Molecular data in R

Phylogeny, evolution & R

Vojtěch Zeisek

Department of Botany, Faculty of Science, Charles University in Prague
Institute of Botany, Czech Academy of Sciences, Průhonice
http://trapa.cz/, zeisek@natur.cuni.cz

January 14 to 16, 2015

Outline I

Outline II

Outline III

Outline IV

What we will and what we will not do...

We will go through...

- Basic introduction into R
- Various methods and data for analysing phylogeny and evolution
  - DNA sequences, SNP, SSRs, AFLP, ...
  - NJ, UPGMA, PCoA, DAPC, Bayesian clustering, ML, maximum parsimony, ...
- Plotting, maps, ...
- Basic creation of scripts

We will not dig deep into...

- All differences among operating systems (OS; not big, but there are some...)
- Theory behind used methods (with some exceptions)
- Programming in R
- Other software related to the methods used but not directly related to R (with exceptions of applications called from R)

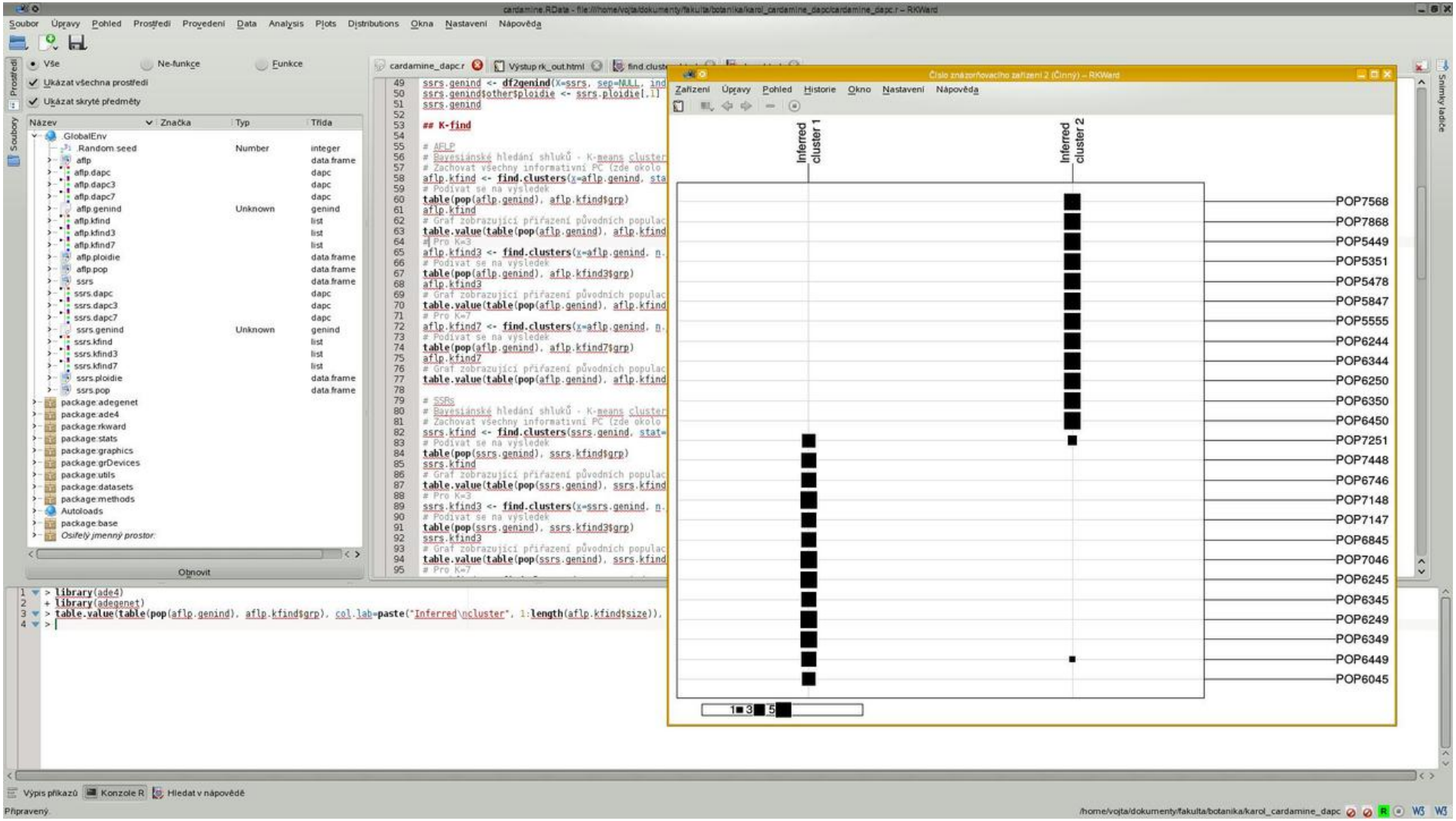
About R

- Project for Statistical Computing
- open-source — freely available with source code — anyone can use it and contribute its development
- Development is organised by non-governmental non-profit organisation from Wien
- Thousands of packages extending its functionality are available — all fields of computing
- Provides only command line interface — full control over analysis, easy to rerun and/or modify analysis in the future, easy creation of scripts for batch analysis
- Several projects provide convenient graphical interfaces (GUI)
- More details: http://www.r-project.org/

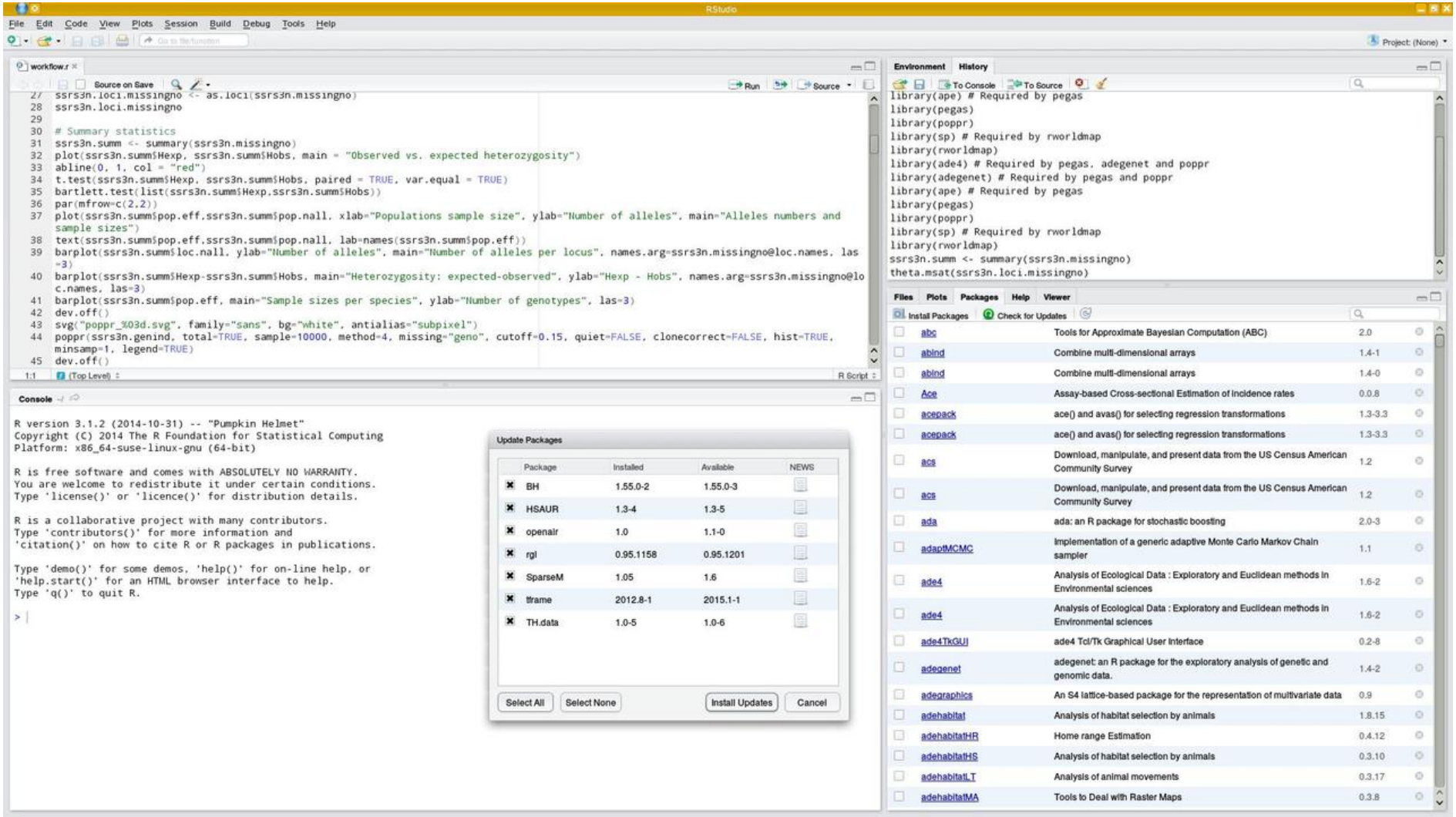
Graphical user interfaces

- Provide more comfortable interface for work with scripts (source code highlight), overview of loaded packages and variables, easier work with figures, ...
- RStudio http://www.rstudio.com/ — probably the most common, multiplatform, very powerful
- RKWard https://rkward.kde.org/ — feature very rich, developed mainly for Linux, although available also for another operating systems
- R commander (Rcmdr) http://www.rcommander.com/ — multiplatform, not so rich as previous
- Tinn-R http://nbcgib.uesc.br/lec/software/editores/tinn-r/en — Windows only
- Pick one you like and install it...

RKWard


Inferred cluster 1
Inferred cluster 2
POP7568
POP7868
POP5449
POP5351
POP5478
POP5847
POP5555
POP6244
POP6344
POP6250
POP6350
POP6450
POP7251
POP7448
POP6746
POP7148
POP7147
POP6845
POP7046
POP6245
POP6345
POP6249
POP6349
POP6449
POP6045

RStudio


RStudio
File Edit Code View Plots Session Build Debug Tools Help
workflow.r
Source on Save
Run
Source
ssrs3n.loci.missingno <- as.loci(ssrs3n.missingno)
ssrs3n.loci.missingno
# Summary statistics
ssrs3n.summ <- summary(ssrs3n.missingno)
plot(ssrs3n.summ$Hexp, ssrs3n.summ$Hobs, main = "Observed vs. expected heterozygosity")
abline(0, 1, col = "red")
t.test(ssrs3n.summ$Hexp, ssrs3n.summ$Hobs, paired = TRUE, var.equal = TRUE)
bartlett.test(list(ssrs3n.summ$Hexp,ssrs3n.summ$Hobs))
par(mfrow=c(2,2))
plot(ssrs3n.summ$pop.eff,ssrs3n.summ$pop.nall, xlab="Populations sample size", ylab="Number of alleles", main="Alleles numbers and sample sizes")
text(ssrs3n.summ$pop.eff,ssrs3n.summ$pop.nall, lab=names(ssrs3n.summ$pop.eff))
barplot(ssrs3n.summ$loc.nall, ylab="Number of alleles", main="Number of alleles per locus", names.arg=ssrs3n.missingno@loc.names, las=3)
barplot(ssrs3n.summ$Hexp-ssrs3n.summ$Hobs, main="Heterozygosity: expected-observed", ylab="Hexp - Hobs", names.arg=ssrs3n.missingno@loc.names, las=3)
barplot(ssrs3n.summ$pop.eff, main="Sample sizes per species", ylab="Number of genotypes", las=3)
dev.off()
svg("popprr_%03d.svg", family="sans", bg="white", antialias="subpixel")
poppr(ssrs3n.genind, total=TRUE, sample=10000, method=4, missing="geno", cutoff=0.15, quiet=FALSE, clonecorrect=FALSE, hist=TRUE, minsamp=1, legend=TRUE)
dev.off()
(Top Level)
R Script
Console
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-suse-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Update Packages
Package Installed Available NEWS
BH 1.55.0-2 1.55.0-3
HSAUR 1.3-4 1.3-5
openair 1.0 1.1-0
rgl 0.95.1158 0.95.1201
SparseM 1.05 1.6
ttrame 2012.8-1 2015.1-1
TH.data 1.0-5 1.0-6
Select All
Select None
Install Updates
Cancel
Environment History
To Console
To Source
library(ape) # Required by pegas
library(pegas)
library(poppr)
library(sp) # Required by rworldmap
library(rworldmap)
library(ade4) # Required by pegas, adegenet and poppr
library(adegenet) # Required by pegas and poppr
library(ape) # Required by pegas
library(pegas)
library(poppr)
library(sp) # Required by rworldmap
library(rworldmap)
ssrs3n.summ <- summary(ssrs3n.missingno)
theta.msat(ssrs3n.loci.missingno)
Files Plots Packages Help Viewer
Install Packages
Check for Updates
abc Tools for Approximate Bayesian Computation (ABC) 2.0
abind Combine multi-dimensional arrays 1.4-1
abind Combine multi-dimensional arrays 1.4-0
Ace Assay-based Cross-sectional Estimation of Incidence rates 0.0.8
acepack ace() and avas() for selecting regression transformations 1.3-3.3
acepack ace() and avas() for selecting regression transformations 1.3-3.3
acs Download, manipulate, and present data from the US Census American Community Survey 1.2
acs Download, manipulate, and present data from the US Census American Community Survey 1.2
ada ada: an R package for stochastic boosting 2.0-3
adaptMCMC Implementation of a generic adaptive Monte Carlo Markov Chain sampler 1.1
ade4 Analysis of Ecological Data : Exploratory and Euclidean methods in Environmental sciences 1.6-2
ade4 Analysis of Ecological Data : Exploratory and Euclidean methods in Environmental sciences 1.6-2
ade4TkGUI ade4 Tcl/Tk Graphical User Interface 0.2-8
adegenet adegenet: an R package for the exploratory analysis of genetic and genomic data. 1.4-2
adegraphics An S4 lattice-based package for the representation of multivariate data 0.9
adehabitat Analysis of habitat selection by animals 1.8.15
adehabitatHR Home range Estimation 0.4.12
adehabitatHS Analysis of habitat selection by animals 0.3.10
adehabitatLT Analysis of animal movements 0.3.17
adehabitatMA Tools to Deal with Raster Maps 0.3.8

MS Windows, Apple Mac OS X

- Got to http://cran.at.r-project.org/
- Download appropriate version and install as usual
- Download and install selected GUI
- Most of packages are available as pre-compiled and can be immediately installed from R
- Convenient, but usually not tuned for particular computer architecture (type of CPU)
- Usually there are some problems every time new version of OS is released — it takes time to modify and recompile packages for new version of OS
- You have to check for new version of R manually

Linux — general and Debian/Ubuntu

- R, and usually also GUI, is available in repositories — use standard package management according to distribution
- Linux repositories provide automatic updates
- Packages are also partially available in repositories and can be installed and updated as usual application or from R
- Packages commonly have to be compiled – R will do it automatically, just install basic Linux packages for building (in Debian and Ubuntu install package **build-essential**)
- Debian (and derivatives): follow instructions at http://cran.at.r-project.org/bin/linux/debian/
- Ubuntu (and derivatives): follow instructions at http://cran.at.r-project.org/bin/linux/ubuntu/
- As <**my.favorite.cran.mirror**> select cran.at.r-project.org

Linux — openSUSE

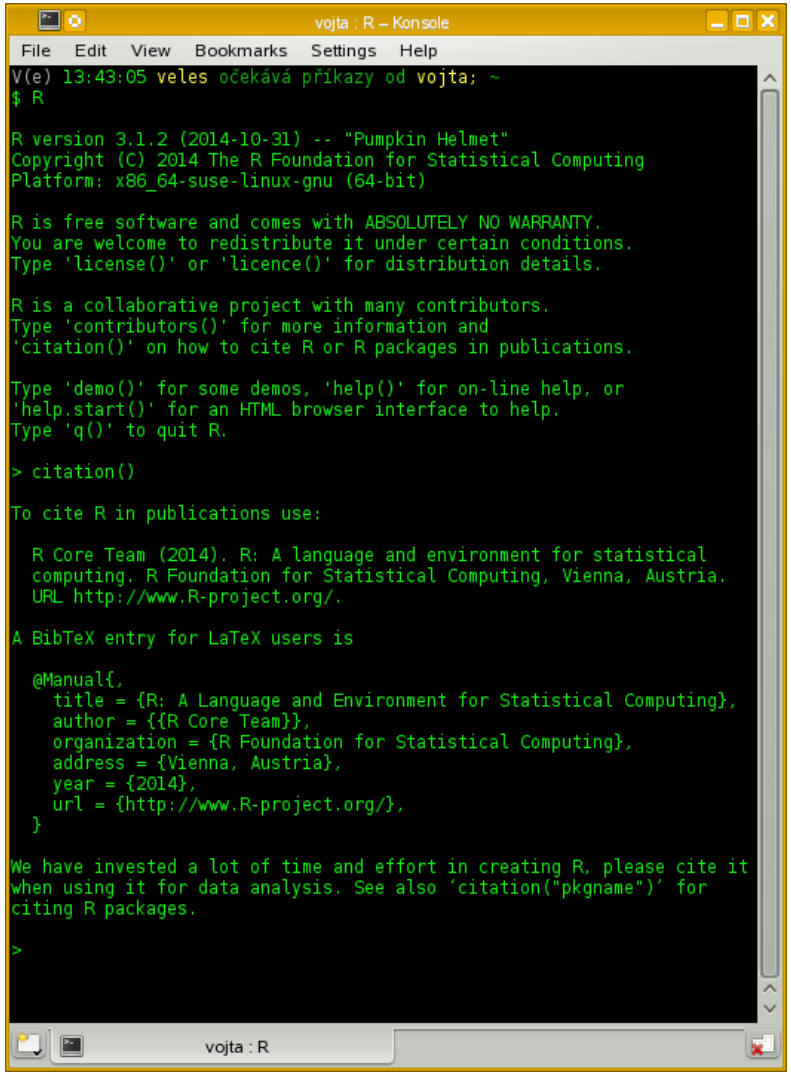
- See instructions at http://cran.r-project.org/bin/linux/suse/
- Add repositories for your distribution
  - http://download.opensuse.org/repositories/devel:/languages:/R:/base/
  - http://download.opensuse.org/repositories/devel:/languages:/R:/patched/
  - http://download.opensuse.org/repositories/devel:/languages:/R:/released/
- Install packages **R-patched** (the R) and **rstudio** and/or **rkward**
- Install package **patterns-openSUSE-devel_basis** for compilation of R packages when installing them from R (only some R packages are available in repositories)
- Compilation takes longer time and there are sometimes issues, but it can then provide higher performance...

Sources of R packages

- R CRAN http://cran.r-project.org/ — main and largest source of R packages (over 6000 packages)
- Bioconductor http://bioconductor.org/ — mainly bioinformatic packages, genomic data (∼1000 packages)
- R-Forge http://r-forge.r-project.org/ (over 1800 packages)
- RForge http://www.rforge.net/ (much smaller)
- Omega Project for Statistical Computing http://www.omegahat.org/
- And more...
- Some packages are available from more resources — same name for function can be used in different packages — to distinguish them you can call functions like this: **muscle::read.fasta()** vs. **seqinr::read.fasta()** — call function **read.fasta()** from package **muscle** or **seqinr** (and their parameters can be different...)

First steps in R

- Linux: open any terminal, type **R** and hit Enter, Windows and Mac: find it as normal application in menu
- R is ruled by typing commands... :-)
- Use arrows up and down to navigate in history
- Ctrl+R works as reverse search — searches typed string in history of commands
- We will mostly use GUI



How it works

- General look of R commands:
- `function(argument1="SomeName", argument2=SomeVariable, argument3=8)`
- `ModifiedObject <- SomeFunction(argument1=MyData, argument2=TRUE)`
- Object is on the left: "<-" says to insert result of the function on the right into the object
- Functions have various parameters/arguments (in brackets, separated by comas): argument=ItsValue
- Arguments are named — if you keep order, no need to name them:
- `SomeFunction(MyData, TRUE, 123, "SomeName")`
- When only some of the arguments are in use, use the names (order doesn't matter any more)
- `SomeFunction(argument2=TRUE, argument3=123, argument1=MyData)`

Get help in R

```
# "#" marks comments - notes within code which are not executed
help(function) # Help for particular function (package must be loaded)
?function # Help for particular function (package must be loaded)
??SearchedTerm # Search for the term within all installed packages
help.search("searched phrase") # Search for the phrase within all
  # installed packages - return list of hits sorted according to
  # type and package (i.e. package::function)
install.package("sos") # More comprehensive search from package sos
library(sos)
findFN("function")# Search for function name
```

? shows help for questioned function:

- Name of the package (top left)
- Function name (headline)
- Description
- Usage
- Comments on arguments
- Details
- About author(s)
- References to cite
- Example code

Where we are?

- In Linux, R starts in current directory (use **cd** to change it before launching R)
- Set and check working directory in R:

```
1 setwd("/some/path/") # Or "~/...". In Windows "C:\..."
2 getwd() # Verifies where we are
3 dir() # Lists files and folders on the disk
4 ls() # Lists currently available R objects
```

- In Windows (File | working directory) or in RStudio (Session | Set working directory) set it in menu or by above command
- R saves history of commands into **.Rhistory** file within working directory (by default hidden in Linux/Mac)
- When closing R by **q()** you can save all R data in **.RData** file and it can be loaded next time
- RStudio and RKWard help with this very much

Materials to help you...

- Follow all code we will use at https://trapa.cz/sites/default/rcourse/course_commands.html
- Download the script from https://trapa.cz/sites/default/rcourse/course_commands.r, use it and write your comments and notes to it during the course
- Note: Open the R script in some good text editor — showing syntax highlight, line numbers, etc. (NO Windows notepad); the file is in UTF-8 encoding and with Linux end of lines (so that too silly programs like Windows notepad won't be able to open it correctly)
- Information about the course: https://trapa.cz/en/course-molecular-data-r-2015 or https://trapa.cz/cs/kurz-molekularni-data-r

Types of objects

- Vectors — numbers or characters
- Matrices — columns are of same type (numeric, character, etc.) and the same length
- Arrays — like matrices, but with possibly more dimensions
- Data frames — more general — columns can be of different type
- Lists — ordered collections of objects (vectors, matrices, ...) — not necessarily of the same type
- Factors — a vector of levels, e.g. populations, colours, etc.
- More "advanced" objects to store plots, genetic data, ...
  - Commonly called "S3" and "S4" objects
  - Technically commonly just lists putting together various information
  - We will meat many of them...
- Functions require particular object types — take care about it

Popular object classes (we are going to use) I

- dist — distance matrices
- genind — stores various genetic information for individuals
- genpop — like genind, but on population level
- genlight — variant of genind to store large multiple genomes
- SNPbin — stores large SNP data for single genome
- DNAbin — stores DNA sequences (aligned or not)
- haplotype — unique sequences from DNAbin
- alignment — aligned sequences (seqinr)
- phyDat — "preparation" of data for some phylogenetic analysis
- loci – extension of DF, stores information about loci
- hclust – output of hierarchical clustering, can be converted to phylo
- treeshape — derived from hclust

Popular object classes (we are going to use) II

- phylo — phylogenetic information, typically trees
- phylo4 — derived from phylo, S4 instead of S3
- matching – binary phylogenetic trees
- haplonet — networks without reticulation
- spca — results of sPCA
- pco; dudi — results of PCA, PCoA, ...
- dapc — results of DAPC
- and more...common task is converting among formats...
- ...not all formats are (easily) convertable amon each other...

To get information about content of each data type see

```
getClassDef("genind") # Or any other class name
```

There are information about slots within that classes you can access.

Conversions among data types I

From	To	Command	Package
phylo	phylo4	as(x, "phylo4")	phylobase
phylo	matching	as.matching(x)	ape
phylo	treeshape	as.treeshape(x)	apTreeshape
phylo	hclust	as.hclust(x)	ape
phylo	prop.part	prop.part(x)	ape
phylo	splits	as.splits(x)	phangorn
phylo	evonet	evonet(x, from, to)	ape
phylo	network	as.network(x)	ape
phylo	igraph	as.igraph(x)	ape
phylo4	phylo	as(x, "phylo")	phylobase
matching	phylo	as.phylo(x)	ape
treeshape	phylo	as.phylo(x)	apTreeshape
splits	phylo	as.phylo(x)	phangorn

Conversions among data types II

From	To	Command	Package
splits	networx	as.networx(x)	phangorn
evonet	phylo	as.phylo(x)	ape
evonet	networx	as.networx(x)	ape
evonet	network	as.network(x)	ape
evonet	igraph	as.igraph(x)	ape
haploNet	network	as.network(x)	pegas
haploNet	igraph	as.igraph(x)	pegas
hclust	phylo	as.phylo(x)	ape
hclust	dendrogram	as.dendrogram(x)	stats
DNAbin	character	as.character(x)	ape
DNAbin	alignment	as.alignment(x)	ape
DNAbin	phyDat	as.phyDat(x)	phangorn
DNAbin	genind	DNAbin2genind(x)	adegenet
character	DNAbin	as.DNAbin(x)	ape

Conversions among data types III

From	To	Command	Package
character	loci	as.loci(x)	pegas
alignment	DNAbin	as.DNAbin(x)	ape
alignment	phyDat	as.phyDat(x)	phangorn
alignment	character	as.matrix(x)	seqinr
alignment	genind	alignment2genind(x)	adegenet
phyDat	DNAbin	as.DNAbin(x)	phangorn
phyDat	character	as.character(x)	phangorn
loci	genind	loci2genind(x)	pegas
loci	data frame	class(x) <- "data.frame"	
genind	loci	genind2loci(x)	pegas
data frame	phyDat	as.phyDat(x)	phangorn
data frame	loci	as.loci(x)	pegas
data frame	genind	df2genind(x)	adegenet
matrix	phyDat	as.phyDat(x)	phangorn

Basic operations with data I

```
x <- c(1, 2, 3, 4, 5) # Creates vector (see also ?rep)
c() # Is generic function to concatenate objects into new one
length(x) # Length of the object - for matrices and DF use dim()
str(x) # Information about structure of the object, try also ls.str()
mode(x) # Gets type of storage mode of the object
class(x) # Shows class of the object
x[2] # Shows second element of the object
x <- x[-5] # Removes fifth element
y <- matrix(data=5:20, nrow=4, ncol=4) # Creates a matrix
is.matrix(y) # Is it matrix? Try is.<TAB><TAB>
# TAB key shows available functions and objects starting by typed text
y # Prints the matrix
y[,2] # Prints second column
y[3,] # Prints third row
y[4,3] # Prints element from fourth row and third column
x <- y[2,] # Replaces "x" by second row of "y" (no warning)
# R doesn't ask neither notifies when overwriting objects! Be careful!
```

Basic operations with data II

```
rm(x) # Deletes x
y[,1:3] # Prints first through third column of the matrix
y[3,] <- rep(x=20, each=4) # Replaces third line by value of 20
y[y==20] <- 10 # If value of y's element is 20, replace it by 10
summary(y) # Basic statistics - according to columns
colnames(y) <- c("A", "B", "C", "D") # Set column names
# Objects and functions are without quotation marks; files, text with
colnames(y) # Prints column names, use rownames() in same way
y[,"C"] # Prints column C (R is case sensitive!)
t(y) # Transposes the matrix
y <- as.data.frame(y) # Turns into DF (see other functions as.*)
y[y==17] <- "NA" # Removes values of 17
y$B # Gets variable B of data frame y ($ works similarly in S3 objects)
save(list=ls(), file="test.RData") # Saves all objects during the work
load("test.RData") # Loads saved R environment with all objects
# When loading saved project, you have to load again libraries and
# scripts (see further), data objects are restored
```

Set repositories

```
# Basic package installation
install.packages("PackageName") # Case sensitive!
?install.packages # Shows all available parameters
# We will need extra repositories:
options(repos=c("http://cran.at.r-project.org",
  "http://r-forge.r-project.org/", "http://www.rforge.net/",
  "http://bioconductor.statistik.tu-dortmund.de/packages/3.1/bioc",
  "http://bioconductor.statistik.tu-dortmund.de/packages/3.1/data/
  annotation", "http://bioconductor.statistik.tu-dortmund.de/
  packages/3.1/data/experiment",
  "http://bioconductor.statistik.tu-dortmund.de/packages/3.1/extra"))
getOption("repos") # Shows actual repositories
options() # Generic function to modify various settings
?options # Gives details
```

- All the time keep newest version of R and and newest versions of packages!
- Installation of multiple packages may sometimes fail — install then packages in smaller groups or one by one

Install needed packages

```
install.packages(pkgs=c("Geneland", "PBSmapping", "ParallelStructure",
  "RandomFields", "RgoogleMaps", "TeachingDemos", "XML", "ade4",
  "adegenet", "adephylo", "akima", "ape", "colorspace", "combinat",
  "corrplot", "fields", "gplots", "grid", "ips", "lattice", "mapdata",
  "mapproj", "maps", "maptools", "muscle", "pegas", "permute",
  "phangorn", "phyloch", "phytools", "polysat", "poppr", "rworldmap",
  "seqinr", "sp", "spam", "tcltk", "vegan"), repos=getOption("repos"),
  dependencies=TRUE)
library(package) # Loads installed package (we will do it on the fly)
update.packages(repos=getOption("repos")) # Updates installed packages
```

Installation of packages in GUI

- RStudio: set repositories by command from slide 28 and in bottom right pane select "Packages" and click on "Install Packages"...
- RKWard: go to menu "Settings" | "Configure 'RKWard"' and select "Packages R". Add URLs pf repositories from slide 28. OK. Go to menu "Settings" | "Manage R packages", click to "Install...", select and install desired packages...

Bioconductor

- Tools for analysis of genomic data, see http://bioconductor.org/
- To install it, add appropriate repositories (repository use to have same version number as R — change them accordingly when upgrading R) or use Bioconductor's special function (recommended by Bioconductor):

```
# Load basic Bioconductor function
source("http://bioconductor.org/biocLite.R")
# Get help how to use it
?biocLite
# Install packages
biocLite(c("package1", "package2", "..."))
# Upgrades installed Bioconductor packages
biocLite("BiocUpgrade")
```

- Explore available packages: http://bioconductor.org/packages/release/BiocViews.html

Population genetics and phylogenetics in R

Microsatellites, AFLP, SNP & sequences

- Now we will use mainly packages adegenet and poppr
- Other important genetic packages: ape, ade4 and pegas
- Dominant/co-dominant marker data of any ploidy level including SSRs, SNP, and AFLP
- Most of methods are able for polyploids (although not all)
- Some methods are unavailable for dominant (presence/absence) data
- Mixing of ploidy levels is tricky — it doesn't matter when data are encoded as PA, otherwise it is problematic

```
1 library(ape)
2 library(ade4)
3 library(adegenet)
4 library(pegas)
5 library(poppr)
```

Load data

Source data:

```
    pop msta93 msta101 msta102 msta103 msta105 msta131...
H01 He 269/269 198/198 221/223 419/419 197/197 196/196...
H02 He 275/283 198/198 221/223 419/419 193/193 168/190...
... ...... ... ... ... ... ... ...
```

```
1 # Load training data (Taraxacum haussknechtii from Macedonia)
2 hauss.loci <- read.loci(file="http://trapa.cz/sites/default/rcourse/
3   haussknechtii_ssrs.txt", header=TRUE, loci.sep="\t", allele.sep="/",
4   col.pop=2, col.loci=3:14, row.names=1) # \t means TAB key
5 # Data control
6 hauss.loci
7 print(hauss.loci, details=TRUE)
```

First line starts with empty cell (if **header** is presented), there can be any extra column, just take care about **col.loci**. **row.names** are individual names (firs column). Take care about **loci.sep** (here TAB) and **allele.sep** according to data formatting.

Prepare genind object for analysis and load coordinates

```
# Conversion of loci to genind - used for many analysis
hauss.genind <- loci2genind(hauss.loci)
# See population names
pop(hauss.genind)
hauss.genind$pop.names # "$" separates extra slots within object
```

Coordinates can be in any projection or scale — according to aim

Source data:

Ind	lon	lat
H01	21.3333	41.1
...	...	...

```
# Read coordinates
hauss.coord <- read.csv("http://trapa.cz/sites/default/rcourse/
  haussknechtii_coordinates.csv", header=TRUE, sep="\t", quote="",
  dec=".", row.names=1)
hauss.coord
```

Take care about parameters of **read.csv()**! See **?read.csv** for details.

Add coordinates to genind and create genpop object

```
# Add coordinates - note identification of slots within object
hauss.genind$other$xy <- hauss.coord
# See result
hauss.genind$other$xy
hauss.genind
# Removes missing data - see ?missingno for types of dealing them
hauss.genind <- missingno(pop=hauss.genind, type="mean", cutoff=0.1)
# Convert corrected genind to loci
hauss.loci.cor <- genind2loci(hauss.genind)
# Writes loci file to the disk
write.loci(hauss.loci.cor, file="hauss.loci.cor.txt",
  loci.sep="\t", allele.sep="/")
# Conversion to genpop - for population-level analysis
hauss.genpop <- genind2genpop(hauss.genind, missing="NA",
  process.other=TRUE)
# See result
hauss.genpop
```

Import existing data set from popular software

```
read.genalex() # poppr - reads *.csv file
read.fstat() # adegenet - reads *.dat files, only haploid/diploid data
read.genetix() # adegenet - reads *.gtx files, only haploid/diploid data
read.genepop() # adegenet - reads *.gen files, only haploid/diploid data
read.structure() # adegenet - reads *.str files, only haploids/diploids
import2genind() # adegenet - more automated version of above functions
```

One function rules them all...

All those functions (including read-loci and read.csv) are only modifications of **read.table()**. You can use it directly to import any data. Look at **?read.table** and play with it. Take care about parameters. Does the table use quotes to mark cell (quote=...)? How are columns separated (sep=...)? Is there a header with names of populations/loci/whatever (header=T/F)? What is decimal separator (dec=...)? Are there row names (used typically as names of individuals; row.names=...)? Check data after import!

Import of polyploid microsatellites

- adegent, poppr and related packages can for most of functions handle any ploidy level (but no mixing of various ploidy levels for microsatellites)
- polysat can handle mixed ploidy levels for microsatellites, but range of methods is limited

As for AFLP, we need two files: the data matrix and individual's populations:

	msat58	msat31	msat78	msat61	...
ala1	124/124/124	237/237/237	164/164/172	136/136/138	...
ala2	124/124/124	237/237/237	164/164/172	136/136/138	...
ala4-1	124/124/124	237/237/237	164/164/172	136/136/138	...
...	...	...	...	...	...

How to import polyploid microsatellites

```
# Import if table is as usual. Last column contains populations
tarax3n.table <- read.table("http://trapa.cz/sites/default/rcourse/
  tarax3n.txt", header=TRUE, sep="\t", quote="", row.names=1)
# Check the data
tarax3n.table
class(tarax3n.table)
dim(tarax3n.table)
# See parameter "X" - we don't import whole tarax3n.table as last column
# contains populations - this column we use for "pop" parameter (note
# different style of calling the column - just to show the possibility).
# Check "ploidy" and "ncode" (how many digits code one allele - must be
# same everywhere). See ?df2genind for more details.
tarax3n.genind <- df2genind(X=tarax3n.table[,1:6], sep="/", ncode=3,
  pop=tarax3n.table[["pop"]], missing=NA, ploidy=3, type="codom")
# See resulting genind object
tarax3n.genind
summary(tarax3n.genind)
```

Import of AFLP data — background

Source data (two files - AFLP data with individual names and populations)

AFLP – presence/absence data:

```
     L1 L2 L3 L4 L5 L6 L7 L8 L9 ...
Ind1 0  0  1  1  1  0  0  0  1  ...
IndG 0  0  1  1  0  0  0  0  0  ...
...   ...   ...
```

Individual's populations:

```
POP
pop1
popZ
...
```

- Use any names, just keep one word (no spaces) and don't use special characters
- Keep names of loci as simple as possible, there are some issues when they contain dots
- As soon as one line of data = one individual, ploidies and their mixing doesn't matter
- Not all methods are available/meaningful for PA

Import of AFLP data — the code

```
amara.aflp <- read.table(file="http://trapa.cz/sites/default/rcourse/
  amara_aflp.txt", header=TRUE, sep="\t", quote="")
amara.aflp
dim(amara.aflp)
class(amara.aflp) # Must be matrix or data frame
# Populations - just one column with population names for all inds
amara.pop <- read.table(file="http://trapa.cz/sites/default/rcourse/
  amara_pop.txt", header=TRUE, sep="\t", quote="")
amara.pop
# You can use just one file, where populations are in last column and
# in df2genind() use for example X=aflp[,1:XXX] and pop=aflp[,YYY]
# Create genind object - ind.names and loc.names are taken from X
aflp.genind <- df2genind(X=amara.aflp, sep=NULL, ind.names=NULL,
  loc.names=NULL, pop=amara.pop[,1], missing=NA, type="PA")
aflp.genind
summary(amara.aflp)
# You can add any other variables into genind$other$XXX
```

Another data manipulation

SNPs can be into genind imported in same way as AFLP

```
genind2df() # adegenet - export into data frame
genind2genalex() # poppr - export for genalex
splitcombine() # poppr - edits population hierarchy
popsub() # poppr - extracts only selected population(s)
clonecorrect() # poppr - corrects for clones
informloci() # poppr - removes uninformative loci
seppop() # adegenet - separates populations from genind or genlight
seploc() # adegenet - splits genind, genpop or genlight by markers
# seppop and seploc return lists of objects - for further analysis
# read manual pages (?...) of those functions before usage
```

Notes about getting data into R

- When importing fragmentation data, we somehow use function **read.table()** — it is important to understand it
- I recommend to use TAB (TSV — tab separated values; encoded as "\t" in R) to separate columns (no quotation marks, no commas)
- When importing microsatellites, all alleles **must** have same number of digits. Separate alleles by "/", "|" or something similar and correctly specify it in **read.loci()** or **df2genind()**
- Do not use underscores ("_") to name objects in R — only numbers, letters, dots and minuses
- **read.loci()** sometimes doesn't work correctly on AFLP or polyploid microsatellites — try **write.table()** instead...
- Theoretically, genind object could be currently able to store mixing ploidy data, but most of analysis will probably fail...

Import of DNA data

```
# Reading FASTA (reads also another formats, see ?read.dna)
# Sequences of flu viruses from various years
usflu.dna <- read.dna(file="http://adegenet.r-forge.r-project.org/
  files/usflu.fasta", format="fasta")
# Another possibility (only for FASTA alignments, same result):
usflu.dna <- fasta2DNAbin(file="http://adegenet.r-forge.r-project.org/
  files/usflu.fasta") # Normally keeps only SNP -- see ?fasta2DNAbin
# Check the object
usflu.dna
class(usflu.dna)
as.character(usflu.dna)[1:5,1:10]
# Read anotations
usflu.annot <- read.csv("http://adegenet.r-forge.r-project.org/files/
  usflu.annot.csv", header=TRUE, row.names=1)
head(usflu.annot) # See result
# Convert DNAbin to genind - only polymorphic loci (SNPs) are retained
# When converting DNAbin to genind, the sequences must be aligned!
usflu.genind <- DNAbin2genind(x=usflu.dna, pop=usflu.annot[["year"]])
```

Import sequences from GeneBank

```
# read.fasta() from seqinr package reads DNA or AA in FASTA format
# returns a list (DNAbin is for us now better choice)
usflu.dna <- read.fasta(file="http://adegenet.r-forge.r-project.org/
  files/usflu.fasta", seqtype="DNA"
# Importingsequences according to sequence ID
# data from http://www.ncbi.nlm.nih.gov/popset/608602125 (Meles meles)
meles.dna <- read.GenBank(c("KJ161355.1", "KJ161354.1", "KJ161353.1",
  "KJ161352.1", "KJ161351.1", "KJ161350.1", "KJ161349.1", "KJ161348.1",
  "KJ161347.1", "KJ161346.1", "KJ161345.1", "KJ161344.1", "KJ161343.1",
  "KJ161342.1", "KJ161341.1", "KJ161340.1", "KJ161339.1", "KJ161338.1",
  "KJ161337.1", "KJ161336.1", "KJ161335.1", "KJ161334.1", "KJ161333.1",
  "KJ161332.1", "KJ161331.1", "KJ161330.1", "KJ161329.1", "KJ161328.1"))
meles.dna
class(meles.dna)
# Converts DNAbin to genind - extracts SNP - for large datasets can be
# computationally very intensive
meles.genind <- DNAbin2genind(meles.dna)
```

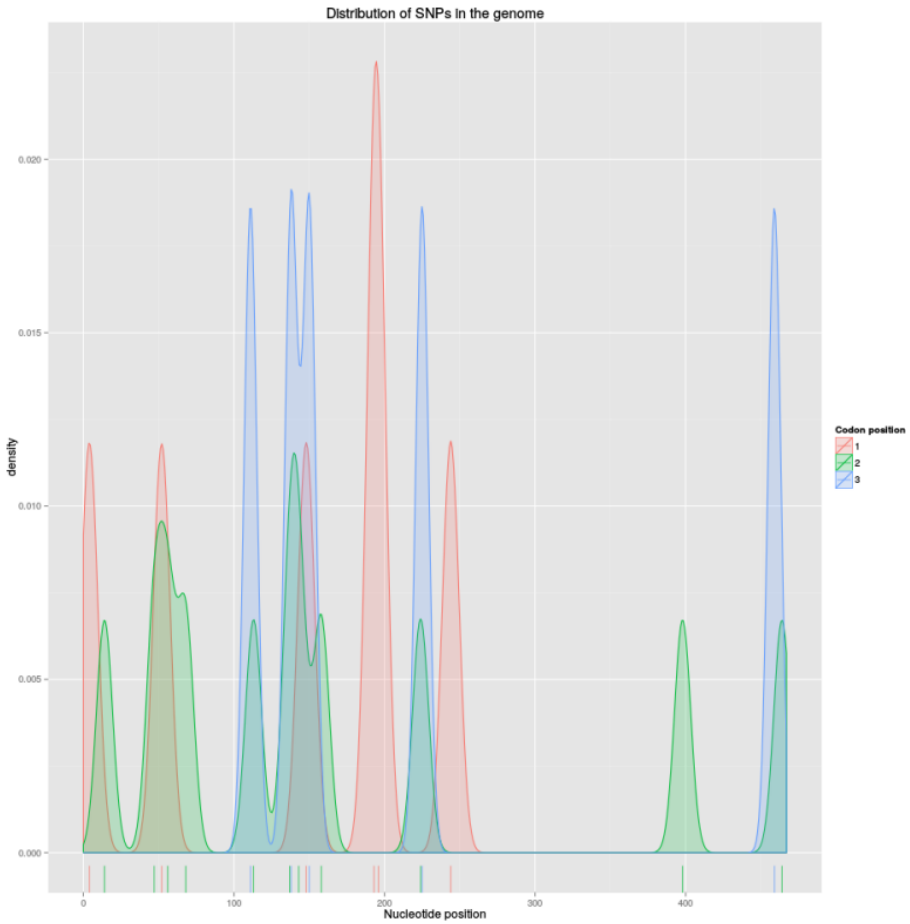
To query on-line database as through web interface we use seqinr.

Query on-line sequence databases

```
library(seqinr)
choosebank() # Genetic banks available for seqinr
choosebank("embl") # Choose some bank
?query # See how to construct the query
# Query selected database - there are much possibilities
nothofagus <- query(listname="nothofagus",
  query="SP=Nothofagus AND K=rbcl", verbose=TRUE)
# See the sequences information
nothofagus$req
# Get the sequences as a list
nothofagus.sequences <- getSequence(nothofagus$req)
# See sequences
nothofagus.sequences
# Get annotations
nothofagus.annot <- getAnnot(nothofagus[["req"]])
# Close the bank when work is over
closebank()
# Convert sequences from a list to DNAbin (functions as.DNAbin*)
nothofagus.dna <- as.DNAbin.list(nothofagus.sequences)
```

Checking SNPs

```
# Position of polymorphism within
# alignment - snpposi.plot requires
# input data in form of matrix
snpposi.plot(x=as.matrix(
  meles.dna), codon=FALSE)
# Position of polymorphism within
# alignment - differentiating codons
snpposi.plot(as.matrix(meles.dna))
# When converting to genind object,
# only polymorphic loci are kept -
# threshold for polymorphism can
# be arbitrary
meles.genind <- DNAbin2genind(x=
  meles.dna, polyThres=0.01)
# Test of random distribution of SNP
snpposi.test(as.matrix(meles.dna))
```



Importing SNP

Import from PLINK requires saving of data with option "-recodeA":

```
read.PLINK(file="PLINKfile", ...) # See ?readPLINK
```

Extracting SNP from alignments reads FASTA alignments and keep only SNPs. The method is relatively efficient even for large data sets with several genomes:

```
usflu.genlight <- fasta2genlight(file=
  "http://adegenet.r-forge.r-project.org/files/usflu.fasta",
  quiet=FALSE, saveNbAlleles=TRUE)
?fasta2genlight # Function has several options to speed up reading
# If it crashes (on Windows), try add parameter "parallel=FALSE"
```

- For small datasets it doesn't matter if resulting dataset is genind or genlight, for large genlight is more efficient

```
# Convert genlight to genind
usflu.genind <- as.genind(usflu.genlight)
# Resulting data matrix might be slightly different than when using
# DNAbin2genind - depends on used settings
```

Notes about using genlight (vs. genind)

- Genlight is "just" version of more common genind object to store large data sets with (nearly) complete multiple genomes
- "Large" is tricky — there is no easy criterion — try genind and when work fails because of not enough computer resources, go on with genlight
- Can be quickly converted to genind (not back)
- Use is basically same as when working with genind — but not all functions are able to deal with it (on the other hand, others are optimised to work well on large data sets)
- SNPbin is version of genind/genlight to store one large genome — serves basically as storage, no need to deal with it
- genlight allows varying ploidy level — genind has issues with it

Export data

```
# Convert genind into DF using genind2genotype() or genind2df()
hauss.df <- genind2genotype(x=hauss.genind, pop=hauss.genind@pop,
  res.type="matrix")
hauss.df <- genind2df(x=hauss.genind, pop=NULL, sep="/",
  usepop=TRUE, oneColPerAll=FALSE)
write.table(x=hauss.df, file="haussdata.txt", quote=FALSE,
  sep="\t", na="NA", dec=".", row.names=TRUE, col.names=TRUE)
# Export of DNA sequences into FASTA format
write.dna(x=usflu.dna, file="usflu.fasta", format="fasta",
  append=FALSE, nbcol=6)
write.fasta(sequences=meles.dna, names=names(meles.dna),
  file.out="meles.fasta", open="w")
# Export trees (objects of class phylo)
# Writes tree(s) in NEWICK format
write.tree(phy=hauss.nj.bruvo, file="haussknechtii.nwk")
# Writes tree(s) in NEXUS format
write.nexus(hauss.nj.bruvo, file="haussknechtii.nexus")
```

Descriptive statistics I

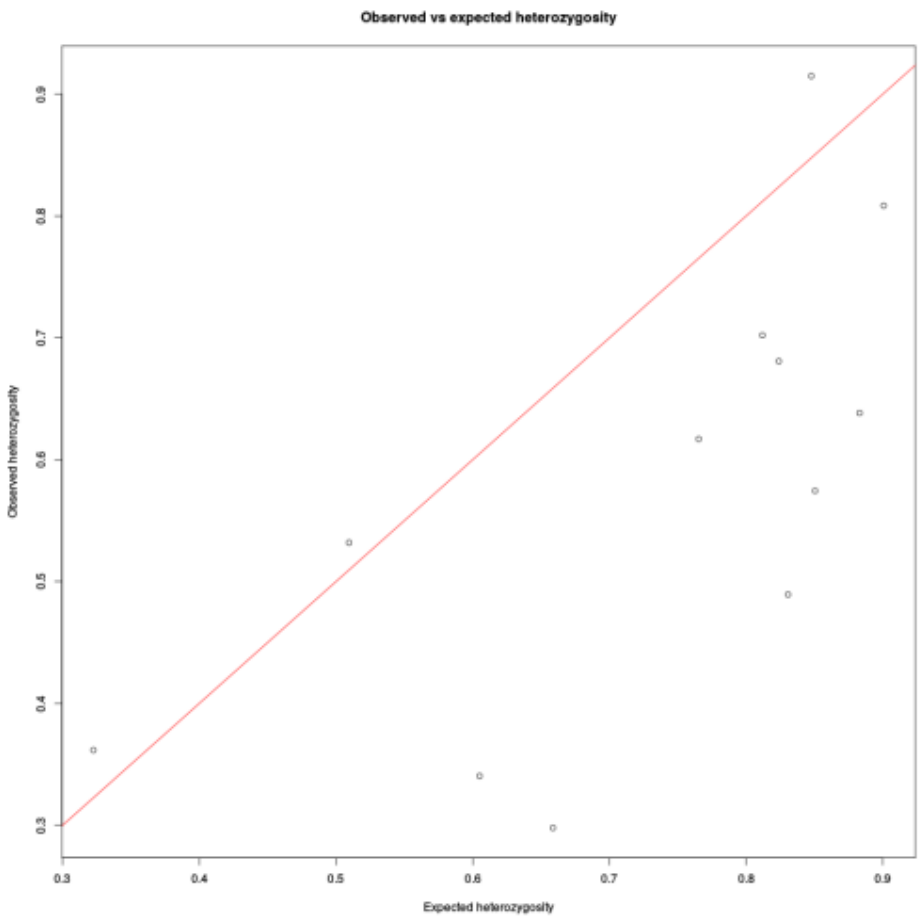
We will now work mainly with diploid SSRs for *Taraxacum haussknechtii*, you can try other data examples by yourselves

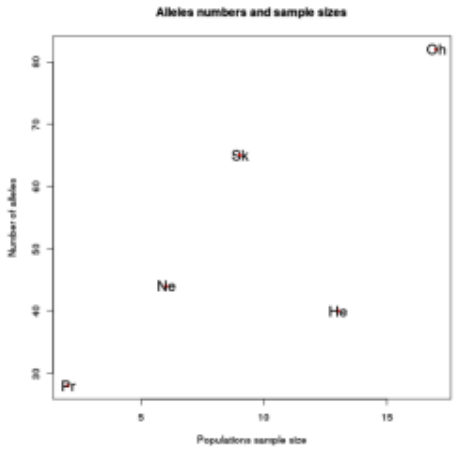
```
# Get summary - names and sizes of populations,
# heterozygosity, some info about loci
hauss.summ <- summary(hauss.genind)
# Plot expected vs. observed heterozygosity
# it looks like big difference
plot(x=hauss.summ$Hexp, y=hauss.summ$Hobs,
  main="Observed vs expected heterozygosity",
  xlab="Expected heterozygosity", ylab="Observed heterozygosity")
abline(0, 1, col="red")
# T-test of difference between
# observed and expected heterozygosity - strongly significant
t.test(x=hauss.summ$Hexp, y=hauss.summ$Hobs, paired=TRUE, var.equal=T)
# Bartlett's K-squared of difference
# between observed and expected heterozygosity - not significant
bartlett.test(list(hauss.summ$Hexp,hauss.summ$Hobs))
```

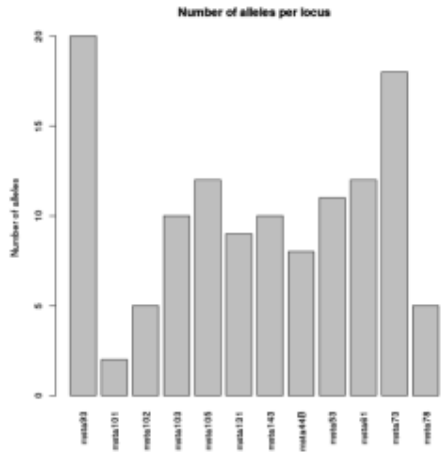
Descriptive statistics II

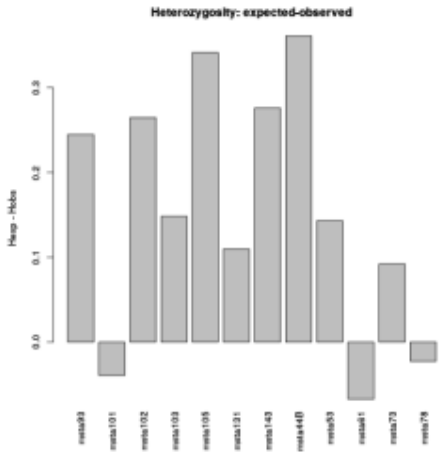
```
# Create pane with some information
par(mfrow=c(2,2)) # Divide graphical devices into 4 smaller spaces
# Plot alleles number vs. population sizes
plot(x=hauss.summ$pop.eff, y=hauss.summ$pop.nall, xlab="Populations
  sample size", ylab="Number of alleles", main="Alleles numbers and
  sample sizes", col="red", pch=20)
# Add text description to the point
text(x=hauss.summ$pop.eff, y=hauss.summ$pop.nall,
  lab=names(hauss.summ$pop.eff), cex=1.5)
# Barplots of various data
barplot(height=hauss.summ$loc.nall, ylab="Number of alleles",
  main="Number of alleles per locus",
  names.arg=hauss.genind@loc.names, las=3)
barplot(height=hauss.summ$Hexp-hauss.summ$Hobs, main="Heterozygosity:
  expected-observed", ylab="Hexp - Hobs",
  names.arg=hauss.genind@loc.names, las=3)
barplot(height=hauss.summ[["pop.eff"]], main="Sample sizes per
  population", ylab="Number of genotypes", las=3)
```

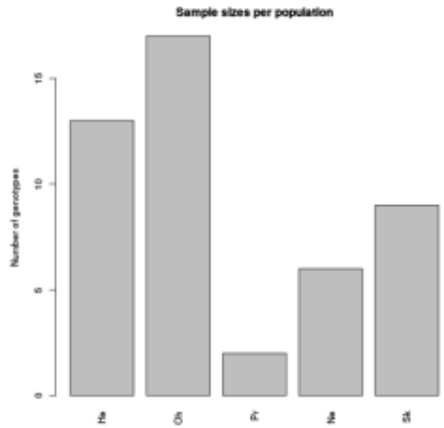
Graphs from previous two slides












```
dev.off() # Closes graphical device - otherwise following
          # graphs would still be divided into 4 parts
```

Population statistics by poppr

It returns various population statistics — **png()** will produce set of figures for each population. You can similarly use **svg()**, **pdf()**, ...

```
1 png(filename="poppr_%03d.png", width=720, height=720, bg="white")
2   # Output figures will be saved to the disk
3 poppr(dat=hauss.genind, total=TRUE, sample=1000, method=4,
4   missing="geno", cutoff=0.15, quiet=FALSE, clonecorrect=FALSE,
5   hist=TRUE, minsamp=1, legend=TRUE)
6 dev.off() # Closes graphical device - needed after use of functions
7   # like png(), svg(), pdf(), ... to write the file(s) to the disk
```

- Pop — Population name (Note that "Total" also means "Pooled")
- N — Number of individuals observed
- MLG — Number of multilocus genotypes (MLG) observed
- eMLG — The number of expected MLG at the smallest sample size 10 based on rarefaction
- SE — Standard error based on eMLG
- H — Shannon-Wiener Index of MLG diversity
- G — Stoddart and Taylor's Index of MLG diversity
- Hexp — Nei's 1978 genotypic diversity (corrected for sample size), or Expected Heterozygosity
- E.5 — Evenness, E5
- Ia — The index of association, IA
- rbarD — The standardized index of association
- See poppr's manual for details

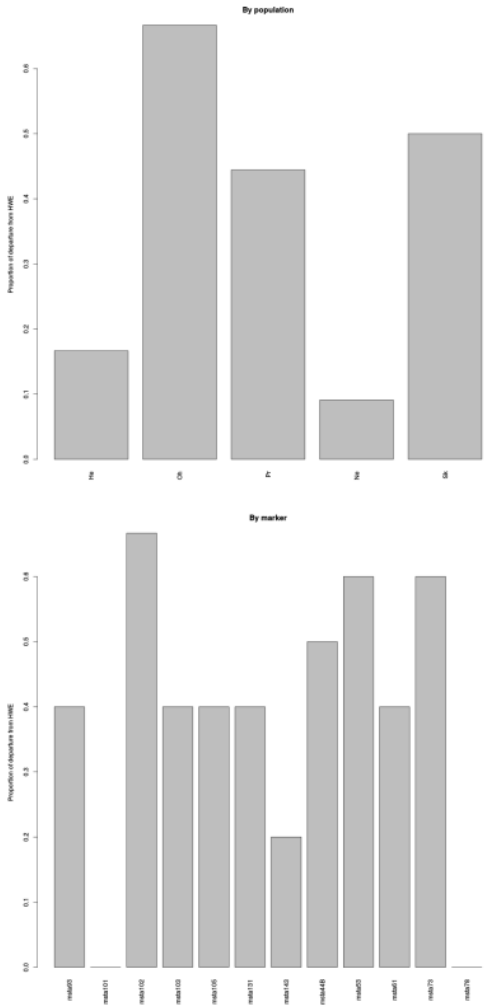
Departure from Hardy-Weinberg equilibrium

```
# Departure from HWE
hauss.hwe <- HWE.test.genind(x=hauss.genind, permut=TRUE,
  nsim=1000, res.type="matrix")
# According to populations
barplot(apply(X=hauss.hwe<0.05, 1, FUN=mean, na.rm=TRUE),
  las=3, ylab="Proportion of departure from HWE", main="By
  population", names.arg=hauss.genind@pop.names)
# According to loci
barplot(apply(X=hauss.hwe<0.05, 2, FUN=mean, na.rm=TRUE),
  las=3, ylab="Proportion of departure from HWE", main="By marker")
hauss.hwe.test <- hw.test(x=hauss.loci, B=1000)
hauss.hwe.test
# List of lists for all combinations of loci and populations
hauss.hwe.full <- HWE.test.genind(x=hauss.genind, permut=TRUE,
  nsim=1000, res.type="full")
hauss.hwe.full # Very looong list...
```

Departure from HWE — results

```
> hauss.hwe.test
```

	chi^2	df	Pr(chi^2 >)	Pr.exact
msta93	383.5519728	190	3.552714e-15	0.000
msta101	0.6927242	1	4.052393e-01	0.657
msta102	83.0741964	10	1.250111e-13	0.000
msta103	77.1819098	45	1.998865e-03	0.000
msta105	282.3441356	66	0.000000e+00	0.000
msta131	175.8647805	36	0.000000e+00	0.000
msta143	155.4567196	45	4.474199e-14	0.000
msta44B	112.6283892	28	4.149125e-12	0.000
msta53	110.2581985	55	1.436375e-05	0.002
msta61	70.7719907	66	3.215148e-01	0.070
msta73	310.9811293	153	7.842615e-13	0.000
msta78	4.4688908	10	9.237260e-01	0.569



F-statistics and population theta

Functions return tables of Fst/thetas values for populations/loci

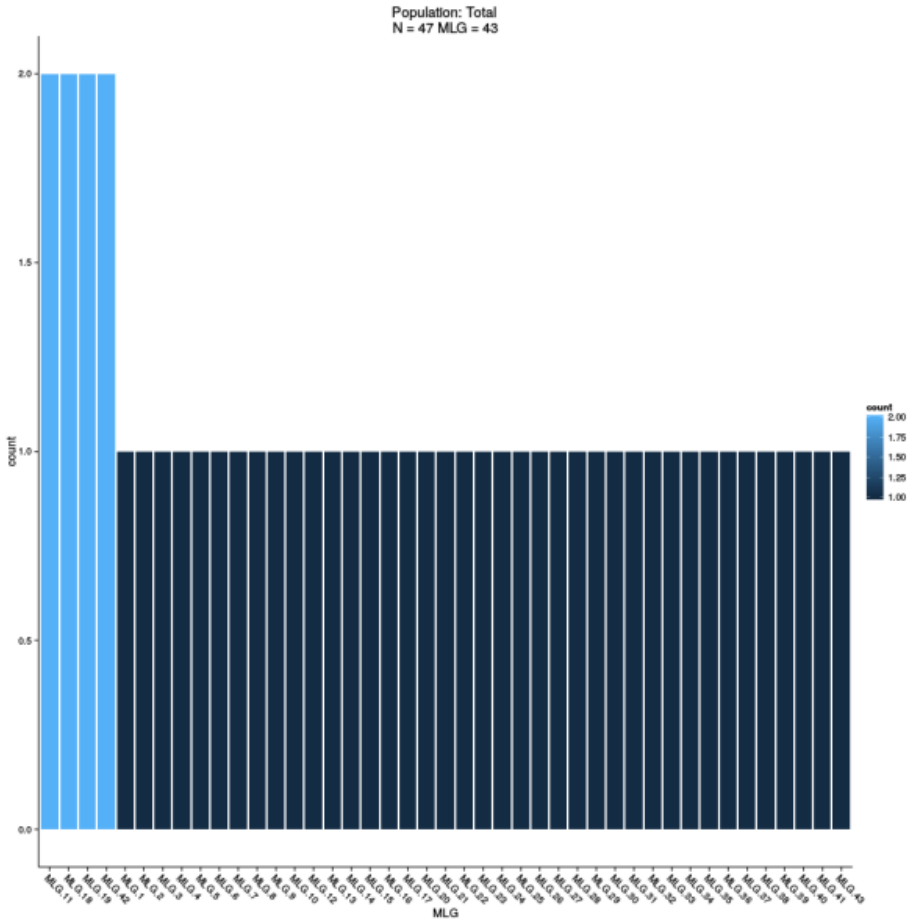
```
# Fit, Fst and Fis for each locus
Fst(x=hauss.loci.cor, pop=1)
# multilocus estimators of variance components and F-statistics,
# alternative to Fst
fstat(x=hauss.genind, pop=NULL, fstonly=FALSE)
# Nei's pairwise Fst between all pairs of populations.
# Difference in res.type="dist"/"matrix" is only in format of output
pairwise.fst(x=hauss.genind, pop=NULL, res.type="matrix", truenames=T)
# Estimates of population theta according to Kimmel et al. 1998
theta.msat(hauss.loci.cor)
```

```
Fst(x=hauss.loci.cor, pop=1)
```

	Fit	Fst	Fis
msta93	0.31835291	0.17867087	0.17006829
msta101	-0.09968472	0.04064928	-0.14628018
...	...	...	...

Multi locus genotypes

```
# Total number of MLGs
# (simple value)
mlg(pop=hauss.genind, quiet=FALSE)
# MLGs shared among populations
mlg.crosspop(pop=hauss.genind,
  df=TRUE, quiet=FALSE)
# Detailed view on distribution
# of MLGs into populations
# (table and/or plot)
mlg.table(pop=hauss.genind,
  bar=TRUE, total=TRUE,
  quiet=FALSE)
# See manual pages for details
```



Functions from poppr package — the best for microsatellites, although available also from another data types

Basic distances

```
# Euclidean distance for individuals
hauss.dist <- dist(x=hauss.genind, method="euclidean", diag=T, upper=T)
hauss.dist
# Nei's distance (not Euclidean) for populations
# (other methods are available, see ?dist.genpop)
hauss.dist.pop <- dist.genpop(x=hauss.genpop, method=1, diag=T, upper=T)
# Test if it is Euclidean
is.euclid(hauss.dist.pop, plot=TRUE, print=TRUE, tol=1e-10) # FALSE = No
# Turn to be Euclidean
hauss.dist.pop <- cailliez(distmat=hauss.dist.pop, print=FALSE,
  tol=1e-07, cor.zero=TRUE)
# Test if it is Euclidean
is.euclid(hauss.dist.pop, plot=TRUE, print=TRUE, tol=1e-10) # TRUE = OK
# Show it
hauss.dist.pop
```

Most of analysis based on distances more or less require Euclidean (non-negative) distances. If the distance matrix contains non-Euclidean distances, the result can be weird...

Distances reflecting microsatellite repeats

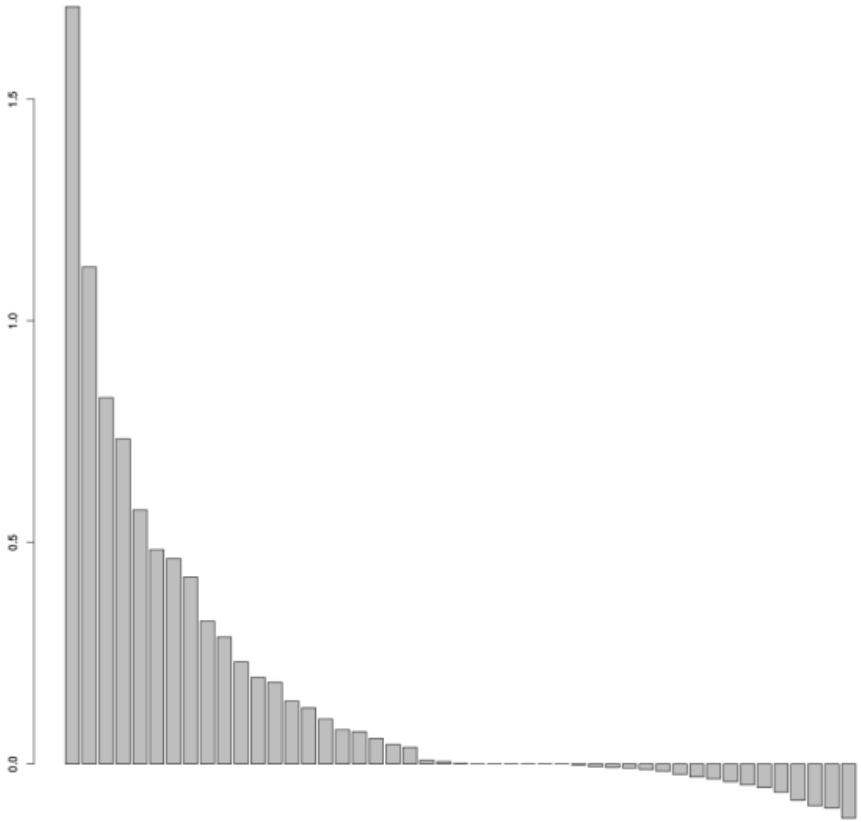
```
# Bruvo's distances weighting SSRs repeats - take care about replen
# parameter - requires repetition length for every SSRs locus
hauss.dist.bruvo <- bruvo.dist(pop=hauss.genind, replen=rep(2, 12),
  loss=TRUE)
# Test if it is Euclidean
is.euclid(hauss.dist.bruvo, plot=TRUE, print=TRUE, tol=1e-10)
# Turn to be Euclidean
hauss.dist.bruvo <- cailliez(distmat=hauss.dist.bruvo, print=FALSE,
  tol=1e-07, cor.zero=TRUE)
# Test if it is Euclidean
is.euclid(hauss.dist.bruvo, plot=TRUE, print=TRUE, tol=1e-10)
# Show it
hauss.dist.bruvo
```

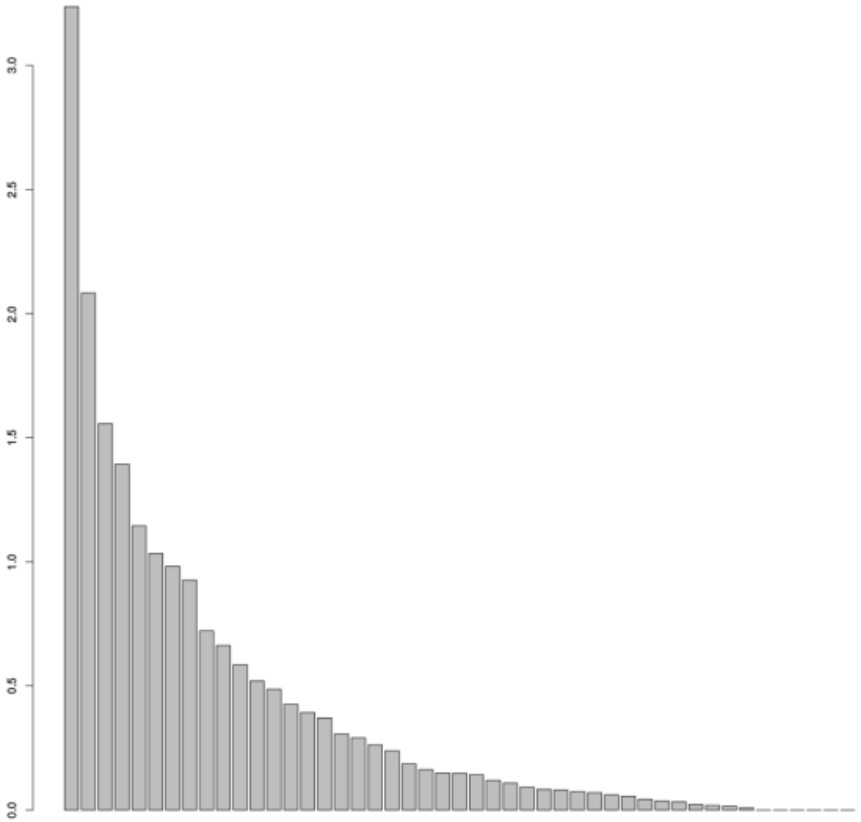
See poppr's manual and manual pages of the functions for details and different possibilities of settings.

Turning distance matrix into Euclidean is controversial...

How to deal with zero distances in original matrix? There is no really good solution...

Histograms of Bruvo distance before and after transformation:





More distances...

```
# Nei's distance (not Euclidean) for individuals
# (other methods are available, see ?nei.dist from poppr package)
hauss.dist.nei <- nei.dist(x=hauss.genind, warning=TRUE)
hauss.dist.nei
# Dissimilarity matrix returns a distance reflecting the number of
# allelic differences between two individuals
hauss.dist.diss <- diss.dist(x=hauss.genind, percent=FALSE, mat=TRUE)
hauss.dist.diss
```

Import own distance matrix from another software:

```
   Fe        He       Oh        ...
Fe 0.00000   132.019  109.159   ...
He 132.0191  0.00000  9.89111   ...
Oh 109.1590  9.89111  0.00000   ...
Pr 139.5669  8.55312  4.40562   ...
Ne 156.7619  9.96143  16.6927   ...
... ...      ...      ...       ...
```

```
MyDistance <- read.csv("distances.
  txt", header=TRUE, sep="\t",
  dec=".", row.names=1)
MyDistance <- as.dist(MyDistance)
class(MyDistance)
dim(MyDistance)
MyDistance
```

Distances among DNA sequences

The sequences must be aligned before calculating distances among them!

```
?dist.dna # There are various models available
usflu.dist <- dist.dna(x=usflu.dna, model="TN93")
usflu.dist
class(usflu.dist)
dim(as.matrix(usflu.dist))
meles.dist <- dist.dna(x=meles.dna, model="F81")
meles.dist
class(meles.dist)
dim(as.matrix(meles.dist))
```

Distances and genlight object

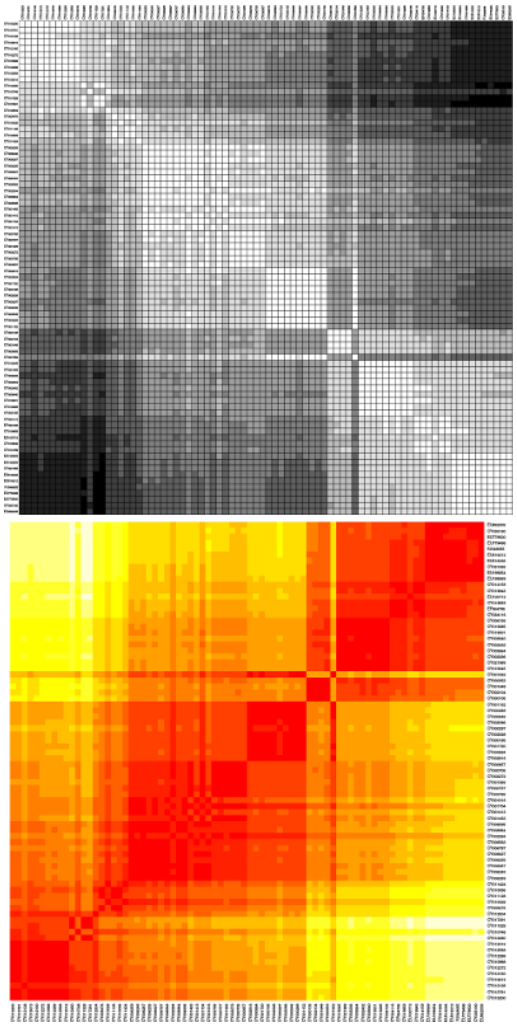
Pairwise genetic distances for each data block (genlight objects with whole genome data) — sensitive to missing data (not useful in every case):

```
usflu.dists <- lapply(X=usflu.genlight, FUN=function(DDD)
  dist(as.matrix(DDD)))
class(usflu.dists)
names(usflu.dists)
class(usflu.dists[[1]])
usflu.distr <- Reduce(f="+", x=usflu.dists)
class(usflu.distr)
usflu.distr
# It is possible to use just basic dist function on whole
# genlight object
usflu.distg <- dist(as.matrix(usflu.genlight))
```

Rationale of this approach is to save resources when dividing whole data set into smaller blocks — useful for huge data, nor for most of the cases

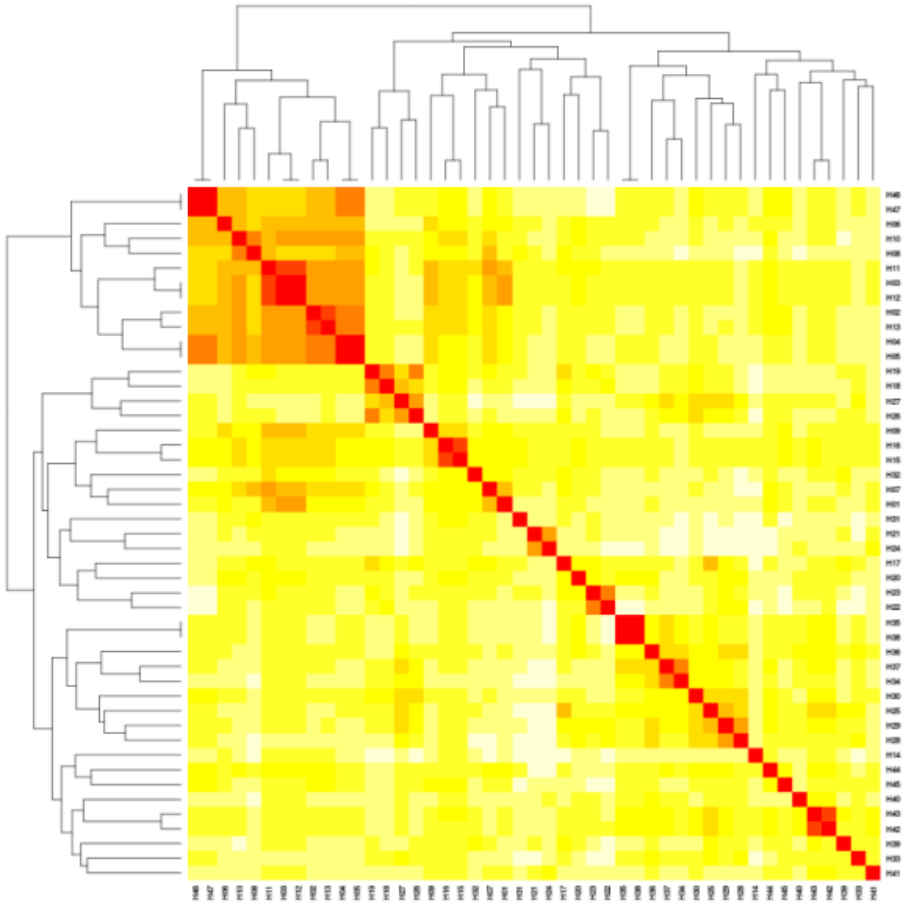
Visualize pairwise genetic similarities

```
# table.paint() requires data
# frame, dist can't be directly
# converted to DF
table.paint(df=as.data.frame(
  as.matrix(usflu.dist)), cleg=0,
  clabel.row=0.5, clabel.col=0.5)
# Same visualization, coloured
# heatmap() reorders values
# because by default it plots
# also dendrograms on the edges
heatmap(x=as.matrix(usflu.dist),
  Rowv=NA, Colv=NA, symm=TRUE)
```



Another possibility is to use function **corrplot()** from package **corrplot** — it plots correlation plots.

Heatmaps



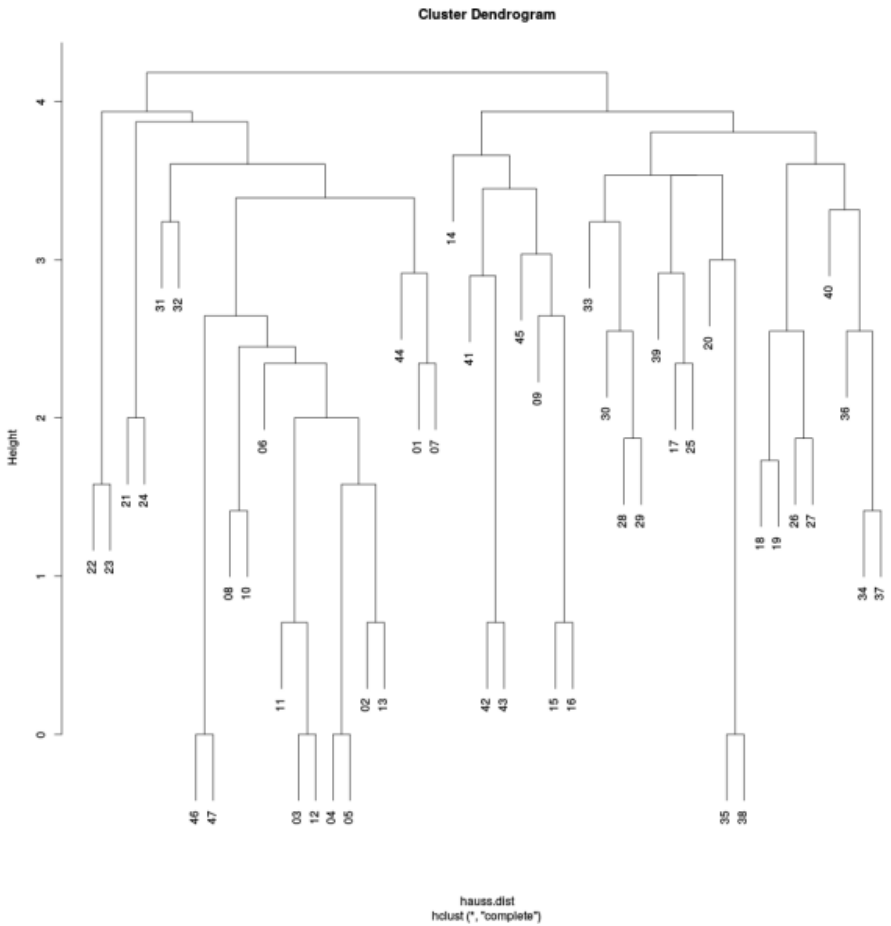
```
1  # Based on various distances
2  heatmap(as.matrix(hauss.dist),
3    symm=TRUE, labRow=rownames(
4    as.matrix(hauss.dist.bruvo)),
5    labCol=colnames(as.matrix(
6    hauss.dist.bruvo)))
7    # hauss.dist doesn't contain
8    # names of individuals - add here
9  heatmap(as.matrix(hauss.dist.pop),
10   symm=TRUE)
11 heatmap(as.matrix(hauss.dist.
12   bruvo), symm=TRUE)
13 heatmap(as.matrix(hauss.dist.
14   diss), symm=TRUE)
```

There are various settings — colours, dendrogram, ...See **?heatmap**.

Hierarchical clustering — UPGMA and others

```
# According to distance used
# see ?hclust for methods
plot(hclust(d=hauss.dist,
  method="complete"))
plot(hclust(d=hauss.dist.pop,
  method="complete"))
plot(hclust(d=hauss.dist.
  bruvo, method="complete"))
```

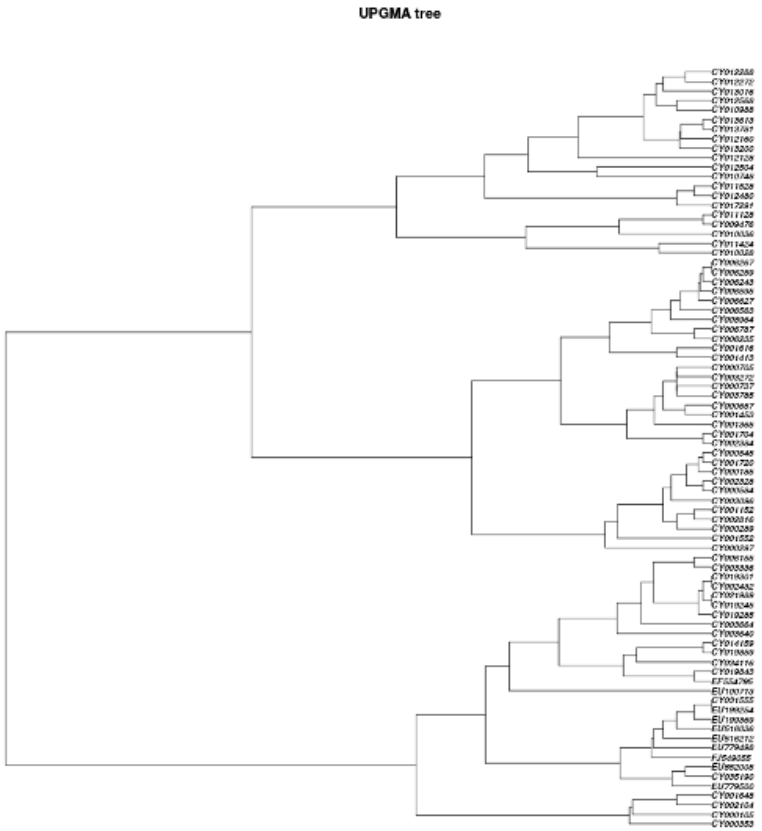
This is very basic function to make dendrogram. There are better possibilities (NJ etc).

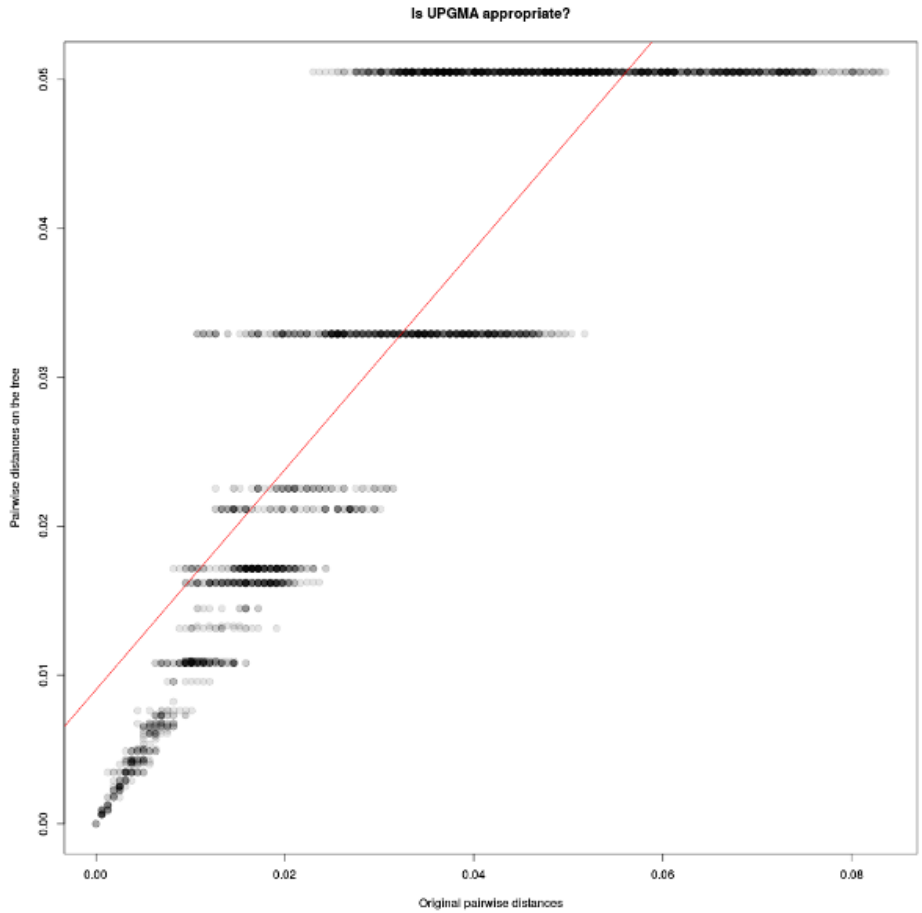


UPGMA and its test

```
# Calculate it
# Saving as phylo object (and not hclust) gives more
# possibilities for further plotting and manipulations
usflu.upgma <- as.phylo(hclust(d=usflu.dist, method="average"))
plot.phylo(x=usflu.upgma, cex=0.75)
title("UPGMA tree")
# Test quality - tests correlation of original distance in the matrix
# and reconstructed distance from hclust object
plot(x=as.vector(usflu.dist), y=as.vector(as.dist(
  cophenetic(usflu.upgma))), xlab="Original pairwise distances",
  ylab="Pairwise distances on the tree", main="Is UPGMA
  appropriate?", pch=20, col=transp(col="black",
  alpha=0.1), cex=2)
abline(lm(as.vector(as.dist(cophenetic(usflu.upgma)))~
  as.vector(usflu.dist)), col="red")
```

UPGMA is not the best choice here...

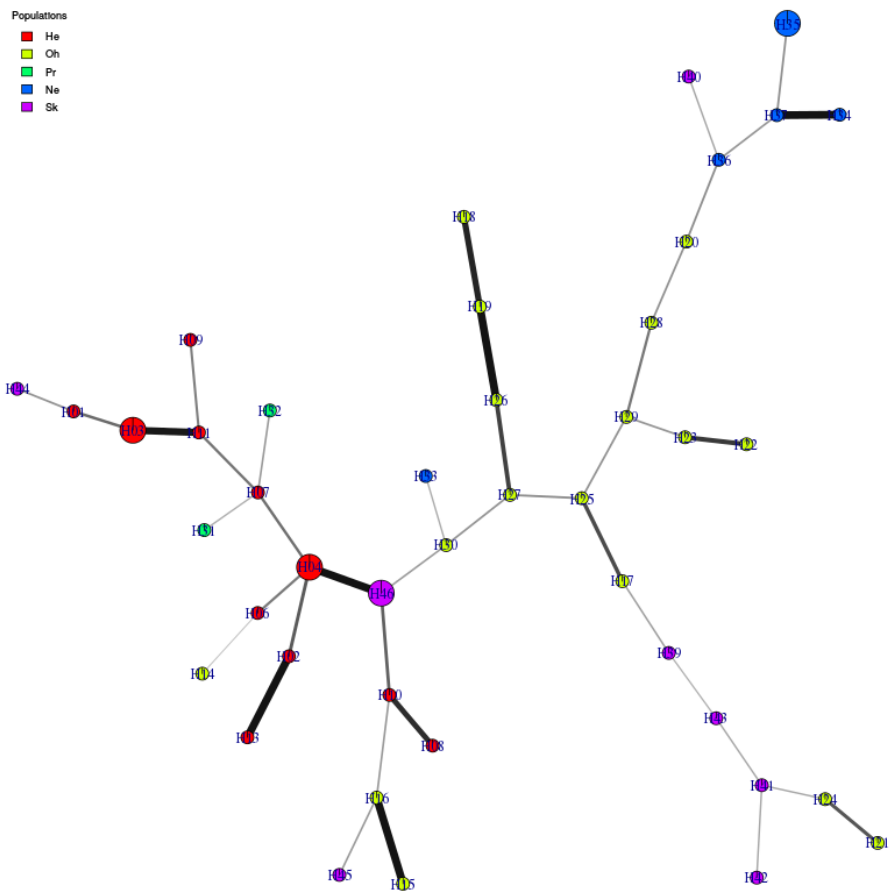




AMOVA

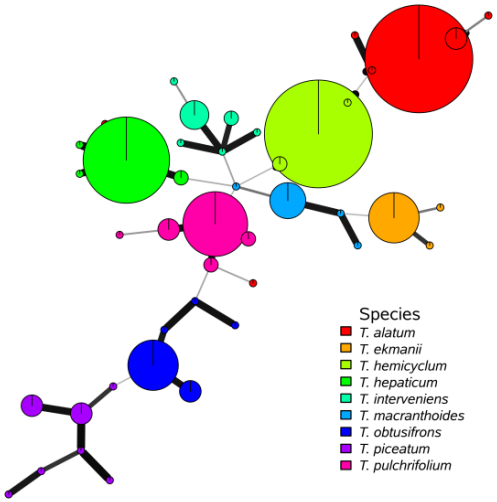
```
# From package pegas (doesn't directly show percentage of variance)
hauss.pop <- hauss.genind$pop
hauss.amova <- amova(hauss.dist~hauss.pop, data=NULL,
  nperm=1000, is.squared=TRUE)
hauss.amova
# Output:
$tab
                SSD      MSD df
hauss.pop  15.11916 3.779790  4
Error      59.62501 1.419643 42
Total      74.74417 1.624873 46
# Another possibility is poppr.amova - for more complicated hierarchy
# see ?poppr.amova
```

Minimum Spanning Network




```
1 bruvo.msn(pop=hauss.genind,
2   replen=rep(2, 12), loss=TRUE,
3   palette=rainbow, vertex.label=
4   "inds", gscale=TRUE,
5   wscale=TRUE, showplot=TRUE)
```

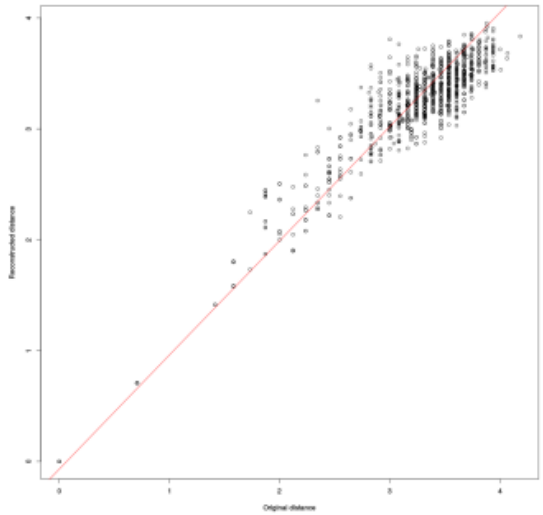
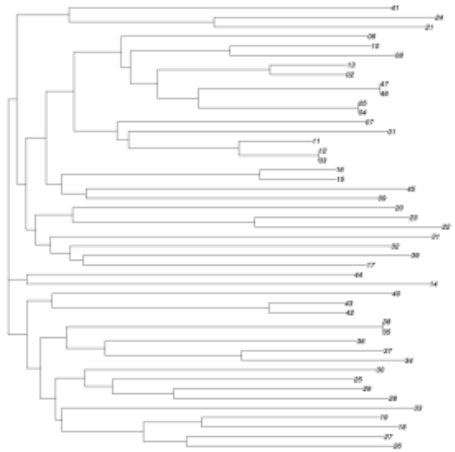
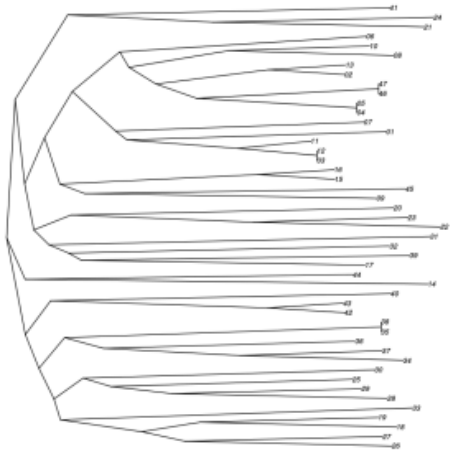
Package poppr, based on Bruvo's distance (for SSRs).

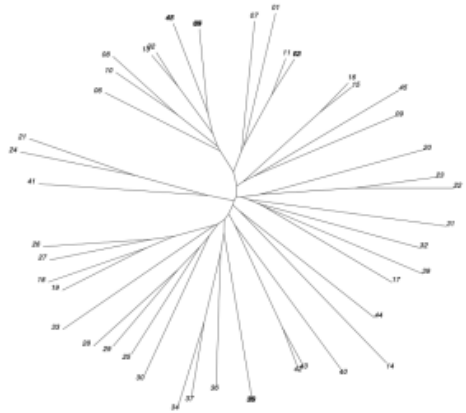


Calculate and test NJ tree

```
# Calculates the tree (try with various distances)
hauss.nj <- nj(hauss.dist)
# Test tree quality - plot original vs. reconstructed distance
plot(as.vector(hauss.dist), as.vector(as.dist(cophenetic(hauss.nj))),
  xlab="Original distance", ylab="Reconstructed distance")
abline(lm(as.vector(hauss.dist) ~
  as.vector(as.dist(cophenetic(hauss.nj)))), col="red")
summary(lm(as.vector(hauss.dist) ~
  as.vector(as.dist(cophenetic(hauss.nj))))) # Prints summary text
# Bootstrap
hauss.boot <- boot.phylo(phy=hauss.nj, x=hauss.loci.cor,
  FUN=function(XXX) nj(dist(loci2genind(XXX))), B=1000)
# Plot a basic tree - see ?plot.phylo for details
plot.phylo(x=hauss.nj, type="phylogram")
plot.phylo(x=hauss.nj, type="cladogram", edge.width=2)
plot.phylo(x=hauss.nj, type="fan", edge.width=2, edge.lty=2)
plot.phylo(x=hauss.nj, type="radial", edge.color="red",
  edge.width=2, edge.lty=3, cex=2)
```

Choose your tree...

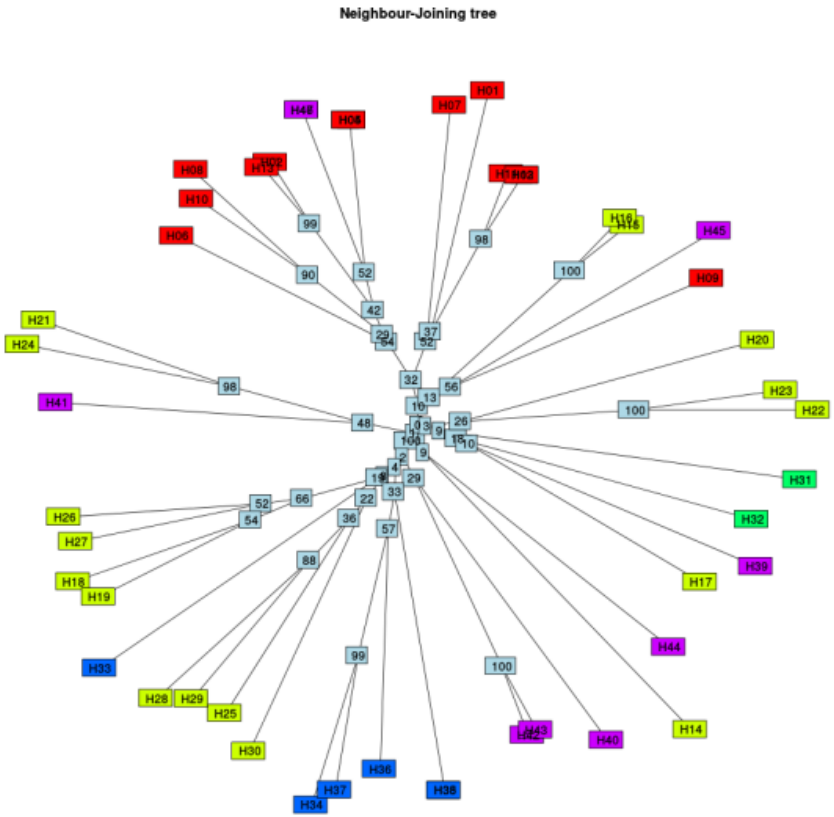




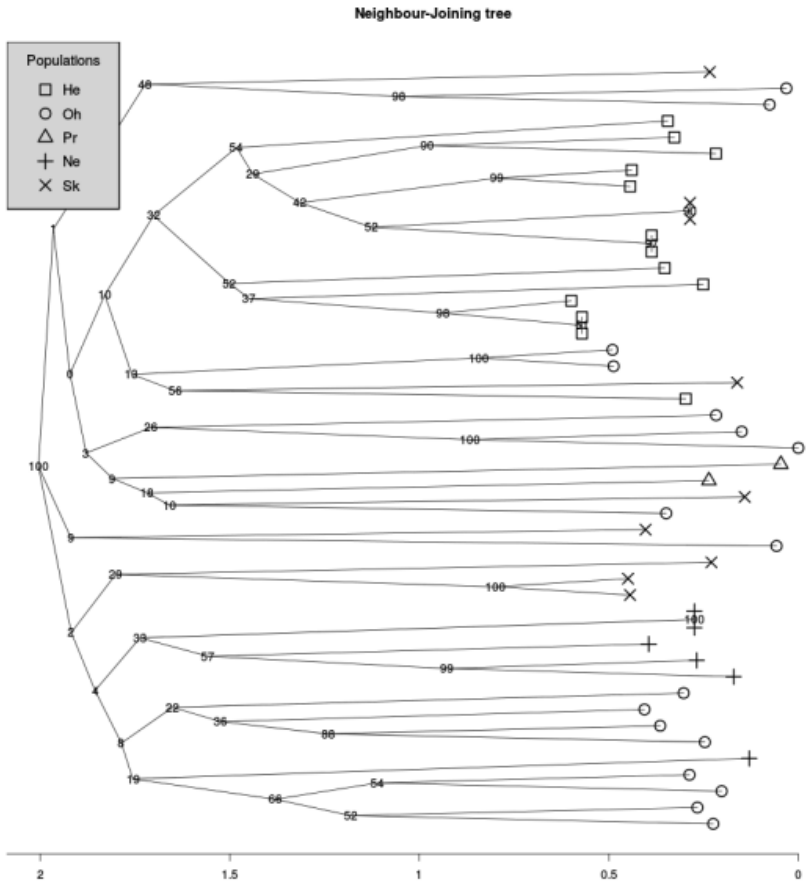



Nicer trees

```
## Plot a nice tree with colored tips
plot.phylo(x=hauss.nj, type="unrooted", show.tip=F, lwd=3, main="NJ")
# Labels for nodes - bootstrap - see ?nodelabels for graphical settings
nodelabels(text=round(hauss.boot/10))
# Colored labels - creates vector of colors according to populations
nj.rainbow<-colorRampPalette(rainbow(length(levels(pop(hauss.genind)))))
tiplabels(text=hauss.genind$ind.names, bg=fac2col(x=hauss.genind$pop,
  col.pal=nj.rainbow)) # Coloured tips
## Plot BW tree with tip symbols and legend
plot.phylo(x=hauss.nj, type="cladogram", show.tip=F, lwd=3, main="NJ")
axisPhylo() # Add axis with distances
# From node labels let's remove unneeded frame
nodelabels(text=round(hauss.boot/10), frame="none", bg="white")
# As tip label we use only symbols - see ?points for graphical details
tiplabels(frame="none", pch=rep(0:4,times=c(13,17,2,6,9)), lwd=2, cex=2)
# Plot a legend explainindg symbols
legend(x="topleft", legend=c("He", "Oh", "Pr", "Ne", "Sk"),
  border="black", pch=0:4, pt.lwd=2, pt.cex=2, bty="o", bg="lightgrey",
  box.lwd=2, cex=1.2, title="Populations")
```

Choose your tree...



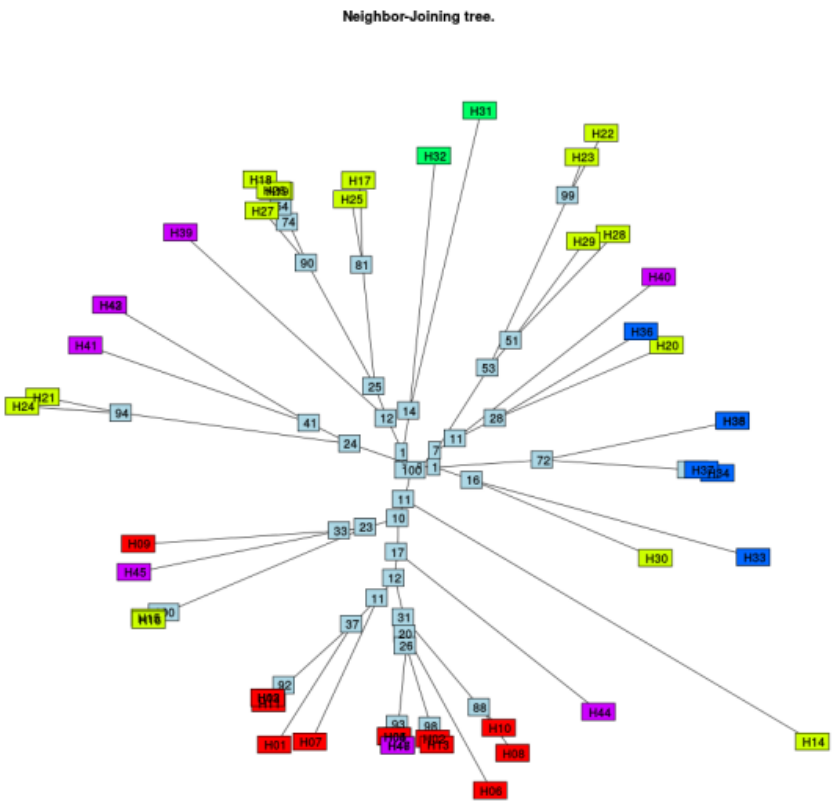


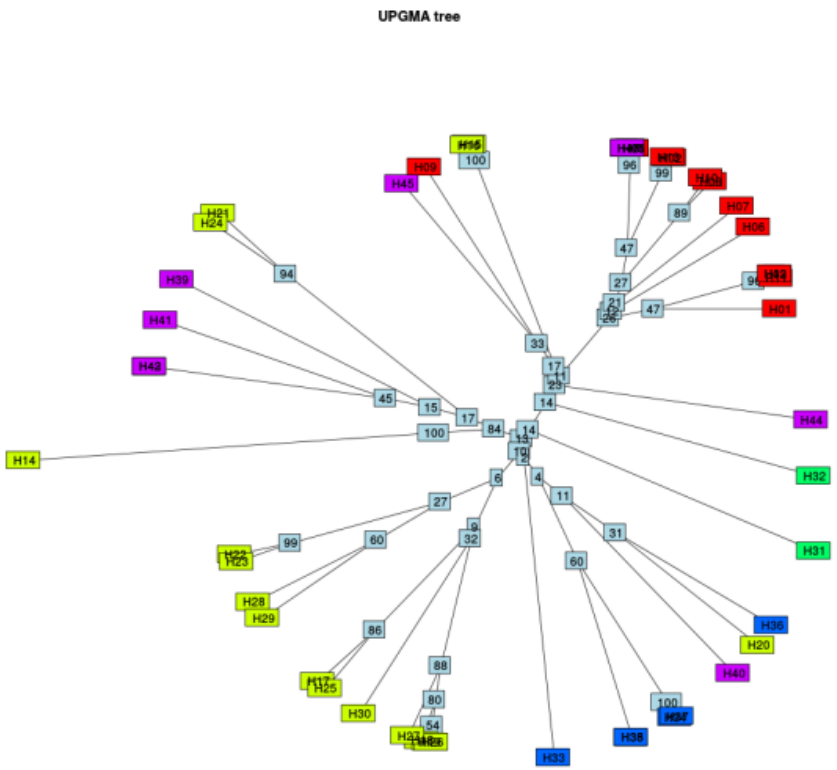
Trees based on Bruvo's distance

Package poppr (bootstrap is incorporated within the function)

```
# NJ
hauss.nj.bruvo <- bruvo.boot(pop=hauss.genind, replen=rep(2, 12),
  sample=1000, tree="nj", showtree=TRUE, cutoff=1, quiet=FALSE)
plot.phylo(x=hauss.nj.bruvo, type="unrooted", show.tip=FALSE,
  lwd=3, main="Neighbor-Joining tree.")
# you can call node labels as phylo$node.labels or phylo[["node.labels"]]
nodelabels(hauss.nj.bruvo[["node.labels"]])
tiplabels(hauss.nj.bruvo[["tip.label"]], bg=fac2col(x=hauss.genind$pop,
  col.pal=nj.rainbow))
# UPGMA
hauss.upgma <- bruvo.boot(pop=hauss.genind, replen=rep(2, 12),
  sample=1000, tree="upgma", showtree=TRUE, cutoff=1, quiet=FALSE)
plot.phylo(hauss.upgma, type="unrooted", show.tip=FALSE, lwd=3,
  main="UPGMA tree")
nodelabels(hauss.upgma[["node.labels"]])
tiplabels(hauss.upgma[["tip.label"]], bg=fac2col(x=hauss.genind@pop,
  col.pal=nj.rainbow))
```

Choose your tree...





NJ tree of populations

```
# Populations
hauss.nj.pop <- nj(hauss.dist.pop)
# Bootstrap - source() loads external scripts
# boot.phylo doesn't work for population trees
source("http://trapa.cz/sites/default/rcourse/boot_phylo_nj_pop.r")
hauss.boot.pop <- boot.phylo.nj.pop(hauss.nj.pop, hauss.genind, 1000)
# Plot a tree
plot(hauss.nj.pop, type="radial", cex=1.2, lwd=3,
  main="Neighbor-Joining tree of populations")
# Labels - bootstrap
nodelabels(round(hauss.boot.pop/10), frame="none")
# Print information about phylo object
print.phylo(hauss.nj.pop)
```

NJ tree based on DNA sequences

```
usflu.tree <- nj(X=usflu.dist)
# Plot it
plot.phylo(x=usflu.tree, type="unrooted", show.tip=FALSE)
title("Unrooted NJ tree")
# Coloured tips
usflu.pal <- colorRampPalette(topo.colors(length(levels(as.factor(
  usflu.annot[["year"]])))))
# Tip labels
tiplabels(text=usflu.annot$year, bg=num2col(usflu.annot$year,
  col.pal=usflu.pal), cex=0.75)
# Legend - describing years - pretty() automatically shows best
# values from given range, num2col() selects colours from colour scale
legend(x="bottomright", fill=num2col(x=pretty(x=1993:2008, n=8),
  col.pal=usflu.pal), leg=pretty(x=1993:2008, n=8), ncol=1)
```

Root the tree

```
# Root the tree - "outgroup" is name of accession (in quotation
# marks) or number (position within phy object)
usflu.tree.rooted <- root(phy=usflu.tree, outgroup=1)
# Plot it
plot.phylo(x=usflu.tree.rooted, show.tip=FALSE, edge.width=2)
title("Rooted NJ tree")
# Labeling of tips
tiplabels(text=annot$year, bg=transp(num2col(x=annot$year,
  col.pal=usflu.pal), alpha=0.7), cex=0.75, fg="transparent")
# Add axis with phylogenetic distance
axisPhylo()
# Legend - describing years - pretty() automatically shows best
# values from given range, num2col() selects colours from colour scale
legend(x="topright", fill=num2col(x=pretty(x=1993:2008, n=8),
  col.pal=usflu.pal), leg=pretty(x=1993:2008, n=8), ncol=1)
```

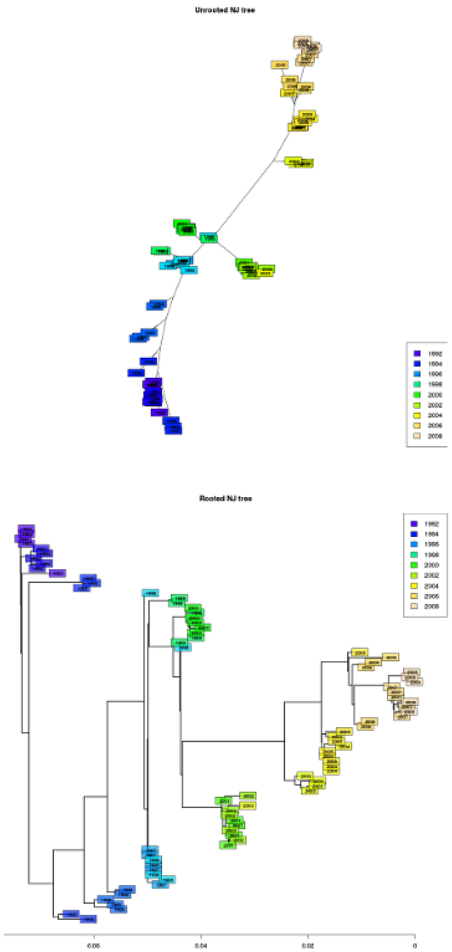
Bootstrap rooted tree

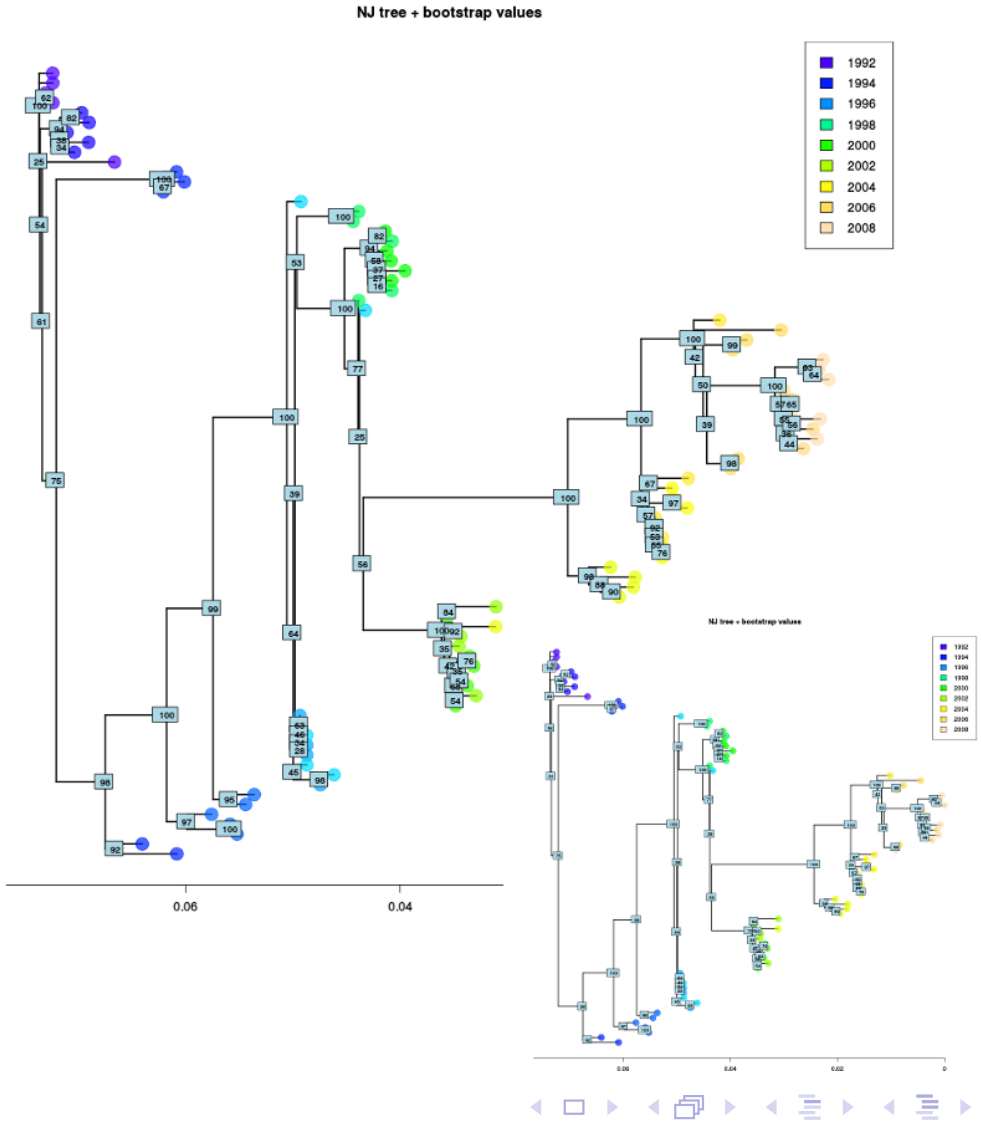
```
# Calculate it
usflu.boot <- boot.phylo(phy=usflu.tree.rooted, x=usflu.dna,
  FUN=function(EEE) root(nj(dist.dna(EEE, model="TN93")),
  outgroup=1), B=1000)
# Plot the tree
plot.phylo(x=usflu.tree.rooted, show.tip=FALSE, edge.width=2)
title("NJ tree + bootstrap values")
tiplabels(frame="none", pch=20, col=transp(num2col(x=annot[["year"]],
  col.pal=usflu.pal), alpha=0.7), cex=3.5, fg="transparent")
axisPhylo()
# Legend - describing years - pretty() automatically shows best
# values from given range, num2col() selects colours from colour scale
legend(x="topright", fill=num2col(x=pretty(x=1993:2008, n=8),
  col.pal=usflu.pal), leg=pretty(x=1993:2008, n=8), ncol=1)
# Plots bootstrap support - note usflu.boot contains raw numbers
# transform it into percent
nodelabels(text=round(usflu.boot/10), cex=0.75)
```

Collapse branches with low bootstrap support

```
usflu.tree.temp <- usflu.tree.rooted
# Determine branches with low support - note BS values are in raw
# numbers - use desired percentage with respect to number of bootstraps
usflu.tocollapse <- match(x=which(usflu.boot < 700)+
  length(usflu.tree.rooted$tip.label), table=usflu.tree.temp$edge[,2])
# Set length of bad branches to zero
usflu.tree.temp$edge.length[usflu.tocollapse] <- 0
# Create new tree
usflu.tree.collapsed <- di2multi(phy=usflu.tree.temp, tol=0.00001)
# Plot the consensus tree
plot.phylo(x=usflu.tree.collapsed, show.tip=FALSE, edge.width=2)
title("NJ tree after collapsing weak nodes")
tiplabels(text=annot$year, bg=transp(num2col(x=annot[["year"]],
  col.pal=usflu.pal), alpha=0.7), cex=0.5, fg="transparent")
axisPhylo()
legend(x="topright", fill=num2col(x=pretty(x=1993:2008, n=8),
  col.pal=usflu.pal), leg=pretty(x=1993:2008, n=8), ncol=1)
```

The trees


Unrooted NJ tree
Rooted NJ tree


NJ tree + bootstrap values
1992
1994
1996
1998
2000
2002
2004
2006
2008
0.06
0.04
NJ tree + bootstrap values

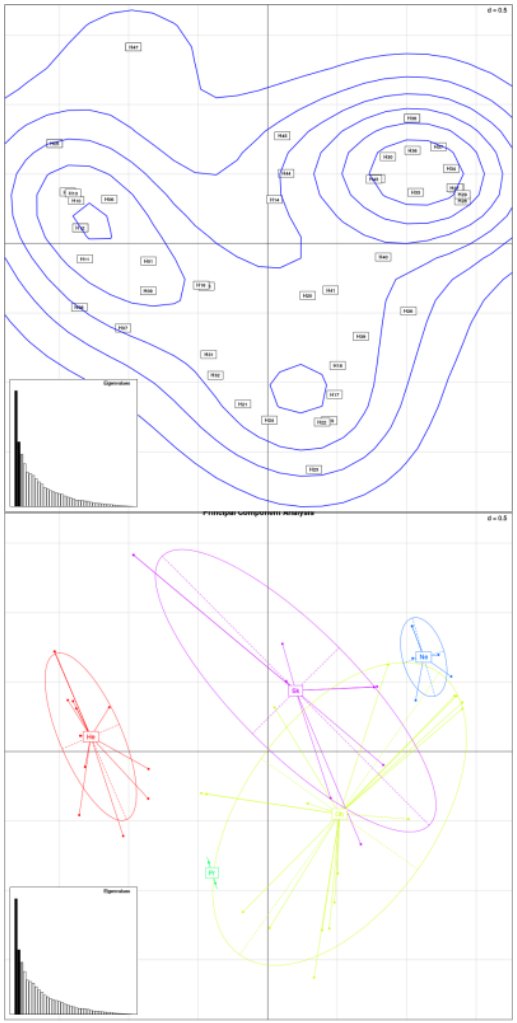
NJ is death. Long live NJ!

- "Basic" NJ has many limitations – there are several tries to overcome them
- Package **phangorn** has functions **NJ()** and unweighted version **UNJ()**
- Package **ape** has functions **njs()** and **bionjs()** which are designed to perform well on distances with (more) missing values
- Function **bionj()** from **ape** implements BIONJ algorithm
- FastME functions (package **ape**) perform the minimum evolution algorithm and aim to be replacement of NJ — read **?fastme** before use
- All those functions read distance matrix and their use is same as with "classical" **nj()** (read manual pages before using them) — it is also from package **ape**

PCoA I

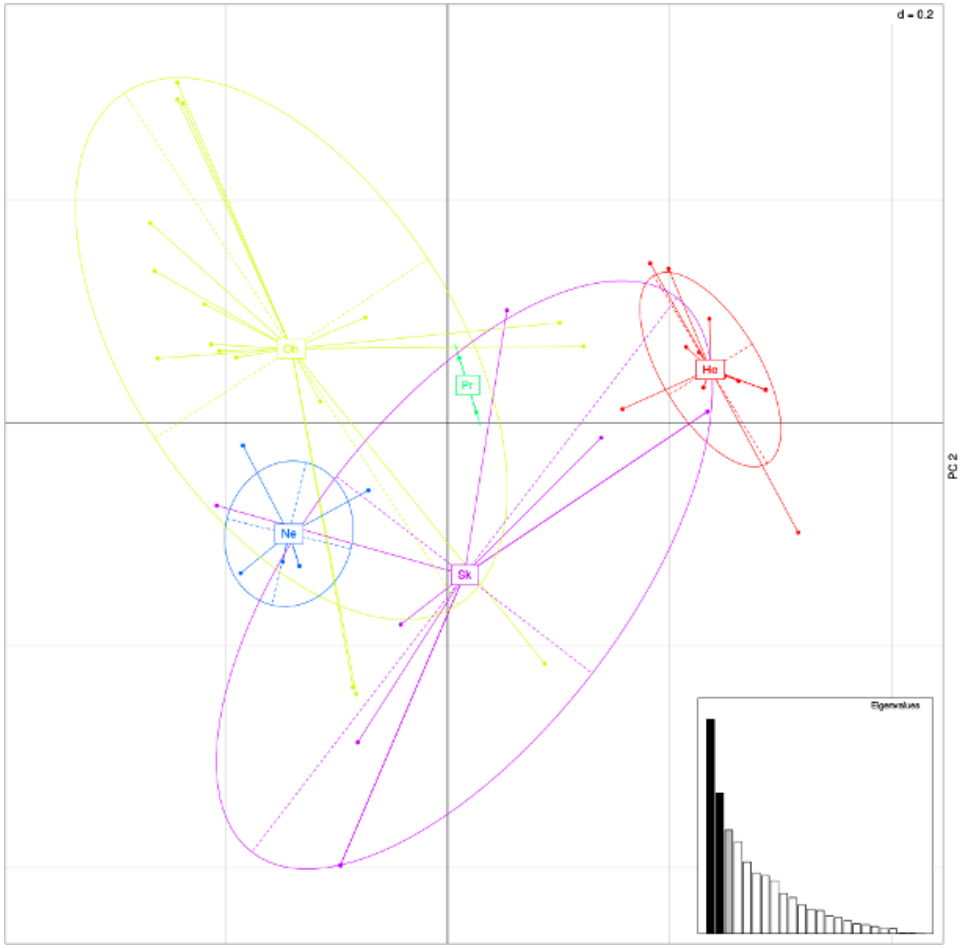
```
hauss.pcoa <- dudi.pco(d=dist(x=scaleGen(x=hauss.genind, center=TRUE,
  scale=FALSE, truenames=TRUE, missing="NA"), method="euclidean"),
  scannf=FALSE, nf=3)
# Basic display
s.label(dfxy=hauss.pcoa$li, clabel=0.75)
# To plot different axes use for example dfxy=hauss.pcoa$li[c(2, 3)]
# Add kernel density
s.kde2d(dfxy=hauss.pcoa$li, cpoint=0, add.plot=TRUE)
# Adds histogram of Eigenvalues
add.scatter.eig(w=hauss.pcoa$eig, nf=3, xax=1, yax=2,
  posi="bottomleft", sub="Eigenvalues")
# Colored display according to populations
# Creates vector of colors according to populations
hauss.pcoa.col <- rainbow(length(levels(pop(hauss.genind))))
s.class(dfxy=hauss.pcoa$li, fac=pop(hauss.genind), col=hauss.pcoa.col)
add.scatter.eig(w=hauss.pcoa$eig, nf=3, xax=1, yax=2,
  posi="bottomleft", sub="Eigenvalues")
title("Principal Coordinates Analysis") # Adds title to the graph
```

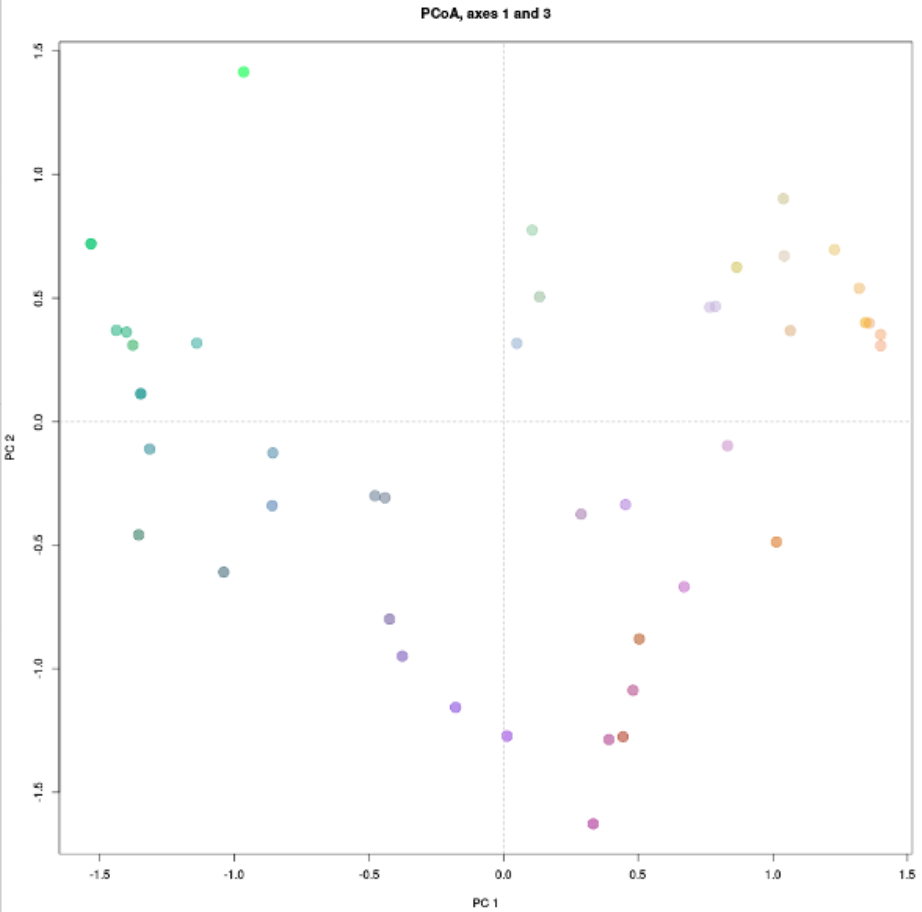
PCoA II



```
hauss.pcoa.bruvo <- dudi.pco(d=
  bruvo.dist(pop=hauss.genind,
  replen=rep(2, 12)), scannf=F,
  nf=3)
s.class(dfxy=hauss.pcoa.bruvo$li,
  fac=pop(hauss.genind),
  col=hauss.pcoa.col)
add.scatter.eig(hauss.pcoa.bruvo$
  eig, posi="bottomright", 3,1,2)
# Another possibility for coloured
# plot (see ?colorplot for details)
colorplot(xy=hauss.pcoa$li[c(1, 2)],
  X=hauss.pcoa$li, transp=TRUE,
  cex=3, xlab="PC 1", ylab="PC 2")
title(main="PCoA, axes 1 and 3")
abline(v=0, h=0, col="grey", lty=2)
```

PCoA — Bruvo and colorplot





DAPC

- Discriminant Analysis of Principal components
- Runs K-means Bayesian clustering on data transformed with PCA (reduces number of variables, speeds up process)
- Finally it runs discriminant analysis
- Various modes of displaying of results — "Structure-like", "PCA-like" and more
- More information at http://adegenet.r-forge.r-project.org/ and **adegenetTutorial("dapc")**
- If following commands would seem to complicated to you, try web interface by this command:

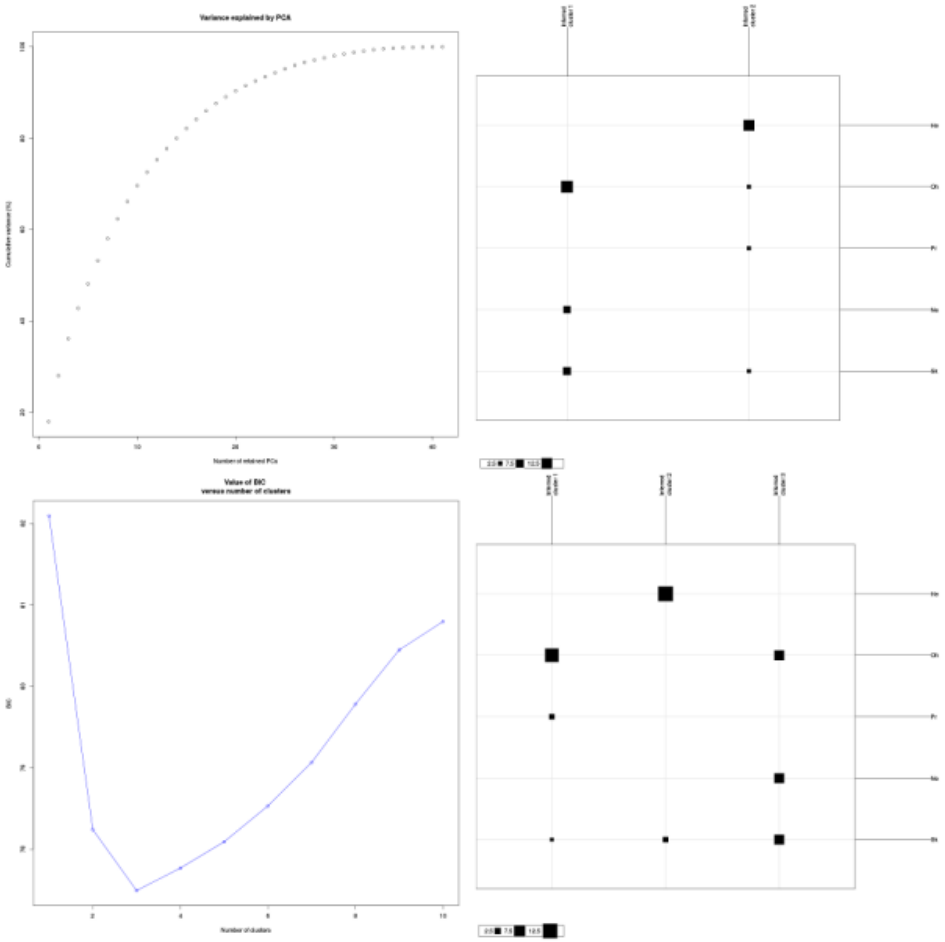
```
adegenetServer("DAPC")
```

K-find — Bayesian K-means clustering

```
# Retain all informative PC (here about 35).
# According to second graph select best K (here 2 or 3).
hauss.kfind <- find.clusters(x=hauss.genind, stat="BIC",
  choose.n.clust=TRUE, max.n.clust=10, n.iter=100000, n.start=100,
  scale=FALSE, truenames=TRUE)
# See results as text
table(pop(hauss.genind), hauss.kfind$grp)
hauss.kfind
# Graph showing table of original and inferred populations and
# assignment of individuals
table.value(df=table(pop(hauss.genind), hauss.kfind$grp),
  col.lab=paste("Inferred\ncluster", 1:length(hauss.kfind$size)), grid=T)
# For K=3 - note parameters n.pca and n.clust - we just rerun the
# analysis and when results are stable, no problem here
hauss.kfind3 <- find.clusters(x=hauss.genind, n.pca=35, n.clust=3,
  stat="BIC", choose.n.clust=FALSE, max.n.clust=10, n.iter=100000,
  n.start=100, scale=FALSE, truenames=TRUE)
```

K-find outputs

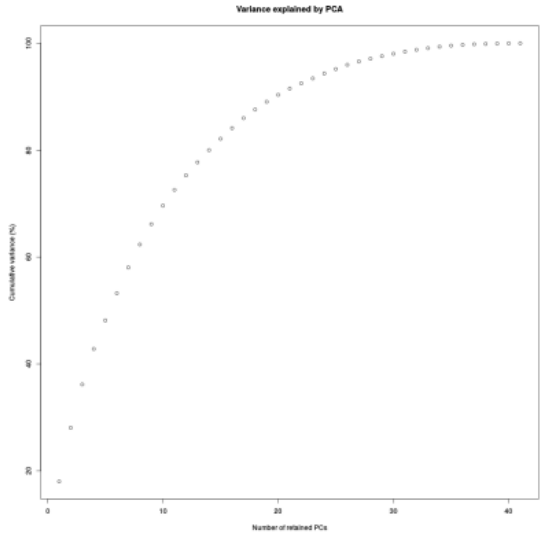
```
# See results as text
table(pop(hauss.genind),
  hauss.kfind3$grp)
hauss.kfind3
# Graph showing table of original
# and inferred populations and
# assignment of individuals
table.value(
  df=table(pop(hauss.
  genind), hauss.kfind3$grp),
  col.lab=paste("Inferred\n
  cluster",
  1:length(hauss.kfind3$size)),
  grid=TRUE)
```

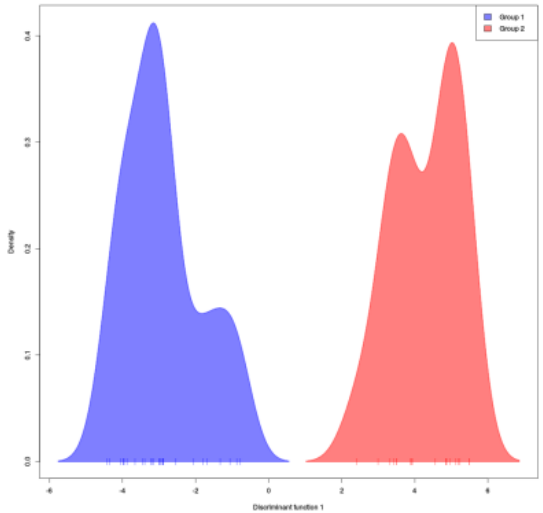


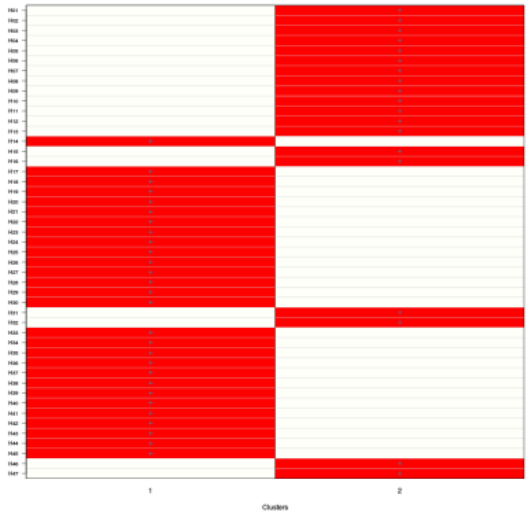
DAPC code I

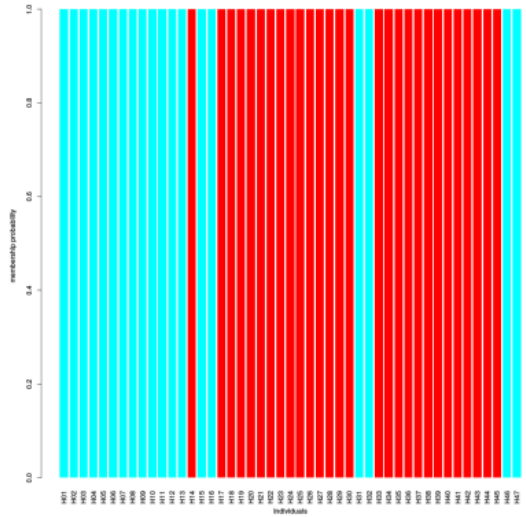
```
## K=2
# Creates DAPC
# Number of informative PC (Here 15, adegenet recommends < N/3). Select
# number of informative DA (here only one is available - no PCA graph)
hauss.dapc <- dapc(x=hauss.genind, pop=hauss.kfind$grp, center=TRUE,
  scale=FALSE, var.contrib=TRUE, pca.info=TRUE, truenames=TRUE)
# Information
hauss.dapc
# Density functions
scatter(x=hauss.dapc, xax=1, yax=1, main="DAPC", bg="white", solid=0.5,
  leg=TRUE, txt.leg=c("Group 1", "Group 2"), posi.leg="topright")
# Assignment of individuals to clusters
assignplot(x=hauss.dapc)
# Structure-like plot
compoplot(x=hauss.dapc, xlab="Individuals", leg=FALSE)
# Loadingplot - alleles the most adding to separation of individuals
loadingplot(x=hauss.dapc$var.contr)
```

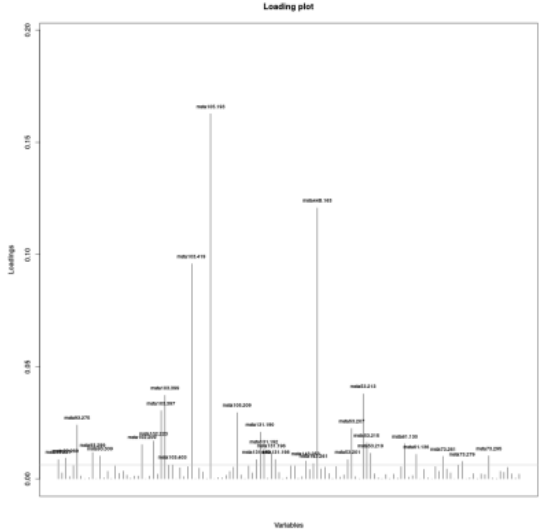
DAPC for K=2

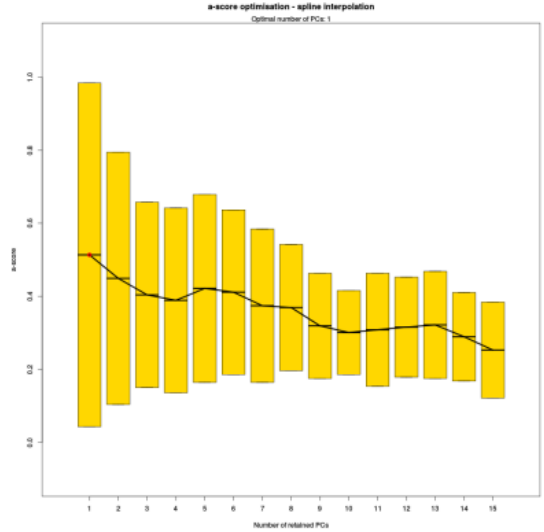

Variance explained by PCA
Cumulative variance (%)
Number of retained PCs


Group 1
Group 2
Density
Discriminant function 1


Clusters


membership probability
Individuals


Loading plot
Loadings
Variables

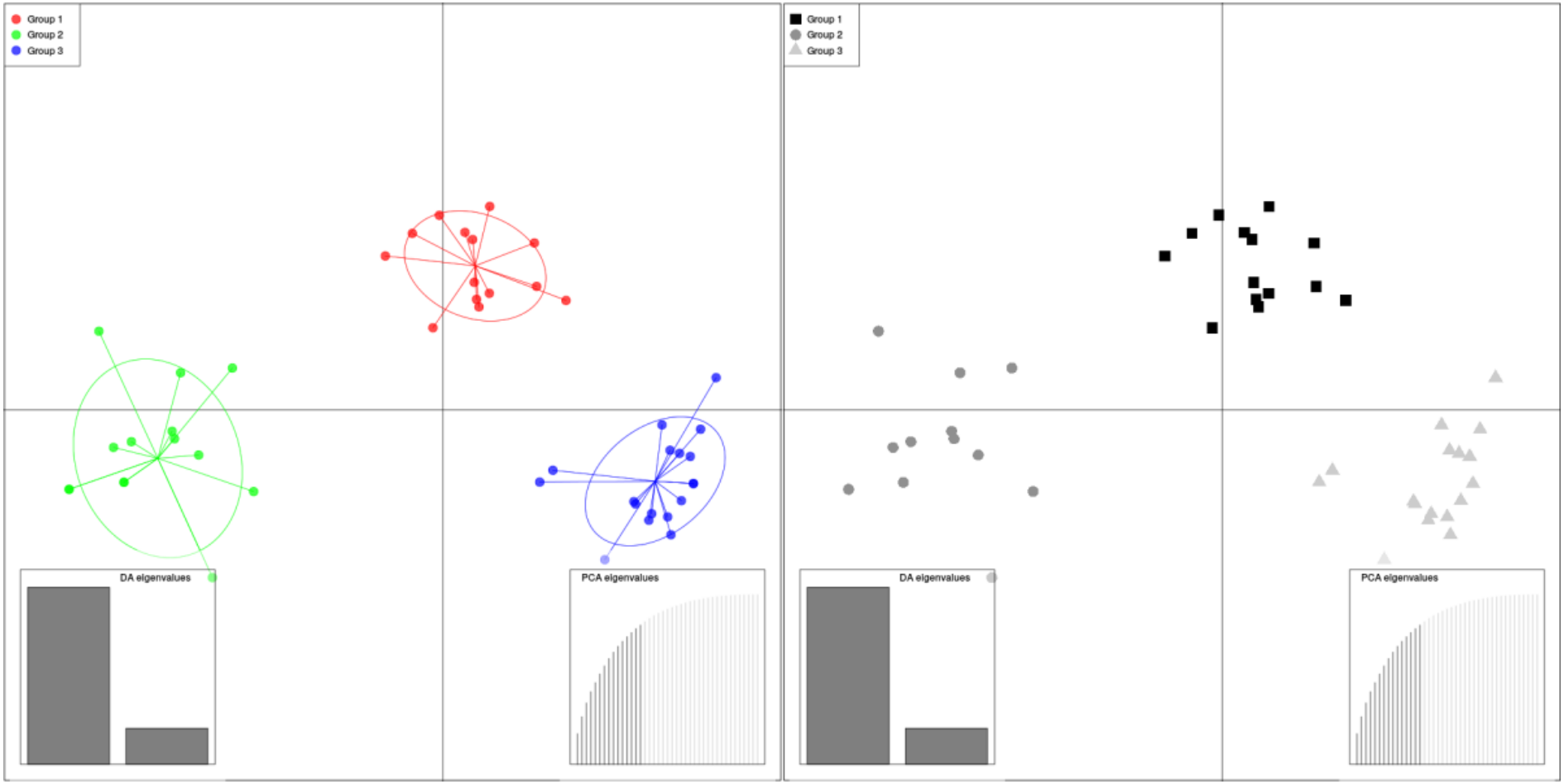

a-score optimisation - spline interpolation
Optimal number of PCs: 1
a-score
Number of retained PCs

DAPC code II

```
## alfa-score - according to number of PC axis
optim.a.score(x=hauss.dapc)
# K=3
# Creates DAPC
# Number of informative PC (Here 15, adegenet recommends < N/3).
# Select number of informative DA (here 2).
hauss.dapc3 <- dapc(x=hauss.genind, pop=hauss.kfind3$grp, center=TRUE,
  scale=FALSE, var.contrib=TRUE, pca.info=TRUE, truenames=TRUE)
# Information
hauss.dapc
# A la PCA graph
scatter(x=hauss.dapc3, main="DAPC, Cardamine", bg="white", cex=3,
  clab=0, col=rainbow(3), posi.da="bottomleft", scree.pca=TRUE,
  posi.pca="bottomright", leg=TRUE, txt.leg=c("Group 1", "Group 2",
  "Group 3"), posi.leg="topleft")
```

Especially graphical parameters have huge possiilities...

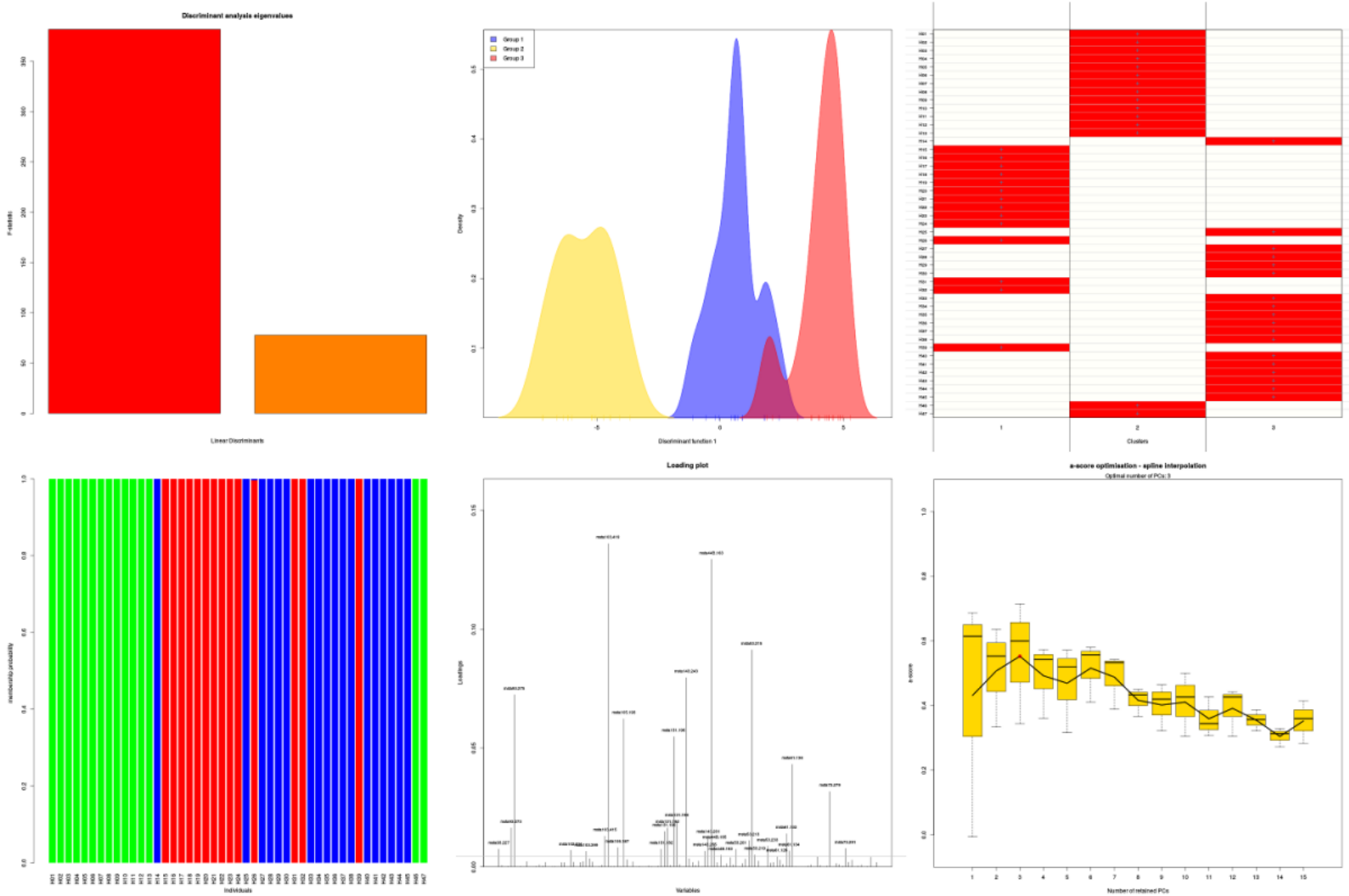
DAPC for K=3


Group 1
Group 2
Group 3
DA eigenvalues
PCA eigenvalues
Group 1
Group 2
Group 3
DA eigenvalues
PCA eigenvalues

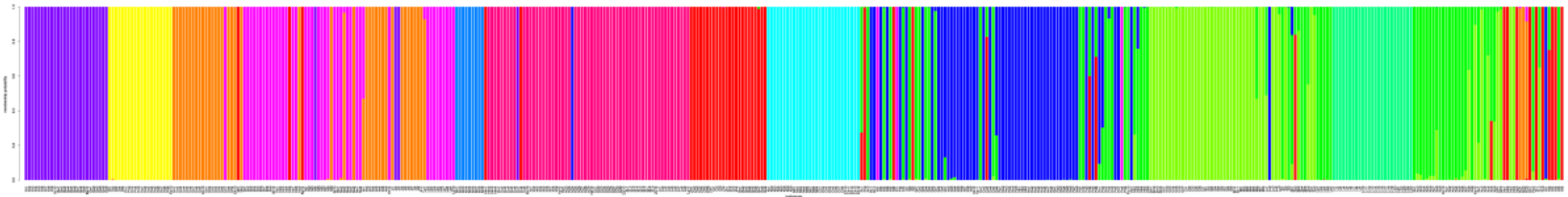
DAPC code III

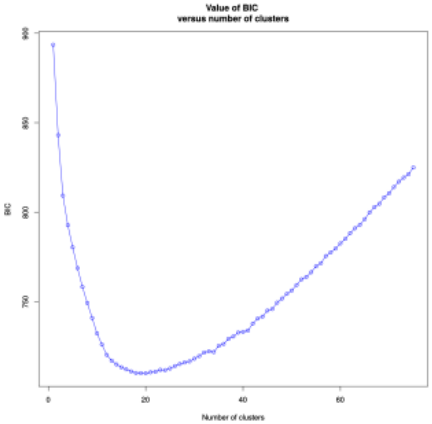
```
# Same in BW
scatter(x=hauss.dapc3, main="DAPC, Cardamine", bg="white", pch=c(15:17),
  cell=0, cstar=0, solid=1, cex=2.5, clab=0, col=grey.colors(3, start=0,
  end=0.8, gamma=2, alpha=0), posi.da="bottomleft", scree.pca=TRUE,
  posi.pca="bottomright", leg=TRUE, txt.leg=c("Group 1", "Group 2",
  "Group 3"), posi.leg="topleft")
# Density functions
scatter(x=hauss.dapc3, xax=1, yax=1, main="DAPC", bg="white", solid=0.5,
  leg=T, txt.leg=c("Group 1", "Group 2", "Group 3"), posi.leg="topleft")
# Assignment of individuals to clusters
assignplot(hauss.dapc3)
# Structure-like plot
compoplot(hauss.dapc3, xlab="Individuals", leg=FALSE)
# Loadingplot - alleles the most adding to separation of individuals
loadingplot(hauss.dapc3$var.contr)
# alfa-score - according to number of PC axis
optim.a.score(hauss.dapc3)
```

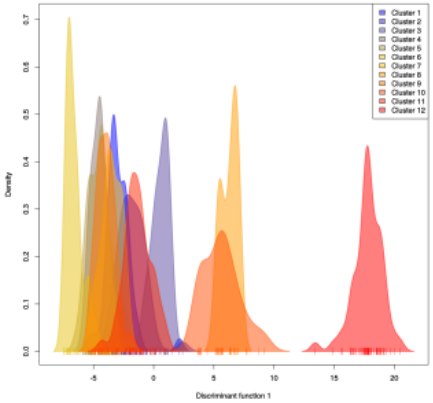
DAPC for K=3, extra information

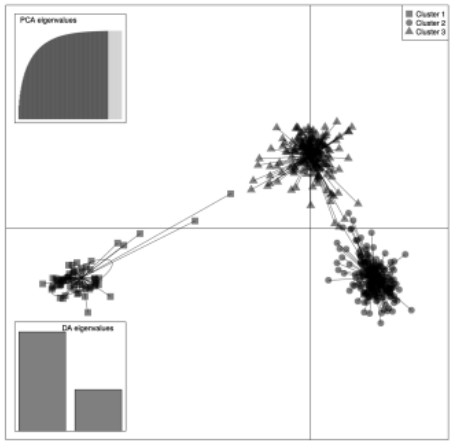


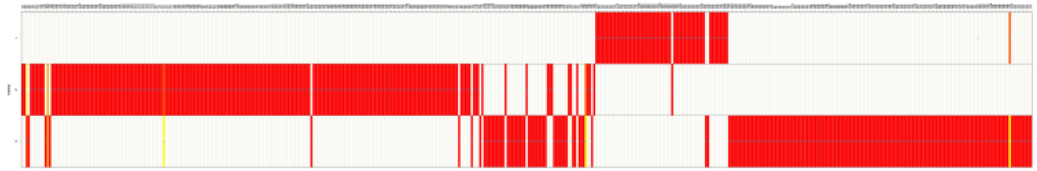
Another DAPC example

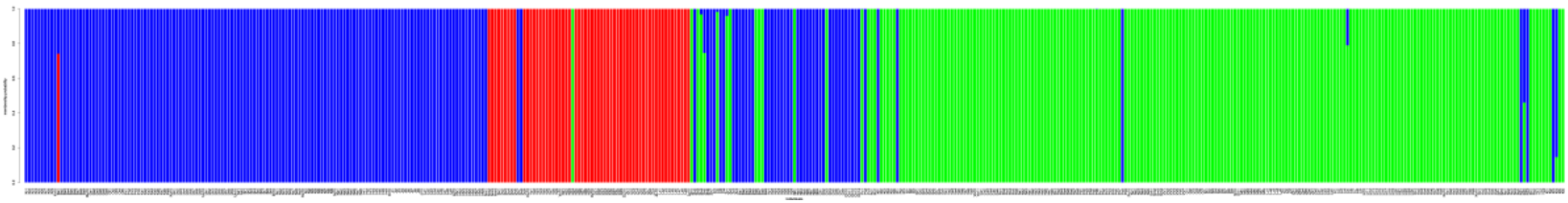












Some functions to work with huge SNP data sets

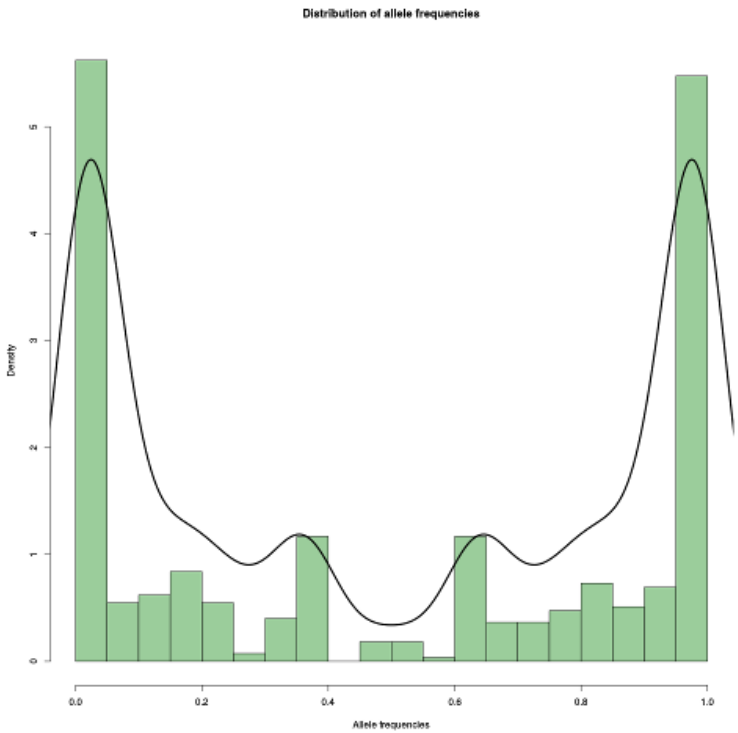
```
# Plot of missing data (white) and number of 2nd alleles
glPlot(x=usflu.genlight, legend=TRUE, posi="topleft")
# Sum of the number of second allele in each SNP
usflu.freq <- glSum(usflu.genlight)
# Plot distribution of (second) allele frequencies
hist(x=usflu.freq, proba=TRUE, col="gold", xlab="Allele
  frequencies", main="Distribution of (second) allele frequencies")
lines(x=density(usflu.freq)$x, y=density(usflu.freq)$y*1.5,
  col="red", lwd=3 )
# Mean number of second allele in each SNP
usflu.mean <- glMean(usflu.genlight)
usflu.mean <- c(usflu.mean, 1-usflu.mean)
# Plot distribution of allele frequencies
hist(x=usflu.mean, proba=TRUE, col="darkseagreen3",
  xlab="Allele frequencies", main="Distribution of allele
  frequencies", nclass=20)
lines(x=density(usflu.mean, bw=0.05)$x, y=density(usflu.mean,
  bw=0.05)$y*2, lwd=3)
```

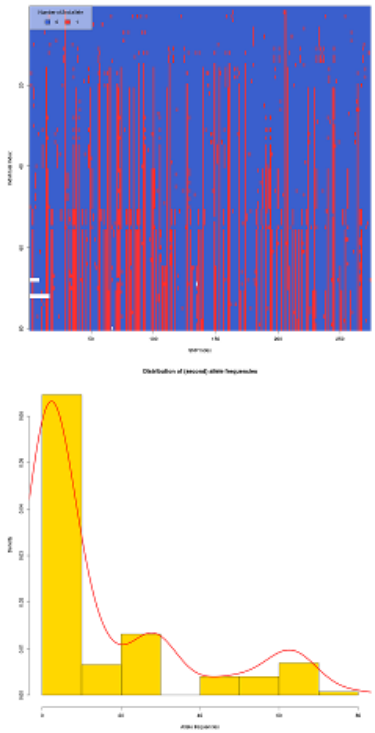
Number of missing values in each locus

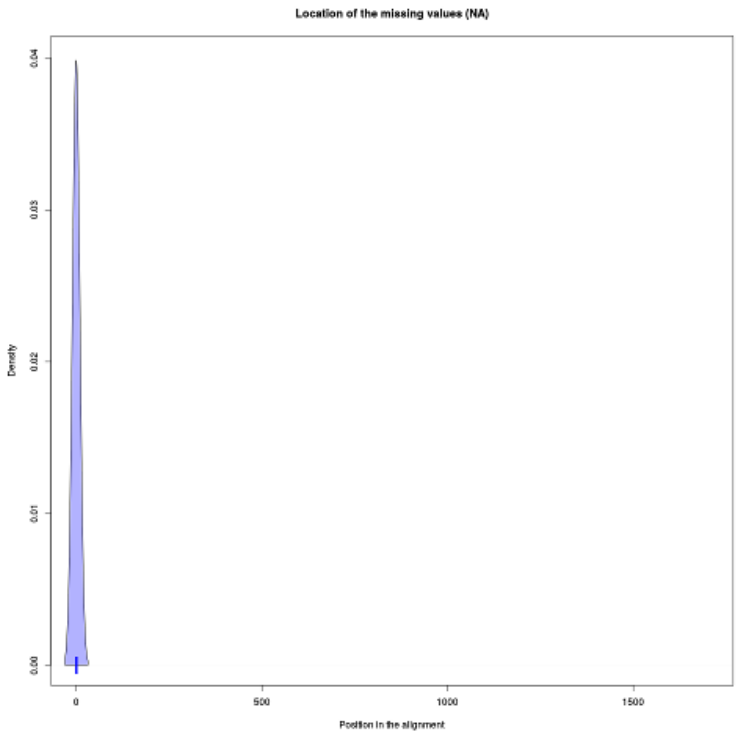
```
# Play with bw parameter to get optimal image
usflu.na.density <- density(glNA(usflu.genlight), bw=10)
# Set range of xlim parameter from 0 to the length
# of original alignment
plot(x=usflu.na.density, type="n", xlab="Position in the alignment",
  main="Location of the missing values (NA)", xlim=c(0, 1701))
polygon(c(usflu.na.density$x, rev(usflu.na.density$x)),
  c(usflu.na.density$y, rep(0, length(usflu.na.density$x))),
  col=transp("blue", alpha=0.3))
points(glNA(usflu.genlight), rep(0, nLoc(usflu.genlight)),
  pch="|", cex=2, col="blue")
```

Those tools are designed mainly for situation when having multiple (nearly) complete genomes. It is not needed for smaller (normal) datasets. Lets keep hoping in fast development of computers...

Basic information about SNP: distribution of 2^{nd} allele frequencies, missing data and number of 2^{nd} allele, distribution of allele frequencies, and number of missing values in each locus



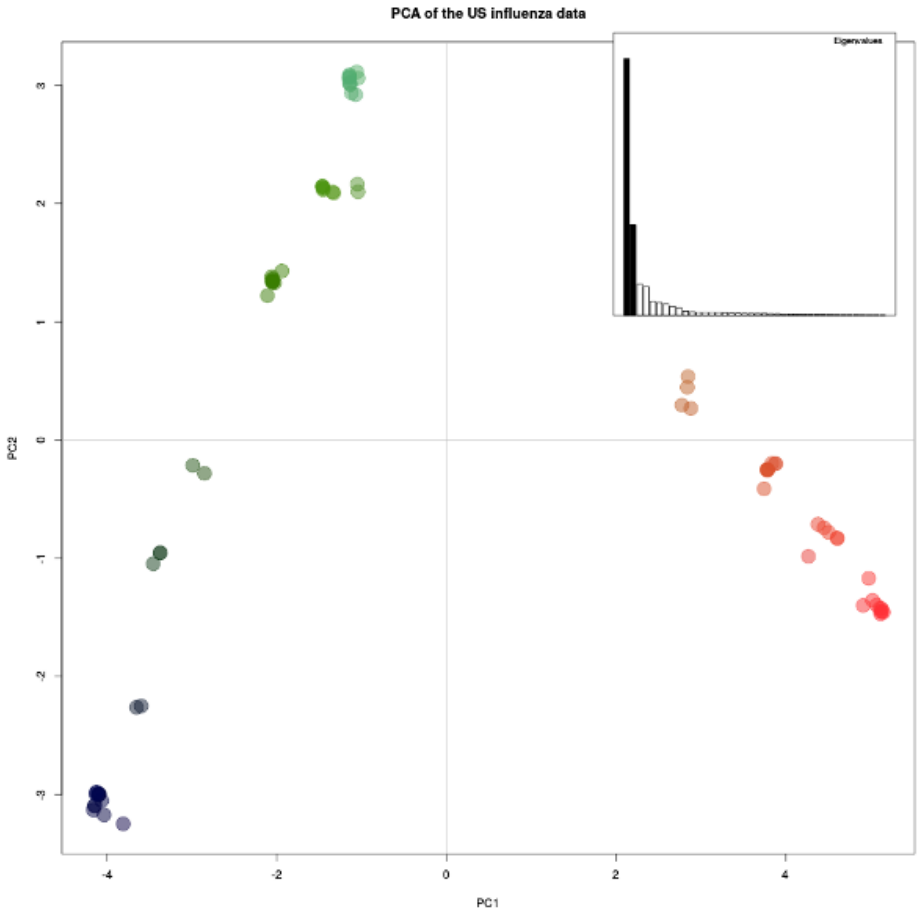


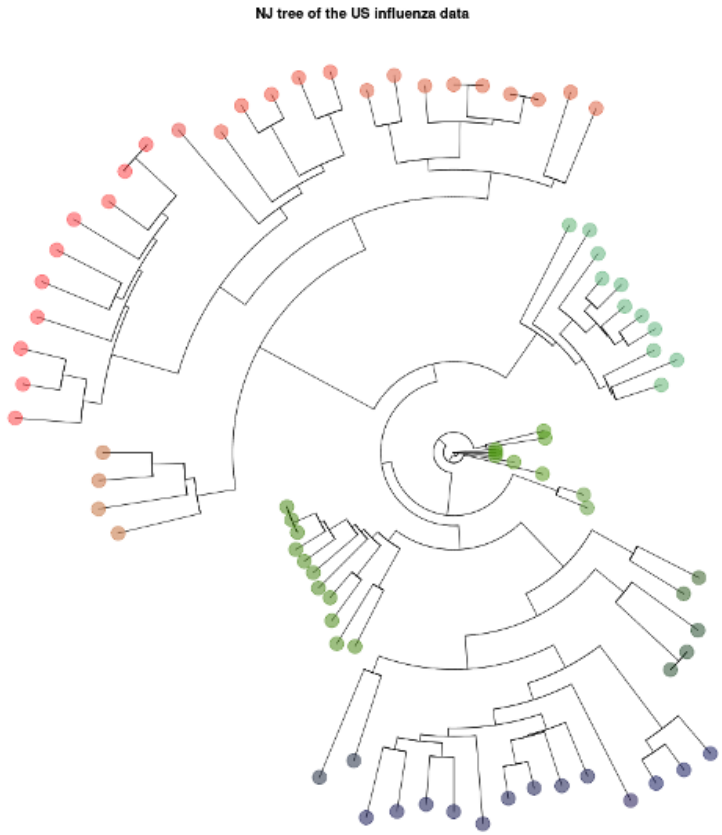


PCA, NJ and genlight objects

```
usflu.pca <- glPca(x=usflu.genlight, center=TRUE, scale=FALSE,
  loadings=TRUE) # Select number of retained PC axes, about 10 here
# Plot PCA
scatter.glPca(x=usflu.pca, posi="bottomright")
title("PCA of the US influenza data")
# Coloured plot
colorplot(usflu.pca$scores, usflu.pca$scores, transp=TRUE, cex=4)
title("PCA of the US influenza data")
abline(h=0, v=0, col="grey")
add.scatter.eig(usflu.pca[["eig"]][1:40], 2, 1, 2, posi="topright",
  inset=0.05, ratio=0.3)
# Calculate phylogenetic tree
usflu.tree.genlight <- nj(dist(as.matrix(usflu.genlight)))
# Plot coloured phylogenetic tree
plot.phylo(x=usflu.tree.genlight, typ="fan", show.tip=FALSE)
tiplabels(pch=20, col=usflu.col, cex=4)
title("NJ tree of the US influenza data")
```

PCA, NJ and genlight objects





Short overview of spatial genetics (in R)

- Basic approaches
  - Moran's *I* — several implementations (here as basic autocorrelation index, in sPCA and in Monmonier's algorithm), generally it is autocorrelation coefficient with broader use
  - Mantel test — several implementations, popular, although recently criticised as biologically irrelevant, generally correlation of two matrices (here genetic and geographical)
  - Bayesian clustering using geographical information as a proxy and showing results in geographical context (here as implemented in Geneland)
  - There are unlimited possibilities with connections with GIS software — check specialised courses and literature

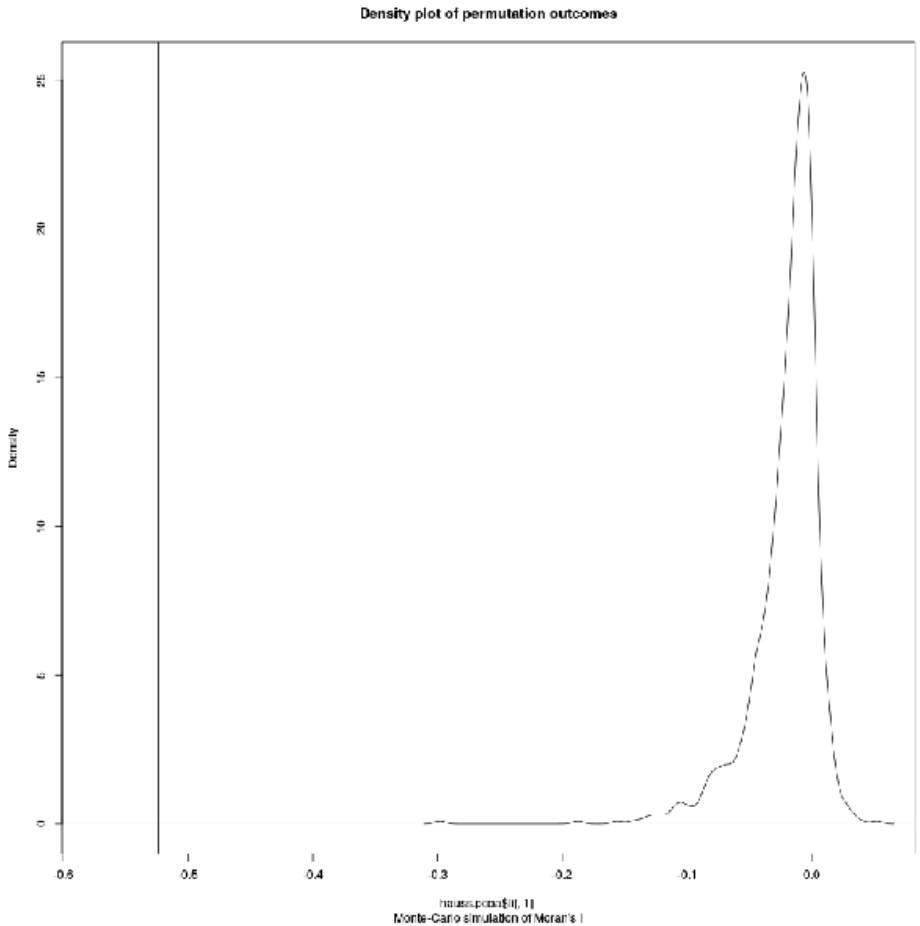
Calculation of Moran's *I* I

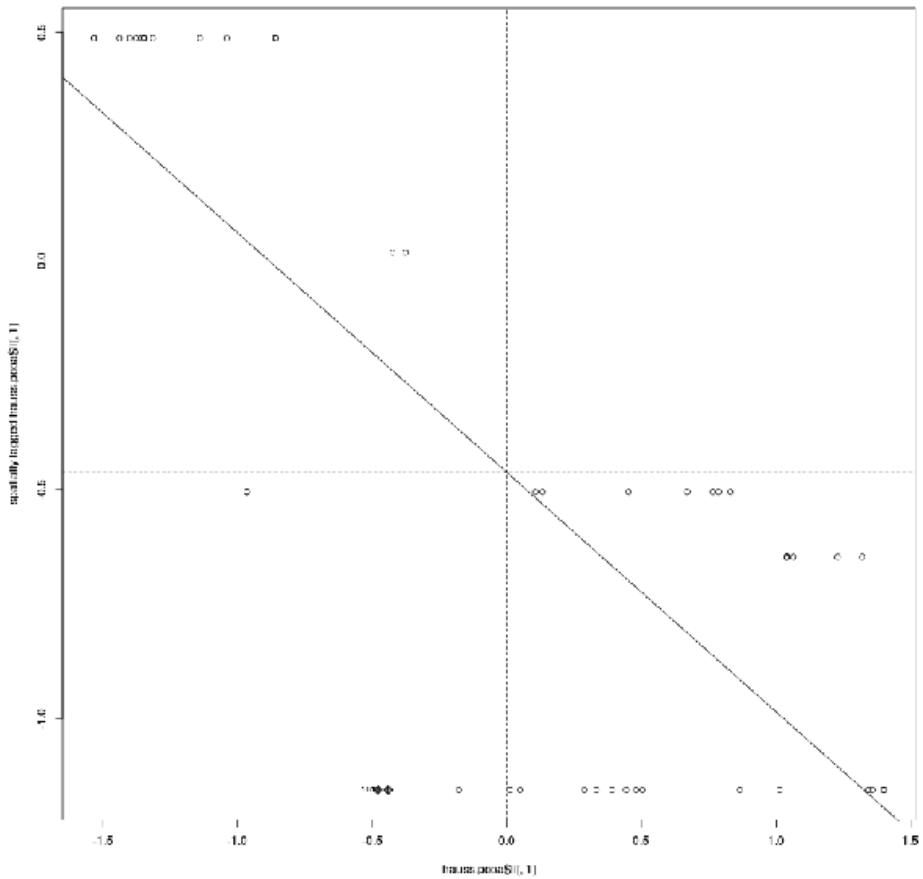
```
# Creates connection network
hauss.connectivity <- chooseCN(xy=hauss.genind$other$xy, type=5,
  d1=0, d2=1, plot.nb=TRUE, result.type="listw", edit.nb=FALSE)
hauss.connectivity
# Output:
Characteristics of weights list object:
Neighbour list object:
Number of regions: 47
Number of nonzero links: 1120
Percentage nonzero weights: 50.70167
Average number of links: 23.82979
Weights style: W
Weights constants summary:
   n   nn S0       S1       S2
W 47 2209 47 4.033607 207.037
###
```

Calculation of Moran's I II

```
# Test of Moran's I for 1st PCoA axis
# Results can be checked against permuted values of moran.mc()
moran.test(x=hauss.pcoa$li[,1], listw=hauss.connectivity,
  alternative="greater", randomisation=TRUE)
# Output:
Moran's I test under randomisation
data:  hauss.pcoa$li[, 1]
weights: hauss.connectivity
Moran I statistic standard deviate = -18.514, p-value = 1
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
    -0.5232003724     -0.0217391304      0.0007336276
hauss.pcoa1.mctest <- moran.mc(x=hauss.pcoa$li[,1],
  listw=hauss.connectivity, alternative="greater", nsim=1000)
hauss.pcoa1.mctest
plot(hauss.pcoa1.mctest) # Plot of densitiy of permutations
moran.plot(x=hauss.pcoa$li[,1], listw=hauss.connectivity) # PC plot
```

Moran's *I* for our 1^{st} axis isn't significant



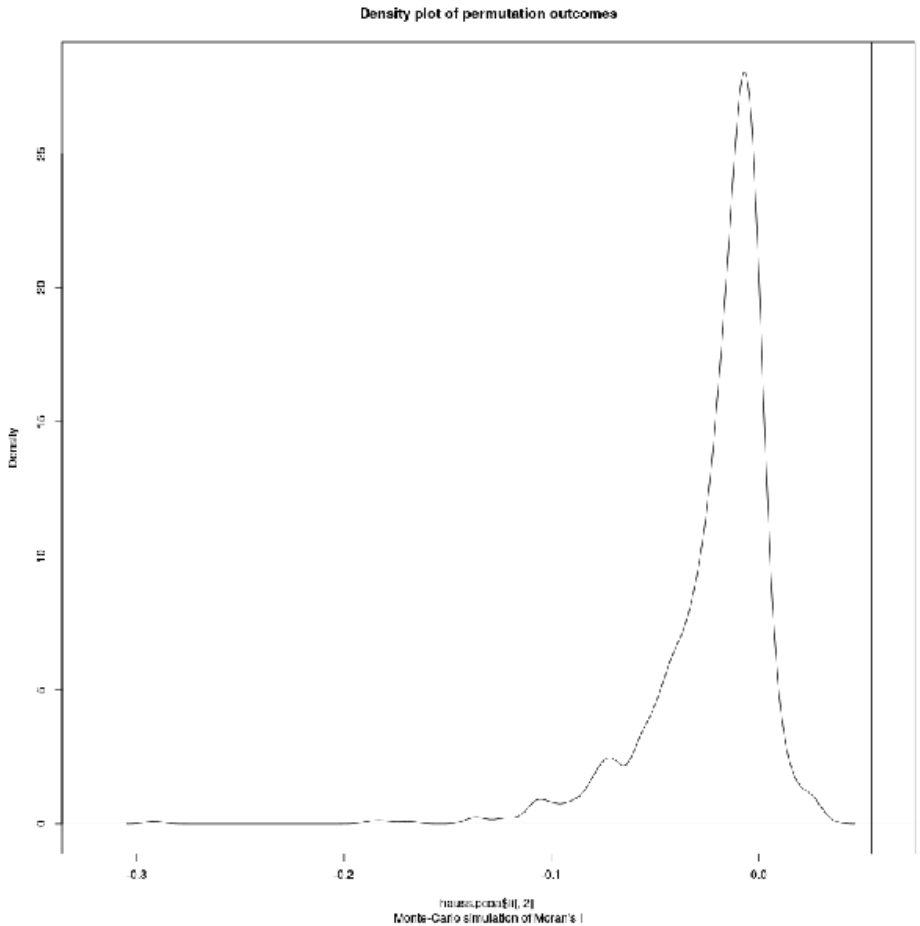


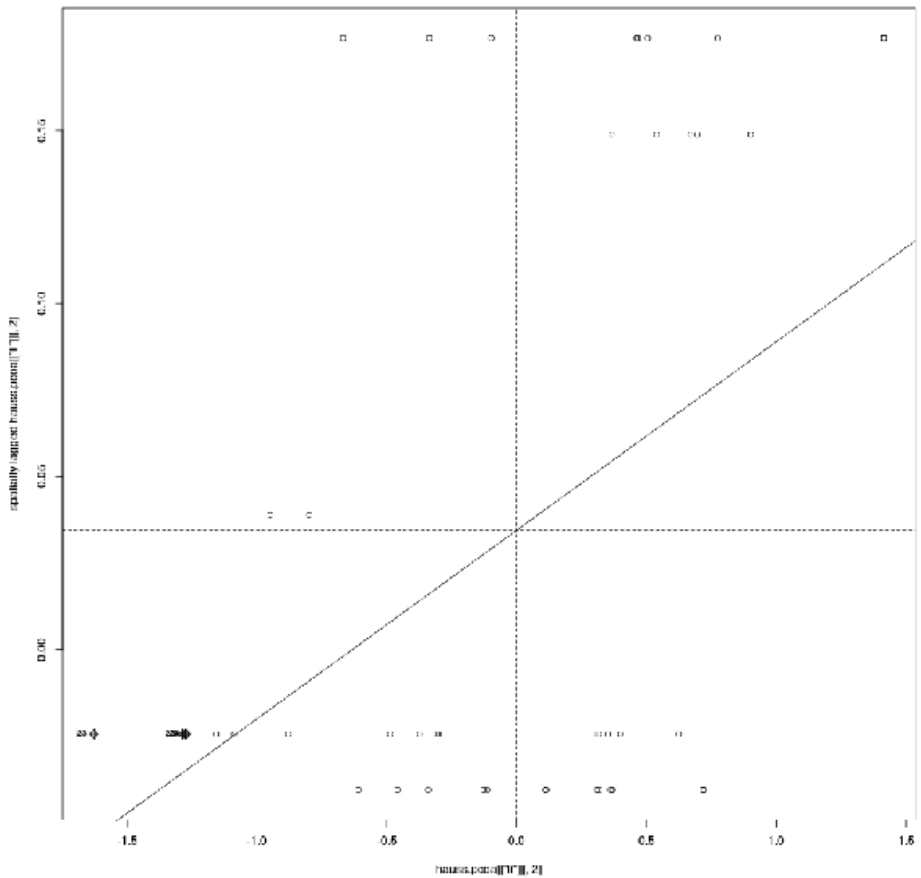
We tested hypothesis "greater" — there is **no** significant positive autocorrelation

Calculation of Moran's *I* (2^{nd} axis)

```
# Test of Moran's I for 2nd PCoA axis
moran.test(x=hauss.pcoa$li[,2], listw=hauss.connectivity,
  alternative="greater", randomisation=TRUE)
hauss.pcoa2.mctest <- moran.mc(x=hauss.pcoa$li[,2],
  listw=hauss.connectivity, alternative="greater", nsim=1000)
hauss.pcoa2.mctest
# Output
Monte-Carlo simulation of Moran's I
data:  hauss.pcoa$li[, 2]
weights: hauss.connectivity
number of simulations + 1: 1001
statistic = 0.0545, observed rank = 1001, p-value = 0.000999
alternative hypothesis: greater
# Plots
plot(hauss.pcoa2.mctest)
moran.plot(x=hauss.pcoa$[,2], listw=hauss.connectivity)
```

Second axis is surprisingly significant





We tested hypothesis "greater" — there **is** significant positive autocorrelation

Spatial Analysis of Principal Components (sPCA)

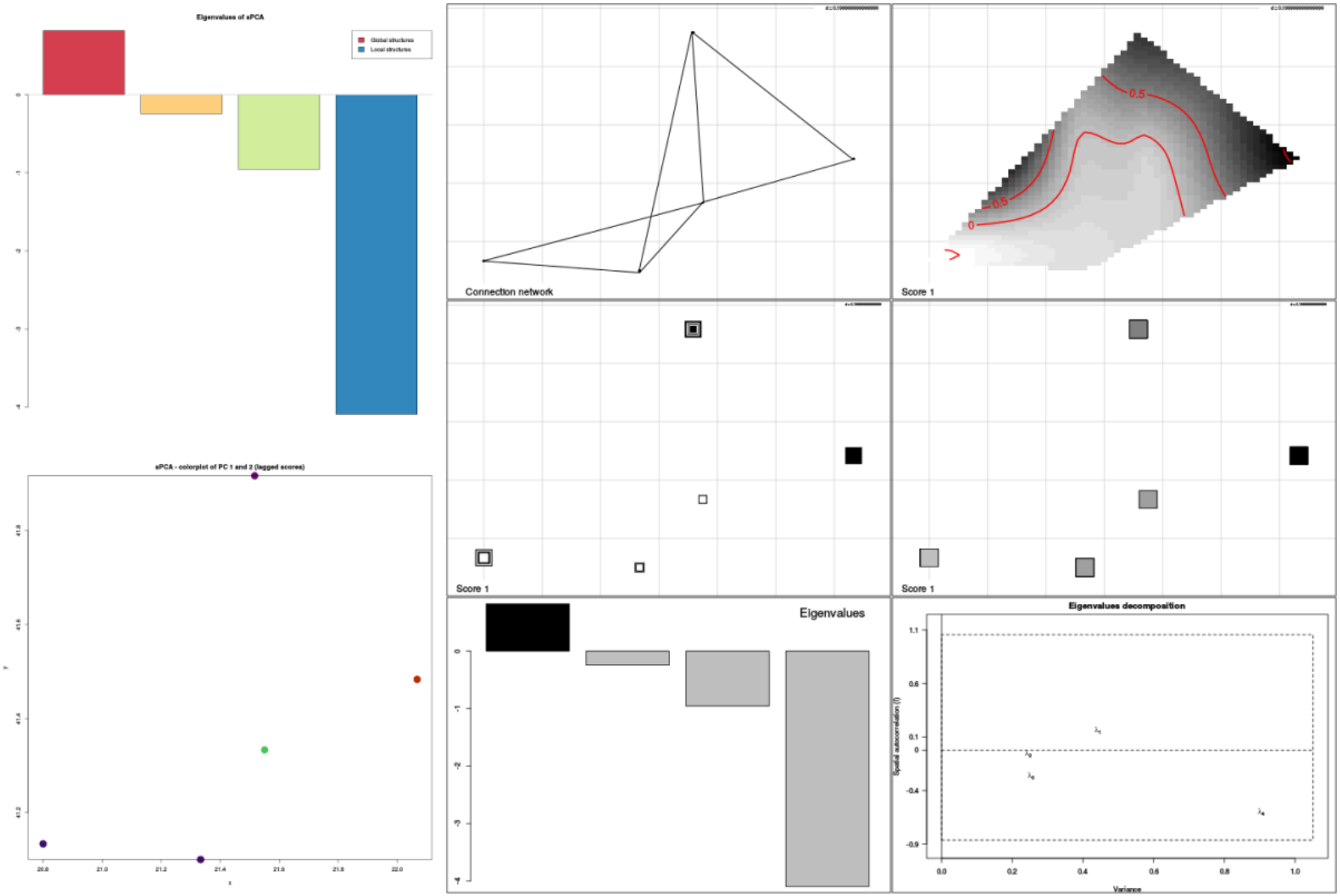
- Implemented in adegenet, see **adegenetTutorial("spca")**
- Analyses matrix of relative allele frequencies of genotypes/populations and spatial weighting matrix
- The geographical matrix is usually (as for Moran's *I*) created by **chooseCN()** — creates connectivity network among entities (genotypes/populations) — spatial coordinates are not directly used
- When using **chooseCN()**, look at the documentation and try various methods with changing settings to see differences

```
1 data(rupica) # Loads adegenet's training dataset
2 # Try various settings vor chooseCN (type=X) - type 1-4 as there
3 # are identical coordinates (multiple sampling from same locality)
4 chooseCN(xy=rupica$other$xy, ask=T, type=5/6/7, plot.nb=T, edit.nb=F)
5 # See ?chooseCN for more details
```

Calculations of sPCA

```
hauss.spca <- spca(obj=hauss.genind, cn=hauss.connectivity,
  scale=TRUE, scannf=TRUE)
# Plot eigenvalues of sPCA - gobal vs. local structure
barplot(height=hauss.spca$eig, main="Eigenvalues of sPCA",
  col=spectral(length(hauss.spca$eig)))
legend("topright", fill=spectral(2), leg=c("Global structures",
  "Local structures")) # Add legend
abline(h=0,col="grey") # Add line showing zero
print.spca(hauss.spca) # Information about sPCA
summary.spca(hauss.spca) # Summary of sPCA results
# Shows connectivity network, 3 different scores
# barplot of eigenvalues and eigenvalues decomposition
plot.spca(hauss.spca)
colorplot.spca(hauss.spca, cex=3) # Display of scores in color canals
title("sPCA - colorplot of PC 1 and 2 (lagged scores)", line=1, cex=1.5)
# Spatial and variance components of the eigenvalues
screeplot.spca(x=hauss.spca, main=NULL)
```

sPCA outputs

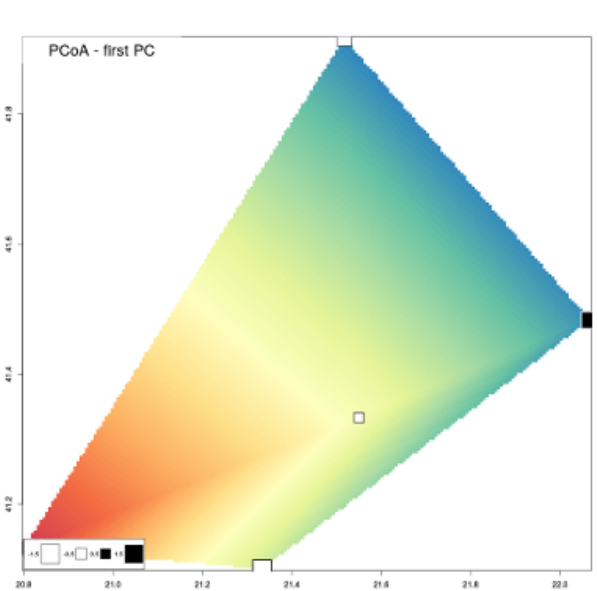


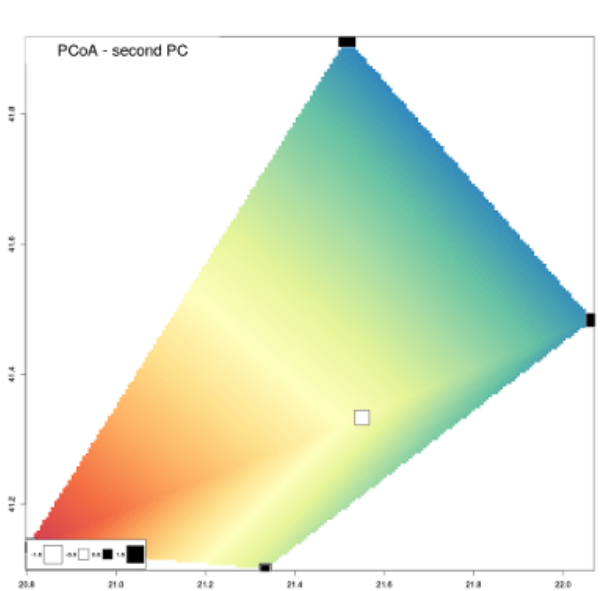
Map of genetic clines

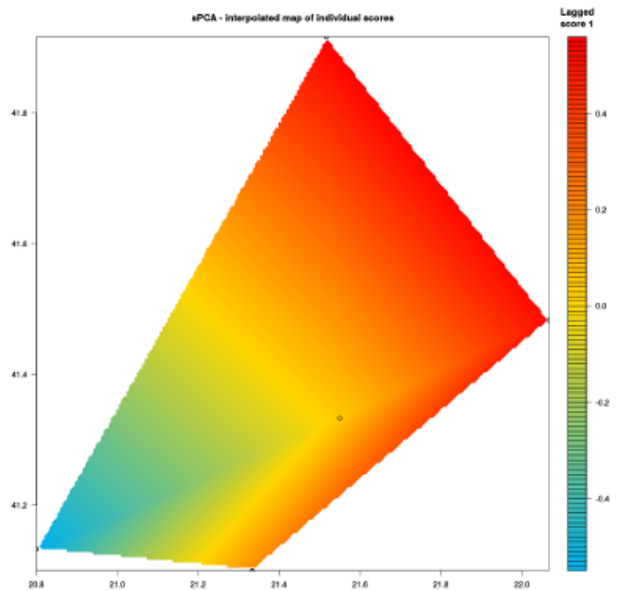
```
library(akima) # This library is needed for manipulation with coordinates
hauss.spca.temp <- interp(other(hauss.genind)$xy[,1], other(
  hauss.genind)$xy[,2], hauss.spca$ls[,1], xo=seq(min(other(
  hauss.genind)$xy[,1]), max(other(hauss.genind)$xy[,1]), le=200),
  yo=seq(min(other(hauss.genind)$xy[,2]), max(other(hauss.genind)$xy[,2]),
  le=200), duplicate="median")
# For 1st axis
image(x=hauss.spca.temp, col=spectral(100))
s.value(dfxy=hauss.genind$other$xy, z=hauss.pcoa$li[,1], add.p=TRUE,
  csize=0.5, sub="PCoA - first PC", csub=2, possub="topleft")
# For 2nd axis
image(x=hauss.spca.temp, col=spectral(100))
s.value(dfxy=hauss.genind$other$xy, z=hauss.pcoa[["li"]][,2], add.p=TRUE,
  csize=0.5, sub="PCoA - second PC", csub=2, possub="topleft")
```

sPCA outputs

```
# Interpolated lagged score on a map
hauss.spca.annot <- function() {
  title("sPCA - interpolated map of individual scores")
  points(other(hauss.genind)$xy[,1], other(hauss.genind)$xy[,2])
  }
filled.contour(hauss.spca.temp, color.pal=colorRampPalette(
  lightseasun(100)), pch=20, nlevels=100, key.title=title("Lagged\n
  score 1"), plot.title=hauss.spca.annot())
```

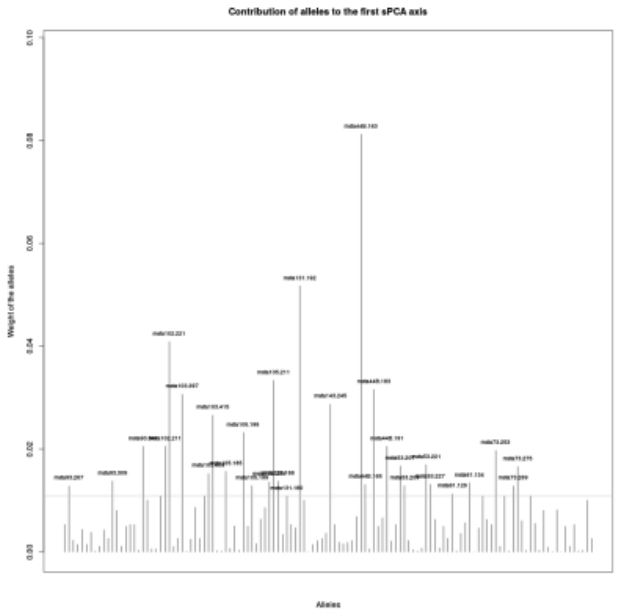


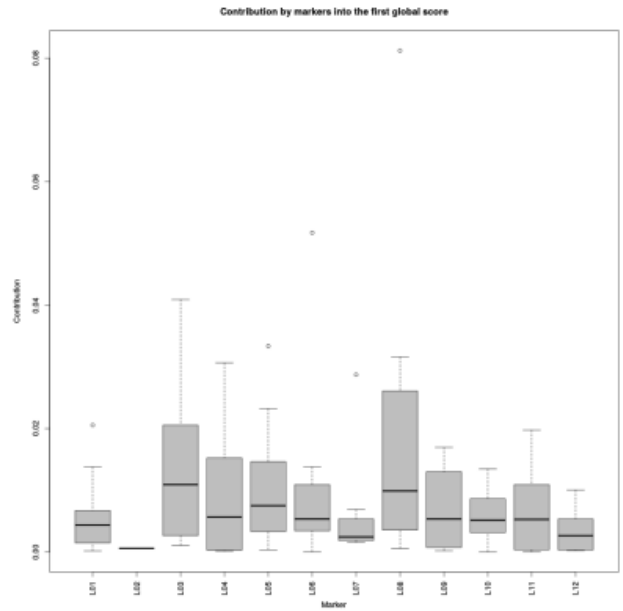




Loading plots - which alleles contribute the most?

```
hauss.spca.loadings <- hauss.spca[["c1"]][,1]^2
names(hauss.spca.loadings) <- rownames(hauss.spca$c1)
loadingplot(x=hauss.spca.loadings, xlab="Alleles", ylab="Weight of the
  alleles", main="Contribution of alleles to the first sPCA axis")
boxplot(formula=hauss.spca.loadings~hauss.genind$loc.fac, las=3,
  ylab="Contribution", xlab="Marker", main="Contribution by markers
  into the first global score", col="grey")
```





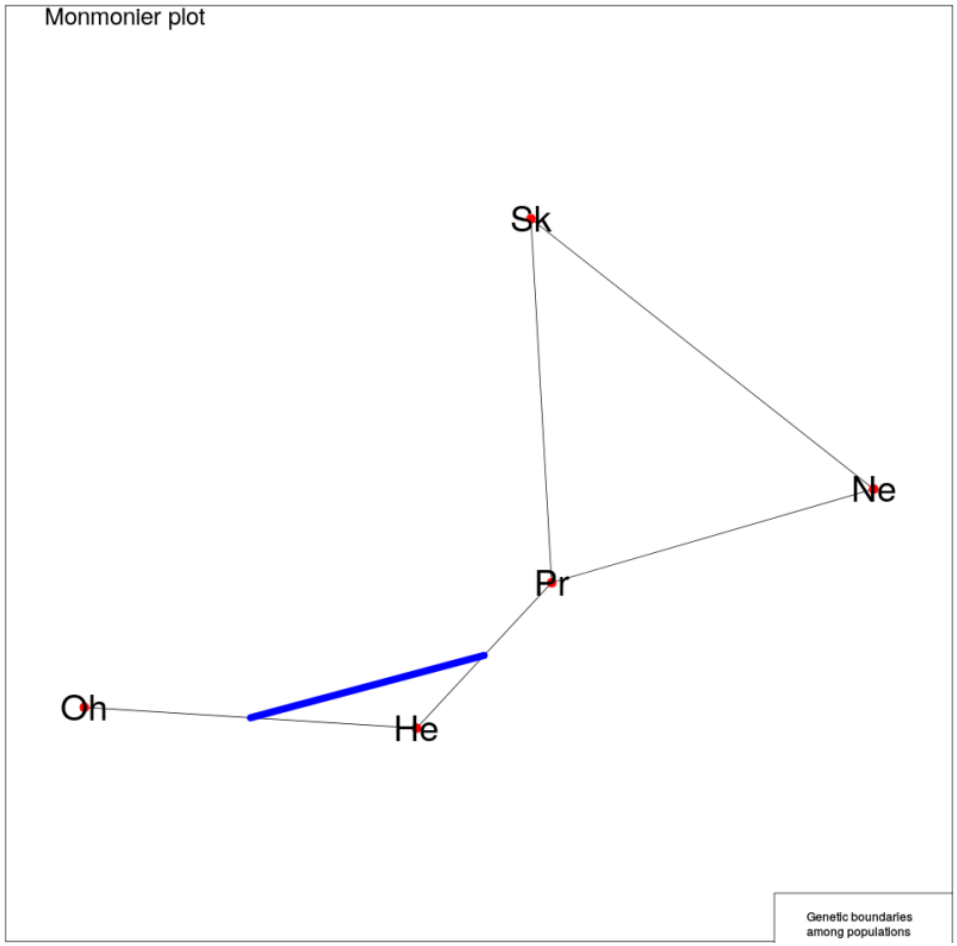
Monmonier's algorithm — genetic boundaries

- Finds boundaries of maximum differences between contiguous polygons of a tesselation
- Detects genetic boundaries among georeferecend genotypes (or populations)
- For more information see **adegenetTutorial("basics")**
- Requires every point to have unique coordinates (use **jitter()** when having same coordinates from one locality)
- Example here is on population level, which is not ideal for all cases

```
1 # Calculates Monmonier's function (for threshold use 'd')
2 hauss.monmonier <- monmonier(xy=hauss.genpop$other$xy, dist=
3   dist(hauss.genpop@tab), cn=chooseCN(hauss.genpop$other$xy,
4   ask=FALSE, type=2, plot.nb=FALSE, edit.nb=FALSE), nrun=1)
5 coords.monmonier(hauss.monmonier) # See result as text
```

Plot genetic boundaries

```
plot.monmonier(hauss.monmonier,
  add.arrows=FALSE, bwd=10,
  sub="Monmonier plot", csub=2)
points(hauss.genpop$other$xy,
  cex=2.5, pch=20, col="red")
text(x=hauss.genpop$other$xy$lon,
  y=hauss.genpop$other$xy$lat,
  labels=hauss.genpop@pop.names,
  cex=3)
legend("bottomright",
  leg="Genetic boundaries\n
  among populations")
# See ?points, ?text and ?legend
```



Monmonier notes

- Sometimes it is needed to get rid of random noise in data. To do so use as parameter **dist** of **monmonier()** table from PCA (**pcaObject$li**):

```
monmonier(..., dist=dudi.pco(d=dist(x=GenindObject$tab),
  scannf=FALSE, nf=1)$li, ...)
```

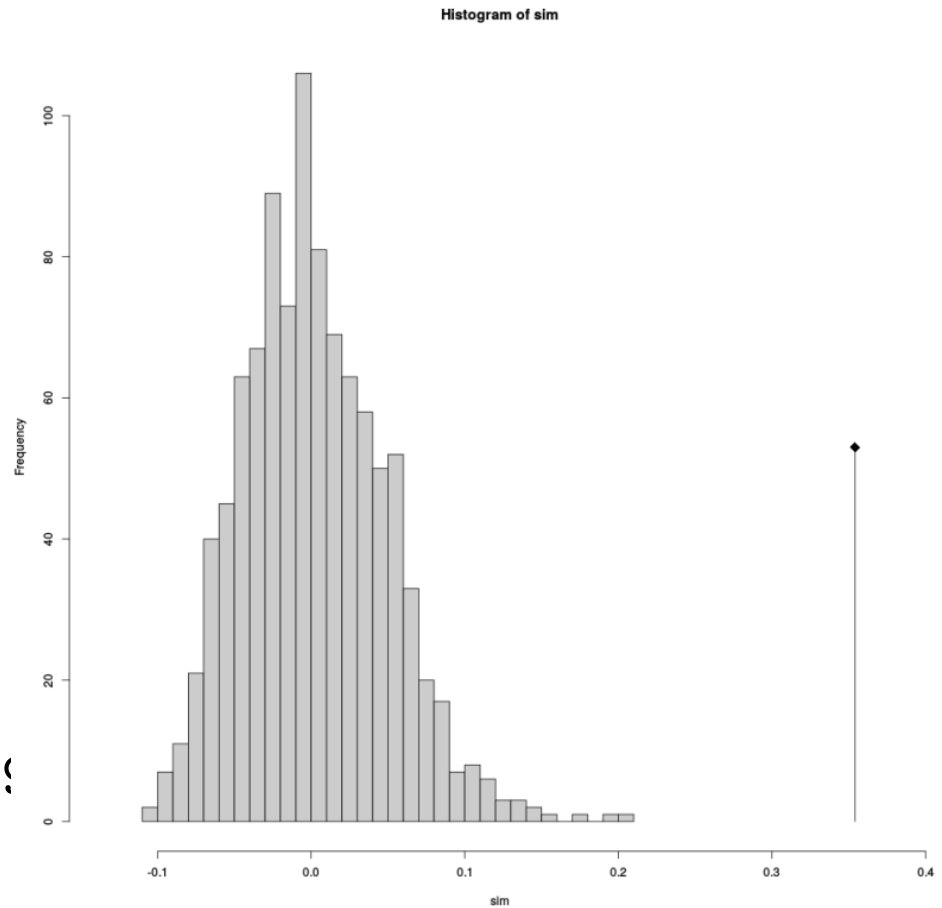
- Generally (when dataset is bigged and more diverse) it is recommended to run it several times (parameter **nrun**) — there will be several iterations
- See **?plot.monmonier** for various graphical parameters to customize the plot
- Use **points()** to add for example coloured symbols of samples and/or **text()** to add text labels

Mantel test — isolation by distance

```
# Geographical distance
hauss.gdist <- dist(x=hauss.genind$other$xy, method="euclidean",
  diag=TRUE, upper=TRUE)
# Mantel test
hauss.mantel <- mantel.randtest(m1=hauss.dist, m2=hauss.gdist,
  nrepet=1000)
hauss.mantel # See text output
plot(hauss.mantel, nclass=30)
# Libraries required by mantel.correlog:
library(permute)
library(lattice)
library(vegan)
# Different implementation of Mantel test testing distance classes
hauss.mantel.cor <- mantel.correlog(D.eco=hauss.dist,D.geo=hauss.gdist,
  XY=NULL, n.class=0, break.pts=NULL, cutoff=FALSE, r.type="pearson",
  nperm=1000, mult="holm", progressive=TRUE)
# See results for respective classes
hauss.mantel.cor
summary(hauss.mantel.cor)
```

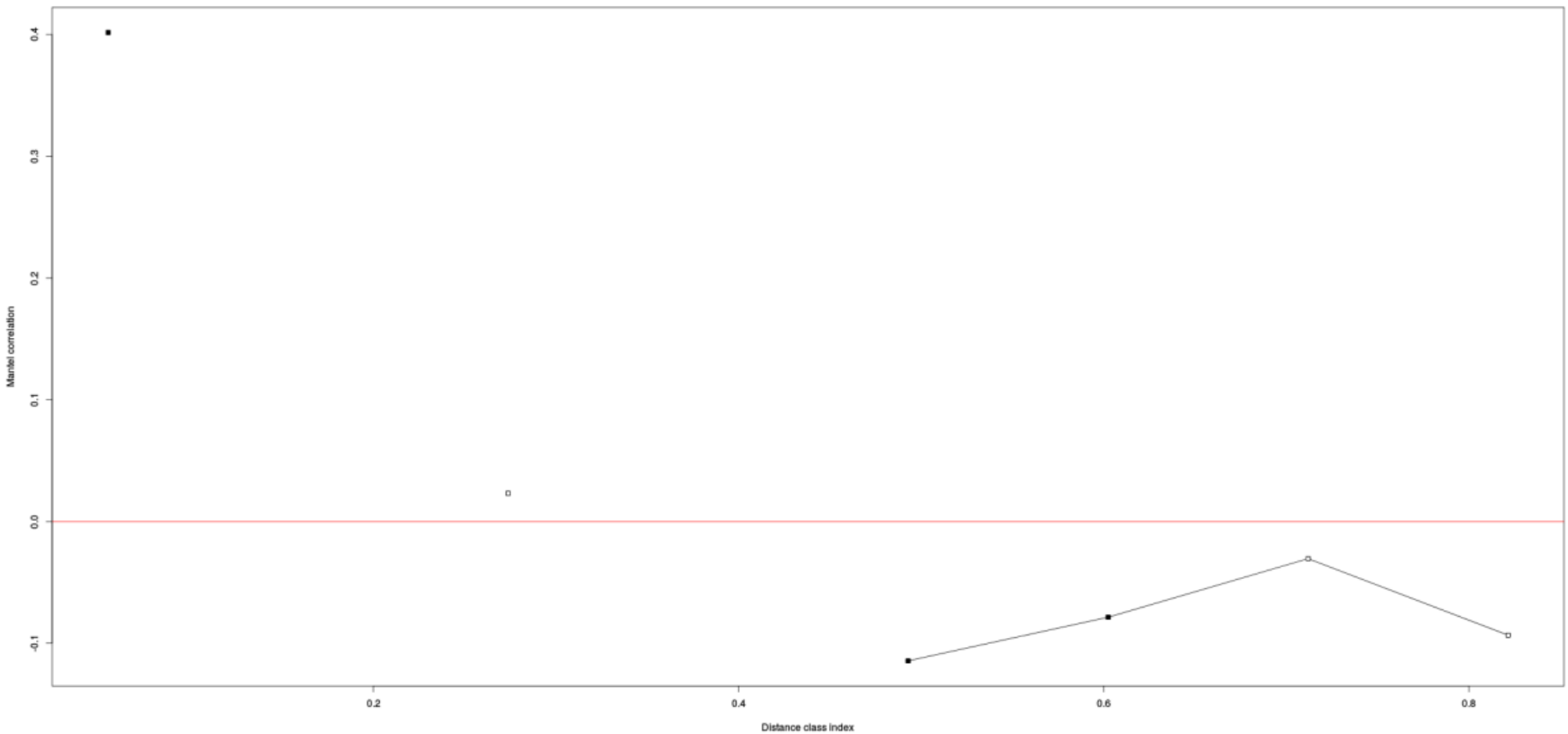
Mantel test outputs — strongly significant

```
> hauss.mantel
Monte-Carlo test
Call: mantel.randtest(m1 =
  hauss.dist, m2 =
  hauss.gdist, nrepet = 1000)
Observation: 0.35409
Based on 1000 replicates
Simulated p-value: 0.000999001
Alternative hypothesis: greater
  Std.Obs Expectation  Variance
7.61967545 0.001687140 0.002138
```




```
1 # Plot correlogram (next slide)
2 plot(hauss.mantel.cor)
```

Plot of Mantel Correlogram Analysis



About Geneland

- Works with haploid and diploid codominant markers (microsattelites or SNPs)
- Spatially explicit Bayesian clustering
- Produces maps of distribution of inferred genetic clusters
- Relative complicated tool with various modelling options

```
# Load needed libraries
library(PBSmapping)
library(RandomFields)
library(fields)
library(spam)
library(grid)
library(maps)
library(tcltk)
library(Geneland)
# Graphical interface is available
# we will use only command line
Geneland.GUI()
```

For more information see http://www2.imm.dtu.dk/~gigu/Geneland/

Loading and conversions of coordinates

```
# Geneland requires specific coordinate space
# hauss.cord is DF, we need just plain matrix
hauss.geneland.coord <- as.matrix(hauss.coord)
colnames(hauss.geneland.coord) <- c("X", "Y")
attr(hauss.geneland.coord, "projection") <- "LL"
attr(hauss.geneland.coord, "zone") <- NA
hauss.geneland.coord.utm <- convUL(hauss.geneland.coord)
dim(hauss.geneland.coord)
hauss.geneland.coord
dim(hauss.geneland.coord.utm)
hauss.geneland.coord.utm # Final coordinates
# Load data (only haploid or diploid data are supported)
# only plain table with alleles
hauss.geneland.data <- read.table(file=
  "http://trapa.cz/sites/default/rcourse/haussknechtii_geneland.txt",
  na.string="-999", header=FALSE, sep="\t")
dim(hauss.geneland.data)
hauss.geneland.data
```

Settings and running MCMC

```
hauss.geneland.nrun <- 5 # Set number of independent runs
hauss.geneland.burnin <- 100 # Set length of burnin chain
hauss.geneland.maxpop <- 10 # Set maximal K (number of populations)
# FOR loop will run several independent runs and produce output maps
# of genetic clusters - outputs are written into subdirectory within
# geneland directory (this has to exist prior launching analysis)
for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
  hauss.geneland.path.mcmc <- paste("geneland/", hauss.geneland.irun,
    "/", sep="") # paste is good especially for joining several texts
  # On Windows, remove following line and create subdirectories from
  # 1 to max K manually (creating subdirs in Windows is complicated)
  system(paste("mkdir ", hauss.geneland.path.mcmc)) # Creates subdirs
  # Inferrence - MCMC chain - see ?MCMC for details
  MCMC(coordinates=hauss.geneland.coord.utm, geno.dip.codom=
    hauss.geneland.data, path.mcmc=hauss.geneland.path.mcmc,
    delta.coord=0.001, varnpop=TRUE, npopmin=1, npopmax=
    hauss.geneland.maxpop, nit=10000, thinning=10,
    freq.model="Uncorrelated", spatial=TRUE)
  # For loop continues on next slide
```

Running MCMC

```
# Start of FOR loop is on previous page
# Post-process chains
PostProcessChain(coordinates=hauss.geneland.coord.utm,
  path.mcmc=hauss.geneland.path.mcmc, nxdom=500, nydom=500,
  burnin=hauss.geneland.burnin)
# Output
# Simulated number of populations
Plotnpop(path.mcmc=hauss.geneland.path.mcmc, printit=TRUE,
  file=paste(hauss.geneland.path.mcmc, "/geneland-number_of_clusters
  .pdf", sep=""), format="pdf", burnin=hauss.geneland.burnin)
dev.off() # We must close graphical device manually
# Map of estimated population membership
PosteriorMode(coordinates=hauss.geneland.coord.utm,
  path.mcmc=hauss.geneland.path.mcmc, printit=TRUE, format="pdf",
  file=paste(hauss.geneland.path.mcmc,"/geneland-map.pdf", sep=""))
dev.off() # We must close graphical device manually
} # End of FOR loop from previous slide
```

Estimate F_{ST}

```
# Prepare list to record values of Fst for all runs
hauss.geneland.fstat <- list()
# Estimate Fst
for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
  hauss.geneland.path.mcmc <- paste("geneland/", hauss.geneland.irun,
    "/", sep="")
  # F-statistics - Fis and Fst
  hauss.geneland.fstat[[hauss.geneland.irun]] <- Fstat.output(
    coordinates=hauss.geneland.coord.utm, genotypes=hauss.geneland.data,
    burnin=hauss.geneland.burnin, ploidy=2,
    path.mcmc=hauss.geneland.path.mcmc)
  }
  # Print Fst output
  hauss.geneland.fstat
```

MCMC inference under the admixture model

```
for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
  hauss.geneland.path.mcmc <- paste("geneland/",
    hauss.geneland.irun, "/", sep="")
  hauss.geneland.path.mcmc.adm <- paste(hauss.geneland.path.mcmc,
    "admixture", "/", sep="")
  # On Windows, remove following line of code and create in each
  # result directory (from 1 to max K) new subdirectory "admixture"
  # (creating subdirs in Windows is complicated)
  system(paste("mkdir ", hauss.geneland.path.mcmc.adm))
  HZ(coordinates=hauss.geneland.coord.utm, geno.dip.codom=
    hauss.geneland.data, path.mcmc.noadm=hauss.geneland.path.mcmc,
    nit=10000, thinning=10, path.mcmc.adm=hauss.geneland.path.mcmc.adm)
  }
```

Currently, there is no much use for admixture results, at lest not without extra work...

Produce maps of respective inferred clusters

```
for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
  hauss.geneland.path.mcmc <- paste("geneland/",
    hauss.geneland.irun, "/", sep="")
  # Maps - tesselations
  PlotTessellation(coordinates=hauss.geneland.coord.utm,
    path.mcmc=hauss.geneland.path.mcmc, printit=TRUE,
    path=hauss.geneland.path.mcmc)
  for (hauss.geneland.irun.img in 1:hauss.geneland.maxpop) {
    dev.off() } # We must close graphical device manually
  }
```

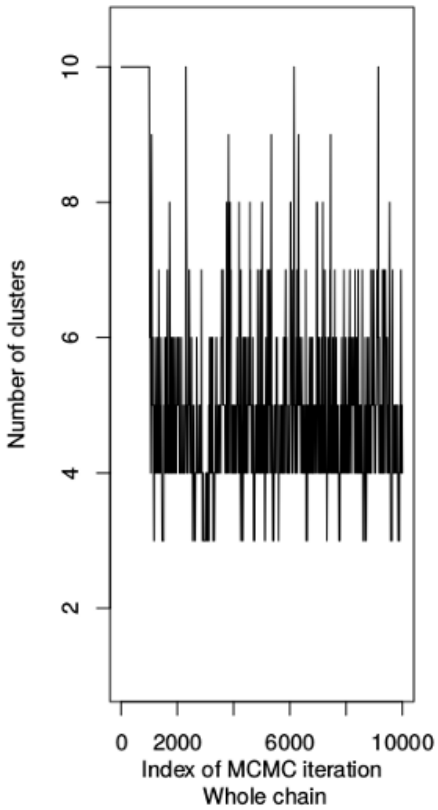
Maps are produced as PS (PostScript) files. Not every graphical software can handle them (try for example GIMP). There are as many plots as was maximal K, but only those up to inferred number of clusters have some content.

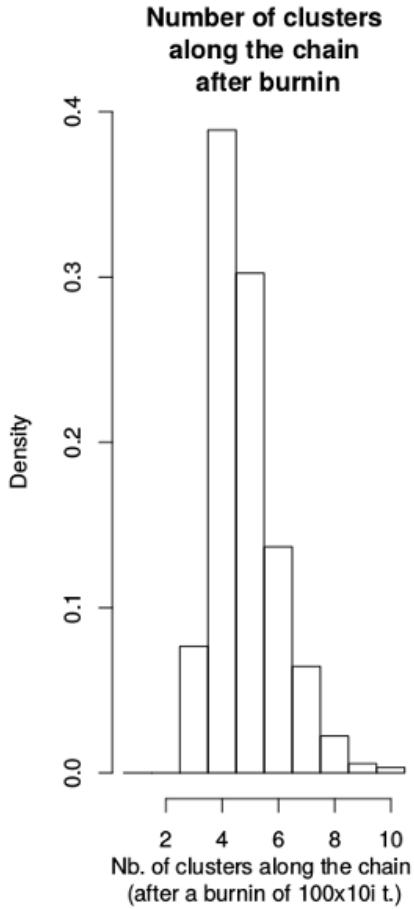
Determine which cluster is the best

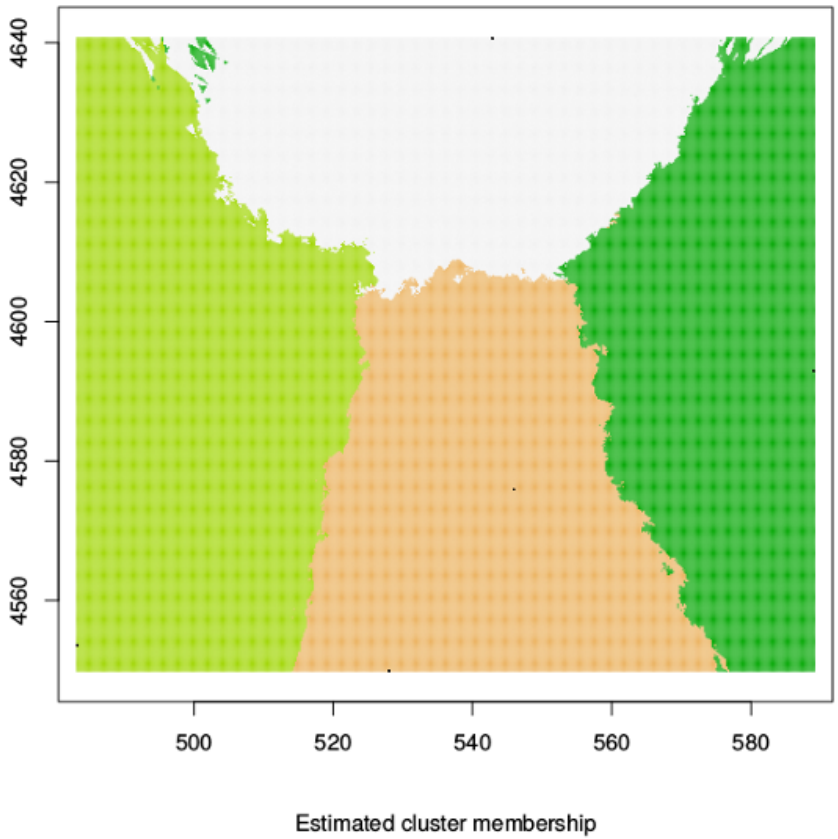
```
# Calculate average posterior probability
hauss.geneland.lpd <- rep(NA, hauss.geneland.nrun)
for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
  hauss.geneland.path.mcmc <- paste("geneland/",
    hauss.geneland.irun, "/", sep="")
  hauss.geneland.path.lpd <- paste(hauss.geneland.path.mcmc,
    "log.posterior.density.txt", sep="")
  hauss.geneland.lpd[hauss.geneland.irun] <-
    mean(scan(hauss.geneland.path.lpd)[-(1:hauss.geneland.burnin)])
  }
# Sorts runs according to decreasing posterior probability
# the first one is the best
order(hauss.geneland.lpd, decreasing=TRUE)
hauss.geneland.lpd
```

We will use figures and F_{ST} outputs only from the best run.

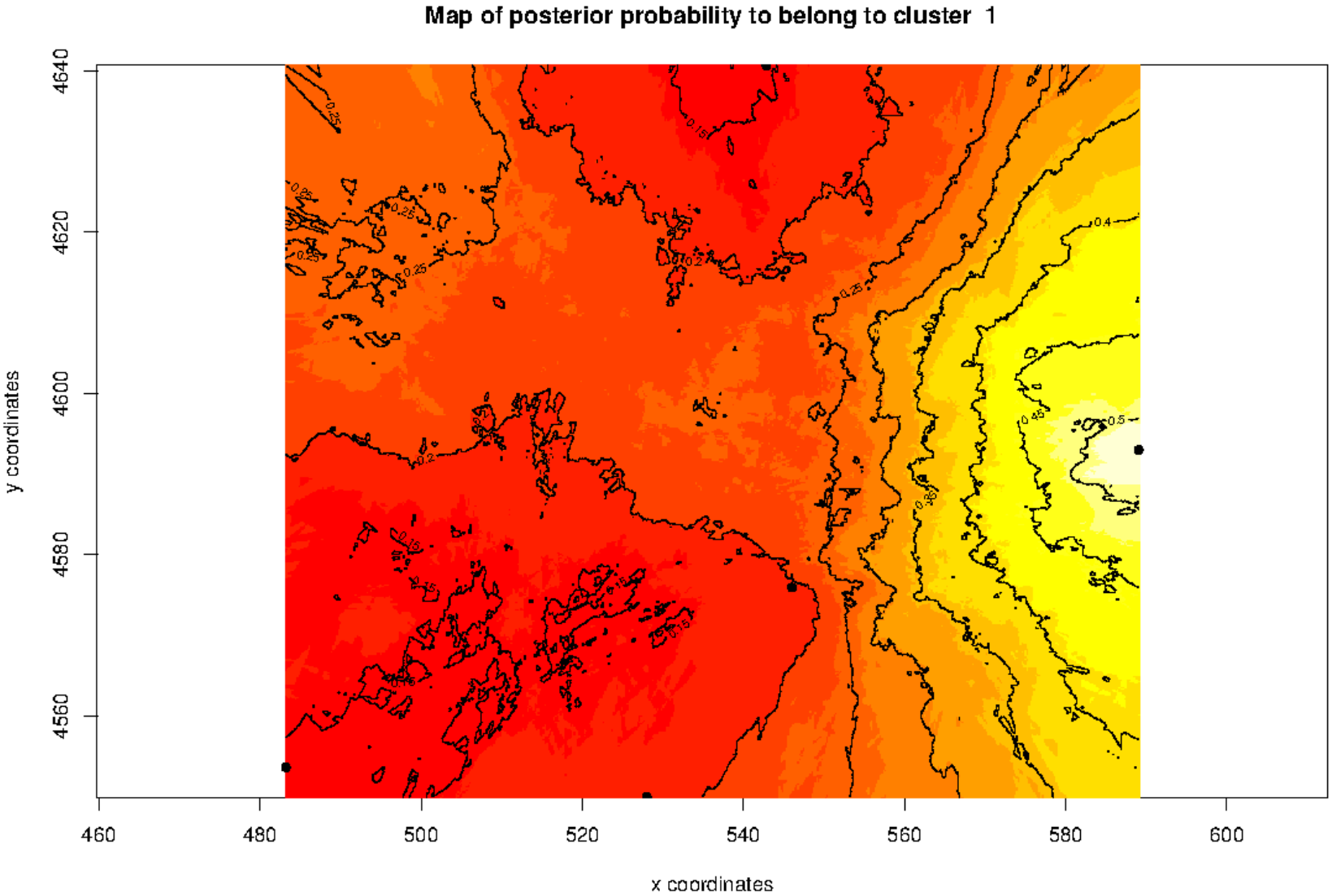
MCMC chain, number of clusters and their map







Map of posterior probability of belonging into cluster 1

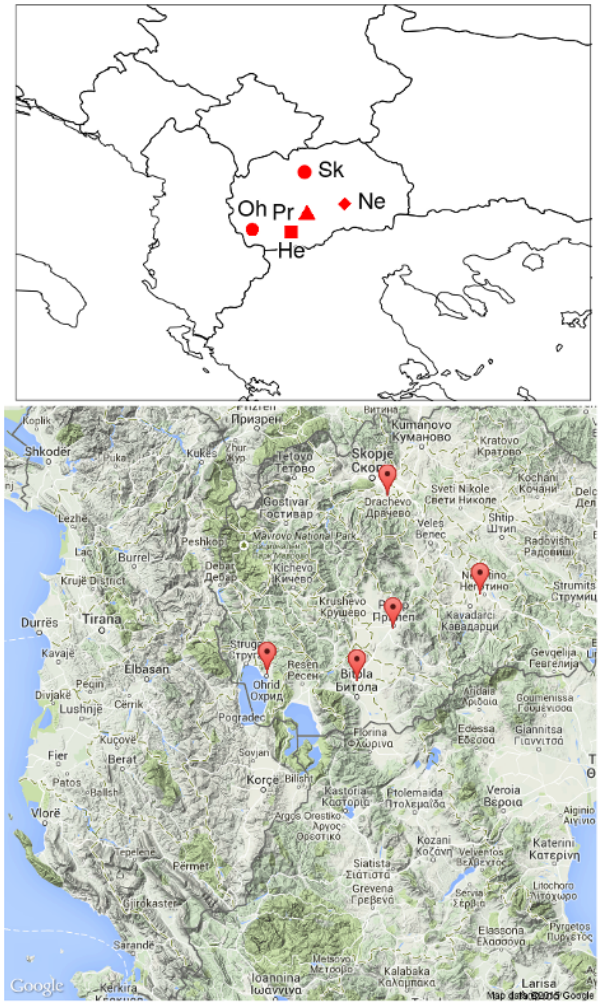


Very basic mapping in R

```
# Load libraries
library(sp)
library(rworldmap) # Basic world maps
library(TeachingDemos) # To be able to move text little bit
library(RgoogleMaps) # Google and OpenStreetMaps
# Plot basic map with state boundaries within selected range
plot(x=getMap(resolution="high"), xlim=c(19, 24), ylim=c(39, 44),
  asp=1, lwd=1.5)
box() # Add frame around the map
# Plot location points
points(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$xy
  [["lat"]], pch=15:19, col="red", cex=4)
# Add text descriptions for points. Text is aside and with background
shadowtext(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$xy
  [["lat"]], labels=as.vector(hauss.genind@pop.names), col="black",
  bg="white", theta=seq(pi/4, 2*pi, length.out=8), r=0.15,
  pos=c(1, 3, 2, 4, 4), offset=0.75, cex=1.5)
```

Basic map and Google map

```
# Insert legend
legend(x="topright", inset=1/50,
  legend=c("He", "Oh", "Pr", "Ne",
  "Sk"), col="red", border="black",
  pch=15:19, pt.cex=2, bty="o",
  bg="lightgrey", box.lwd=1.5,
  cex=1.5, title="Populations")
# Google map is produced into a
# file. Parametr markers contain
# data frame with coordinates and
# possibly with another information
hauss.gmap <- GetMap(center=c(lat=
  41, lon=21), size=c(640, 640),
  destfile="gmap.png", zoom=8,
  markers=hauss.coord,
  maptype="terrain")
```



OpenStreetMaps and datasets from mapproj

```
# Plot on OpenStreeMap - server is commonly overloaded and doesn't
# respond correctly - in fact, it rarely works...
GetMap.OSM(lonR=c(18, 24), latR=c(39, 44), scale=200000, destfile=
  "osmmap.png", format="png", RETURNIMAGE=TRUE, GRAYSCALE=FALSE,
  NEWMAP=TRUE, verbose=1)
# Plot on datasets from mapproj package
library(maps) # Various maping tools (plotting, ...)
library(mapdata) # More detailed maps, but political boundaries often
      # outdatet, see http://cran.r-project.org/web/packages/mapdata/
library(mapproj) # Converts latitude/longitude into projected coordinates
# Plot a map, check parameters
# Check among others "projection" and ?mapproject for its details
map(database="worldHires", boundary=TRUE, interior=TRUE, fill=TRUE,
  col="lightgrey", plot=TRUE, xlim=c(16, 27), ylim=c(37, 46))
# If you'd use projection, use - mapproject to convert also coordinates!
# See ?mapproject for details
points(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$xy
  [["lat"]], pch=15:19, col="red", cex=4)
```

Plotting on SHP files I

Get SHP files from http://trapa.cz/sites/default/rcourse/macedonia.zip

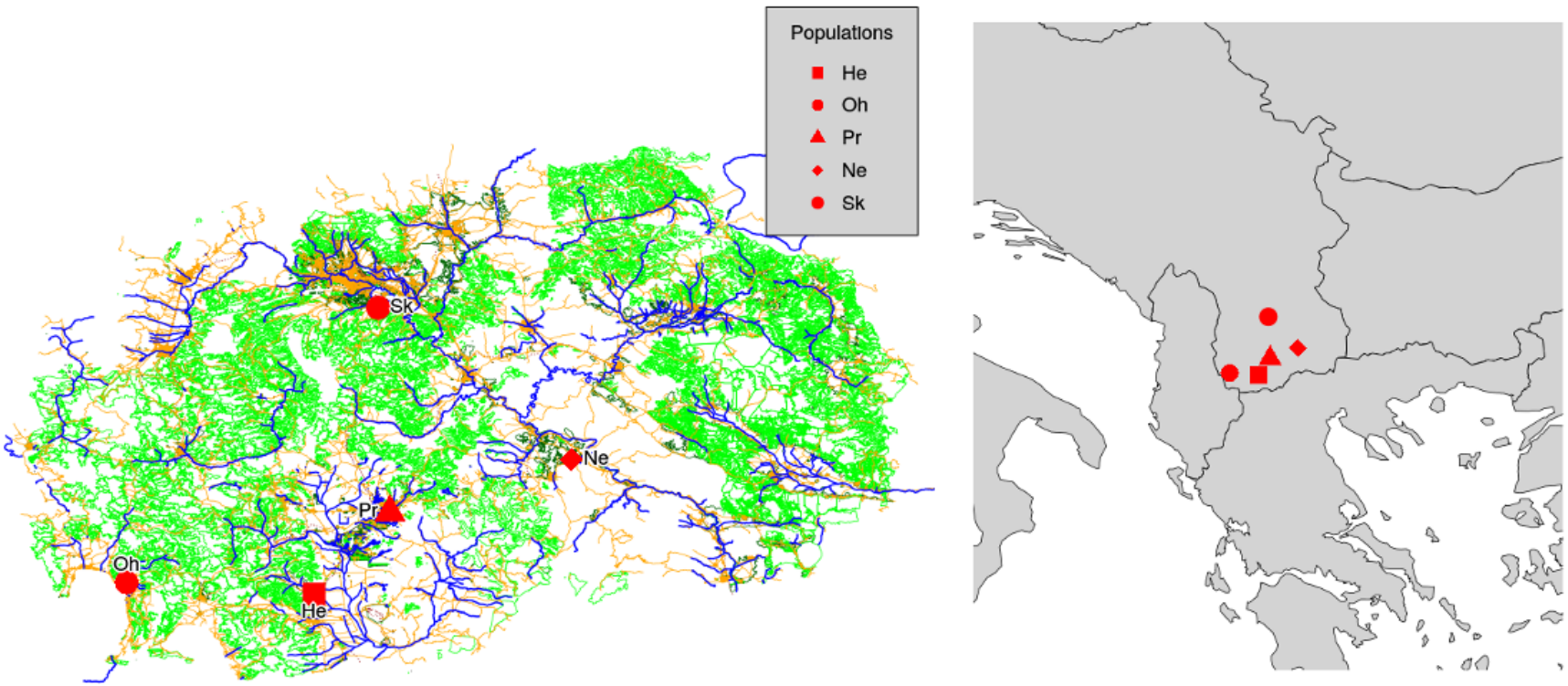
```
1  library(maptools)
2  # Load SHP file
3  # Data from http://download.geofabrik.de/europe/macedonia.html
4  # Directory has to contain also respective DBF and SHX files
5  # (same name, only different extension)
6  # There are several functions readShape* - select appropriate
7  # according to data stored in respective SHP file
8  macedonia_building <- readShapeLines(fn="macedonia_buildings.shp")
9  plot(macedonia_building)
10 macedonia_landuse <- readShapeLines(fn="macedonia_landuse.shp")
11 plot(macedonia_landuse)
12 macedonia_natural <- readShapeLines(fn="macedonia_natural.shp")
13 plot(macedonia_natural)
14 macedonia_railways <- readShapeLines(fn="macedonia_railways.shp")
15 plot(macedonia_railways)
16 macedonia_roads <- readShapeLines(fn="macedonia_roads.shp")
```

Plotting on SHP files II

```
plot(macedonia_roads)
macedonia_waterways <- readShapeLines(fn="macedonia_waterways.shp")
plot(macedonia_waterways)
# Plot all layers into single image
plot(macedonia_building)
plot(macedonia_landuse, add=TRUE, col="darkgreen", fill=TRUE)
plot(macedonia_natural, add=TRUE, col="green", fill=TRUE)
plot(macedonia_railways, add=TRUE, col="brown", lty="dotted")
plot(macedonia_roads, add=TRUE, col="orange")
plot(macedonia_waterways, add=TRUE, col="blue", lwd=2)
# Add sampling points
points(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$
  xy[["lat"]], pch=15:19, col="red", cex=4)
# Add description of psampling points
shadowtext(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$
  xy[["lat"]], labels=as.vector(hauss.genind@pop.names), col="black",
  bg="white", theta=seq(pi/4, 2*pi, length.out=8), r=0.15,
  pos=c(1, 3, 2, 4, 4), offset=0.75, cex=1.5)
```

Plotting on SHP files III

```
# Add legend
legend(x="topright", inset=1/50, legend=c("He", "Oh", "Pr", "Ne",
  "Sk"), col="red", border="black", pch=15:19, pt.cex=2, bty="o",
  bg="lightgrey", box.lwd=1.5, cex=1.5, title="Populations")
```



Running Structure in parallel with R

- Population genetic software for Bayesian clustering Structure requires several runs for each K and various Ks. It can result in hundreds runs
- With should use power of modern multi-core CPUs
- ParallelStructure library can optimally distribute computations of independent Structure runs among CPU cores
- When using it, cite Besnier & Glover 2013
- I show slightly modified way from The Molecular Ecologist

```
# Prepare special directory and set working directory
setwd("~/dokumenty/fakulta/vyuka/r_mol_data/priklady/structure/")
library(ParallelStructure) # Load the library
# See Structure manual and function's documentation
# It takes more or less same parameter as for normal Structure
?parallel_structure
# Author's recommend to run it without GUI and not on Windows
```

Preparing for ParallelStructure

Within working "structure" directory you need

- Subdirectory for results
- Text file describing jobs ("joblist")
  - One row for one Structure run
  - Every line contains name of run, list of populations separated by comas (e.g. 1,2,3,4,5) — you don't have to use all populations in all runs
  - K for actual run
  - Length of burnin chain
  - Number of steps
  - Columns are separated by spaces (or TABs)
- Data input file
  - Make it as simple as possible — remove any unneeded columns
  - For population names use subsequent number from 1 to number of populations

Input files

Joblist file:

S02	1,2,3,4,5	1	500	10000
S06	1,2,3,4,5	2	500	10000
S08	1,2,3,4,5	2	500	10000
...	...	...	...	...

Input file:

			msta93	msta101	msta102	msta103	...
H01	1	0	269	198	221	419	...
H01	1	0	269	198	223	419	...
H02	1	0	275	198	221	419	...
H02	1	0	283	198	223	419	...
...	...	...	...	...	...	...	...

Running ParallelStructure

```
parallel_structure(joblist="joblist.txt", n_cpu=7, structure_path=
  "~/bin/", infile="hauss_stru.in", outpath="results/", numinds=47,
  numloci=12, plot_output=1, label=1, popdata=1, popflag=1,
  phenotypes=0, markernames=1, mapdist=0, onerowperind=0, phaseinfo=0,
  extracol=0, missing=-9, ploidy=2, usepopinfo=0, revert_convert=1,
  printqhat=1, locdata=0, recessivealleles=0, phased=0, noadmix=0,
  linkage=0, locprior=0, inferalpha=1)
```

- Choose **n_cpu** according to your computer
- **structure_path** points to directory containing Structure binary
- **outpath** should aim to empty directory
- **plot_output=1** will produce plots for all runs
- Check all other settings according to Structure manual and your needs
- Get input file https://trapa.cz/sites/default/rcourse/hauss_stru.in and joblist https://trapa.cz/sites/default/rcourse/joblist.txt

ParallelStructure and Windows

- Authors do not recommend to run ParallelStructure on Windows...
- **parallel_structure()** is missing paralleling library which is not available on Windows, instead try

```
# Install Rmpi library required by ParallelStructure for
# parallelisation on Windows
install.packages("Rmpi")
library(Rmpi)
# Instead of parallel_structure() use MPI_structure()
# with same arguments
MPI_structure(...) # Same arguments as on previous slide
```

- If this fails, look for some UNIX machine (Linux, Mac OS X, BSD, ...)...

Postprocess Structure results — select the best K

- Using Structure-sum-2009 R script by Dorothee Ehrich

```
# Load the script
source("structure-sum-2009.r")
# Create new directory with result files results_job_*_f and set
# working directory accordingly
setwd("/home/vojta/dokumenty/fakulta/vyuka/r_mol_data/priklady/
  structure/structure_sum/")
```

- When using it, cite Ehrich 2006. If you don't have the script, ask (it is not my so I don't want to post it on the net)
- Prepare list_k.txt containing on each line K and name of output file
- Get list K from https://trapa.cz/sites/default/rcourse/list_k.txt
- Joblist look like this:

```
2    results_job_S010_f
3    results_job_S011_f
...  ...
```

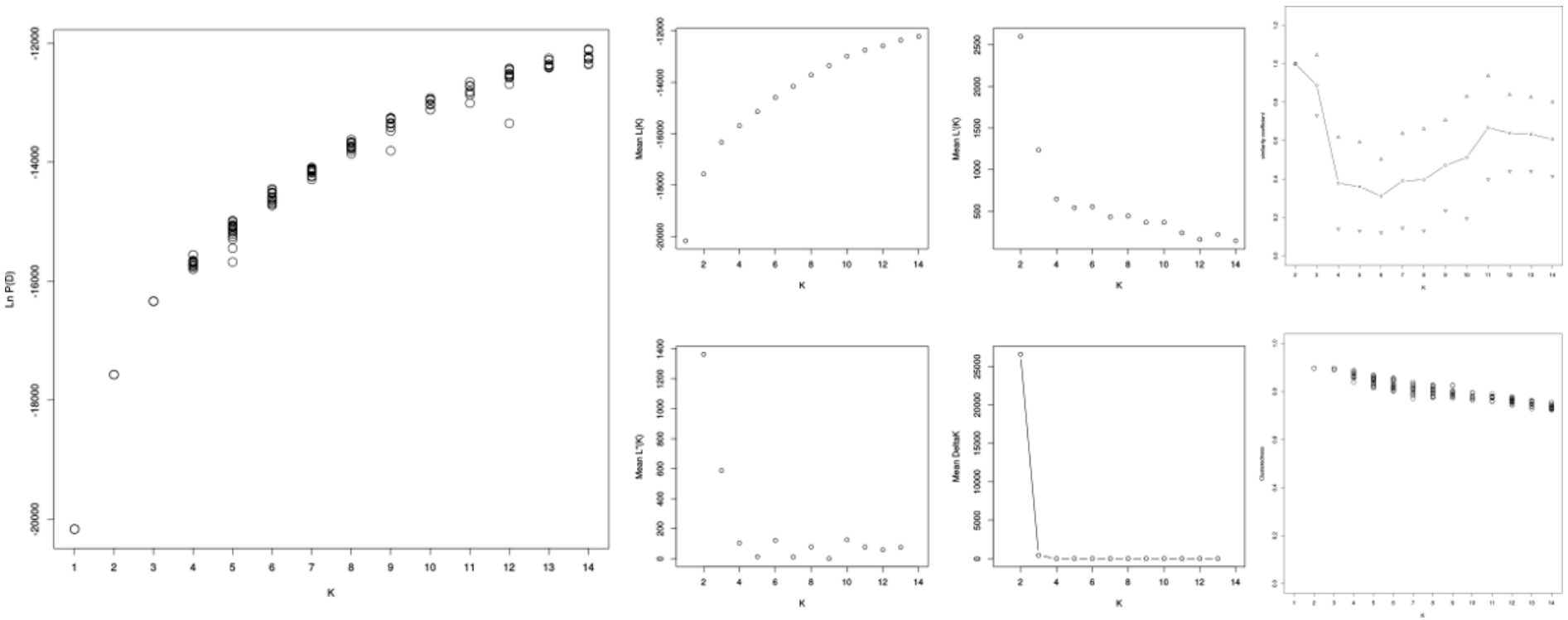
Run Structure-sum

```
# See documentation for details. Functions take as an argument
# list_k file and number of populations
Structure.table("list_k.txt", 5)
Structure.simil("list_k.txt", 5)
Structure.deltaK("list_k.txt", 5)
graphics.off() # Close graphics
Structure.cluster("list_k.txt", 5)
# Reordering ("alignment") of runs to get same clusters in same
# columns (prepare respective list_k files - one for each K)
Structure.order("list_k_02.txt", 5)
Structure.order("list_k_03.txt", 5)
Structure.order("list_k_04.txt", 5)
Structure.order("list_k_05.txt", 5)
Structure.order("list_k_06.txt", 5)
Structure.order("list_k_07.txt", 5)
# Continue with CLUMPP and distruct...
```

Details: https://trapa.cz/en/structure-r-linux

Outputs of Structure-sum — the best K is 2, may be 3 — stability of runs, good posterior probability

Results from different data set, not from our toy



Multiple sequence alignment — package muscle

```
# Muscle has its own functions read.fasta and write.fasta
library(muscle)
# Reads local file and writes R object or external file
# with alignment, you can use any command line argument
# available for muscle, see ?muscle and its manual
# Write result to external file
muscle(seqs="usflu.fasta", out="usflu.aln", quiet=FALSE)
# Write result to R object
usflu.muscle <- muscle(seqs="usflu.fasta", out=NULL, quiet=FALSE)
# Prints the alignment
print(usflu.muscle, from=1, to=usflu.muscle[["length"]])
class(usflu.muscle)
```

- Package **muscle** contains MUSCLE binary
- Read MUSCLE documentation to get correct parameters (if you are not fine with defaults)

Multiple sequence alignment

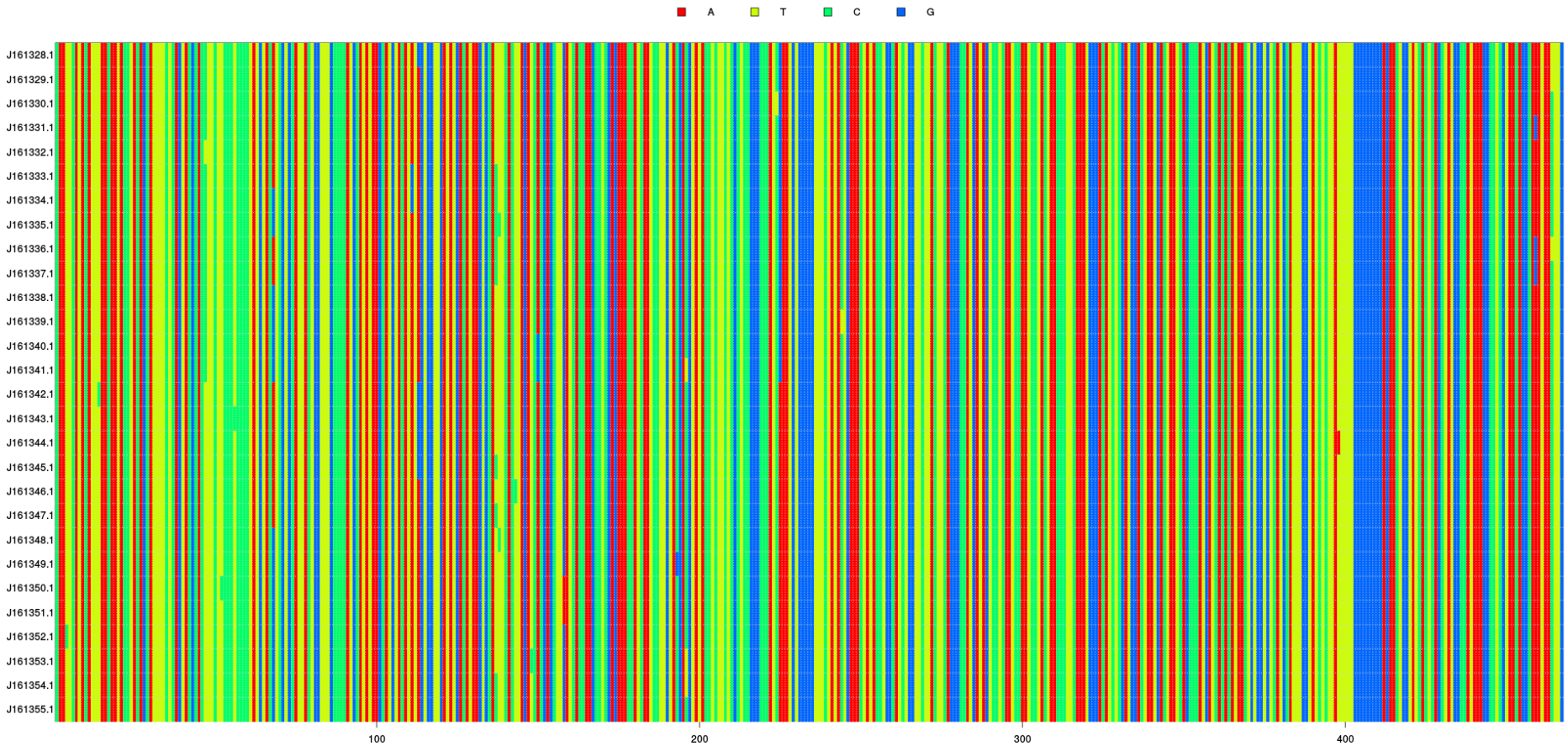
- MUSCLE is available in package **muscle** (above) and in package **ape** — first one reads FASTA files and writes external FASTA files or R objects of class "muscle". The latter reads and writes object of class "DNAbin"
- ape also contains functions to use Clustal and T-Coffee — both read and write DNAbin
- MAFFT is available from (same author) in packages **ips** and **phyloch** — both read and write DNAbin

```
1 library(colorspace) # Libraries needed by phyloch/ips
2 library(XML)
3 library(phyloch) # Alignment with mafft, you can also try package ips
4 # Requires mafft binary on specific location
5 # you might be needed to copy it or make symlink or so
6 # read ?mafft and mafft's documentation
7 meles.mafft <- mafft(x=meles.dna, method="localpair", maxiterate=100)
8 meles.mafft
```

Clustal, MUSCLE and T-Coffee from ape

```
class(meles.mafft)
# read ?clustal and documentation of Clustal, Muscle and T-Coffee
# when using them to set correct parameters
clustal(x=meles.dna, pw.gapopen=10, pw.gapext=0.1, gapopen=10,
  gapext=0.2, quiet=FALSE, original.ordering=TRUE)
meles.muscle <- muscle(x=meles.dna, exec="muscle",
  quiet=FALSE, original.ordering=TRUE)
meles.muscle
class(meles.muscle)
# Plot the alignment - you can select which bases to plot
# and/or modify colours
image(x=meles.muscle, c("a", "t", "c" ,"g", "n"), col=rainbow(5))
# Add grey dotted grid
grid(nx=ncol(meles.muscle), ny=nrow(meles.muscle), col="lightgrey")
# Remove gaps from alignment - destroy it
meles.nogaps <- del.gaps(meles.muscle)
```

Multiple sequence alignment with MUSCLE



Read and write tree and drop tips

```
# Read trees in NEWICK format - single or multiple tree(s)
oxalis.trees <- read.tree
  ("http://trapa.cz/sites/default/rcourse/oxalis.nwk")
summary(oxalis.trees)
length(oxalis.trees)
names(oxalis.trees)
# Export trees in NEWICK format
write.tree(phy=oxalis.trees, file="trees.nwk")
# Drop a tip from multiPhylo
plot.multiPhylo(x=oxalis.trees)
oxalis.trees.drop <- lapply(X=oxalis.trees, FUN=drop.tip, "TaxaOut")
class(oxalis.trees.drop) <- "multiPhylo"
plot.multiPhylo(x=oxalis.trees.drop)
# Drop a tip from single tree
plot.phylo(hauss.nj)
hauss.nj.drop <- drop.tip(phy=hauss.nj, tip=47)
plot.phylo(hauss.nj.drop)
```

Extract clades from trees and drop extinct tips

```
# Interactively extract tree
# Plot source tree
plot.phylo(hauss.nj)
# Select clade to extract by clicking on it
hauss.nj.extracted <- extract.clade(phy=hauss.nj, interactive=TRUE)
# See new extracted tree
plot.phylo(hauss.nj.extracted)
# Non-interactively extract tree
hauss.nj.extracted <- extract.clade(phy=hauss.nj, node=60,
  interactive=FALSE)
# See new extracted tree
plot.phylo(hauss.nj.extracted)
# Drop "extinct" tips - those who don't reach end the tree
# tolerance is respective to the used metrics
plot.phylo(hauss.nj)
axisPhylo()
hauss.nj.fossil <- drop.fossil(phy=hauss.nj, tol=0.25)
plot.phylo(hauss.nj.fossil)
```

Join two trees, rotate tree

```
# Bind two trees into one
hauss.nj.bind <- bind.tree(x=hauss.nj.fossil, y=hauss.nj.extracted,
  where="root", position=0, interactive=FALSE)
plot.phylo(hauss.nj.bind)
# Bind two trees interactively
# Plot tree receiving the new one
plot.phylo(hauss.nj.fossil)
# Select where to bind new tree to
hauss.nj.bind <- bind.tree(x=hauss.nj.fossil, y=hauss.nj.extracted,
  interactive=TRUE)
plot.phylo(hauss.nj.bind)
# Rotate tree
plot.phylo(hauss.nj)
hauss.nj.rotated <- rotate(phy=hauss.nj, node="70")
plot.phylo(hauss.nj.rotated)
```

Ladderize and (un)root the tree

```
# Ladderize the tree
plot.phylo(hauss.nj)
hauss.nj.ladderized <- ladderize(hauss.nj)
plot.phylo(hauss.nj.ladderized)
# Root the tree
plot.phylo(hauss.nj)
print.phylo(hauss.nj)
hauss.nj.rooted <- root(phy=hauss.nj, outgroup="10")
print.phylo(hauss.nj.rooted)
plot.phylo(hauss.nj.rooted)
# Root the tree interactive
plot.phylo(hauss.nj)
hauss.nj.rooted <- root(phy=hauss.nj, interactive=TRUE)
plot.phylo(hauss.nj.rooted)
# unroot the tree
unroot()
# Check if it is rooted
is.rooted()
```

Check tree and compute branch lengths and times

```
# Check if the tree is ultrametric - is variance of distances
# of all tips to node 0? It is required for some analysis
is.ultrametric()
# Make tree ultrametric
chronos()
?chronos # Check it for mode how to calculate the lengths
# chronos has more uses - it is mainly used for dating
# Compute branch lengths for trees withou branch lengths
?compute.brlen # Check it for mode how to calculate the lengths
compute.brlen()
# Computes the branch lengths of a tree giving its branching
# times (aka node ages or heights)
compute.brtime()
?compute.brtima # Check it for mode how to calculate the lengths
```

Class "multiPhylo" is just a list of "phylo" objects to store multiple trees — you can perform most of analysis on it as on phylo, commonly using **lapply** function (afterwards use class(x) <- "multiPhylo" to ensure other functions will see it as multiPhylo object).

Topographical distances among trees I

We have plenty of trees. How much are their topologies different?

```
1  library(gplots)
2  library(corrplot)
3  library(phytools)
4  # Prepare matrix for distances
5  oxalis.trees.d <- matrix(nrow=length(oxalis.trees),
6    ncol=length(oxalis.trees))
7  # Calculate pairwise topographic distances
8  for (i in 1:length(oxalis.trees)) {
9    for (j in i:length(oxalis.trees)) {
10     print(c(i,j))
11     oxalis.trees.d[i,j] <- dist.topo(oxalis.trees[[i]],
12       oxalis.trees[[j]])
13   }
14 }
15 # Basic information about the distance matrix
16 dim(oxalis.trees.d)
17 head.matrix(oxalis.trees.d)
```

Topographical distances among trees II

Postprocess the matrix and plot it. There are several methods for calculating distance matrices among the trees — some take branch lengths into account, some only topology.

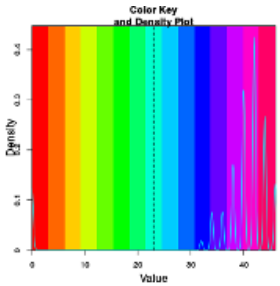
```
# Add names of columns and rows
colnames(oxalis.trees.d) <- names(oxalis.trees)
rownames(oxalis.trees.d) <- names(oxalis.trees)
# Make matrix symetric
oxalis.trees.d[lower.tri(oxalis.trees.d)]=
  t(oxalis.trees.d)[lower.tri(oxalis.trees.d)]
# Create heatmaps using heatmap.2 function from gplots package
heatmap.2(x=oxalis.trees.d, Rowv=FALSE, Colv="Rowv", dendrogram="none",
  symm=TRUE, scale="none", na.rm=TRUE, revC=FALSE, col=rainbow(15),
  cellnote=oxalis.trees.d, notecex=1, notecol="white", trace="row",
  linecol="black", labRow=names(oxalis.trees),
  labCol=names(oxalis.trees), key=TRUE, keysize=2,
  density.info="density", symkey=FALSE, main="Correlation matrix of
  topographical distances", xlab=names(oxalis.trees),
  ylab=names(oxalis.trees))
```

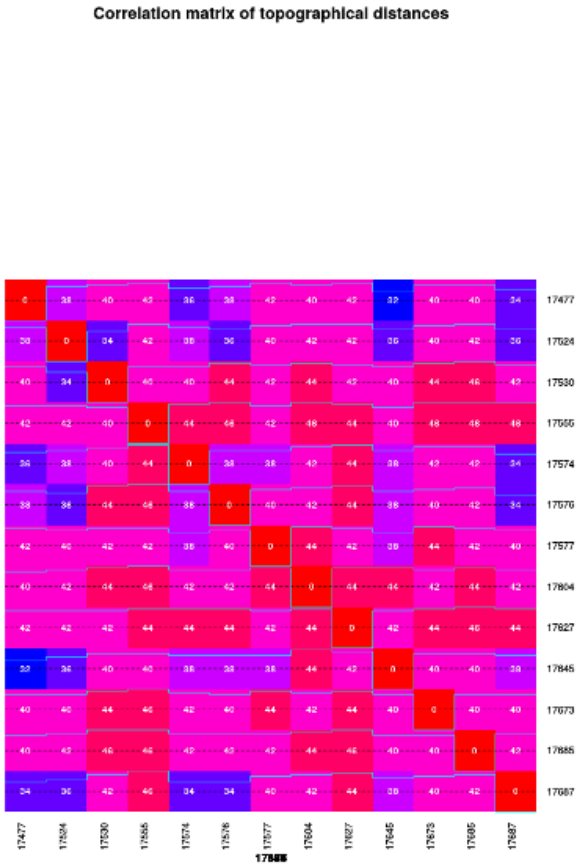
Topographical distances among trees III

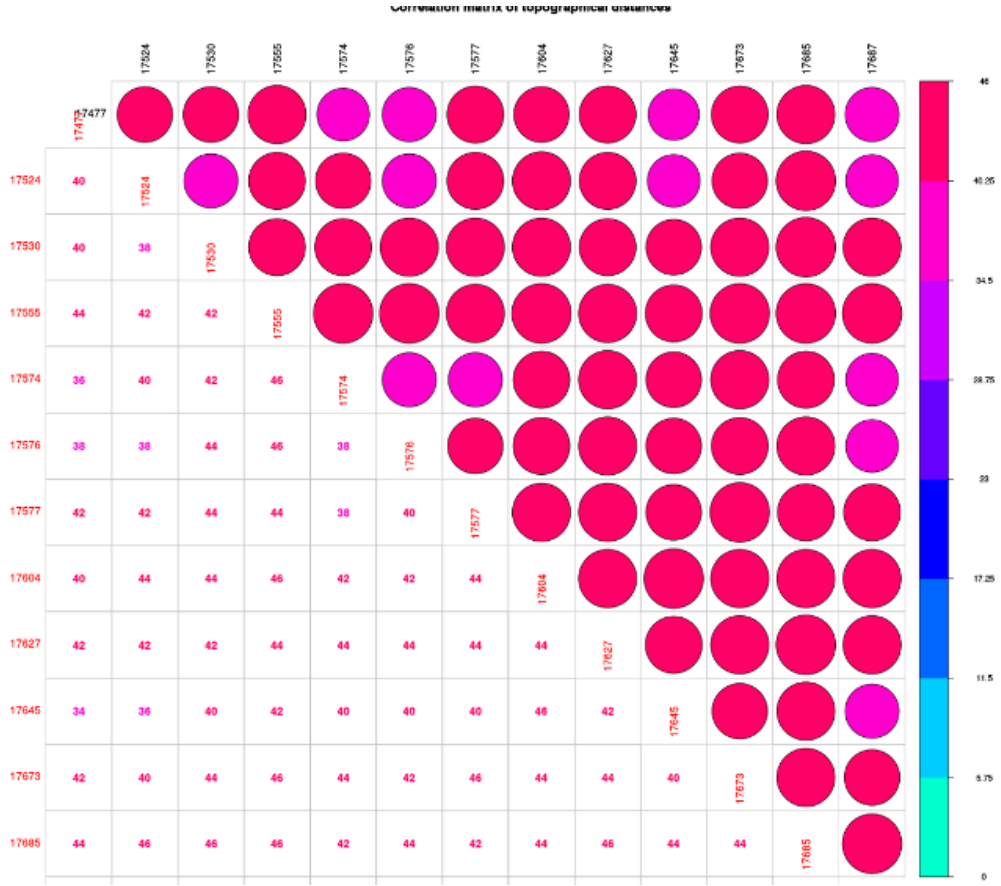
Calculate Robinsons-Foulds distance (allow compare trees created by different algorithms) matrix among trees and plot it.

```
# Robinsons-Foulds distance
oxalis.trees.d.rf <- multiRF(oxalis.trees)
# Add names of columns and rows
colnames(oxalis.trees.d.rf) <- names(oxalis.trees)
rownames(oxalis.trees.d.rf) <- names(oxalis.trees)
# Create heatmap using corrplot function from corrplot package
corrplot(corr=oxalis.trees.d.rf, method="circle", type="upper",
  col=rainbow(15), title="Correlation matrix of topographical
  distances", is.corr=FALSE, diag=FALSE, outline=TRUE,
  order="alphabet", tl.pos="lt", tl.col="black")
corrplot(corr=oxalis.trees.d.rf, method="number", type="lower",
  add=TRUE, col=rainbow(15), title="Correlation matrix of
  topographical distances", is.corr=FALSE, diag=FALSE,
  outline=FALSE, order="alphabet", tl.pos="ld", cl.pos="n")
```

Topographical distances among trees IV — the matrices

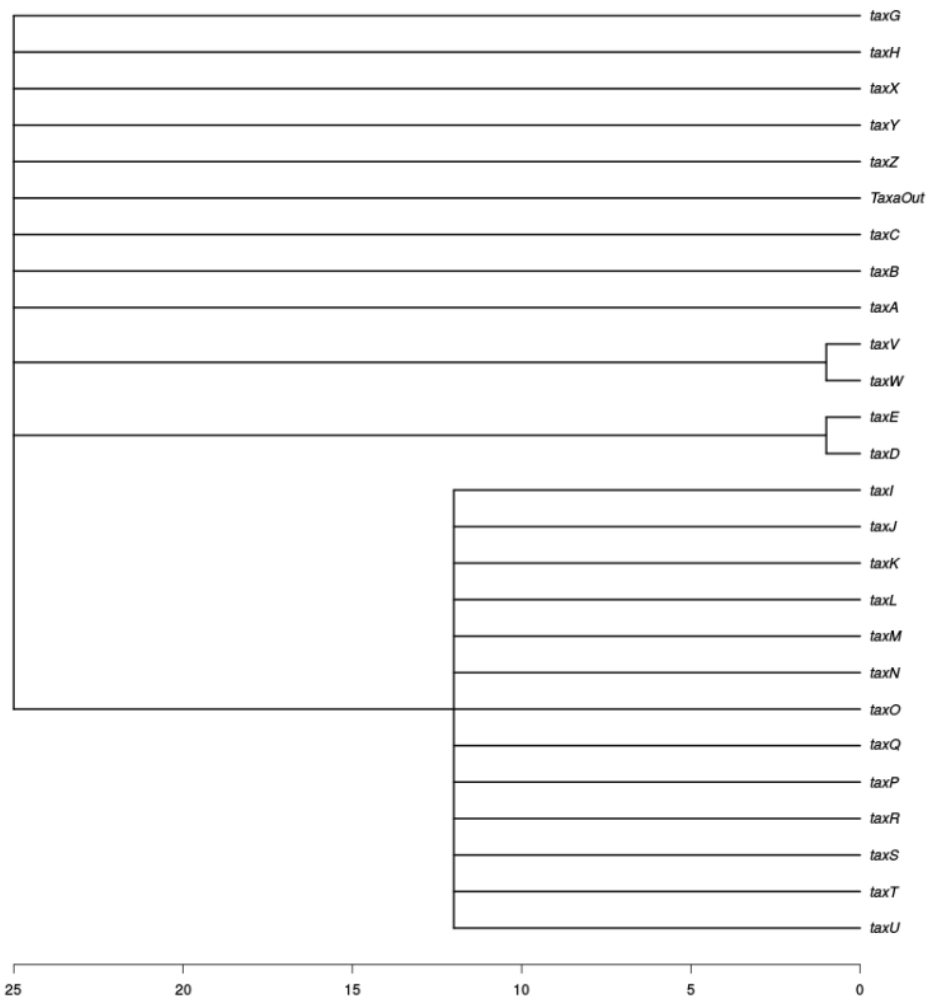




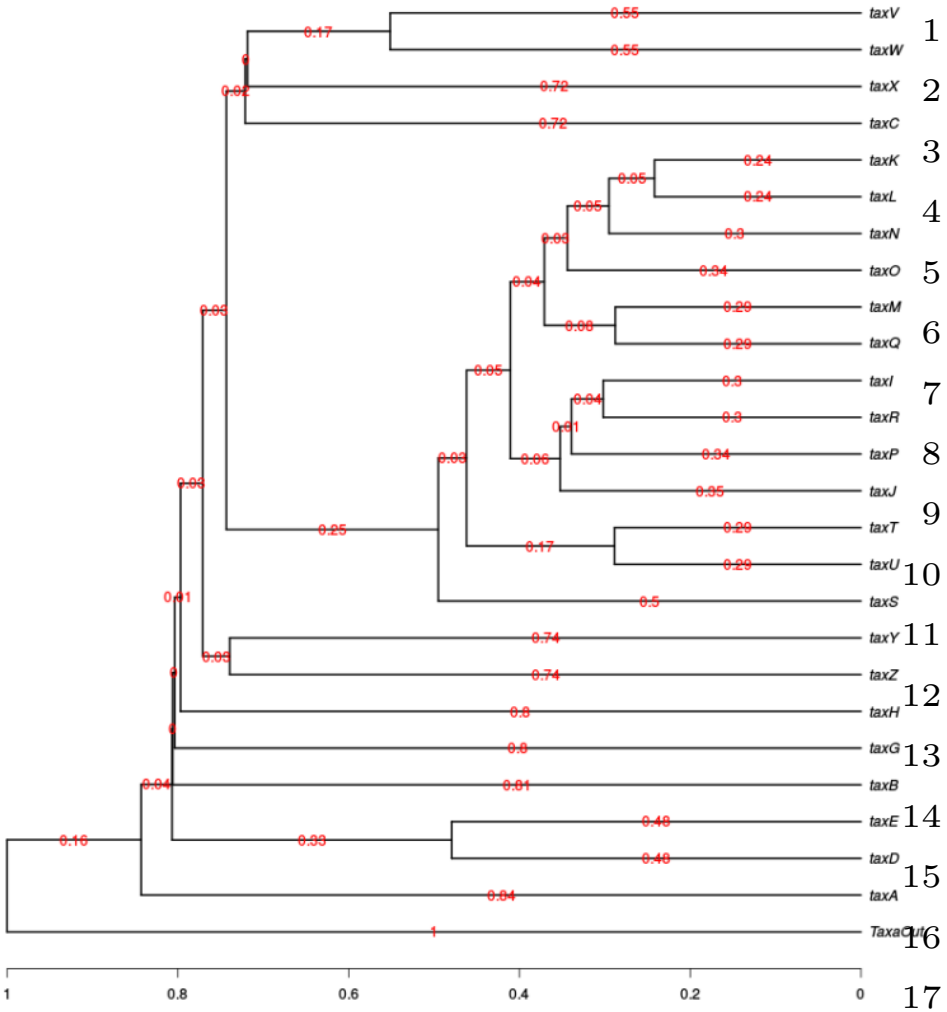


Consensus tree

```
# Root all trees
oxalis.trees.rooted <- lapply
  (X=oxalis.trees, FUN=root,
  "TaxaOut")
class(oxalis.trees.rooted) <-
  "multiPhylo"
# Consenus tree (50 \% rule)
oxalis.tree.con <- consensus
  (oxalis.trees.rooted, p=0.5,
  check.labels=TRUE)
print.phylo(oxalis.tree.con)
# Plot the tree
plot.phylo(oxalis.tree.con,
  edge.width=2, label.offset=0.3)
axisPhylo(side=1)
# What a nice tree... :-P
```



Species tree — all trees must be ultrametric

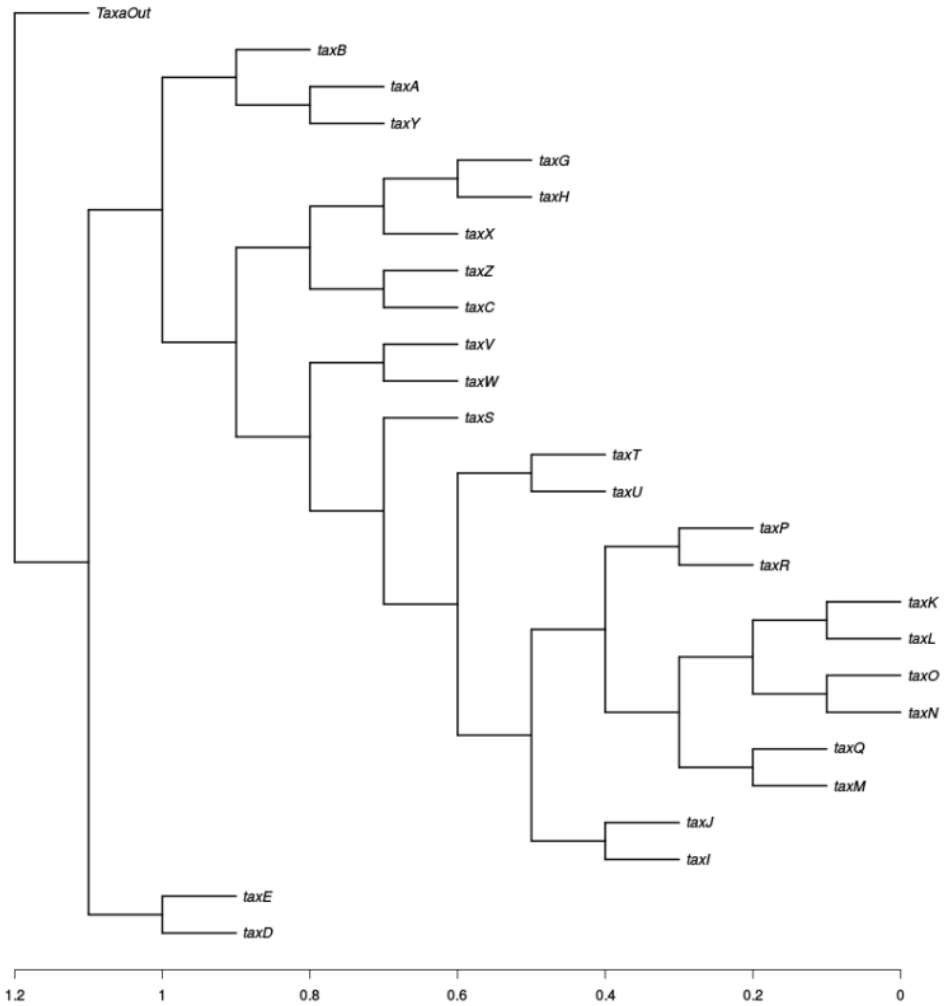



```
# Chronos scale trees
oxalis.trees.ultra <- lapply
  (X=oxalis.trees.rooted,
  FUN=chronos, model="correlated")
class(oxalis.trees.ultra) <-
  "multiPhylo"
# Mean distances
oxalis.tree.sp.mean <- speciesTree
  (oxalis.trees.ultra, mean)
# Plot the tree
plot.phylo(oxalis.tree.sp.mean,
  edge.width=2, label.offset=0.01)
edgelabels(text=round(oxalis.
  tree.sp.mean[["edge.length"]],
  digits=2), frame="none",
  col="red", bg="none")
axisPhylo(side=1)
```

Parsimony super tree

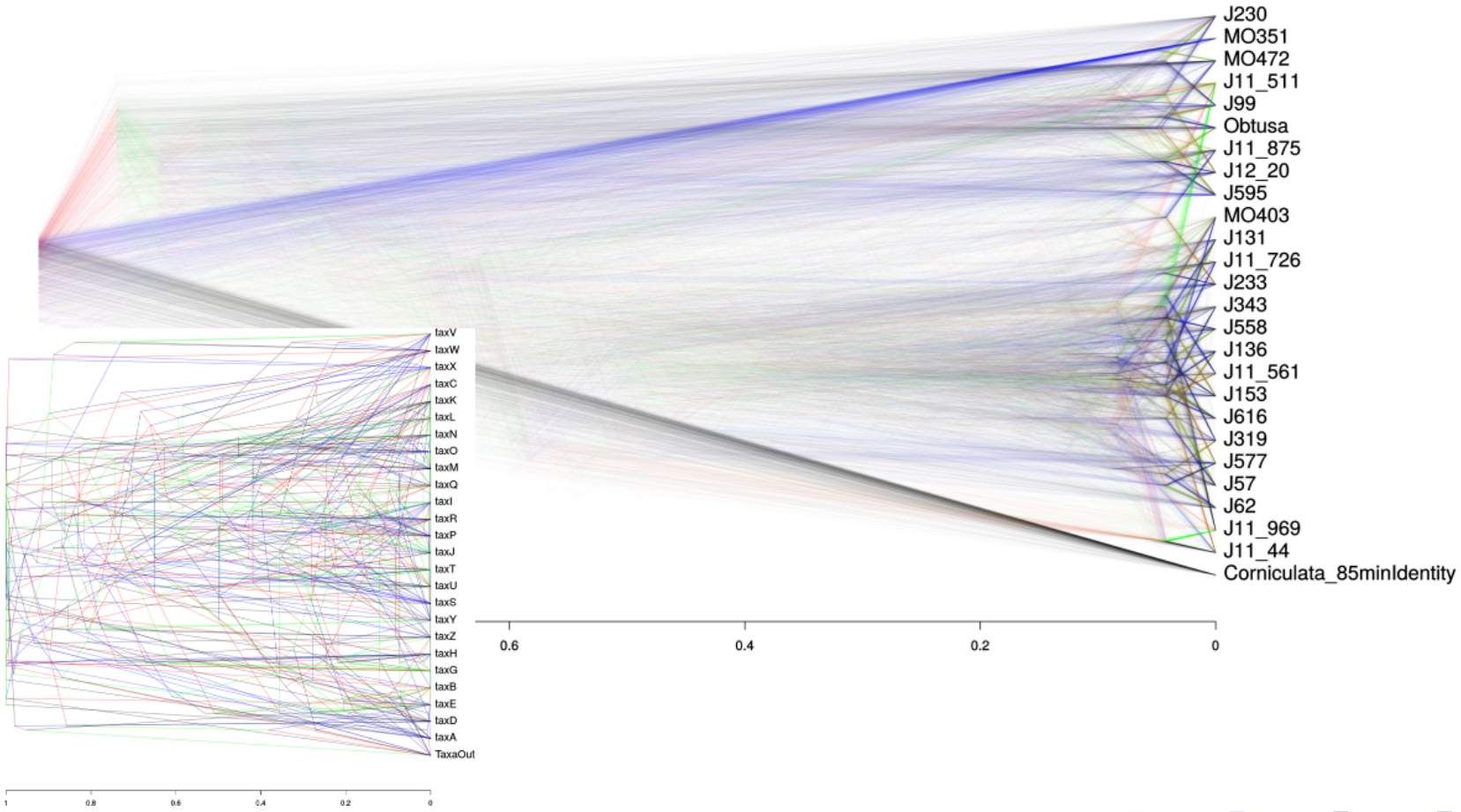
```
library(phangorn)
oxalis.tree.sp <- superTree(tree=
  oxalis.trees.rooted, method=
  "optim.parsimony", rooted=TRUE)
print.phylo(oxalis.tree.sp)
plot.phylo(oxalis.tree.sp,
  edge.width=2, label.offset=0.01)
axisPhylo(side=1)
```

Currently, there is no package for coalescence based approach for creation of species tree from multiple gene trees. This function was in package phybase, but it is abandoned and doesn't install in R 3 without hacks



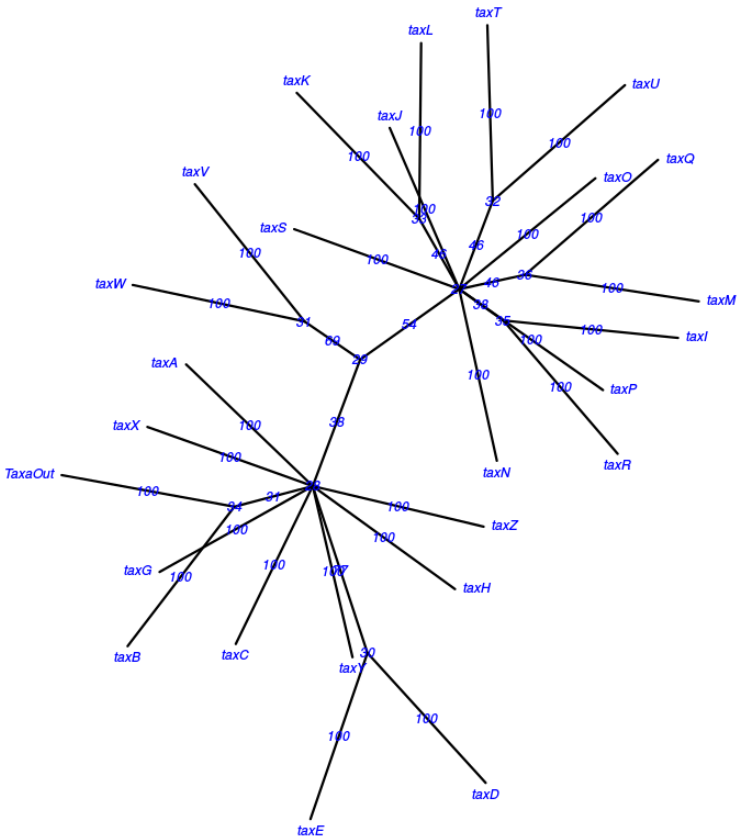
Density tree

```
1 densiTree(x=oxalis.trees.ultra, type="cladogram", alpha=0.5,
2   consensus=oxalis.tree.sp.mean, scaleX=TRUE, col=c("black",
3   "green", "blue", "red"), cex=1.5)
```



Networks

```
oxalis.tree.net <- consensusNet
  (oxalis.trees.rooted, prob=0.25)
```




```
plot.networx(x=oxalis.tree.net,
  planar=FALSE, type="2D",
  use.edge.length=TRUE,
  show.tip.label=TRUE,
  show.edge.label=TRUE,
  show.node.label=TRUE,
  show.nodes=TRUE,
  edge.color="black",
  tip.color="blue")
plot.networx(x=oxalis.tree.net,
  planar=FALSE, type="3D",
  use.edge.length=TRUE,
  show.tip.label=TRUE,
  show.edge.label=TRUE,
  show.node.label=TRUE,
  show.nodes=TRUE, edge.color=
  "black", tip.color="blue")
```

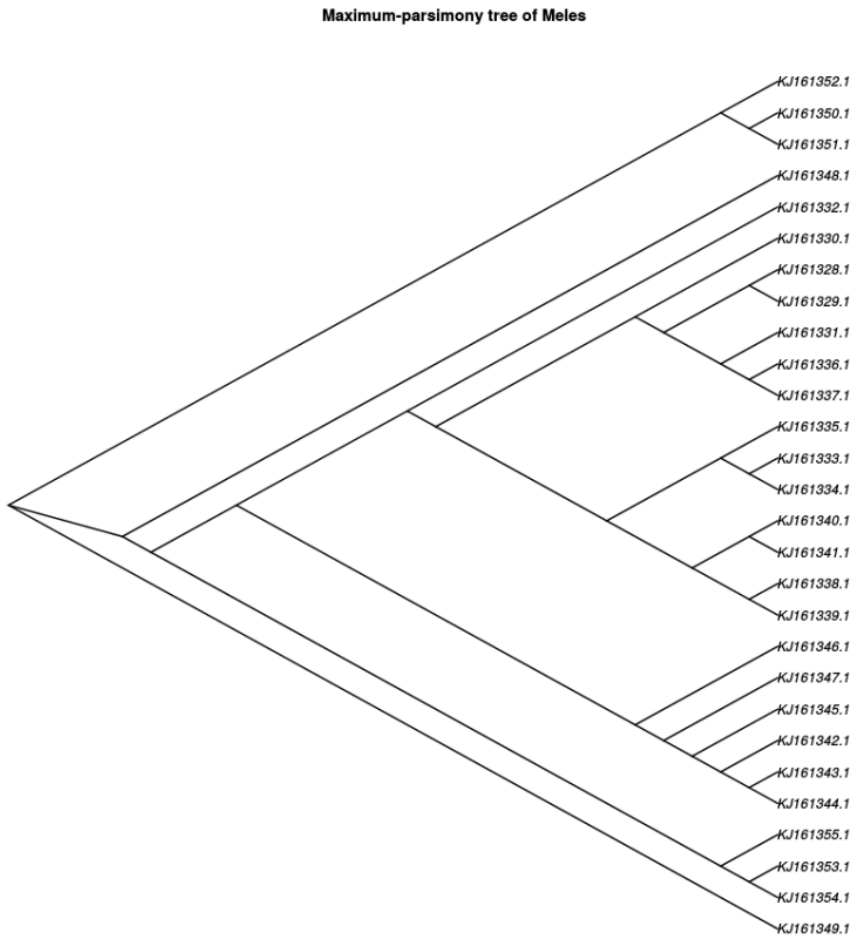
Kronoviz — see all trees on same scale

```
kronoviz(x=oxalis.trees.rooted,
  layout=length(oxalis.trees.
  rooted), horiz=TRUE)
# Close graphical device to
# cancel division of plotting
# device
dev.off()
```

The plot can be very long and it can be hard to see details. But one can get impression if all trees are more or less in same scale (have comparable length) or not.

Maximum parsimony

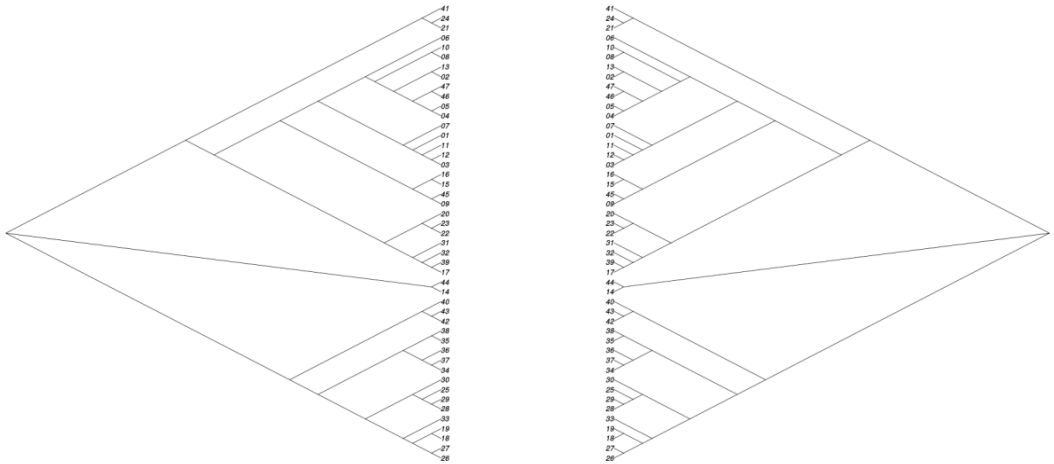
```
# Conversion to phyDat
meles.phydat <-
  as.phyDat(meles.dna)
# Prepare starting tree
meles.tre.ini <- nj(dist.
  dna(x=meles.dna,model="raw"))
# Parsimony: ?parsimony
# Maximum parsimony score
parsimony(tree=meles.tre.ini,
  data=meles.phydat)
# Optimisation
# Maximum parsimony tree
meles.tre.pars <- optim.
  parsimony(tree=meles.tre.
  ini, data=meles.phydat)
# Draw a tree
plot.phylo(x=meles.tre.pars,
  type="clad", edge.width=2)
```



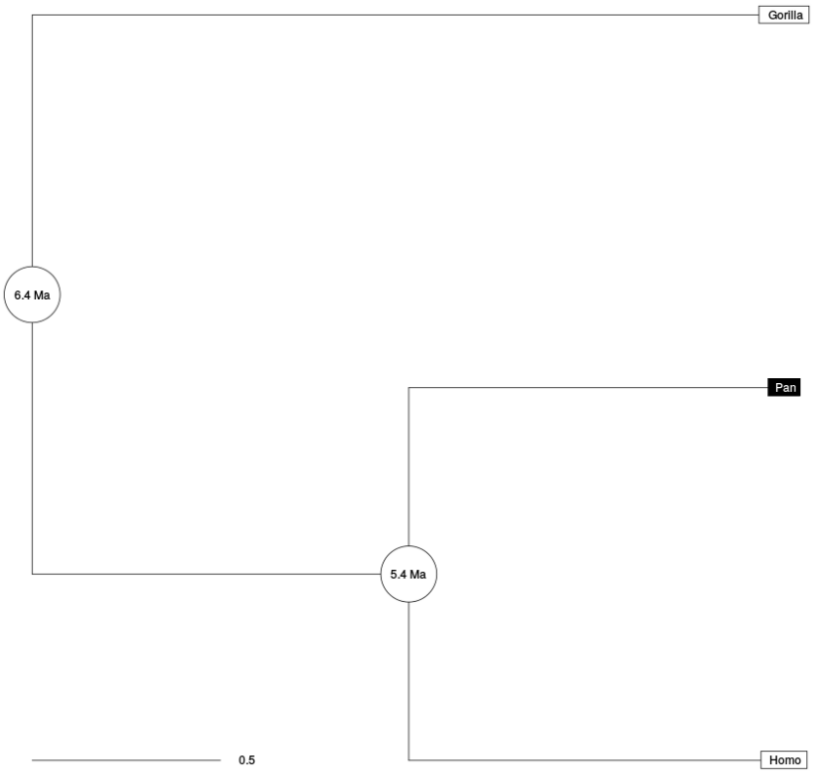
Change orientation of plots

plot.phylo() has plenty of possibilities to influence — check **?plot.phylo** and **?par**

```
?plot.phylo # check it for various possibilities what to influence
par(mfrow=c(1, 2)) # Plot two plots in one row
plot.phylo(x=hauss.nj, type="cladogram", use.edge.length=FALSE,
  direction="rightwards")
plot.phylo(x=hauss.nj, type="cladogram", use.edge.length=FALSE,
  direction="leftwards")
dev.off() # Close graphical device to cancel par() settings
```



Highlighted labels




```
1  # Nice tiplabels and
2  # higlighted tiplabel
3  trape <- read.tree(text=
4    "((Homo, Pan), Gorilla);")
5  plot.phylo(x=trape,
6    show.tip.label=FALSE)
7  tiplabels(trape[["tip.label"]],
8    bg=c("white", "black",
9    "white"), col=c("black",
10   "white", "black"))
11 nodelabels(text=c("6.4 Ma",
12   "5.4 Ma"), frame="circle",
13   bg="white")
14 add.scale.bar()
```

Phylogenetic independent contrast

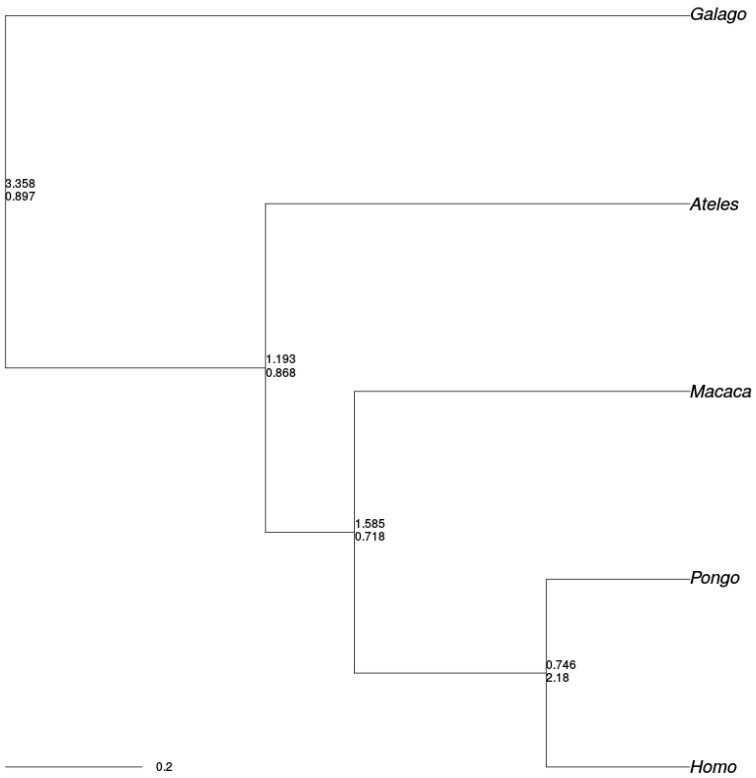
- When analysing comparative data takes phylogeny into account
- If we assume that a continuous trait evolves randomly in any direction (i.e., the Brownian motion model), then the "contrast" between two species is expected to have a normal distribution with mean zero, and variance proportional to the time since divergence

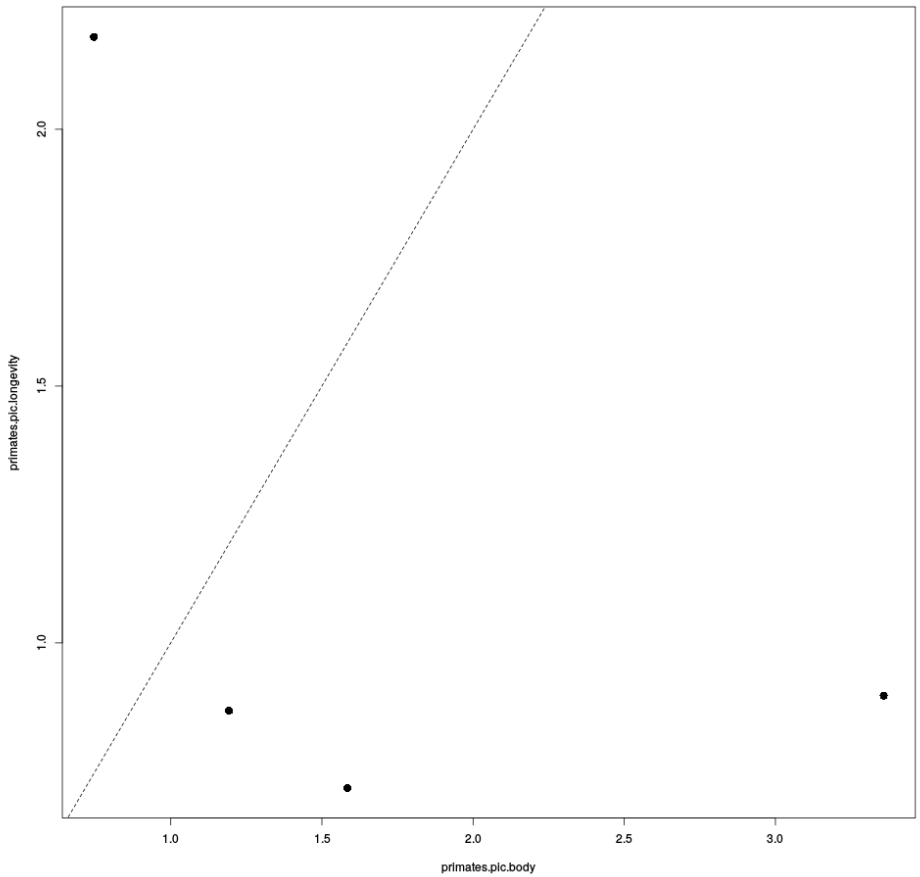
```
# Prepare the data # Body mass of primates
primates.body <- c(4.09434, 3.61092, 2.37024, 2.02815, 1.46968)
# Longevity of primates
primates.longevity <- c(4.74493, 3.3322, 3.3673, 2.89037, 2.30259)
# Add names to the values
names(primates.body) <- names(primates.longevity) <- c("Homo", "Pongo",
  "Macaca", "Ateles", "Galago")
# Create a tree in Newick format
primates.tree <- read.tree(text="((((Homo:0.21, Pongo:0.21):0.28,
  Macaca:0.49):0.13, Ateles:0.62):0.38, Galago:1.00);")
plot.phylo(primates.tree)
```

PIC and its plotting

```
primates.pic.body <- pic(x=primates.body, phy=primates.tree,
  scaled=TRUE, var.contrasts=FALSE, rescaled.tree=FALSE)
primates.pic.longevity <- pic(x=primates.longevity, phy=primates.tree,
  scaled=TRUE, var.contrasts=FALSE, rescaled.tree=FALSE)
# Plot a tree with PIC values
plot.phylo(x=primates.tree, lwd=2, cex=1.5, offset=0.2)
nodelabels(round(primates.pic.body, digits=3), adj=c(0, -0.5),
  frame="none")
nodelabels(round(primates.pic.longevity, digits=3), adj=c(0, 1),
  frame="none")
add.scale.bar()
# Plot PIC
plot(x=primates.pic.body, y=primates.pic.longevity, pch=16, cex=1.5)
abline(a=0, b=1, lty=2) # x=y line
# correlation coefficient of both PICs
cor(x=primates.pic.body, y=primates.pic.longevity, method="pearson")
[1] -0.5179156
```

Plot of PIC (on the tree)





Test it

```
lm(formula=primates.pic.longevity~primates.pic.body)
Coefficients:
  (Intercept)  primates.pic.body
       1.6957            -0.3081
# Because PICs have expected mean zero - such linear regressions
# should be done through the origin (the intercept is set to zero)
lm(formula=primates.pic.longevity~primates.pic.body-1)
Coefficients:
primates.pic.body
           0.4319
# Permutation procedure to test PIC
lmorigin(formula=primates.pic.longevity~primates.pic.body, nperm=1000)
Regression through the origin
Permutation method = raw data
Coefficients and parametric test results
                   Coefficient Std_error t-value Pr(>|t|)
primates.pic.body      0.43193   0.28649  1.5077   0.2288
F-statistic: 2.273067 on 1 and 3 DF:
  permutational p-value: 0.2377622
```

Phylogenetic autocorrelation

- Autocorrelation coefficient to quantify whether the distribution of a trait among a set of species is affected or not by their phylogenetic relationships
- In the absence of phylogenetic autocorrelation, the mean expected value of I and its variance are known - it is thus possible to test the null hypothesis of the absence of dependence among observations

```
# Let's choose weights as wij = 1/dij, where the d's is the distances
# measured on the tree - cophenetic() calculates cophenetic distances
# can be just cophenetic(primates.tree) or some other transformation
primates.weights <- 1/cophenetic(primates.tree)
primates.weights # See it
class(primates.weights)
diag(primates.weights) <- 0 # Set diagonal to 0
```

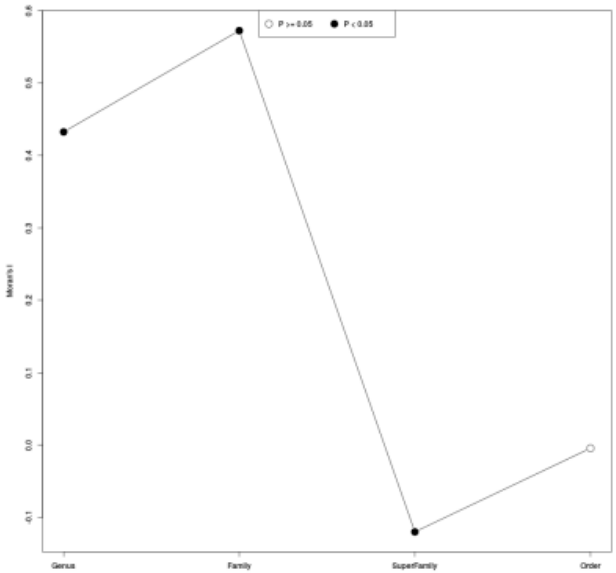
Testing of Moran's *I*

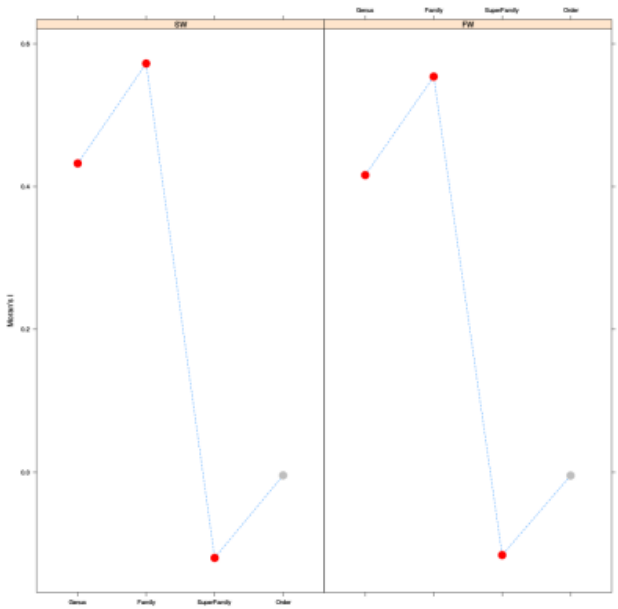
```
# Calculate Moran's I
# Slighly significant positive phylogenetic correlation among body mass
Moran.I(x=primates.body, weight=primates.weights,
  alternative="greater")
# Positive, but non-significant
Moran.I(x=primates.longevity, weight=primates.weights,
  alternative="greater")
# Test of Moran's with randomisation procedure
# Body is significant - nonrandom, longevity not (random)
gearymoran(bilis=primates.weights, X=data.frame(primates.body,
  primates.longevity), nrepet=1000)
# Test of Abouheif designed to detect phylogenetic autocorrelation in a
# quantitative trait - in fact Moran's I test using a particular
# phylogenetic proximity between tips
library(adephylo)
abouheif.moran(x=cbind(primates.body, primates.longevity),
  W=primates.weights, method="oriAbouheif", a=1, nrepet=1000,
  alter="greater")
```

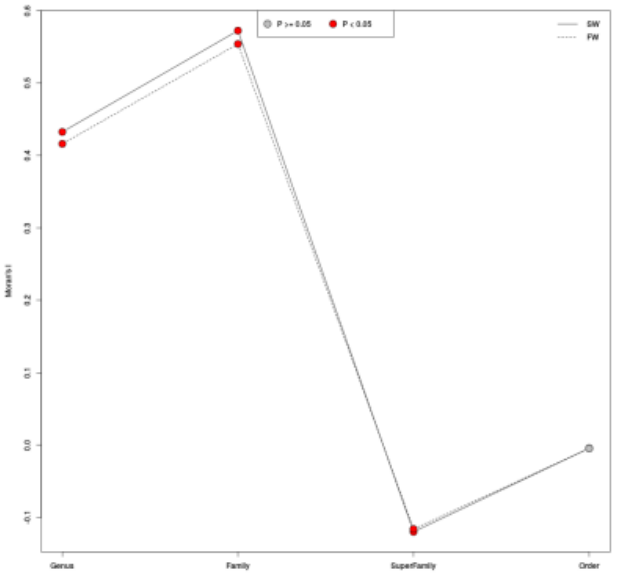
Correlogram to visualize results of phylogenetic autocorrelative analysis

```
data(carnivora) # Loads training data set
head(carnivora) # Look at the data
# Calculate the correlogram
carnivora.correlogram <- correlogram.formula
  (formula=SW~Order/SuperFamily/Family/Genus, data=carnivora)
carnivora.correlogram # See results
# Calculate the correlogram - test for both body masses
carnivora.correlogram2 <- correlogram.formula
  (formula=SW+FW~Order/SuperFamily/Family/Genus, data=carnivora)
carnivora.correlogram2 # See results
plot.correlogram(x=carnivora.correlogram, legend=TRUE,
  test.level=0.05, col=c("white", "black")) # Plot it
# Plot it - test for both body masses - two or one graph(s)
plot.correlogramList(x=carnivora.correlogram2, lattice=TRUE,
  legend=TRUE, test.level=0.05)
plot.correlogramList(x=carnivora.correlogram2, lattice=FALSE,
  legend=TRUE, test.level=0.05)
```

Correlograms of SW and SW+FW (in one or two graphs) depending on taxonomical level with marked significance



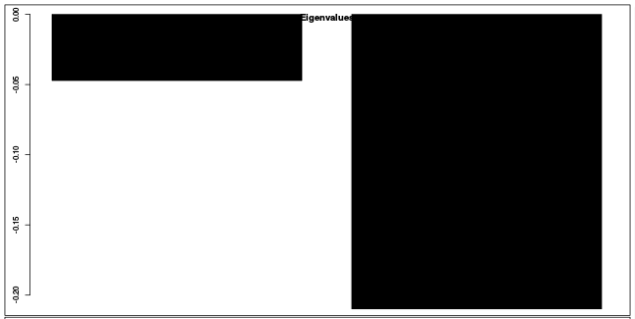


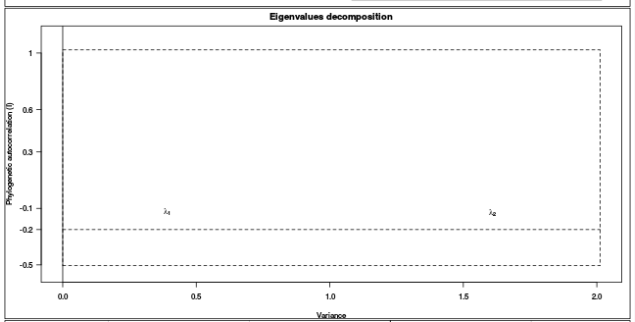


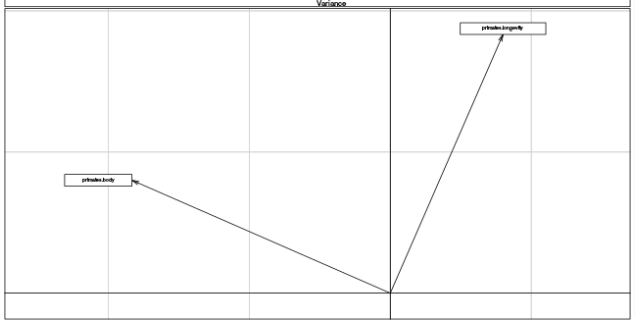
Phylogenetic principal component analysis

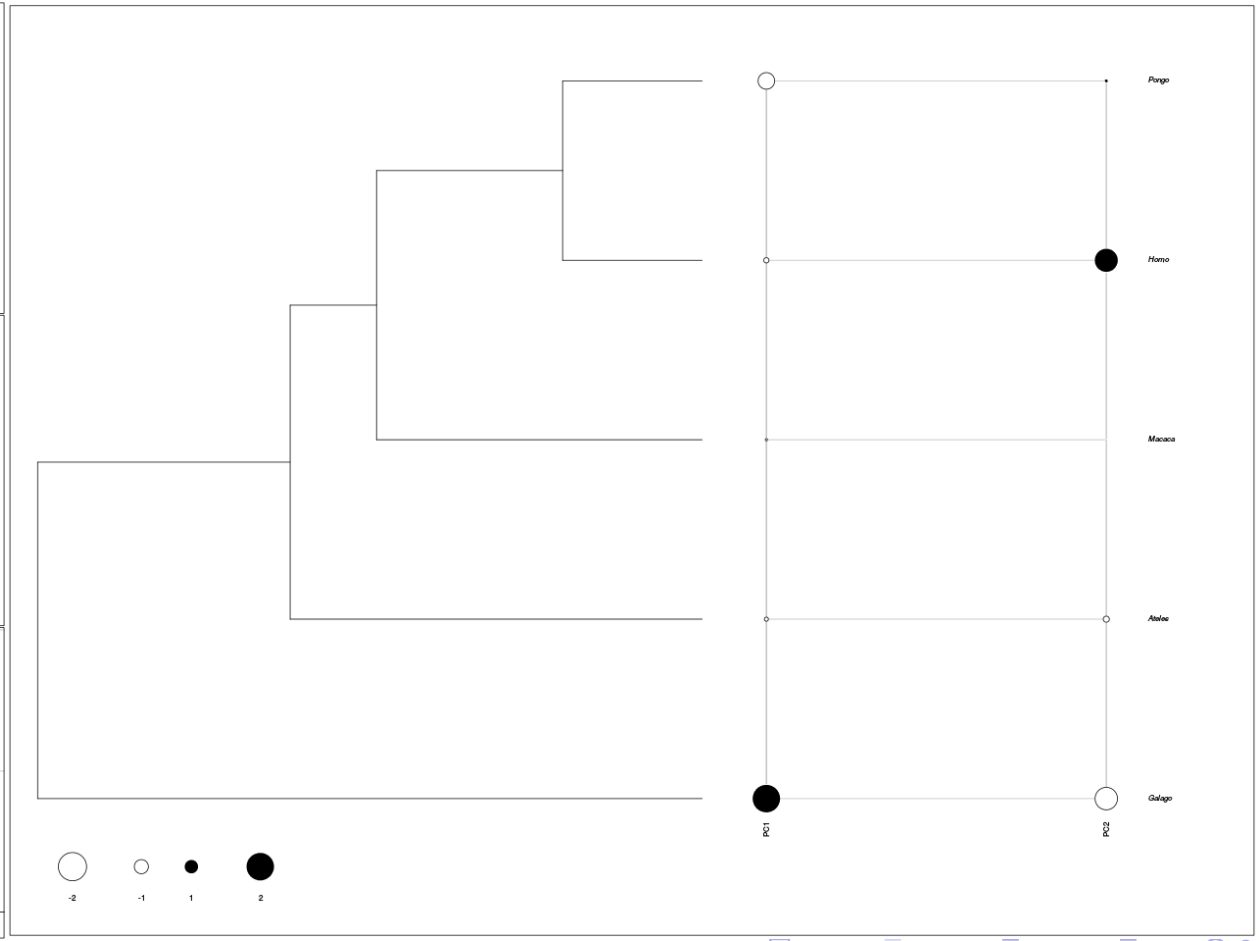
```
# Library needed to create phylo4d object required by ppca
library(phylobase)
# Calculate pPCA
primates.ppca <- ppca(x=phylo4d(x=primates.tree, cbind(
  primates.body, primates.longevity)), method="patristic",
  a=1, center=TRUE, scale=TRUE, scannf=TRUE, nfposi=1, nfnega=0)
# Print results
print.ppca(primates.ppca)
# See summary information
summary.ppca(primates.ppca)
# See PCA scores for variables on phylogenetic tree
scatter.ppca(primates.ppca)
# See decomposition of pPCA eigenvalues
screeplot.ppca(primates.ppca)
# Plot pPCA results - global vs. local structure, decomposition
# of pPCA eigenvalues, PCA plot of variables and PCA scores
# for variables on phylogenetic tree
plot.ppca(primates.ppca)
```

Plot pPCA results - global vs. local structure, decomposition of pPCA eigenvalues, PCA plot of variables and PCA scores for variables on phylogenetic tree









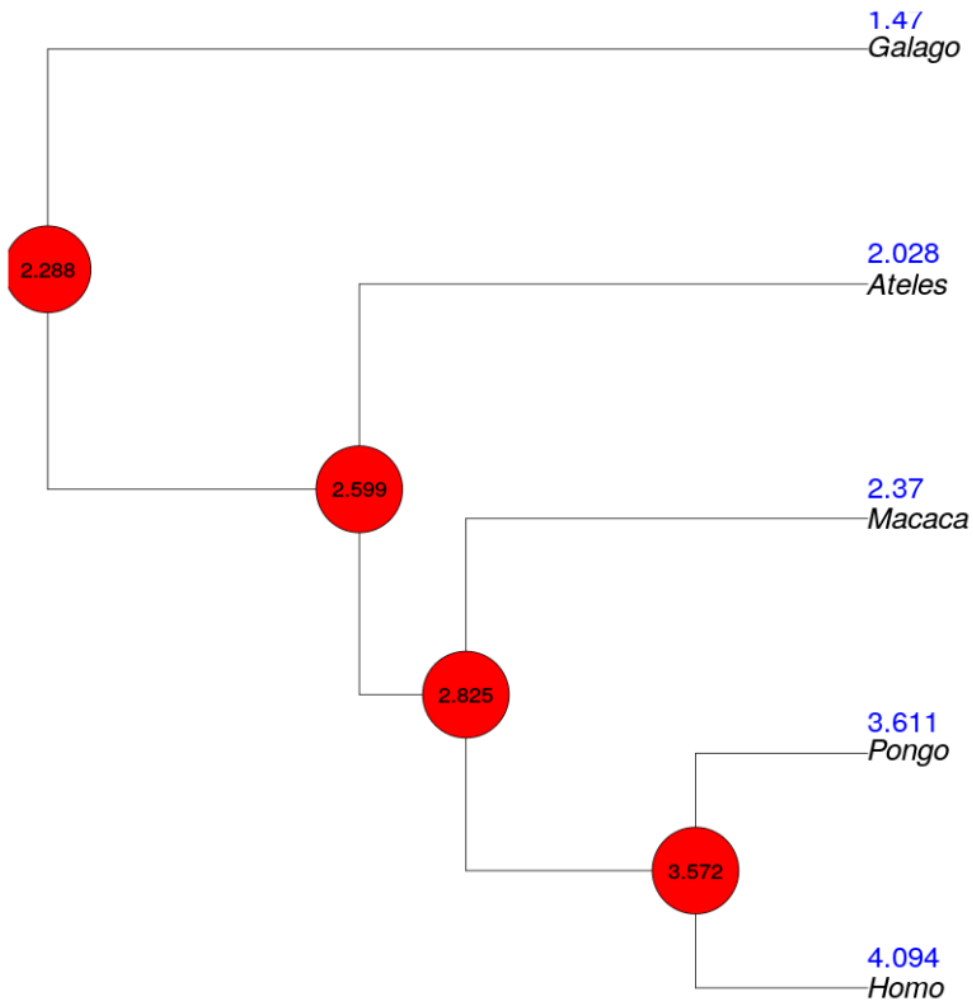
Ancestral state reconstruction

By default **ace** performs estimation for continuous characters assuming a Brownian motion model fit by maximum likelihood

```
# See ?ace for possible settings
primates.body.ace <- ace(x=primates.body, phy=primates.tree,
  type="continuous", method="REML",
  corStruct=corBrownian(value=1, phy=primates.tree))
# See result - reconstructions are in $ace
# To be plotted on nodes - 1st column are node numbers
primates.body.ace
# Plot it
plot(primates.tree, lwd=2, cex=2)
tiplabels(round(primates.body, digits=3), adj=c(0, -1),
  frame="none", col="blue", cex=2)
nodelabels(round(primates.body.ace$ace, digits=3),
frame="circle", bg="red", cex=1.5)
```

Other implementations are available in packages geiger (functions fitContinuousMCMC and fitDiscrete), phangorn and more in ape (MPR)

Ancestral state reconstructions of primates body weights



Citations

- To correctly cite R launch **citation()** and see information there — it is slightly different for every version of R
- Cite used packages — launch **citation("PackageName")** — if this information is missing, go to its manual page and/or homepage and find the information there
- Packages/functions commonly provide various methods to calculate desired task — check function's help page (**?FunctionName**) and find references there and cite them accordingly

Where to look for the help I

- R phylogeny mailing list https://stat.ethz.ch/mailman/listinfo/r-sig-phylo
- R genetics mailing list https://stat.ethz.ch/mailman/listinfo/r-sig-genetics
- Bioconductor mailing list https://stat.ethz.ch/mailman/listinfo/bioconductor and Bioconductor help pages http://master.bioconductor.org/help/
- R phylo wiki http://www.r-phylo.org/wiki/Main_Page
- R phylogenetics at CRAN http://cran.r-project.org/web/views/Phylogenetics.html
- Little Book of R for Bioinformatics http://a-little-book-of-r-for-bioinformatics.readthedocs.org/en/latest/

Where to look for the help II

- Adegenet web http://adegenet.r-forge.r-project.org/ and help forum https://lists.r-forge.r-project.org/cgi-bin/mailman/listinfo/adegenet-forum
- APE home page http://ape-package.ird.fr/
- Phytools http://phytools.org/ and its blog http://blog.phytools.org/
- Poppr documentation and forum http://grunwaldlab.cgrb.oregonstate.edu/poppr-r-package-population-genetics/manual-documentation-news-and-changes
- ade4 home page and documentation http://pbil.univ-lyon1.fr/ade4/home.php?lang=eng
- Phangorn resources http://cran.r-project.org/web/packages/phangorn/index.html
- Information and manual about pegas http://ape-package.ird.fr/pegas.html

Where to look for the help III

- R help mailing list https://stat.ethz.ch/mailman/listinfo/r-help (web interface http://r.789695.n4.nabble.com/)
- R announce mailing list https://stat.ethz.ch/mailman/listinfo/r-announce
- R manuals http://cran.r-project.org/manuals.html
- Books about R http://www.r-project.org/doc/bib/R-books.html
- R at StackOverflow StackExchange https://stackoverflow.com/questions/tagged/r
- R at CrossValidated StackExchange https://stats.stackexchange.com/questions/tagged/r
- The R journal http://journal.r-project.org/
- R-bloggers — aggregation of R blogs http://www.r-bloggers.com/

Where to look for the help IV

- R on The Molecular Ecologist http://www.molecularecologist.com/category/software/r/
- R tutorial http://www.r-tutor.com/
- Cookbook for R http://www.cookbook-r.com/
- spatial R https://sites.google.com/site/spatialr/
- Statistics with R http://zoonek2.free.fr/UNIX/48_R/all.html
- Springer R series https://www.springer.com/series/6991?detailsPage=titles
- Biostars — general bioinformatics forum https://www.biostars.org/
- Biology — general forum about biology at StackExchange https://biology.stackexchange.com/

Where to look for the help V

- Do not hesitate to ask on the forum or contact author of package with which you have problem, preferably through some public forum or mailing list, they usually respond quickly and helpfully...
- Uncle Google is your friend (*"how to XXX in R"*)...
- R packages commonly contain vignettes (tutorials) — list them by **vignette()** and load selected by **vignette("VignetteName")**
- List available training datasets from various R packages by **data()** and load selected by **data(DatasetName)**

Further reading


Michael J. Crawley
The R Book, second edition
Wiley, 2012


Emmanuel Paradis
Analysis of Phylogenetics and Evolution with R, second edition
Springer, 2012
http://ape-package.ird.fr/APER.html


Paurush Praveen Sinha
Bioinformatics with R Cookbook
Packt Publishing, 2014

The end

Our course is over...

...I hope it was helpful for You...

...any feedback is welcomed...

...happy ***R*** hacking...

Typesetting using XeLaTeX on openSUSE GNU/Linux