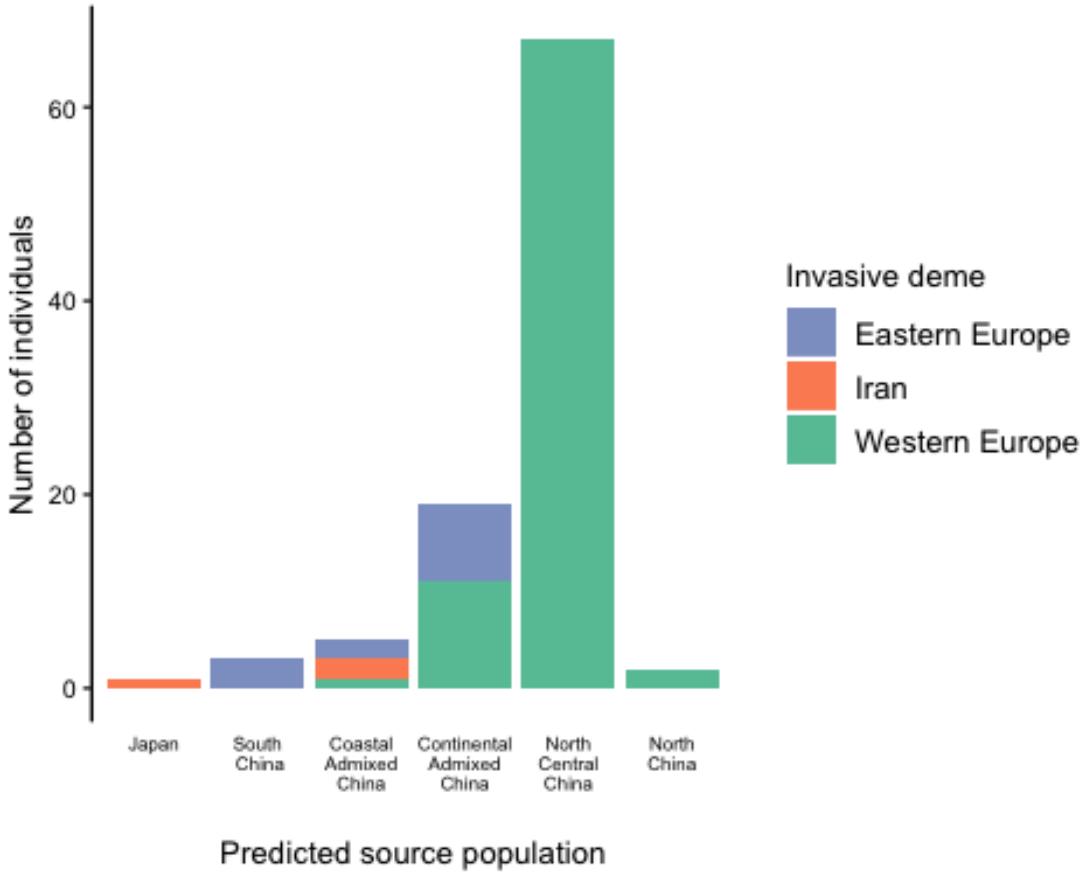

```
#-------------------------------------------------------------
# Identify informative loci
#-------------------------------------------------------------
# In some cases, using a subset of high FST loci may produce similar
assignment accuracy with using all available loci.
# Identification of these loci not only could help reduce time and cost in
preparing samples in the future
# but also could help identify loci that might be associated with functional
genes.
# This function should only be used for the cross-validation results based on
# resampling high FST training loci (loci.sample = "fst") in the assign.MC()
analysis.
# The 200 best loci on 3,000
source("Sources/check.loci.R")
check.loci(dir = "AssignPOP/assignPOP.MC.svm/", top.loci = 20)
# Results are saved in a 'High_Fst_Locus_Freq.txt' file in the directory.
# top N informative loci in N rows, and each row has a list of loci sorted by
its occurrence
best.loci = read.table("AssignPOP/assignPOP.MC.svm/High_Fst_Locus_Freq.txt",
sep=" ",
                       header = FALSE, skip = 1, fill = TRUE)
# A large data set can be reduced to a smaller data set of the most
informative loci with lots of advantages
```

```
# including reduced calculation time and memory usage

#============================================================
#  ASSIGNMENT
#============================================================
# Predict source of unknown individuals (invasive populations)
# All European populations at once
genetic.assign = read.Genepop("Data/genepop_invasive.txt") # unknown
populations to assign to native populations
#------------------------------------------------------------
# Use training with MC
# 1.Perform assignment test using genetic data
# Results stored in a file 'AssignmentResults.txt'
assign.X(x1=genetic.training, x2=genetic.assign,
dir="AssignPOP/assignPOP.MC.svm/", model="svm", mplot = TRUE)
# membership.plot(dir = "AssignPOP/assignPOP.MC.svm/")

# Results stored in a file:
res=read.table(("AssignPOP/assignPOP.MC.svm/AssignmentResult.txt"),
header=TRUE)
table(res$pred.pop, res$Ind.ID) # predicted populations as a function of
invasive population
# Validate the results: print mean and standard deviation across assignment
tests
assign.matrix(dir="AssignPOP/assignPOP.MC.svm/", train.inds=c(0.9),
train.loci=c(0.5))
# Distribution of predicted source populations
table(res$pred.pop)
plot(res$pred.pop)
# Distribution of predicted source populations for each deme
plot(res$pred.pop[which(res$Ind.ID %in% c("Aus", "Bel", "Hun", "Ita", "Pol",
"Spa", "UK"))], main = "Assignment test of Western Europe")
plot(res$pred.pop[which(res$Ind.ID %in% c("Bul1", "Bul2", "Tur"))], main =
"Assignment test of Eastern Europe")
plot(res$pred.pop[which(res$Ind.ID %in% c("Ira"))], main = "Assignment test
of Iran")

# Distribution of predictions probabilities as error assessment
# Same plot for posterior probability as for assignment accuracy of training
apply(res[,-c(1,2)], 1, max) # probabilities of predicted populations for
each individual
hist(apply(res[,-c(1,2)], 1, max), breaks = 20, main = "Distribution of
posterior probabilities", xlab = "Posterior probability", cex = 2)

# 2.Perform assignment test using decision tree
assign.X(x1=genetic.training, x2=genetic.assign,
dir="AssignPOP/assignPOP.MC/", model="tree", mplot = TRUE)
# 3.Perform assignment test using random forest
assign.X(x1=genetic.training, x2=genetic.assign,
dir="AssignPOP/assignPOP.MC/", model="randomForest", ntree=100, mplot = TRUE)
```

```
#-----------------------------------------------------------
# Use training with K fold
assign.X(x1=genetic.training, x2=genetic.assign,
dir="AssignPOP/assignPOP.Kfold/", model="naiveBayes", mplot = TRUE)
membership.plot(dir = "AssignPOP/assignPOP.Kfold/")
assign.X(x1=genetic.training, x2=genetic.assign,
dir="AssignPOP/assignPOP.Kfold/", model="tree", mplot = TRUE)
assign.X(x1=genetic.training, x2=genetic.assign,
dir="AssignPOP/assignPOP.Kfold/", model="randomForest", ntree=100, mplot =
TRUE)
#===========================================================
# ASSIGNMENT TO SOURCE DEMES
#===========================================================
# Use training with SVM
# 1.Perform assignment test using genetic data
# Results stored in a file 'AssignmentResults.txt'
assign.X(x1=genetic.training, x2=genetic.assign,
dir="AssignPOP/assignPOP.MC.svm.demes/", model="svm", mplot = TRUE)
# membership.plot(dir = "AssignPOP/assignPOP.MC/")

# Results stored in a file:
res=read.table(("AssignPOP/assignPOP.MC.svm.demes/AssignmentResult.txt"),
header=TRUE)
table(res$pred.pop, res$Ind.ID) # predicted populations as a function of
invasive population

##
##                              Aus Bel Bul1 Bul2 Hun Ira Ita Pol Spa Tur UK
##   Coastal_Admixed_China        1   0    1    3   0   5   1   0   0   4  0
##   Continental_Admixed_China    1   2    9    3   0   0  17   0   1  13  0
##   Japan                        0   0    0    0   0   3   0   0   0   0  0
##   North_Central_China         15  13    0    0  20   2   0  11  17   0 13
##   North_China                  0   2    0    0   2   0   0   1   1   0  1
##   South_China                  0   0    0    4   0   0   0   0   0   2  0

# Distribution of predicted source populations for each deme
# par(mfrow=c(1,3))
# plot(res$pred.pop[which(res$Ind.ID %in% c("Aus", "Bel", "Hun", "Ita",
"Pol", "Spa", "UK"))], main = "Assignment test of Western Europe")
# plot(res$pred.pop[which(res$Ind.ID %in% c("Bul1", "Bul2", "Tur"))], main =
"Assignment test of Eastern Europe")
# plot(res$pred.pop[which(res$Ind.ID %in% c("Ira"))], main = "Assignment test
of Iran")
# par(mfrow=c(1,1))
# Distribution of predictions probabilities as error assessment
# Same plot for posterior probability as for assignment accuracy of training
# apply(res[,-c(1,2)], 1, max) # probabilities of predicted populations for
each individual
# hist(apply(res[,-c(1,2)], 1, max), breaks = 20, main = "Distribution of
posterior probabilities", xlab = "Posterior probability", cex = 2)
# Relative probabilities, as implemented in Schmidt et al. 2019
```

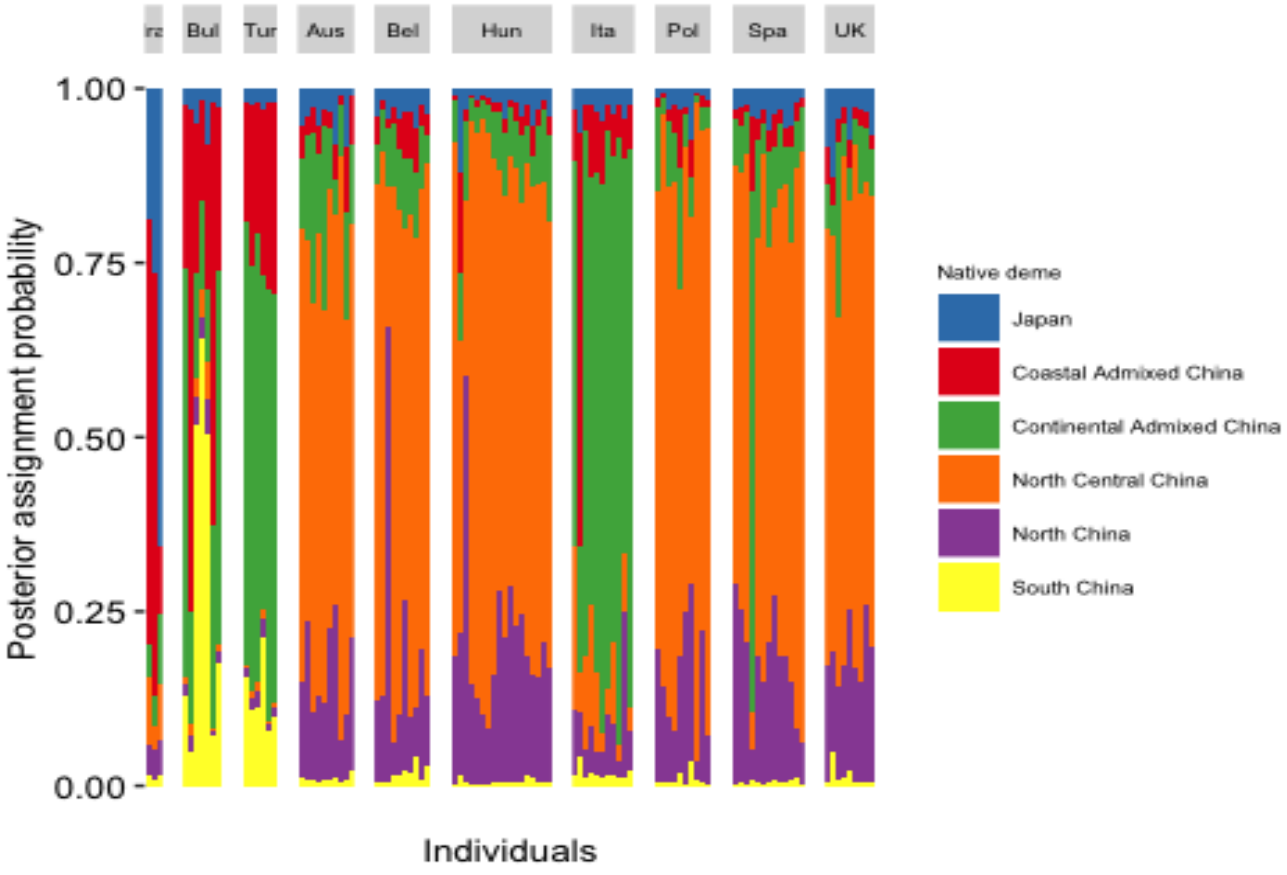
```
# probability of membership to the most likely population divided by the
probability of membership to the second most likely population
# Individuals are considered as correctly assigned if the relative
probability > 2
# i.e. the first probability is at least twice the next one.
rel.proba = c()
for (i in 1:nrow(res)) {
  rel.proba[i] = as.numeric(sort(res[i,-c(1,2)], decreasing =
TRUE)[1]/sort(res[i,-c(1,2)], decreasing = TRUE)[2])
}
# hist(as.numeric(rel.proba), breaks = 20)
sum(rel.proba > 2) # 100/168 individuals can be considered as correctly
assigned (59%), this is much better than assignment to populations

## [1] 97

res$rel.proba = as.numeric(rel.proba)
# table(res$pred.pop[res$rel.proba>2])
# Main sources were Continental admixed China and North Central China
res.trim = res[res$rel.proba>2,] # 3 invasive demes represented (e.g. 4
successes for Iran)
# Distribution of predictions probabilities of trimmed individuals
# Same plot for posterior probability as for assignment accuracy of training
# apply(res.trim[,3:8], 1, max) # probabilities of predicted populations for
each individual
# hist(apply(res.trim[,3:8], 1, max), breaks = 20, main = "Distribution of
posterior probabilities", xlab = "Posterior probability", cex = 2)

#---------------------------------
cols.invasive = c("#8DA0CB", "#FC8D62", "#66C2A5")
# Replace pop name by deme name, for legend
res.trim$Invasive.deme = as.character(res.trim$Ind.ID)
res.trim$Invasive.deme[res.trim$Invasive.deme %in% c("Ira")] = "Iran"
res.trim$Invasive.deme[res.trim$Invasive.deme %in% c("Bul1", "Bul2", "Tur")]
= "Eastern Europe"
res.trim$Invasive.deme[res.trim$Invasive.deme %in% c("Aus", "Bel", "Hun",
"Ita", "Pol", "Spa", "UK")] = "Western Europe"
# Demes are displayed in this order...
res.trim$pred.pop=factor(res.trim$pred.pop, levels =
c("Japan","South_China","Coastal_Admixed_China","Continental_Admixed_China","
North_Central_China","North_China"))
PredSource.plot = ggplot(data=res.trim, aes(x=pred.pop, fill=Invasive.deme))+
    geom_bar(width=0.9) +
    scale_x_discrete(labels = c("Japan" = "Japan","South_China" = "South\n
China",
"Coastal_Admixed_China" =
"Coastal\nAdmixed\nChina","Continental_Admixed_China" =
"Continental\nAdmixed\nChina","North_Central_China" =
"North\nCentral\nChina", "North_China" = "North\nChina")) +
    scale_fill_manual(values = cols.invasive) +
```

```
    labs(x="\nPredicted source population", y="Number of individuals",
fill="Invasive deme") +
    theme(axis.line = element_blank(),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank(),
          plot.title = element_text(color="black", size=10,
face="bold.italic",hjust = 0.5),
          plot.subtitle = element_text(color="black",size=10,hjust = 0.5),
          axis.title.x = element_text(color="black", size=10),
          axis.title.y = element_text(color="black", size=10),
          axis.text=element_text(size=8, colour="black"),
          axis.text.x=element_text(size=6, colour="black"),
          strip.text=element_text(size=10, colour="black"),
          axis.ticks.x = element_blank(),
          axis.line.y =
element_line(color="black",size=0.5,linetype="solid"),
          # axis.line.x =
element_line(color="black",size=1,linetype="solid"),
          legend.key = element_rect(fill = "white", size = 1),
          legend.text=element_text(size=10),
          legend.title=element_text(size=10))
PredSource.plot
```




```
df = res.trim[,-c(2,9)]
df$Ind = as.factor(seq(1,nrow(df)))
df.melt=melt(df)
colnames(df.melt)=c("Pop","inv.demes","Ind","source.demes","Posterior_assignm
ent_probability")
cols.demes=c("#377EB8", "#E41A1C", "#4DAF4A", "#FF7F00", "#984EA3",
"#FFFF33", "#A65628", "#F781BF","#8DD3C7", "#FFFFB3", "#BEBADA", "#FB8072",
"#80B1D3", "#FDB462", "#B3DE69", "#FCCDE5", "#D9D9D9")
# Levels in order: Iran, Eastern Europe and Western Europe
# df.melt$inv.demes = as.factor(df.melt$inv.demes)
# df.melt$inv.demes = factor(df.melt$inv.demes, levels = c("Iran", "Eastern
Europe", "Western Europe"))
df.melt$Pop = as.character(df.melt$Pop)
df.melt$Pop[df.melt$Pop %in% c("Bul1", "Bul2")] = "Bul"
df.melt$Pop = as.factor(df.melt$Pop)
df.melt$Pop = factor(df.melt$Pop, levels = c("Ira", "Bul", "Tur","Aus",
"Bel", "Hun", "Ita", "Pol", "Spa", "UK"),labels = c("Ira", "Bul",
"Tur","Aus", "Bel", "Hun", "Ita", "Pol", "Spa", "UK"))
PredSource.prop.plot = ggplot(data=df.melt, aes(x=Ind, y =
Posterior_assignment_probability, fill=source.demes))+
    geom_col(width=1) +
    facet_grid(~Pop, scales = "free",space="free_x") +
    scale_fill_manual(values = cols.demes, labels = c("Japan" =
```

```
"Japan","South_China" = "South China","Coastal_Admixed_China" = "Coastal
Admixed China",
"Continental_Admixed_China" = "Continental Admixed China",
"North_Central_China" = "North Central China","North_China" = "North China"))
+
    labs(x="Individuals", y="Posterior assignment probability", fill="Native
deme") +
    theme(axis.line = element_blank(),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank(),
          plot.title = element_text(color="black", size=10,
face="bold.italic",hjust = 0.5),
          plot.subtitle = element_text(color="black",size=10,hjust = 0.5),
          axis.title.x = element_text(color="black", size=10),
          axis.title.y = element_text(color="black", size=10),
          axis.text=element_text(size=10, colour="black"),
          axis.text.x=element_blank(),
          axis.ticks.x=element_blank(), # No x axis
          strip.text=element_text(size=6, colour="black"),
          legend.key = element_rect(fill = "white", size = 1),
          legend.text=element_text(size=6),
          legend.title=element_text(size=6))
PredSource.prop.plot
```



Coaslescent simulations with DIY ABC

This script performed the sampling of populations included in different scenarios in DIY ABC, and the output of files formatted for DIY ABC. Then DIY ABC was parameterised and simulations were produced…

```
# RE-IMPORT ABC.DF
load("Data/abc.df.Rda")
N.loci=ncol(abc.df)-3
#-----------------------------------------------
# Custom data set for scenarios
################################
# Scenarios for routes of invasion in Europe:
# S3 systematically failed in population assignment ('assignPOP')
# Besides, sample size is only 8, with high admixture rates in this
particulaer population
# Hence, S3 was discarded for any subsequent analysis

############################
# Western European invasion (SIMPLE)
abc.df.tmp=rbind(abc.df[which(abc.df$pop %in% c("S1", "S2",
"S16","S10","S11",
                                                "S13", "S14", "S15", "S17",
                                                "Aus", "Bel", "Hun", "Pol",
```

```
"Spa", "UK", "Ita")),]
)
# Scenario with 3 candidate native pop. (the one tested)
# Cluster sampled locations into demes
abc.df.tmp[,3]=as.character(abc.df.tmp[,3])
abc.df.tmp[which(abc.df.tmp$pop %in% c("Aus", "Bel", "Hun", "Pol", "Spa",
"UK", "Ita")),3]="1" # Western Europe
abc.df.tmp[which(abc.df.tmp$pop %in% c("S1", "S2", "S16")),3]="2" # North
central China, without S3 (unresolved deme assignment)
abc.df.tmp[which(abc.df.tmp$pop %in% c("S13", "S14", "S15", "S17")),3]="3" #
North China
abc.df.tmp[which(abc.df.tmp$pop %in% c("S10","S11")),3]="4" # Admixed
continental China
# abc.df.tmp[which(abc.df.tmp$pop %in% c("S4","S6")),3]="5" # Admixed coastal
China
# Reorder the dataset, DIY ABC names pop in incoming order
abc.df.tmp = abc.df.tmp[order(abc.df.tmp$pop),]
abc.df.tmp$pop
file.create("Data/ABC/ABC_Pparva_SourceWesternEurope.txt")
write.table("Phylogeography of Pseudorasbora parva: Western Europe Origin
(simple) <NM=1.0NF>
<MAF=hudson>","Data/ABC/ABC_Pparva_SourceWesternEurope.txt",sep="
",quote=FALSE,row.names=F,col.names=F)
write(paste("IND SEX POP", paste(rep("A",N.loci),collapse = " "), sep="
"),file="Data/ABC/ABC_Pparva_SourceWesternEurope.txt",append=TRUE)
for (i in 1:nrow(abc.df.tmp)) {
  write(paste(abc.df.tmp[i,],collapse="
"),file="Data/ABC/ABC_Pparva_SourceWesternEurope.txt",append=TRUE)
}
rm(abc.df.tmp)
############################
# Eastern European invasion: we tested Eastern Europe source from 3 most
probable origins
abc.df.tmp=rbind(abc.df[which(abc.df$pop %in% c("S4", "S6", "S10", "S11",
"S9", "S18", "S19", "S20",
                                                "Bul1", "Bul2", "Tur")),]
)
# Cluster sampled locations into demes
abc.df.tmp[,3]=as.character(abc.df.tmp[,3])
abc.df.tmp[which(abc.df.tmp$pop %in% c("Bul1", "Bul2", "Tur")),3]="1" #
Eastern Europe
abc.df.tmp[which(abc.df.tmp$pop %in% c("S4","S6")),3]="2" # Admixed coastal
China
abc.df.tmp[which(abc.df.tmp$pop %in% c("S10","S11")),3]="3" # Admixed
continental China
abc.df.tmp[which(abc.df.tmp$pop %in% c("S9", "S18", "S19", "S20")),3]="4" #
South China
# Reorder the dataset, DIY ABC names pop in incoming order
abc.df.tmp = abc.df.tmp[order(abc.df.tmp$pop),]
abc.df.tmp$pop
```

```
file.create("Data/ABC/ABC_Pparva_SourceEasternEurope.txt")
write.table("Phylogeography of Pseudorasbora parva: Eastern Europe Origin
(simple) <NM=1.0NF>
<MAF=hudson>","Data/ABC/ABC_Pparva_SourceEasternEurope.txt",sep="
",quote=FALSE,row.names=F,col.names=F)
write(paste("IND SEX POP", paste(rep("A",N.loci),collapse = " "), sep="
"),file="Data/ABC/ABC_Pparva_SourceEasternEurope.txt",append=TRUE)
for (i in 1:nrow(abc.df.tmp)) {
  write(paste(abc.df.tmp[i,],collapse="
"),file="Data/ABC/ABC_Pparva_SourceEasternEurope.txt",append=TRUE)
}
rm(abc.df.tmp)

############################
# Iranese invasion (SIMPLE): we tested Iran source from 3 most probable
source origins
abc.df.tmp=rbind(abc.df[which(abc.df$pop %in% c("S4", "S6", "Jap", "S1",
"S2", "S16",
                                                "Ira")),]
)
# Cluster sampled locations into demes
abc.df.tmp[,3]=as.character(abc.df.tmp[,3])
abc.df.tmp[which(abc.df.tmp$pop %in% c("Ira")),3]="1" # Iran
abc.df.tmp[which(abc.df.tmp$pop %in% c("S4","S6")),3]="2" # Admixed coastal
China
abc.df.tmp[which(abc.df.tmp$pop %in% c("Jap")),3]="3" # Japan
abc.df.tmp[which(abc.df.tmp$pop %in% c("S1", "S2", "S16")),3]="4" # North
central China
# Reorder the dataset, DIY ABC names pop in incoming order
abc.df.tmp = abc.df.tmp[order(abc.df.tmp$pop),]
abc.df.tmp$pop
file.create("Data/ABC/ABC_Pparva_SourceIran.txt")
write.table("Phylogeography of Pseudorasbora parva: Iran Origin (simple)
<NM=1.0NF> <MAF=hudson>","Data/ABC/ABC_Pparva_SourceIran.txt",sep="
",quote=FALSE,row.names=F,col.names=F)
write(paste("IND SEX POP", paste(rep("A",N.loci),collapse = " "), sep="
"),file="Data/ABC/ABC_Pparva_SourceIran.txt",append=TRUE)
for (i in 1:nrow(abc.df.tmp)) {
  write(paste(abc.df.tmp[i,],collapse="
"),file="Data/ABC/ABC_Pparva_SourceIran.txt",append=TRUE)
}
rm(abc.df.tmp)

#=========================================================
# GLOBAL SCENARIOS OF INVASION
#=========================================================
# Worldwide pathways, including all invasive demes in the same scenario
# Scenarios were constructed based on 3 previous independent analysis (demes
analysis)
abc.df.tmp=rbind(abc.df[which(abc.df$pop %in% c("S4", "S6", "S1", "S2",
```

```
"S16","S10","S11",
                                                    "S9", "S18", "S19", "S20",
"Jap",
                                                    "Aus", "Bel", "Hun", "Ita",
"Pol", "Spa", "UK",
                                                    "Bul1", "Bul2", "Tur",
"Ira")),]
)
# Cluster sampled locations into demes
abc.df.tmp[,3]=as.character(abc.df.tmp[,3])
abc.df.tmp[which(abc.df.tmp$pop %in% c("Aus", "Bel", "Hun", "Ita", "Pol",
"Spa", "UK")),3]="1" # Western Europe
abc.df.tmp[which(abc.df.tmp$pop %in% c("Bul1", "Bul2", "Tur")),3]="2" #
Eastern Europe
abc.df.tmp[which(abc.df.tmp$pop %in% c("Ira")),3]="3" # Iran
abc.df.tmp[which(abc.df.tmp$pop %in% c("S1", "S2", "S16")),3]="4" # North
central China
abc.df.tmp[which(abc.df.tmp$pop %in% c("S4","S6")),3]="5" # Admixed coastal
China
abc.df.tmp[which(abc.df.tmp$pop %in% c("S10","S11")),3]="6" # Admixed
continental China
abc.df.tmp[which(abc.df.tmp$pop %in% c("S9","S18", "S19", "S20")),3]="7" #
South China
abc.df.tmp[which(abc.df.tmp$pop %in% c("Jap")),3]="8" # Japan
# Reorder the dataset, DIY ABC names pop in incoming order
abc.df.tmp = abc.df.tmp[order(abc.df.tmp$pop),]
abc.df.tmp$pop
file.create("Data/ABC/ABC_Pparva_Global.txt")
write.table("Phylogeography of Pseudorasbora parva: Global invasion pathways
<NM=1.0NF> <MAF=hudson>","Data/ABC/ABC_Pparva_Global.txt",sep="
",quote=FALSE,row.names=F,col.names=F)
write(paste("IND SEX POP", paste(rep("A",N.loci),collapse = " "), sep="
"),file="Data/ABC/ABC_Pparva_Global.txt",append=TRUE)
for (i in 1:nrow(abc.df.tmp)) {
  write(paste(abc.df.tmp[i,],collapse="
"),file="Data/ABC/ABC_Pparva_Global.txt",append=TRUE)
}
rm(abc.df.tmp)
```

ABC Model selection & parameters estimates

10,000 simulations were produced for each scenario in each dataset. Model selection with Machine-Learning does not require a great amount of simulations, hence reducing computational load.

```
#============================================================
# SIMULATIONS WITH DIY ABC
#============================================================
# DIY ABC performs efficient and fast simulations for SNPs and complex
genealogical scenarios for ABC analysis
# Confidence in priors and SuSt must be assessed before any model choice
```

```
# This step was performed in DIY ABC GUI
# Iterative process to make simulated data close to the observed data
# (i.e. observed summary statistics must be in the distribution of summary
satitstics produced by simulations)
# Iterative process of adjusting priors and topologies with a low number of
simulations (e.g. 10,000 per scenario)
# PATH to the project (choose your project to analyze)
path = "SourceWesternEurope"
# path = "SourceEasternEurope"
# path = "SourceIran"
# path = "Global"
if (path == "SourceWesternEurope") {PrjDir =
"/Pparva_SourceWesternEurope2_2019_5_21-1"} # 400,000 simulations
if (path == "SourceEasternEurope") {PrjDir =
"/Pparva_SourceEasternEurope_2019_4_25-1"} # 400,000 simulations
if (path == "SourceIran") {PrjDir = "/Pparva_SourceIran_2019_4_17-1"} #
400,000 simulations
if (path == "Global") {PrjDir = "/Pparva_Global_2019_4_30-2"} # 30,000
simulations
# Observed summary statistics
obs.stats = read.table(paste(DIYABCdir, PrjDir, "/statobs.txt",sep=""),
header = TRUE)
# Read reftable.bin in a DIY ABC project directory
reftable = readRefTable(filename = paste(DIYABCdir, PrjDir,
"/reftable.bin",sep=""),
                        header = paste(DIYABCdir, PrjDir,
"/header.txt",sep=""), N = 40000) # Keep only 40,000 simulations for model
selection
scenarios = reftable$scenarios # Vector of scenarios simulated
stats = reftable$stats # A data frame of summary statistics
params = reftable$params # A data frame of parameters drawn from prior
distributions
colnames(stats) = colnames(obs.stats)
# 4 scenarios to test
# Split statistics and parameters' distributions between different scenarios
stats.1 = stats[which(scenarios=="1"),]
stats.2 = stats[which(scenarios=="2"),]
stats.3 = stats[which(scenarios=="3"),]
stats.4 = stats[which(scenarios=="4"),]
params.1 = params[which(scenarios=="1"),]
params.2 = params[which(scenarios=="2"),]
params.3 = params[which(scenarios=="3"),]
params.4 = params[which(scenarios=="4"),]

#---------------------------------------------------------
# Choice of priors
#---------------------------------------------------------
# Perform a priori goodness of fit using the two first components obtained
with PCA
# To see if we can discriminate among the different scenarios and if
```

```
simulated SuSt encompasses the observed SuSt
# If priors and topology are good, the observed SuSt must be inside the
clouds of Sut
mod.gfit.pca = gfitpca(target = obs.stats, sumstat = stats,
                              index = scenarios, cprob=0.1)
plot(mod.gfit.pca)
summary(mod.gfit.pca)

##################
# Perform a test for goodness-of-fit
# The null distribution is estimated using already performed simulations
contained in sumstat as pseudo-observed datasets.
# For each pseudo-observed dataset, the rejection algorithm is performed to
obtain a value of the goodness-of-fit statistic
# !!!!! Warning ! This test is usually performed on lots of simulations (>>
10,000 required for random forest)
# Hence, this is just a 'rough' preliminary assessment than models are
getting close to the data
mod1.gfit = gfit(target = obs.stats, sumstat = stats.1, nb.replicate = 100,
tol = 0.01)
mod2.gfit = gfit(target = obs.stats, sumstat = stats.2, nb.replicate = 100,
tol = 0.01)
mod3.gfit = gfit(target = obs.stats, sumstat = stats.3, nb.replicate = 100,
tol = 0.01)
mod4.gfit = gfit(target = obs.stats, sumstat = stats.4, nb.replicate = 100,
tol = 0.01)
# Plot
par(mfrow=c(2,2))
plot(mod1.gfit, sub = paste("Scenario 1 with p =
",summary(mod1.gfit)$pvalue,sep=""))
plot(mod2.gfit, sub = paste("Scenario 2 with p =
",summary(mod2.gfit)$pvalue,sep=""))
plot(mod3.gfit, sub = paste("Scenario 3 with p =
",summary(mod3.gfit)$pvalue,sep=""))
plot(mod4.gfit, sub = paste("Scenario 4 with p =
",summary(mod4.gfit)$pvalue,sep=""))
par(mfrow=c(1,1))
# Summary for p-value
summary(mod1.gfit)
summary(mod2.gfit)
summary(mod3.gfit)
summary(mod4.gfit)

# Prior calibration was performed both with DIY ABC implemented methods and
'abc' gfit
# Calibration was run iteratively until a good prior distribution was found,
fitting well the data
# It was never possible to reach the median of the simulated statistics
distribution with the observed data
# Single population statistics were failing more often than others, but two-
```

```
pop and 3-pop statistics were pretty robust
# but, we achieved to fit significantly the data
# Once the models fit well the data, simulations can be run to test scenarios
probabilities...

#=========================================================
# MODEL SELECTION WITH RANDOM FOREST
#=========================================================
# The 'abcrf' package allows to select a model and estimates parameters with
Random Forest
# And with a low number of simulations (around 10,000 per scenario)
# Random forest don't need a lot of simulations: 10,000 simulations per
scenario is enough
data = data.frame(scenarios, stats)
# Construct a random forest from a reference table towards performing an ABC
model choice
abc.mod = abcrf(scenarios~., data = data, lda = FALSE, ntree = 1000,
          sampsize=min(100000, nrow(data)), paral = TRUE, ncores = 4)
# The trained model was saved in a R object
save(abc.mod, file = paste(DIYABCdir, PrjDir, "/RandomForest.Rda",sep=""))
load(file = paste(DIYABCdir, PrjDir, "/RandomForest.Rda",sep=""))
# Calculate and plot for different numbers of tree, the out-of-bag errors
associated with an ABC-RF object
err.abcrf(abc.mod, training = data, paral = TRUE, ncores = 4)
# Variable importance plot from a random forest: contribution of the most
important variables
variableImpPlot(abc.mod)
# Visualize variable importance plot of a model choice ABC-RF object and the
projection of the reference table on the LDA axes
# plot(abc.mod, training = data)
# Predict and evaluate the posterior probability of the MAP for new data
using an ABC-RF object
abc.pred = predict(abc.mod, obs = obs.stats, training = data, paral = TRUE,
ncores = 4)
save(abc.pred, file = paste(DIYABCdir, PrjDir,
"/RandomForest.pred.Rda",sep=""))

PrjDir = "/Pparva_SourceWesternEurope2_2019_5_21-1"
load(file = paste(DIYABCdir, PrjDir, "/RandomForest.pred.Rda",sep=""))
abc.pred

##   selected model votes model1 votes model2 votes model3 votes model4
## 1              4          175          175          192          458
##   post.proba
## 1  0.8738333

#=========================================================
# MODEL SELECTION WITH NEURAL NETWORK
#=========================================================
#---------------------------------------------------------
# POSTERIOR PROBABILITIES
```

```r
# Selecting which model is the best with a Neural Network algorithm
# Tolerance must be calibrated
# If tolerance too stringent, too much rejection and
# Neural Net's performances declines quickly
# But i
stats.clean = as.matrix(stats)
colnames(stats.clean) = NULL
mod.sel = postpr(target = obs.stats, index = as.vector(scenarios), sumstat =
stats.clean, tol=0.05, method="neuralnet", corr = TRUE, kernel =
"epanechnikov", numnet = 1000, sizenet = 12, lambda = c(0.0001, 0.001, 0.01),
trace = TRUE, maxit = 1000, MaxNWts = 100000)
# Model selection with Neural network was computationnally expensive
# The trained model was saved in a R object
save(mod.sel, file = paste(DIYABCdir, PrjDir, "/NeuralNet.Rda",sep=""))

load(file = paste(DIYABCdir, PrjDir, "/NeuralNet.Rda",sep=""))
summary(mod.sel)

## Call:
## postpr(target = obs.stats, index = as.vector(scenarios), sumstat =
stats.clean,
##     tol = 0.2, method = "neuralnet", corr = TRUE, kernel = "epanechnikov",
##     numnet = 10000, sizenet = 12, lambda = c(1e-04, 0.001, 0.01),
##     trace = TRUE, maxit = 1000, MaxNWts = 1e+05)
## Data:
##  postpr.out$values (8000 posterior samples)
## Models a priori:
##  1, 2, 3, 4
## Models a posteriori:
##  1, 2, 3, 4

## Warning: Posterior model probabilities are corrected for unequal number of
## simulations per models.

##
##
## Proportion of accepted simulations (rejection):
##      1      2      3      4
## 0.1652 0.1587 0.1552 0.5209
##
## Bayes factors:
##        1      2      3      4
## 1 1.0000 1.0297 1.0759 0.3155
## 2 0.9712 1.0000 1.0449 0.3064
## 3 0.9294 0.9570 1.0000 0.2933
## 4 3.1692 3.2633 3.4098 1.0000
##
##
## Posterior model probabilities (neuralnet):
##      1      2      3      4
## 0.0272 0.0190 0.1018 0.8521
```

```
##
## Bayes factors:
##          1       2       3       4
## 1  1.0000  1.4322  0.2670  0.0319
## 2  0.6982  1.0000  0.1865  0.0223
## 3  3.7446  5.3631  1.0000  0.1194
## 4 31.3512 44.9014  8.3723  1.0000
```

In this output for Neural Network, we see that nnet() ran with 12 units in the hidden layer, and 10000 networks. A tolerance rate of 0.2 means that the 20% of simulations closest to the observed dataset were retained. Tolerance rate is the threshold fixed for the rejection algorithm. Proportions of accepted simulations (rejection) corresponds to this selection. We can see that more than 50% of accepted simulations were from scenario 4. Furthermore, the neural network was trained on these accepted simulations, and analyses were more powerful to infer a scenario. The scenario 4 was selected with 85% of posterior probability. Prior error rate computed by cross-validation was of 0.02.

```
#-----------------------------------------------------------
# Cross-validation: testing for model selection performance
# Pseudo-observed data (i.e. simulations) were reassigned to a scenario
# Accuracy is given by the proportion of re-assignment to the good scenario
(under which the pseudo-observed dataset was simulated)
# Long time for computation
# Bug in the original function cv4postpr(), not passing supplementary
arguments to neural network nnet()
# Modified function to pass efficiently arguments to nnet (e.g. MaxNWts)
cv.modsel = cv4postpr_nnet(index = as.vector(scenarios), sumstat =
stats.clean, nval = 100, tol = 0.05, method = "neuralnet",corr = TRUE, kernel
= "epanechnikov", numnet = 1000, sizenet = 10,lambda = c(0.0001, 0.001,
0.01), trace = TRUE, maxit = 1000, MaxNWts = 100000)
# Cross-validation with Neural network was computationnally expensive
# Results were saved in a R object
save(cv.modsel, file = paste(DIYABCdir, PrjDir, "/cv.NeuralNet.Rda",sep=""))
load(file = paste(DIYABCdir, PrjDir, "/cv.NeuralNet.Rda",sep=""))
# ?plot.cv4abc
plot(cv.modsel)
png(paste("Figures/ABC/", path,"/Neuralnet_Confusion_matrix.png",
sep=""),res=150,width=800,height=600)
plot(cv.modsel)
dev.off()
summary(cv.modsel)
# Prior error rate is simply the proportion of misclassification among all
cross-validation iterations
cv.modsel$true
unlist(cv.modsel$estim)
1-sum(cv.modsel$true == unlist(cv.modsel$estim))/length(cv.modsel$true)
#-----------------------------------------------------------
# RESULTS
#-----------------------------------------------------------
############################
# SOURCE OF WESTERN INVASION
############################
```

```
# Out-of-bag prior error rate: 5.95% (i.e. re-assignment to the good scenario
error rate)
# We had a good but not excellent power of random forest classification

# Confusion matrix:
#   1    2    3    4 class.error
# 1 9711  164  146   97  0.04022534
# 2  136 9050  812   34  0.09788676
# 3  102  580 9161   28  0.07192787
# 4  122   87   72 9698  0.02815913
# Most of errors came from confusion between model 2 & 3, i.e. single
introduction from admixed coastal China or admixed continental China

# The Out-of-bag prior error rate as a function of number of growing trees
stabilized around the mean value
# Hence, we achieved a sufficient number of growing trees for statistical
precision (ntree = 1000)

# Model prediction with Random Forest: which scenario fit best the data?
# selected model votes model1 votes model2 votes model3 votes model4
post.proba
#        4           96           68          222          614        0.70255

# Model 4 was selected: single introduction from admixture in the native
range between 3 most probable sources
# 614 votes on 1,000 possible votes (posterior probability 0.70)

# With the Neural Network:
# Prior error rate = 0.02
## Posterior model probabilities (neuralnet):
##      1      2      3      4
## 0.0272 0.0190 0.1018 0.8521
##
## Bayes factors:
##         1       2       3       4
## 1  1.0000  1.4322  0.2670  0.0319
## 2  0.6982  1.0000  0.1865  0.0223
## 3  3.7446  5.3631  1.0000  0.1194
## 4 31.3512 44.9014  8.3723  1.0000

############################
# SOURCE OF EASTERN INVASION
############################
# Out-of-bag prior error rate: 5.6075%
# Low, but not perfect prior error rate. Yet, we had a good power to
discriminate between scenarios

# Confusion matrix:
#   1    2    3    4 class.error
```

```
# 1 9652  157  108   90  0.03547517
# 2  147 9426  622   40  0.07904250
# 3   96  684 9073   33  0.08223751
# 4  104   82   80 9606  0.02694489
# Once again (see Western Invasion), the algorithm didn't performed well for
discriminating scenarios 2 & 3
# But classification errors were not dramatic

# selected model votes model1 votes model2 votes model3 post.proba
#        1          859          124           17         0.4755167

# Scenario 4 was selected without uncertainty, with 859 votes on 1,000 and a
posterior probability
# depsite a weaker posterior probability than other scenario sets (0.47)
# single origin from an admixture of the 3 probable putative sources

# With the Neural Network:
# Prior error rate = 0.04

# Posterior model probabilities (neuralnet):
#   1      2      3      4
# 0.0220 0.0241 0.0316 0.9223
#
# Bayes factors:
#   1       2       3       4
# 1  1.0000  0.9142  0.6963  0.0239
# 2  1.0938  1.0000  0.7616  0.0261
# 3  1.4363  1.3130  1.0000  0.0343
# 4 41.8989 38.3042 29.1721  1.0000

############################
# SOURCE OF IRAN INVASION
############################
# Out-of-bag prior error rate: 6.9425%
# Prior error rate similar to other invasive demes.

# Confusion matrix:
#   1    2    3    4 class.error
# 1 9511  219   84  105  0.04113318
# 2  170 8981  907   50  0.11149584
# 3   92  842 9035   22  0.09568612
# 4  111   95   80 9696  0.02865157
# Noticeable confusion between scenarios 2 & 3 but nothing dramatic

# selected model votes model1 votes model2 votes model3 votes model4
post.proba
#       4           60          210           36             694       0.8047

# The fourth model was chosen, with 69% of votes:
```

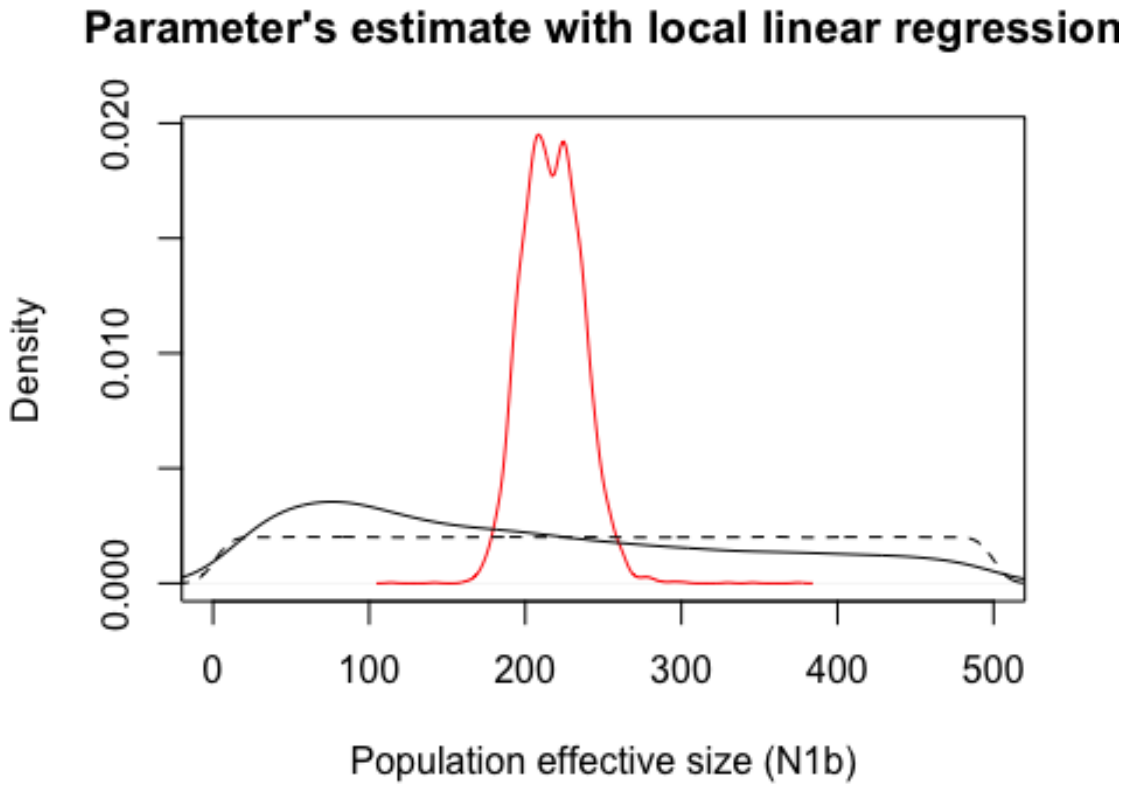
```
# single origin from an admixture of the 3 probable putative sources

# With the Neural Network:
# Prior error rate = 0.045

#   Posterior model probabilities (neuralnet):
#   1      2      3      4
# 0.2909 0.1671 0.1144 0.4276
# 42% of posterior probability in favor to the model 4

# Bayes factors:
#   1      2      3      4
# 1 1.0000 1.7410 2.5414 0.6802
# 2 0.5744 1.0000 1.4598 0.3907
# 3 0.3935 0.6850 1.0000 0.2676
# 4 1.4702 2.5596 3.7364 1.0000

#-----------------------------------------------------------
# SECOND STEP OF HIERARCHICAL ANALYSES
# Since there was admixture in invasive demes, what can we infer at a global
scale?
# Was it a single introduction that splitted in 3 invasive demes
# or 3 independent introductions that colonised different regions?

############################
# WORLDWIDE ROUTES OF INVASION
############################
# Out-of-bag prior error rate: 3.4967%
#
# Confusion matrix:
#         1    2    3  class.error
# 1 9950    2    0 0.0002009646
# 2   12 9392  618 0.0628617043
# 3    2  415 9609 0.0415918612

# selected model votes model1 votes model2 votes model3 votes model4
post.proba
#       4           60          210           36             694       0.8047

# The fourth model was chosen, with 69% of votes:
# single origin from an admixture of the 3 probable putative sources

# With the Neural Network:
# Prior error rate = 0.09333333

# Posterior model probabilities (neuralnet):
#         1      2      3
# 0.9897 0.0059 0.0044
# Model 1 accepted with high confidence
```

```
# 3 independent introductions from 3 independent admixture events

# Bayes factors:
#         1        2        3
# 1   1.0000 166.9438 225.4166
# 2   0.0060   1.0000   1.3503
# 3   0.0044   0.7406   1.0000

#==========================================================
# PARAMETER ESTIMATES WITH RANDOM FOREST
#==========================================================

# A reference table including 100,000 datasets is a good starting point for
this step of parameter estimates (Raynal et al. 2017)
# 500 growing trees is a default choice
# Perform a random forest regression on the response variable (1 parameter to
estimate)
# For the selected scenario only
# 100,000 simulations wasn't performing well for parameters' estimates
# So, re-import 1,000,000 simulations instead of 100,000 (N=0 means all
simulations)
# Trial on selected scenarios at step 1: Western Europe
# PrjDir = "/Pparva_SourceWesternEurope_Params_2019_5_3-1" # Works for
Western Europe scenario
# PrjDir = "/Pparva_SourceEasternEurope_Params_2019_4_25-1"
PrjDir = "/Pparva_SourceIran_Params_2019_4_17-1" # 1,000,000 simulations

# Observed summary statistics
obs.stats = read.table(paste(DIYABCdir, PrjDir, "/statobs.txt",sep=""),
header = TRUE)
# Reference table
reftable = readRefTable(filename = paste(DIYABCdir, PrjDir,
"/reftable.bin",sep=""),
                        header = paste(DIYABCdir, PrjDir,
"/header.txt",sep=""), N = 0)
stats = reftable$stats # A data frame of summary statistics
params = reftable$params # A data frame of parameters drawn from prior
distributions
colnames(stats) = colnames(obs.stats)

#----------------------------------------------------------
#   N1b, EFFECTIVE POPULATION SIZE DURING BOTTLENECK
# Here, estimate the effective population size during the bottleneck, in the
scenario 4
rm(data.reg)
params = as.data.frame(params)
data.reg = data.frame(N1b = params$N1b,
                      stats)
# min.node.size = 5 for regression (?ranger)
reg.abc.N1b = regAbcrf(N1b~., data = data.reg, ntree = 1000,
```

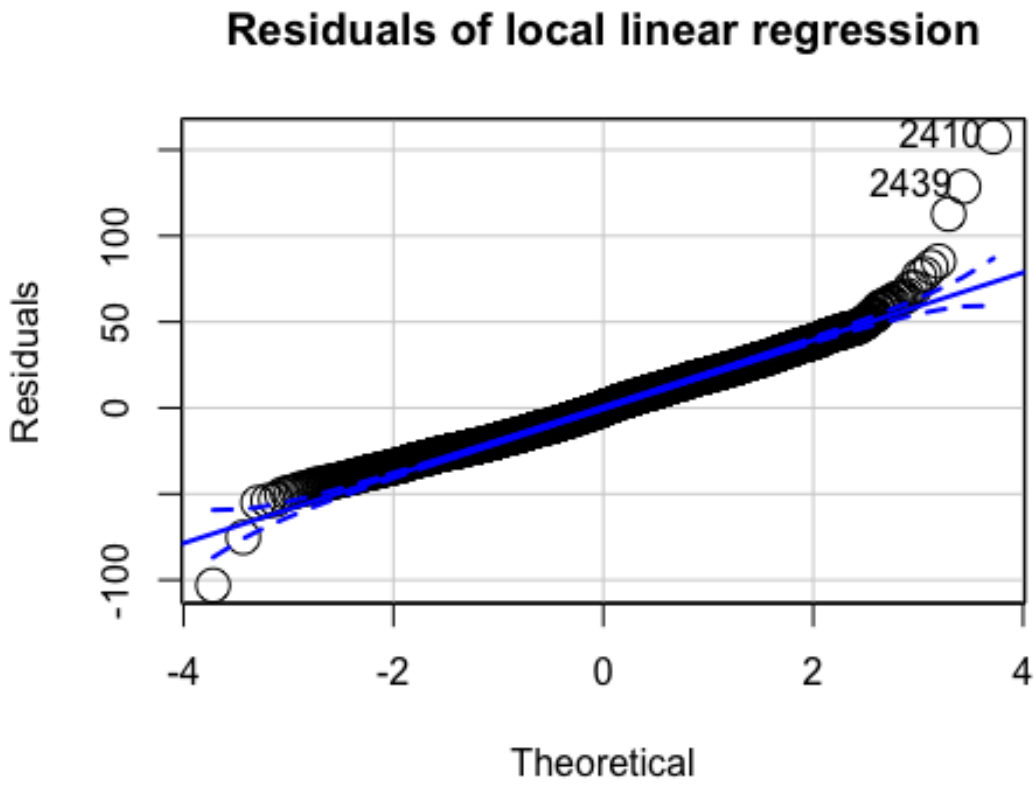
```
                        sampsize=min(100000, nrow(data.reg)), paral = TRUE,
ncores = 4,
                        min.node.size = 5)
reg.abc.N1b
save(reg.abc.N1b, file = paste(DIYABCdir, PrjDir, "/reg.abc.N1b.Rda",sep=""))
load(file = paste(DIYABCdir, PrjDir, "/reg.abc.N1b.Rda",sep=""))

# plot.regAbcrf provides a variable importance plot used to construct the
reg-ABC-RF object
plot(reg.abc.N1b)
# Based on a reg-ABC-RF object this function predicts the posterior
expectation,
# median, variance, quantiles for the corresponding parameter given new
dataset
pred.reg.N1b = predict(reg.abc.N1b, obs = obs.stats, training = data.reg,
quantiles=c(0.025,0.975),
                        paral = TRUE, ncores = 4)
pred.reg.N1b
err.regAbcrf(reg.abc.N1b, training = data.reg,
             paral = TRUE, ncores = 4)
#-----------------------
# Plot The Posterior Density Given A New Summary Statistic
# Given a reg-ABC-RF object and a new value of the summary statistics,
# densityPlot gives the corresponding posterior density plot of the
parameter,
# as well as the prior
densityPlot(reg.abc.N1b, obs = obs.stats, training = data.reg,
            paral = TRUE, ncores = 4)
#-----------------------
# EVALUATE CONFIDENCE IN PARAMETER ESTIMATES
# Predict Out-Of-Bag Posterior Expectation, Median, Variance, Quantiles And
Error Measures Using A Reg-ABC-RF Object
# Based on a reg-ABC-RF object this function predicts the out-of-bag
posterior expectation, median, variance, quantiles, mean squared errors,
normalized mean absolute errors, credible interval coverage and relative mean
range,
# for the corresponding parameter using the out-of-bag observations of the
training data set.
# Mean squared errors and normalized mean absolute errors are computed both
with mean and median of the response variable.
# Memory allocation issues might be encountered when the size of the training
data set is large.
confidence.reg.N1b = predictOOB(reg.abc.N1b, training = data.reg,
                                paral = TRUE, ncores = 4)
confidence.reg.N1b
save(confidence.reg.N1b, file = paste(DIYABCdir, PrjDir,
"/confidence.reg.N1b.Rda",sep=""))
load(file = paste(DIYABCdir, PrjDir, "/confidence.reg.N1b.Rda",sep=""))

#=========================================================
```

```
# PARAMETERS' ESTIMATES WITH LOCAL LINEAR REGRESSION
#=========================================================
?abc
# Tolerance: 0.05 and 0.01
# This function performs multivariate parameter estimation based on summary
statistics using an ABC algorithm
# And return an object of class 'abc'
stats.C = as.data.frame(stats)
colnames(stats.C) = colnames(obs.stats)
# Local linear regression is the faster method to compute parameters'
estimates,
# hence it was used before computationnaly intensive Neural Network
# Tolerance rate allows to adjust the number of simulations retained for
regression
# Only simulations closest to the dataset were retained,
# and design was to retain not too much (inaccurate estimates) or not too few
(biased estimates)
# Starting with local linear regression with 1% tolerance (a common value in
ABC parameters' esitmates)
mod.abc=abc(target = obs.stats, param = params[,c(7,9)], sumstat = stats.C,
tol = 0.005,
            method = "loclinear", transf = "none", corr = TRUE, kernel =
"epanechnikov")
# Local linear regression with 0.1% tolerance (1000 simulations closest to
data)
# mod.abc=abc(target = obs.stats, param = params, sumstat = stats.C, tol =
0.001,
#             method = "loclinear", transf = "none", corr = TRUE, kernel =
"epanechnikov")
# summary(mod.abc)
# 0.1% gave better estimates with lower CI within the wide CI at 1% but
residuals wasn't gaussian
# Keeping only 1% tolerance rate for further analyses (10,000 sampled
simulations)
save(mod.abc, file = paste(DIYABCdir, PrjDir,
"/abcparam.LocLinear.Rda",sep=""))
load(file = paste(DIYABCdir, PrjDir, "/abcparam.LocLinear.Rda",sep=""))
summary(mod.abc)
# Checking quality of predictions
plot(mod.abc, param = params[,c(7,9)], ask = FALSE)
#-----------------------------------------
# POSTERIOR DISTRIBUTIONS
# Diagnostic plots and parameters density

load(file = paste(DIYABCdir, PrjDir, "/abcparam.LocLinear.Rda",sep=""))
# N1b density plot of posterior with LocLinear
# Dashed line was the prior distribution
# Solid black line was the estimates with rejection algorithm
# Red solid line was the estimates with the local linear regression
plot(density(mod.abc$adj.values[,2]), main = "Parameter's estimate with local
```

```
linear regression",
     xlab = "Population effective size (N1b)", cex = 2, xlim = c(0, 500), col
= "red")
lines(density(mod.abc$unadj.values[,2]), col = "black") # add N1b posterior
density plot
lines(density(params[,9]), col = "black", lty = 2) # add N1b priors density
plot
```




```
# Residuals of the model lsfit()
qqPlot(mod.abc$residuals[,2], main = "Residuals of local linear regression",
       xlab = "Theoretical", ylab = "Residuals", cex = 2) # N1b density plot
of priors
```




```
## [1] 2410 2439
```

Confidence interval of the estimates was accurate, and residuals followed the normal law. Nevertheless, mode of rejection estimates was different of the mode of loc linear regression.

```
# Density of T1, time of introduction
plot(density(mod.abc$adj.values[,1]), main = "Parameter's estimate with local
linear regression",
     xlab = "Time of introduction (T1)", cex = 2, xlim = c(0, 100), col =
"red") # T1 density plot of posterior with LocLinear
lines(density(mod.abc$unadj.values[,1]), col = "black") # add T1 posterior
density plot
lines(density(params[,7]), col = "black", lty = 2) # add T1 priors density
plot
# Residuals of the model lsfit()
qqPlot(mod.abc$residuals[,1], main = "Residuals of local linear regression",
       xlab = "Theoretical", ylab = "Residuals", cex = 2) # N1b density plot
of priors

# Summaries of posterior samples by ABC
# Calculates simple summaries of posterior samples: the minimum and maximum,
the weighted mean, median, mode, and credible intervals
summary(mod.abc)
```

```
# Cross-validation of parameters' estimates precision
cv.mod.abc = cv4abc(param = params[,c(7,9)], sumstat = stats.C, abc.out =
NULL, nval = 1000, tols = c(0.01, 0.005),
                    statistic = "median", method = "loclinear", hcorr = TRUE,
transf = "none",
                    kernel = "epanechnikov")
save(cv.mod.abc, file = paste(DIYABCdir, PrjDir,
"/cv.abcparam.LocLinear.Rda",sep=""))
load(file = paste(DIYABCdir, PrjDir, "/cv.abcparam.LocLinear.Rda",sep=""))
plot(cv.mod.abc)
plot(cv.mod.abc)
summary(cv.mod.abc)
# Prior error rate is simply the proportion of misclassification among all
cross-validation iterations
cv.mod.abc$true
unlist(cv.mod.abc$estim)
1-sum(cv.mod.abc$true == unlist(cv.mod.abc$estim))/length(cv.mod.abc$true)

#=========================================================
# PARAMETERS' ESTIMATES WITH NEURAL NETWORK
#=========================================================
# Given results on local linear regression, keeping only 0.1% tolerance rate
mod.nnet.abc=abc(target = obs.stats, param = params, sumstat = stats.C, tol =
0.01,
             method = "neuralnet", transf = "none", corr = TRUE, kernel =
"epanechnikov",
             numnet = 500, sizenet = 4, lambda = c(0.0001, 0.001, 0.01),
trace = TRUE,
             maxit = 1000, MaxNWts = 100000)
save(mod.nnet.abc, file = paste(DIYABCdir, PrjDir,
"/abcparam.NeuralNet.Rda",sep=""))
load(file = paste(DIYABCdir, PrjDir, "/abcparam.NeuralNet.Rda",sep=""))
#-----------------------------------------
# POSTERIOR DISTRIBUTIONS
# Histogram of posterior samples
# ?hist.abc
hist(x=mod.nnet.abc, ask = FALSE)
# hist(x=mod4.abc, file = paste(DIYABCdir, PrjDir,
"/Figures/abcparam.LocLinear.pdf",sep=""))

# Diagnostic plots
plot(density(params[,9]), main = "Parameter's estimate with neural network
regression",
     xlab = "Population effective size (N1b)", cex = 2, ylim = c(0, 0.1)) #
N1b density plot of priors
lines(density(mod.nnet.abc$adj.values[,9]), col = "red") # add N1b posterior
density plot
lines(density(mod.nnet.abc$unadj.values[,9]), col = "black") # add N1b
unadjusted posterior density plot
```

```
# Residuals of the model lsfit()
qqPlot(mod.abc$residuals[,9], main = "Residuals of neural network
regression",
       xlab = "Theoretical", ylab = "Residuals", cex = 2) # N1b density plot
of priors

# Summaries of posterior samples by ABC
# Calculates simple summaries of posterior samples: the minimum and maximum,
the weighted mean, median, mode, and credible intervals
# ?summary.abc
summary(mod.nnet.abc)
# Cross-validation of parameters' estimates precision
cv.mod.nnet.abc = cv4abc(param = params[,c(7,9)], sumstat = stats.C, abc.out
= NULL, nval = 1000, tols = c(0.01, 0.005),
                         method = "neuralnet", transf = "none", corr = TRUE,
kernel = "epanechnikov",
                         numnet = 500, sizenet = 4, lambda = c(0.0001, 0.001,
0.01), trace = TRUE,
                         maxit = 1000, MaxNWts = 100000)
save(cv.mod.nnet.abc, file = paste(DIYABCdir, PrjDir,
"/cv.abcparam.NeuralNet.Rda",sep=""))
load(file = paste(DIYABCdir, PrjDir, "/cv.abcparam.NeuralNet.Rda",sep=""))
plot(cv.mod.nnet.abc)
plot(cv.mod.nnet.abc)
summary(cv.mod.nnet.abc)
# Prior error rate is simply the proportion of misclassification among all
cross-validation iterations
cv.mod.nnet.abc$true
unlist(cv.mod.nnet.abc$estim)
1-sum(cv.mod.nnet.abc$true ==
unlist(cv.mod.nnet.abc$estim))/length(cv.mod.nnet.abc$true)

#-------------------------------------------------------------
# RESULTS
#-------------------------------------------------------------
############################
# SOURCE OF WESTERN INVASION
############################
#--------------------------------------
# LOCAL LINEAR REGRESSION
# N1b (population effective size during bottleneck)
# Mean = 383.6812 (CI = 98.2315; 630.3740)
# t1, time of introduction
# Mean = 40.5615 (CI = 19.9269; 62.8547)

############################
# SOURCE OF EASTERN INVASION
############################
#--------------------------------------
# LOCAL LINEAR REGRESSION
```

```
# N1b (population effective size during bottleneck)
# Mean = 89.7669 (CI = 32.4534; 148.2742)
# t1, time of introduction
# Mean = 38.8207 (CI = 18.2563; 61.5819)

############################
# SOURCE OF IRAN INVASION
############################

#--------------------------------------
# LOCAL LINEAR REGRESSION
# N1b (population effective size during bottleneck)
# Mean = 217.2987 (CI = 185.2318; 253.4952)
# t1, time of introduction
# Mean = 65.7185 (CI = 44.3454; 89.9336)

#==========================================================
# POSTERIOR MODEL VALIDATION AND ERRORS
#==========================================================

# The quality of the selected model was evaluated  y posterior predictive
check
# Simulate 10,000 times the selected scenario with priors drawn from
posterior distribution
# Compare goodness-of-fit of simulations to the observed data
##################
# Perform a priori goodness of fit using the two first components obtained
with PCA
# To see if we can discriminate among the different scenarios and if
simulated SuSt encompasses the observed SuSt
# If priors and topology are good, the observed SuSt must be inside the
clouds of Sut
mod.gfit.pca = gfitpca(target = obs.stats, sumstat = stats,
                       index = scenarios, cprob=0.1)
plot(mod.gfit.pca)
summary(mod.gfit.pca)

##################
# Perform a test for goodness-of-fit
# The null distribution is estimated using already performed simulations
contained in sumstat as pseudo-observed datasets.
# For each pseudo-observed dataset, the rejection algorithm is performed to
obtain a value of the goodness-of-fit statistic
mod.gfit = gfit(target = obs.stats, sumstat = stats, nb.replicate = 100, tol
= 0.01)
# Plot
plot(mod.gfit, sub = paste("p = ",summary(mod.gfit)$pvalue,sep=""))
```

Final figures for report & oral defense

This part was designed to produce figures for the report and the MSc oral defense only. Results saved in R objects were finally loaded and maps automatically produced.

```
#=========================================================
# FINAL MAP FOR REPORT
#=========================================================
# On native side, structure pie charts
# On invasive side, population assignment with AssignPOP
# Prepare the data set for mapping
# Prepare the data set for mapping
strK6=read.table(paste(STRdir,"/2b_Native/CLUMPP/K6.outfile",sep=""))
Native.pies.str=cbind(as.character(labelPops$Location_name[strK6[,4]]),as.dat
a.frame(strK6[,(c(1,2,3,5,4,6)+5)])) # Clusters are selected to correspodn to
DAPC colors
names(Native.pies.str)=c("Population", "Cluster 1", "Cluster 2", "Cluster 3",
"Cluster 4", "Cluster 5", "Cluster 6")
Native.pie.str_means=sapply(split(Native.pies.str[2:7],Native.pies.str$Popula
tion), colMeans) ## This just makes population cluster averages for each
cluster
res=read.table(("AssignPOP/assignPOP.MC.svm.demes/TrimAssignmentResult.txt"),
header=TRUE)
res.table = data.frame(Population = row.names(table(res$Ind.ID,
res$pred.pop)),
                       Coastal_Admixed_China = table(res$Ind.ID,
res$pred.pop)[,1],
                       Continental_Admixed_China = table(res$Ind.ID,
res$pred.pop)[,2],
                       Japan = table(res$Ind.ID, res$pred.pop)[,3],
                       North_Central_China = table(res$Ind.ID,
res$pred.pop)[,4],
                       North_China = table(res$Ind.ID, res$pred.pop)[,5],
                       South_China = table(res$Ind.ID, res$pred.pop)[,6]) #
predicted populations as a function of invasive population
Invasive.pies.means=sapply(split(res.table[2:7],res.table$Population),
colMeans) ## This just makes population cluster averages for each cluster
# Download database of river network at scale 50
# rivers50=ne_download(scale = 50, type = 'rivers_lake_centerlines', category
= 'physical')
# save(rivers50,file="Data/rivers50.Rda")
load("Data/rivers50.Rda")
map("world", xlim=c(-15,150), ylim=c(20,59), col="gray90",  fill=TRUE) # Plot
the map of European area
# Nomenclature
north.arrow(149,56,1, lab="N", lab.pos="above")
scalebar(c(135,20),1000, bg=NA, border=NA, division.cex=0.5)
# INVASIVE
## Pies ##
# Draw a pie with assignment proportions for each population
# Set a vector of colors for clusters
```

```
cols.assign=c("#E41A1C", "#4DAF4A", "#377EB8", "#FF7F00", "#984EA3",
"#FFFF33")
for (i in 1:ncol(Invasive.pies.means)) {
  add.pie(Invasive.pies.means[,i],x=invasive.coords$X[i],
y=invasive.coords$Y[i],labels="",radius=1.8,edges=200,clockwise=T,    col =
cols.assign, cex=1, font=2)
  # Add labels to pies
}
# NATIVE
# Labels of sampled sites
text(invasive.coords$X, y = invasive.coords$Y, invasive.coords$Pop,pos =
c(rep(4,10), 2), offset =0.5, cex = 0.8, font = 2)
# Labels of sampled sites
# Jitter the 13rd label
text(native.coords$X, y = native.coords$Y, native.coords$Pop,pos = c(4, 4, 2,
2, 4, 4, 4, 2, 2, 2, 4, 4, 4, 4, 4, 4, 2, 4), offset =0.5, cex = 0.8, font =
2)
## Pies ##
# Draw a pie with admixture proportions for each population
# Set a vector of colors for clusters
cols.native=c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00",
"#FFFF33", "#A65628", "#F781BF","#8DD3C7", "#FFFFB3", "#BEBADA", "#FB8072",
"#80B1D3", "#FDB462", "#B3DE69", "#FCCDE5", "#D9D9D9")
for (i in 1:ncol(Native.pie.str_means)) {
  add.pie(Native.pie.str_means[,i],x=native.coords$X[i],
y=native.coords$Y[i], labels = "",radius=1.8,edges=200,clockwise=T,    col =
cols.native, cex=1, font=2)
  # Add labels to pies
}

#============================================================
# FINAL MAP FOR ORAL DEFENSE
#============================================================
# On both sides, structure pie charts
strK9Eu=read.table(paste(STRdir,"/3a_Invasive/CLUMPP/K9.outfile",sep=""))
Invasive.pies.str=cbind(as.character(labelPops$Location_name[strK9Eu[,4]]),as
.data.frame(strK9Eu[,(c(1,2,3,4,5,6,7,8,9)+5)])) # Clusters are selected to
correspodn to DAPC colors
names(Invasive.pies.str)=c("Population", "Cluster 1", "Cluster 2", "Cluster
3", "Cluster 4","Cluster 5", "Cluster 6", "Cluster 7", "Cluster 8", "Cluster
9")
Invasive.pies.str_means=sapply(split(Invasive.pies.str[2:10],Invasive.pies.st
r$Population), colMeans) ## This just makes population cluster averages for
each cluster

#---------------------------------
# WORLDWIDE MAP OF SAMPLED SITES FOR M&M (ALL IN ONE MAP)
# Download database of river network at scale 50
# rivers50=ne_download(scale = 50, type = 'rivers_lake_centerlines', category
= 'physical')
```

```
# save(rivers50,file="Data/rivers50.Rda")
load("Data/rivers50.Rda")
map("world", xlim=c(-15,150), ylim=c(20,59), col="gray90",  fill=TRUE) # Plot
the map of European area
# Nomenclature
north.arrow(149,56,1, lab="N", lab.pos="above")
scalebar(c(135,20),1000, bg=NA, border=NA, division.cex=0.5)
# INVASIVE
## Pies ##
# Draw a pie with admixture proportions for each population
# Set a vector of colors for clusters
cols.invasive = c("#A6DBA0", "#D9F0D3",  "#ABD9E9", "#993404",
"#C2A5CF","#E0F3F8","#FE9929", "#FEC44F", "#762A83")
for (i in 1:ncol(Invasive.pies.str_means)) {
  add.pie(Invasive.pies.str_means[,i],x=invasive.coords$X[i],
y=invasive.coords$Y[i],labels="",
          radius=1.8,edges=200,clockwise=T, col =   cols.invasive, cex=1,
font=2)
  # Add labels to pies
}
# NATIVE
# Labels of sampled sites
text(invasive.coords$X, y = invasive.coords$Y, invasive.coords$Pop,
     pos = c(rep(4,10), 2), offset =0.5, cex = 0.8, font = 2)
# Labels of sampled sites
# Jitter the 13rd label
text(native.coords$X, y = native.coords$Y, native.coords$Pop,
     pos = c(4, 4, 2, 2, 4, 4, 4, 2, 2, 2, 4, 4, 4, 4, 4, 4, 2, 4), offset
=0.5, cex = 0.8, font = 2)
### Pies ##
# Draw a pie with admixture proportions for each population
# Set a vector of colors for clusters
cols.native=c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00",
"#FFFF33", "#A65628", "#F781BF",
              "#8DD3C7", "#FFFFB3", "#BEBADA", "#FB8072", "#80B1D3",
"#FDB462", "#B3DE69", "#FCCDE5", "#D9D9D9")
for (i in 1:ncol(Native.pie.str_means)) {
  add.pie(Native.pie.str_means[,i],x=native.coords$X[i],
y=native.coords$Y[i], labels = "",radius=1.8,edges=200,clockwise=T,    col =
cols.native, cex=1, font=2)
  # Add labels to pies
}
```

Supplementary material

Convert the GENIND dataset in GENEPOP

Genepop format is required for multiple R packages for genetic diversity (e.g. diveRsity)

The Input File Format for Genepop(WWW) and LinkDos(WWW) :

First line : Any characters. Use this line to store information about you data.

Second line : the name of the first locus.

Third line : the name of the second locus (if needed)

ETC...

Alternatively, separate loci with commas (or a comma+space) on the same line.

Line N+1 : the name of the Nth locus.

Line N+2 : Type the word “POP” (or “Pop”, or “pop”).

Line N+3 : And example is given below :

ind#001fem, 0101 0202 0000 0410

```
#--------------------------------
# Format in genepop file
genind.df=genind2df(Pparva)
# !!!!! Missing data should be indicated with 00
# Missing data (NA) was transformed in '0000'
genind.df[is.na(genind.df)]="0000"

# ............
# Open connection to new genepop files: 3 dataset (global, Asian and
European) have been printed at once
file.create("Data/genepop.txt")
file.create("Data/genepop_native.txt")
file.create("Data/genepop_invasive.txt")
# print first a title line
write.table("Phylogeography of Pseudorasbora parva","Data/genepop.txt",sep="
",quote=FALSE,row.names=F,col.names=F)
write.table("Phylogeography of Pseudorasbora parva: Native
area","Data/genepop_native.txt",sep=" ",quote=FALSE,row.names=F,col.names=F)
write.table("Phylogeography of Pseudorasbora parva: Invasive
area","Data/genepop_invasive.txt",sep="
",quote=FALSE,row.names=F,col.names=F)

# Locus names: They may be given one per line, or on the same line but
separated by commas.
# write(paste(colnames(genind.df)[-1],collapse
="\n"),file="Data/genepop.txt",append=TRUE)
# write(paste(colnames(genind.df)[-1],collapse
="\n"),file="Data/genepop_native.txt",append=TRUE)
# write(paste(colnames(genind.df)[-1],collapse
="\n"),file="Data/genepop_invasive.txt",append=TRUE)

write(paste(colnames(genind.df)[-1],collapse =" ,
```

```
"),file="Data/genepop.txt",append=TRUE)
write(paste(colnames(genind.df)[-1],collapse =" ,
"),file="Data/genepop_native.txt",append=TRUE)
write(paste(colnames(genind.df)[-1],collapse =" ,
"),file="Data/genepop_invasive.txt",append=TRUE)

# Then, one individual per line, with the following nomenclature 'ind_name ,
locus1 locus2 ...'
# Each sample from a different geographical original is declared by a line
with a POP statement.

# Print each population at a time
# Lists of population names with their associated coordinates
native.names=c("Jap", "S1", "S2", "S3", "S4", "S6","S9",
               "S10", "S11", "S13", "S14", "S15", "S16", "S17", "S18", "S19",
"S20", "Tib")
invasive.names=c("Aus", "Bel", "Bul1", "Bul2", "Hun", "Ira", "Ita", "Pol",
"Spa", "Tur", "UK")

# Native populations were added first(Asia)
for (i in 1:length(native.names)) {
  write("POP",file="Data/genepop.txt",append=TRUE)
  write("POP",file="Data/genepop_native.txt",append=TRUE)
  print(native.names[i])
  subset=genind.df[which(genind.df$pop %in% native.names[i]),]
  for (s in 1:nrow(subset)) {
    write(paste(subset[s,1],paste(subset[s,-1],collapse=" "),sep=",
"),file="Data/genepop.txt",append=TRUE)
    write(paste(subset[s,1],paste(subset[s,-1],collapse=" "),sep=",
"),file="Data/genepop_native.txt",append=TRUE)
  }
}

# Then invasive populations were added (Europe)
for (i in 1:length(invasive.names)) {
  write("POP",file="Data/genepop.txt",append=TRUE)
  write("POP",file="Data/genepop_invasive.txt",append=TRUE)
  print(invasive.names[i])
  subset=genind.df[which(genind.df$pop %in% invasive.names[i]),]
  for (s in 1:nrow(subset)) {
    write(paste(subset[s,1],paste(subset[s,-1],collapse=" "),sep=",
"),file="Data/genepop.txt",append=TRUE)
    write(paste(subset[s,1],paste(subset[s,-1],collapse=" "),sep=",
"),file="Data/genepop_invasive.txt",append=TRUE)
  }
}
```

```
#--------------------------------
# GENEPOP files with demes
genind.df=genind2df(Pparva)
# !!!!! Missing data should be indicated with 00
# Missing data (NA) was transformed in '0000'
genind.df[is.na(genind.df)]="0000"
# Open connection to new genepop files: 2 dataset (Asian and European)
file.create("Data/genepop_native_demes.txt")
file.create("Data/genepop_invasive_demes.txt")
# print first a title line
write.table("Phylogeography of Pseudorasbora parva: demes of Native
area","Data/genepop_native_demes.txt",sep="
",quote=FALSE,row.names=F,col.names=F)
write.table("Phylogeography of Pseudorasbora parva: demes of Invasive
area","Data/genepop_invasive_demes.txt",sep="
",quote=FALSE,row.names=F,col.names=F)

# Locus names: They may be given one per line, or on the same line but
separated by commas.
write(paste(colnames(genind.df)[-1],collapse =" ,
"),file="Data/genepop_native_demes.txt",append=TRUE)
write(paste(colnames(genind.df)[-1],collapse =" ,
"),file="Data/genepop_invasive_demes.txt",append=TRUE)

# Then, one individual per line, with the following nomenclature 'ind_name ,
locus1 locus2 ...'
# Each sample from a different geographical original is declared by a line
with a POP statement.

# Print each population at a time
# Lists of demes
genind.df.demes = genind.df
genind.df.demes[,1]=as.character(genind.df.demes[,1])
genind.df.demes[which(genind.df.demes$pop %in%
c("S4","S6")),1]="AdmixedCoastalChina"
genind.df.demes[which(genind.df.demes$pop %in%
c("S10","S11")),1]="AdmixedContinentalChina"
genind.df.demes[which(genind.df.demes$pop %in% c("S13","S14", "S15",
"S17")),1]="NorthChina"
genind.df.demes[which(genind.df.demes$pop %in% c("S1", "S2", "S3",
"S16")),1]="NorthCentralChina"
genind.df.demes[which(genind.df.demes$pop %in% c("S9", "S18", "S19",
"S20")),1]="SouthChina"

genind.df.demes[which(genind.df.demes$pop %in% c("Aus", "Bel", "Hun", "Ita",
"Pol", "Spa", "UK")),1]="WesternEurope"
genind.df.demes[which(genind.df.demes$pop %in% c("Bul1", "Bul2", "Tur",
"S20")),1]="EasternEurope"

native.demes=c("NorthChina", "NorthCentralChina", "AdmixedContinentalChina",
```

```
"AdmixedCoastalChina",
                "SouthChina", "Jap", "Tib")
invasive.demes=c("WesternEurope", "EasternEurope", "Ira")

# Native demes
for (i in 1:length(native.demes)) {
  write("POP",file="Data/genepop_native_demes.txt",append=TRUE)
  print(native.demes[i])
  subset=genind.df.demes[which(genind.df.demes$pop %in% native.demes[i]),]
  for (s in 1:nrow(subset)) {
    write(paste(subset[s,1],paste(subset[s,-1],collapse=" "),sep=",
"),file="Data/genepop_native_demes.txt",append=TRUE)
  }
}

# Then invasive demes
for (i in 1:length(invasive.demes)) {
  write("POP",file="Data/genepop_invasive_demes.txt",append=TRUE)
  print(invasive.demes[i])
  subset=genind.df.demes[which(genind.df.demes$pop %in% invasive.demes[i]),]
  for (s in 1:nrow(subset)) {
    write(paste(subset[s,1],paste(subset[s,-1],collapse=" "),sep=",
"),file="Data/genepop_invasive_demes.txt",append=TRUE)
  }
}
```

STRUCTURE workflow (bash script for UNIX server)

A first script, 'structure.sh', launched all successive steps of STRUCTURE workflow, with a dependency on a second script 'structure_sampling _chain.sh'. 'structure.sh' was produced automatically by the 'structureLauncher.R' script during STRUCTURE datasets formatting. Here is the script 'structureLauncher.R' followed by an exemple for one dataset only.

```
#============================================================
# GENERATE FILES .SH  TO LAUNCH STRUCTURE ON A LINUX SERVER
#============================================================

structureLauncher=function(project="project",path="",niter=c(1,10),Kmin=1,Kma
x=10,popsize=727){
  # project="project, the name of the project
  # path="", the path of the final working directory in the server
  # niter=c(1,10), numerotation of multiple successive iterations for the
same value of K, necessary for Evanno's method
  # Kmin, minimum number of K to test
  # Kmax, maximum number of K to test
  # popsize=727, population size has to be documented

  # Init a launcher object that will contain all commands
  launcher=c()
  # Go to working directory
```

```
  launcher=rbind(launcher,
                 paste("cd",path,sep=" "),
                 "",
                 "# Set programs as executables",
                 "chmod +x ./structure",
                 "# Convert data files ot Unix encoding",
                 "# Data files can be recognized by the pattern '00_' e.g.
'200_Pparva_structure_native.txt'",
                 "dos2unix *00_*",
                 "",
                 "# Launch batch runs for all values of K and every
iterations")

    for (j in Kmin:Kmax) {
      # iteration is set with a random seed between 1000 and 9999, and output
of structure is saved in a txt file
      iter=paste("nohup ./structure -K ",j," -D ",sample(1:9999,1)," -o
Results/K",j,"run",niter[1]," >
Results/outputK",j,"run",niter[1],".txt",sep="")
      for (i in (niter[1]+1):niter[2]) {
        iter=paste(iter,paste(" & nohup ./structure -K ",j," -D
",sample(1:9999,1)," -o Results/K",j,"run",i," >
Results/outputK",j,"run",i,".txt",sep=""),sep="")
      }
      launcher=rbind(launcher,iter)
    }

    launcher=rbind(launcher,
                   "",
                   "# STRUCTURE Harvester on results to:",
                   "# apply Evanno method (2005) to choose the best K",
                   "# prepare files for CLUMPP",
                   "python structureHarvester.py --dir=./Results/ --
out=./Harvester/ --evanno --clumpp",
                   "",
                   "# Get the values of the sampling chain from 'outputK-run-
.txt'",
                   "# args are in order Kmin, Kmax and number of replicated
runs",
                   "chmod +x ./structure_sampling_chain.sh",
                   paste("./structure_sampling_chain.sh", Kmin, Kmax,
niter[2],sep=" "),
                   "",
                   "# CLUMPP on files formated by STRUCTURE Harvester",
                   "# Launch a CLUMPP task on each K (except K=1, no
segmentation possible on a single cluster)",
                   "chmod +x ./CLUMPP/CLUMPP",
                   paste("for ((i = ",niter[1],"; i < ",niter[2]+1,"; i++ ));
do",sep=""),
                   paste("./CLUMPP/CLUMPP ./CLUMPP/paramfile -i
```

```
'./Harvester/K'$i'.indfile' -p './Harvester/K'$i'.popfile' -o
'./CLUMPP/K'$i'.outfile' -j './CLUMPP/K'$i'.miscfile' -k $i -c ",popsize," -r
",(niter[2]-niter[1]+1)," &",sep=""),
                    "done",
                    "",
                    "# Distruct to display nice graphs of STRUCTURE, with
labels",
                    "# Automatically produce bar plots for each K value",
                    "",
                    "# Output files are copied to the 'data' directory in
'Distruct'",
                    paste("for ((i = ",niter[1],"; i < ",niter[2]+1,"; i++ ));
do",sep=""),
                   "cp './CLUMPP/K'$i'.outfile'
'./Distruct/Data/K'$i'.outfile'",
                    "head -n 18 './Harvester/K'$i'.popfile' >>
'./Distruct/Data/K'$i'.popfile'",
                   "done",
                    "",
                   "cd Distruct",
                    "chmod +x ./distructLinux1.1",
                    paste("for ((i = ",niter[1],"; i < ",niter[2]+1,"; i++ ));
do",sep=""),
                    paste("./distructLinux1.1 -K $i -M 18 -N ",popsize," -p
'Data/K'$i'.popfile' -i 'Data/K'$i'.outfile' -o
'Plots/DistructPlotK'$i'.ps'",sep=""),
                    "done",
                   "",
                   "# Convert .ps as .tiff",
                   "cd Plots/",
                    paste("for ((i = ",niter[1],"; i < ",niter[2]+1,"; i++ ));
do",sep=""),
                    "gs -sDEVICE=tiff64nc -r300 -sPAPERSIZE=a4 -dBATCH -
dNOPAUSE -sOutputFile='DistructPlotK'$i'.tiff' 'DistructPlotK'$i'.ps'",
                    "done",
                   "cd ..",
                   "cd ..")

      ## Write data file for bash
    if (!dir.exists(paste("Data/STRUCTURE/",project,sep=""))){
      dir.create(paste("Data/STRUCTURE/",project,sep=""))
    }

file.create(paste("Data/STRUCTURE/",project,"/structure.sh",sep=""),overwrite
=TRUE)
    write.table(rbind("#!/bin/sh","# file:
structure.sh\n",launcher),paste("Data/STRUCTURE/",project,"/structure.sh",sep
=""),
                quote=F,row.names = F,col.names = F)
  }
```

Here is an example of the bash script to launch a complete STRUCTURE analysis.

```
#!/bin/sh
# file: structure.sh

cd /home/users/tbrazier/STRUCTURE/1c_AllPops

# Set programs as executables
chmod +x ./structure
# Convert data files ot Unix encoding
# Data files can be recognized by the pattern '00_' e.g.
'200_Pparva_structure_native.txt'
dos2unix *00_*

# Launch batch runs for all values of K and every iterations
./structure -K 1 -D 7134 -o Results/K1run1 > Results/outputK1run1.txt &
./structure -K 1 -D 9353 -o Results/K1run2 > Results/outputK1run2.txt &
./structure -K 1 -D 2249 -o Results/K1run3 > Results/outputK1run3.txt &
./structure -K 1 -D 9171 -o Results/K1run4 > Results/outputK1run4.txt &
./structure -K 1 -D 7515 -o Results/K1run5 > Results/outputK1run5.txt &
./structure -K 1 -D 7991 -o Results/K1run6 > Results/outputK1run6.txt &
./structure -K 1 -D 3904 -o Results/K1run7 > Results/outputK1run7.txt &
./structure -K 1 -D 8684 -o Results/K1run8 > Results/outputK1run8.txt &
./structure -K 1 -D 4845 -o Results/K1run9 > Results/outputK1run9.txt &
./structure -K 1 -D 7389 -o Results/K1run10 > Results/outputK1run10.txt &
./structure -K 1 -D 8180 -o Results/K1run11 > Results/outputK1run11.txt &
./structure -K 1 -D 8257 -o Results/K1run12 > Results/outputK1run12.txt &
./structure -K 1 -D 6189 -o Results/K1run13 > Results/outputK1run13.txt &
./structure -K 1 -D 7481 -o Results/K1run14 > Results/outputK1run14.txt &
./structure -K 1 -D 3345 -o Results/K1run15 > Results/outputK1run15.txt &
./structure -K 1 -D 8947 -o Results/K1run16 > Results/outputK1run16.txt &
./structure -K 1 -D 8539 -o Results/K1run17 > Results/outputK1run17.txt &
./structure -K 1 -D 7399 -o Results/K1run18 > Results/outputK1run18.txt &
./structure -K 1 -D 2388 -o Results/K1run19 > Results/outputK1run19.txt &
./structure -K 1 -D 2418 -o Results/K1run20 > Results/outputK1run20.txt
./structure -K 2 -D 2241 -o Results/K2run1 > Results/outputK2run1.txt &
./structure -K 2 -D 9590 -o Results/K2run2 > Results/outputK2run2.txt &
./structure -K 2 -D 3661 -o Results/K2run3 > Results/outputK2run3.txt &
./structure -K 2 -D 3026 -o Results/K2run4 > Results/outputK2run4.txt &
./structure -K 2 -D 4767 -o Results/K2run5 > Results/outputK2run5.txt &
./structure -K 2 -D 1452 -o Results/K2run6 > Results/outputK2run6.txt &
./structure -K 2 -D 4733 -o Results/K2run7 > Results/outputK2run7.txt &
./structure -K 2 -D 3085 -o Results/K2run8 > Results/outputK2run8.txt &
./structure -K 2 -D 5141 -o Results/K2run9 > Results/outputK2run9.txt &
./structure -K 2 -D 4375 -o Results/K2run10 > Results/outputK2run10.txt &
./structure -K 2 -D 1600 -o Results/K2run11 > Results/outputK2run11.txt &
./structure -K 2 -D 6747 -o Results/K2run12 > Results/outputK2run12.txt &
./structure -K 2 -D 3353 -o Results/K2run13 > Results/outputK2run13.txt &
./structure -K 2 -D 7333 -o Results/K2run14 > Results/outputK2run14.txt &
./structure -K 2 -D 5693 -o Results/K2run15 > Results/outputK2run15.txt &
```

```
./structure -K 2 -D 3242 -o Results/K2run16 > Results/outputK2run16.txt &
./structure -K 2 -D 5204 -o Results/K2run17 > Results/outputK2run17.txt &
./structure -K 2 -D 3436 -o Results/K2run18 > Results/outputK2run18.txt &
./structure -K 2 -D 3444 -o Results/K2run19 > Results/outputK2run19.txt &
./structure -K 2 -D 2738 -o Results/K2run20 > Results/outputK2run20.txt
./structure -K 3 -D 1783 -o Results/K3run1 > Results/outputK3run1.txt &
./structure -K 3 -D 7485 -o Results/K3run2 > Results/outputK3run2.txt &
./structure -K 3 -D 3195 -o Results/K3run3 > Results/outputK3run3.txt &
./structure -K 3 -D 8012 -o Results/K3run4 > Results/outputK3run4.txt &
./structure -K 3 -D 5745 -o Results/K3run5 > Results/outputK3run5.txt &
./structure -K 3 -D 7881 -o Results/K3run6 > Results/outputK3run6.txt &
./structure -K 3 -D 3983 -o Results/K3run7 > Results/outputK3run7.txt &
./structure -K 3 -D 7181 -o Results/K3run8 > Results/outputK3run8.txt &
./structure -K 3 -D 2389 -o Results/K3run9 > Results/outputK3run9.txt &
./structure -K 3 -D 1158 -o Results/K3run10 > Results/outputK3run10.txt &
./structure -K 3 -D 3472 -o Results/K3run11 > Results/outputK3run11.txt &
./structure -K 3 -D 3989 -o Results/K3run12 > Results/outputK3run12.txt &
./structure -K 3 -D 3071 -o Results/K3run13 > Results/outputK3run13.txt &
./structure -K 3 -D 2373 -o Results/K3run14 > Results/outputK3run14.txt &
./structure -K 3 -D 6423 -o Results/K3run15 > Results/outputK3run15.txt &
./structure -K 3 -D 2489 -o Results/K3run16 > Results/outputK3run16.txt &
./structure -K 3 -D 4602 -o Results/K3run17 > Results/outputK3run17.txt &
./structure -K 3 -D 6737 -o Results/K3run18 > Results/outputK3run18.txt &
./structure -K 3 -D 9891 -o Results/K3run19 > Results/outputK3run19.txt &
./structure -K 3 -D 2862 -o Results/K3run20 > Results/outputK3run20.txt
./structure -K 4 -D 7426 -o Results/K4run1 > Results/outputK4run1.txt &
./structure -K 4 -D 5198 -o Results/K4run2 > Results/outputK4run2.txt &
./structure -K 4 -D 8043 -o Results/K4run3 > Results/outputK4run3.txt &
./structure -K 4 -D 9042 -o Results/K4run4 > Results/outputK4run4.txt &
./structure -K 4 -D 2526 -o Results/K4run5 > Results/outputK4run5.txt &
./structure -K 4 -D 6947 -o Results/K4run6 > Results/outputK4run6.txt &
./structure -K 4 -D 9915 -o Results/K4run7 > Results/outputK4run7.txt &
./structure -K 4 -D 9438 -o Results/K4run8 > Results/outputK4run8.txt &
./structure -K 4 -D 2300 -o Results/K4run9 > Results/outputK4run9.txt &
./structure -K 4 -D 4612 -o Results/K4run10 > Results/outputK4run10.txt &
./structure -K 4 -D 9752 -o Results/K4run11 > Results/outputK4run11.txt &
./structure -K 4 -D 6599 -o Results/K4run12 > Results/outputK4run12.txt &
./structure -K 4 -D 7435 -o Results/K4run13 > Results/outputK4run13.txt &
./structure -K 4 -D 9827 -o Results/K4run14 > Results/outputK4run14.txt &
./structure -K 4 -D 1418 -o Results/K4run15 > Results/outputK4run15.txt &
./structure -K 4 -D 2944 -o Results/K4run16 > Results/outputK4run16.txt &
./structure -K 4 -D 8378 -o Results/K4run17 > Results/outputK4run17.txt &
./structure -K 4 -D 4282 -o Results/K4run18 > Results/outputK4run18.txt &
./structure -K 4 -D 1124 -o Results/K4run19 > Results/outputK4run19.txt &
./structure -K 4 -D 5421 -o Results/K4run20 > Results/outputK4run20.txt
./structure -K 5 -D 4761 -o Results/K5run1 > Results/outputK5run1.txt &
./structure -K 5 -D 9219 -o Results/K5run2 > Results/outputK5run2.txt &
./structure -K 5 -D 3922 -o Results/K5run3 > Results/outputK5run3.txt &
./structure -K 5 -D 9533 -o Results/K5run4 > Results/outputK5run4.txt &
./structure -K 5 -D 8249 -o Results/K5run5 > Results/outputK5run5.txt &
```

```
./structure -K 5 -D 2255 -o Results/K5run6 > Results/outputK5run6.txt &
./structure -K 5 -D 4454 -o Results/K5run7 > Results/outputK5run7.txt &
./structure -K 5 -D 2905 -o Results/K5run8 > Results/outputK5run8.txt &
./structure -K 5 -D 3640 -o Results/K5run9 > Results/outputK5run9.txt &
./structure -K 5 -D 9618 -o Results/K5run10 > Results/outputK5run10.txt &
./structure -K 5 -D 2420 -o Results/K5run11 > Results/outputK5run11.txt &
./structure -K 5 -D 2763 -o Results/K5run12 > Results/outputK5run12.txt &
./structure -K 5 -D 6617 -o Results/K5run13 > Results/outputK5run13.txt &
./structure -K 5 -D 1990 -o Results/K5run14 > Results/outputK5run14.txt &
./structure -K 5 -D 3670 -o Results/K5run15 > Results/outputK5run15.txt &
./structure -K 5 -D 7742 -o Results/K5run16 > Results/outputK5run16.txt &
./structure -K 5 -D 2314 -o Results/K5run17 > Results/outputK5run17.txt &
./structure -K 5 -D 8351 -o Results/K5run18 > Results/outputK5run18.txt &
./structure -K 5 -D 2468 -o Results/K5run19 > Results/outputK5run19.txt &
./structure -K 5 -D 8909 -o Results/K5run20 > Results/outputK5run20.txt
./structure -K 6 -D 2701 -o Results/K6run1 > Results/outputK6run1.txt &
./structure -K 6 -D 3097 -o Results/K6run2 > Results/outputK6run2.txt &
./structure -K 6 -D 2360 -o Results/K6run3 > Results/outputK6run3.txt &
./structure -K 6 -D 1942 -o Results/K6run4 > Results/outputK6run4.txt &
./structure -K 6 -D 3297 -o Results/K6run5 > Results/outputK6run5.txt &
./structure -K 6 -D 1121 -o Results/K6run6 > Results/outputK6run6.txt &
./structure -K 6 -D 4128 -o Results/K6run7 > Results/outputK6run7.txt &
./structure -K 6 -D 7727 -o Results/K6run8 > Results/outputK6run8.txt &
./structure -K 6 -D 7848 -o Results/K6run9 > Results/outputK6run9.txt &
./structure -K 6 -D 1030 -o Results/K6run10 > Results/outputK6run10.txt &
./structure -K 6 -D 6765 -o Results/K6run11 > Results/outputK6run11.txt &
./structure -K 6 -D 2615 -o Results/K6run12 > Results/outputK6run12.txt &
./structure -K 6 -D 9023 -o Results/K6run13 > Results/outputK6run13.txt &
./structure -K 6 -D 9360 -o Results/K6run14 > Results/outputK6run14.txt &
./structure -K 6 -D 2074 -o Results/K6run15 > Results/outputK6run15.txt &
./structure -K 6 -D 2853 -o Results/K6run16 > Results/outputK6run16.txt &
./structure -K 6 -D 4835 -o Results/K6run17 > Results/outputK6run17.txt &
./structure -K 6 -D 7218 -o Results/K6run18 > Results/outputK6run18.txt &
./structure -K 6 -D 1782 -o Results/K6run19 > Results/outputK6run19.txt &
./structure -K 6 -D 2505 -o Results/K6run20 > Results/outputK6run20.txt
./structure -K 7 -D 8462 -o Results/K7run1 > Results/outputK7run1.txt &
./structure -K 7 -D 9188 -o Results/K7run2 > Results/outputK7run2.txt &
./structure -K 7 -D 4861 -o Results/K7run3 > Results/outputK7run3.txt &
./structure -K 7 -D 1166 -o Results/K7run4 > Results/outputK7run4.txt &
./structure -K 7 -D 1099 -o Results/K7run5 > Results/outputK7run5.txt &
./structure -K 7 -D 7843 -o Results/K7run6 > Results/outputK7run6.txt &
./structure -K 7 -D 5348 -o Results/K7run7 > Results/outputK7run7.txt &
./structure -K 7 -D 2908 -o Results/K7run8 > Results/outputK7run8.txt &
./structure -K 7 -D 1688 -o Results/K7run9 > Results/outputK7run9.txt &
./structure -K 7 -D 7638 -o Results/K7run10 > Results/outputK7run10.txt &
./structure -K 7 -D 1426 -o Results/K7run11 > Results/outputK7run11.txt &
./structure -K 7 -D 3401 -o Results/K7run12 > Results/outputK7run12.txt &
./structure -K 7 -D 2985 -o Results/K7run13 > Results/outputK7run13.txt &
./structure -K 7 -D 9213 -o Results/K7run14 > Results/outputK7run14.txt &
./structure -K 7 -D 4636 -o Results/K7run15 > Results/outputK7run15.txt &
```

```
./structure -K 7 -D 4542 -o Results/K7run16 > Results/outputK7run16.txt &
./structure -K 7 -D 5340 -o Results/K7run17 > Results/outputK7run17.txt &
./structure -K 7 -D 8154 -o Results/K7run18 > Results/outputK7run18.txt &
./structure -K 7 -D 2805 -o Results/K7run19 > Results/outputK7run19.txt &
./structure -K 7 -D 6894 -o Results/K7run20 > Results/outputK7run20.txt
./structure -K 8 -D 4887 -o Results/K8run1 > Results/outputK8run1.txt &
./structure -K 8 -D 5240 -o Results/K8run2 > Results/outputK8run2.txt &
./structure -K 8 -D 7140 -o Results/K8run3 > Results/outputK8run3.txt &
./structure -K 8 -D 6424 -o Results/K8run4 > Results/outputK8run4.txt &
./structure -K 8 -D 2033 -o Results/K8run5 > Results/outputK8run5.txt &
./structure -K 8 -D 4927 -o Results/K8run6 > Results/outputK8run6.txt &
./structure -K 8 -D 2820 -o Results/K8run7 > Results/outputK8run7.txt &
./structure -K 8 -D 1586 -o Results/K8run8 > Results/outputK8run8.txt &
./structure -K 8 -D 8135 -o Results/K8run9 > Results/outputK8run9.txt &
./structure -K 8 -D 1531 -o Results/K8run10 > Results/outputK8run10.txt &
./structure -K 8 -D 6196 -o Results/K8run11 > Results/outputK8run11.txt &
./structure -K 8 -D 1707 -o Results/K8run12 > Results/outputK8run12.txt &
./structure -K 8 -D 6368 -o Results/K8run13 > Results/outputK8run13.txt &
./structure -K 8 -D 3516 -o Results/K8run14 > Results/outputK8run14.txt &
./structure -K 8 -D 7475 -o Results/K8run15 > Results/outputK8run15.txt &
./structure -K 8 -D 3110 -o Results/K8run16 > Results/outputK8run16.txt &
./structure -K 8 -D 9982 -o Results/K8run17 > Results/outputK8run17.txt &
./structure -K 8 -D 3467 -o Results/K8run18 > Results/outputK8run18.txt &
./structure -K 8 -D 9768 -o Results/K8run19 > Results/outputK8run19.txt &
./structure -K 8 -D 7407 -o Results/K8run20 > Results/outputK8run20.txt
./structure -K 9 -D 2922 -o Results/K9run1 > Results/outputK9run1.txt &
./structure -K 9 -D 8243 -o Results/K9run2 > Results/outputK9run2.txt &
./structure -K 9 -D 5550 -o Results/K9run3 > Results/outputK9run3.txt &
./structure -K 9 -D 3790 -o Results/K9run4 > Results/outputK9run4.txt &
./structure -K 9 -D 2497 -o Results/K9run5 > Results/outputK9run5.txt &
./structure -K 9 -D 6587 -o Results/K9run6 > Results/outputK9run6.txt &
./structure -K 9 -D 6778 -o Results/K9run7 > Results/outputK9run7.txt &
./structure -K 9 -D 5137 -o Results/K9run8 > Results/outputK9run8.txt &
./structure -K 9 -D 7031 -o Results/K9run9 > Results/outputK9run9.txt &
./structure -K 9 -D 3618 -o Results/K9run10 > Results/outputK9run10.txt &
./structure -K 9 -D 8349 -o Results/K9run11 > Results/outputK9run11.txt &
./structure -K 9 -D 8335 -o Results/K9run12 > Results/outputK9run12.txt &
./structure -K 9 -D 4459 -o Results/K9run13 > Results/outputK9run13.txt &
./structure -K 9 -D 6652 -o Results/K9run14 > Results/outputK9run14.txt &
./structure -K 9 -D 4347 -o Results/K9run15 > Results/outputK9run15.txt &
./structure -K 9 -D 9451 -o Results/K9run16 > Results/outputK9run16.txt &
./structure -K 9 -D 6699 -o Results/K9run17 > Results/outputK9run17.txt &
./structure -K 9 -D 2930 -o Results/K9run18 > Results/outputK9run18.txt &
./structure -K 9 -D 6371 -o Results/K9run19 > Results/outputK9run19.txt &
./structure -K 9 -D 5585 -o Results/K9run20 > Results/outputK9run20.txt
./structure -K 10 -D 4346 -o Results/K10run1 > Results/outputK10run1.txt &
./structure -K 10 -D 8719 -o Results/K10run2 > Results/outputK10run2.txt &
./structure -K 10 -D 5719 -o Results/K10run3 > Results/outputK10run3.txt &
./structure -K 10 -D 6193 -o Results/K10run4 > Results/outputK10run4.txt &
./structure -K 10 -D 1410 -o Results/K10run5 > Results/outputK10run5.txt &
```

```
./structure -K 10 -D 4536 -o Results/K10run6 > Results/outputK10run6.txt &
./structure -K 10 -D 9442 -o Results/K10run7 > Results/outputK10run7.txt &
./structure -K 10 -D 7244 -o Results/K10run8 > Results/outputK10run8.txt &
./structure -K 10 -D 7091 -o Results/K10run9 > Results/outputK10run9.txt &
./structure -K 10 -D 8545 -o Results/K10run10 > Results/outputK10run10.txt &
./structure -K 10 -D 2113 -o Results/K10run11 > Results/outputK10run11.txt &
./structure -K 10 -D 7568 -o Results/K10run12 > Results/outputK10run12.txt &
./structure -K 10 -D 3753 -o Results/K10run13 > Results/outputK10run13.txt &
./structure -K 10 -D 9170 -o Results/K10run14 > Results/outputK10run14.txt &
./structure -K 10 -D 3490 -o Results/K10run15 > Results/outputK10run15.txt &
./structure -K 10 -D 5349 -o Results/K10run16 > Results/outputK10run16.txt &
./structure -K 10 -D 3238 -o Results/K10run17 > Results/outputK10run17.txt &
./structure -K 10 -D 6528 -o Results/K10run18 > Results/outputK10run18.txt &
./structure -K 10 -D 2050 -o Results/K10run19 > Results/outputK10run19.txt &
./structure -K 10 -D 5769 -o Results/K10run20 > Results/outputK10run20.txt
./structure -K 11 -D 4526 -o Results/K11run1 > Results/outputK11run1.txt &
./structure -K 11 -D 3456 -o Results/K11run2 > Results/outputK11run2.txt &
./structure -K 11 -D 2320 -o Results/K11run3 > Results/outputK11run3.txt &
./structure -K 11 -D 8279 -o Results/K11run4 > Results/outputK11run4.txt &
./structure -K 11 -D 3895 -o Results/K11run5 > Results/outputK11run5.txt &
./structure -K 11 -D 8222 -o Results/K11run6 > Results/outputK11run6.txt &
./structure -K 11 -D 1177 -o Results/K11run7 > Results/outputK11run7.txt &
./structure -K 11 -D 1650 -o Results/K11run8 > Results/outputK11run8.txt &
./structure -K 11 -D 8999 -o Results/K11run9 > Results/outputK11run9.txt &
./structure -K 11 -D 2267 -o Results/K11run10 > Results/outputK11run10.txt &
./structure -K 11 -D 3834 -o Results/K11run11 > Results/outputK11run11.txt &
./structure -K 11 -D 1177 -o Results/K11run12 > Results/outputK11run12.txt &
./structure -K 11 -D 1446 -o Results/K11run13 > Results/outputK11run13.txt &
./structure -K 11 -D 3503 -o Results/K11run14 > Results/outputK11run14.txt &
./structure -K 11 -D 1168 -o Results/K11run15 > Results/outputK11run15.txt &
./structure -K 11 -D 4994 -o Results/K11run16 > Results/outputK11run16.txt &
./structure -K 11 -D 2793 -o Results/K11run17 > Results/outputK11run17.txt &
./structure -K 11 -D 1775 -o Results/K11run18 > Results/outputK11run18.txt &
./structure -K 11 -D 3123 -o Results/K11run19 > Results/outputK11run19.txt &
./structure -K 11 -D 5054 -o Results/K11run20 > Results/outputK11run20.txt
./structure -K 12 -D 3908 -o Results/K12run1 > Results/outputK12run1.txt &
./structure -K 12 -D 7398 -o Results/K12run2 > Results/outputK12run2.txt &
./structure -K 12 -D 2989 -o Results/K12run3 > Results/outputK12run3.txt &
./structure -K 12 -D 4017 -o Results/K12run4 > Results/outputK12run4.txt &
./structure -K 12 -D 7130 -o Results/K12run5 > Results/outputK12run5.txt &
./structure -K 12 -D 6205 -o Results/K12run6 > Results/outputK12run6.txt &
./structure -K 12 -D 8358 -o Results/K12run7 > Results/outputK12run7.txt &
./structure -K 12 -D 4406 -o Results/K12run8 > Results/outputK12run8.txt &
./structure -K 12 -D 2882 -o Results/K12run9 > Results/outputK12run9.txt &
./structure -K 12 -D 5376 -o Results/K12run10 > Results/outputK12run10.txt &
./structure -K 12 -D 2939 -o Results/K12run11 > Results/outputK12run11.txt &
./structure -K 12 -D 4348 -o Results/K12run12 > Results/outputK12run12.txt &
./structure -K 12 -D 5962 -o Results/K12run13 > Results/outputK12run13.txt &
./structure -K 12 -D 5091 -o Results/K12run14 > Results/outputK12run14.txt &
./structure -K 12 -D 3198 -o Results/K12run15 > Results/outputK12run15.txt &
```

```
./structure -K 12 -D 9313 -o Results/K12run16 > Results/outputK12run16.txt &
./structure -K 12 -D 2081 -o Results/K12run17 > Results/outputK12run17.txt &
./structure -K 12 -D 8160 -o Results/K12run18 > Results/outputK12run18.txt &
./structure -K 12 -D 6306 -o Results/K12run19 > Results/outputK12run19.txt &
./structure -K 12 -D 4698 -o Results/K12run20 > Results/outputK12run20.txt
./structure -K 13 -D 1468 -o Results/K13run1 > Results/outputK13run1.txt &
./structure -K 13 -D 6749 -o Results/K13run2 > Results/outputK13run2.txt &
./structure -K 13 -D 3449 -o Results/K13run3 > Results/outputK13run3.txt &
./structure -K 13 -D 8883 -o Results/K13run4 > Results/outputK13run4.txt &
./structure -K 13 -D 3682 -o Results/K13run5 > Results/outputK13run5.txt &
./structure -K 13 -D 6491 -o Results/K13run6 > Results/outputK13run6.txt &
./structure -K 13 -D 2667 -o Results/K13run7 > Results/outputK13run7.txt &
./structure -K 13 -D 7451 -o Results/K13run8 > Results/outputK13run8.txt &
./structure -K 13 -D 1292 -o Results/K13run9 > Results/outputK13run9.txt &
./structure -K 13 -D 6835 -o Results/K13run10 > Results/outputK13run10.txt &
./structure -K 13 -D 7749 -o Results/K13run11 > Results/outputK13run11.txt &
./structure -K 13 -D 4072 -o Results/K13run12 > Results/outputK13run12.txt &
./structure -K 13 -D 3654 -o Results/K13run13 > Results/outputK13run13.txt &
./structure -K 13 -D 6459 -o Results/K13run14 > Results/outputK13run14.txt &
./structure -K 13 -D 7870 -o Results/K13run15 > Results/outputK13run15.txt &
./structure -K 13 -D 5196 -o Results/K13run16 > Results/outputK13run16.txt &
./structure -K 13 -D 8650 -o Results/K13run17 > Results/outputK13run17.txt &
./structure -K 13 -D 2693 -o Results/K13run18 > Results/outputK13run18.txt &
./structure -K 13 -D 5335 -o Results/K13run19 > Results/outputK13run19.txt &
./structure -K 13 -D 7350 -o Results/K13run20 > Results/outputK13run20.txt

# STRUCTURE Harvester on results to:
# apply Evanno method (2005) to choose the best K
# prepare files for CLUMPP
python structureHarvester.py --dir=./Results/ --out=./Harvester/ --evanno --
clumpp

# Get the values of the sampling chain from 'outputK-run-.txt'
# args are in order Kmin, Kmax and number of replicated runs
./structure_sampling_chain.sh 1 13 20

# CLUMPP on files formated by STRUCTURE Harvester
# Launch a CLUMPP task on each K (except K=1, no segmentation possible on a
single cluster)
chmod +x ./CLUMPP/CLUMPP
for ((i = 1; i < 21; i++ )); do
./CLUMPP/CLUMPP ./CLUMPP/paramfile -i './Harvester/K'$i'.indfile' -p
'./Harvester/K'$i'.popfile' -o './CLUMPP/K'$i'.outfile' -j
'./CLUMPP/K'$i'.miscfile' -k $i -c 468 -r 20 &
done

# Distruct to display nice graphs of STRUCTURE, with labels
# Automatically produce bar plots for each K value

# Output files are copied to the 'data' directory in 'Distruct'
```

```
for ((i = 1; i < 21; i++ )); do
cp './CLUMPP/K'$i'.outfile' './Distruct/Data/K'$i'.outfile'
head -n 18 './Harvester/K'$i'.popfile' >> './Distruct/Data/K'$i'.popfile'
done

cd Distruct
chmod +x ./distructLinux1.1
for ((i = 1; i < 21; i++ )); do
./distructLinux1.1 -K $i -M 18 -N 468 -p 'Data/K'$i'.popfile' -i
'Data/K'$i'.outfile' -o 'Plots/DistructPlotK'$i'.ps'
done

# Convert .ps as .tiff
cd Plots/
for ((i = 1; i < 21; i++ )); do
gs -sDEVICE=tiff64nc -r300 -sPAPERSIZE=a4 -dBATCH -dNOPAUSE -
sOutputFile='DistructPlotK'$i'.tiff' 'DistructPlotK'$i'.ps'
done
cd ..
cd ..
```

The script 'structure_sampling _chain.sh' was used to get traces of the MCMC sampling chains in a correct format before importing them in R.

```
#=======================================================================
#           EXTRACTING RESULTS FROM STRUCTURE
#=======================================================================

#=======================================================================#
# Loop to Extract Posterior Estimates & Write to Dataframe
#-----------------------------------------------------------------------#
#!#!#! N.B. the names of standard output files will vary with each Structure
run
    # e.g. outputK1run1.txt
        # 'output' will be standard
        # 'K' & 'run' will be semi-standard
            # integers will vary based on the number of 'K' defined in
parameter set & the number of replicates (i.e. runs)

#-----------------------------------------------------------------------#
# Alpha & posterior estimates of P(D) (sampling chain) -- full chain with
burnin
#-----------------------------------------------------------------------#
# This part get values of Alpha, Ln Likelihood of P(Data) for each MCMC
iteration (burnin + whole length of MCMC), for all replicates (runs) of a
given K and write it into the file trace_alphaK${k}
# !!! One sampling chain per cluster
# Supplementary data: Values of F, D and r

#-----------------------------------------------------------------------#
```

```
# Init K value and number of replicates (number of runs)
# With K in {k1-k2}
# Values are taken in arguments, in this order: Kmin Kmax #Replicates
k1=$1
k2=$2
run=$3

cd Results/

for k in $(seq $k1 $k2);
do
  (echo Run; (grep -A 1 'BURNIN completed' outputK${k}run${run}.txt | grep -v
'BURNIN completed')) | paste - - > sampling_chain_K${k} # Init the file where
to save the trace of alpha and the sampling chain, with names of columns
  # Reformating space characters
  sed -i s/'Ln Like'/'Ln.Like'/'' sampling_chain_K${k}
  sed -i s/'Est Ln P(D)'/'Est.Ln.P'/'' sampling_chain_K${k}
  # print the number of fields
  fields="$(awk '{print NF}' sampling_chain_K${k})"

  for r in $(seq $run)
  do
    (grep -A 11 'Rep#:' outputK${k}run${r}.txt | grep -v 'Rep#:' \
    | grep '[^[:blank:]]' | grep '[[:blank:]$]') | awk 'BEGIN {FS=OFS="\t";
r='$r'};  {print r, (seq $fields)}' >> sampling_chain_K${k}
  done
  # Reformatting iterations: removing ':' character
  sed -i s/':'//'g' sampling_chain_K${k}
  # and ',' characters
  sed -i s/','//'g' sampling_chain_K${k}
  # Reformating BURNIN iterations: those with '--' became '--    --'
  sed -i s/'--'/'--  --'/'' sampling_chain_K${k}

  sed -i 's/\s/#/g' sampling_chain_K${k} # visualise how many spaces
  sed -i 's/--##--##/NA NA/' sampling_chain_K${k}
  sed -i 's/#A/ A/g' sampling_chain_K${k}
  sed -i 's/###0./ 0./' sampling_chain_K${k}
  sed -i 's/##0./ 0./' sampling_chain_K${k}
  sed -i 's/########/ /' sampling_chain_K${k}
  sed -i 's/######/ /' sampling_chain_K${k}
  sed -i 's/####/ /g' sampling_chain_K${k}
  sed -i 's/Like##/Like /' sampling_chain_K${k}
  sed -i 's/###/ /g' sampling_chain_K${k}
  sed -i 's/##/ /g' sampling_chain_K${k}
  sed -i 's/#/ /g' sampling_chain_K${k}

  sed -i 's/\s\s/ /g' sampling_chain_K${k}

  head sampling_chain_K${k}
```

```
done

cd ..
```