# Molecular data in R

## Phylogeny, evolution & R

Vojtěch Zeisek

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

January 18 to 20, 2017

# Outline I

# Outline II

# Outline III

# Outline IV

# Outline V

# The course information

- The course page: https://trapa.cz/en/course-molecular-data-r-2017
  - Česky: https://trapa.cz/cs/kurz-molekularni-data-r-2017
- Subject in SIS: https://is.cuni.cz/studium/eng/predmety/index.php?do=predmet&kod=MB120C16
  - Česky: https://is.cuni.cz/studium/predmety/index.php?do=predmet&kod=MB120C16
  - For students having subscribed the subject, active participation and presence whole course is required
- Working version is available at https://github.com/V-Z/course-r-mol-data – feel free to contribute, request new parts or report bugs

# Materials to help you...

- Download the presentation from https://soubory.trapa.cz/rcourse/r_mol_data_phylogen.pdf
- Follow all code we will use at https://soubory.trapa.cz/rcourse/course_commands.html
- Download the script from https://soubory.trapa.cz/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 UNIX end of lines (so that too silly programs like Windows Notepad won't be able to open it correctly)
  - The best is to open the script (or copy-paste the text) in e.g. RStudio or any other R GUI (see further) and directly work with it
  - Downloaded file must have suffix `*.r`, not `*txt`
  - Never ever open R script in software like MS Word – it destroys quotation marks and other things making script unusable

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

**We will go through...**

- Basic introduction into R
- Analyzing phylogeny and evolution and basic theory
  - DNA sequences, SNP, SSRs, AFLP, ...
  - NJ, UPGMA, PCoA, DAPC, Bayesian clustering, ML, maximum parsimony, ...
  - Character evolution, ancestral state reconstructions, ...
  - Alignments
  - Manipulations with trees
- Plotting
- Maps, spatial analysis, ...
- Basic creation of scripts
- And more...

**We will not dig deep into...**

- Detailed theory behind used methods
- Programming in R
- Other software related to the methods used (with exceptions of applications called from R)
- Other areas of R usage (ecology, biomedicine, ...)

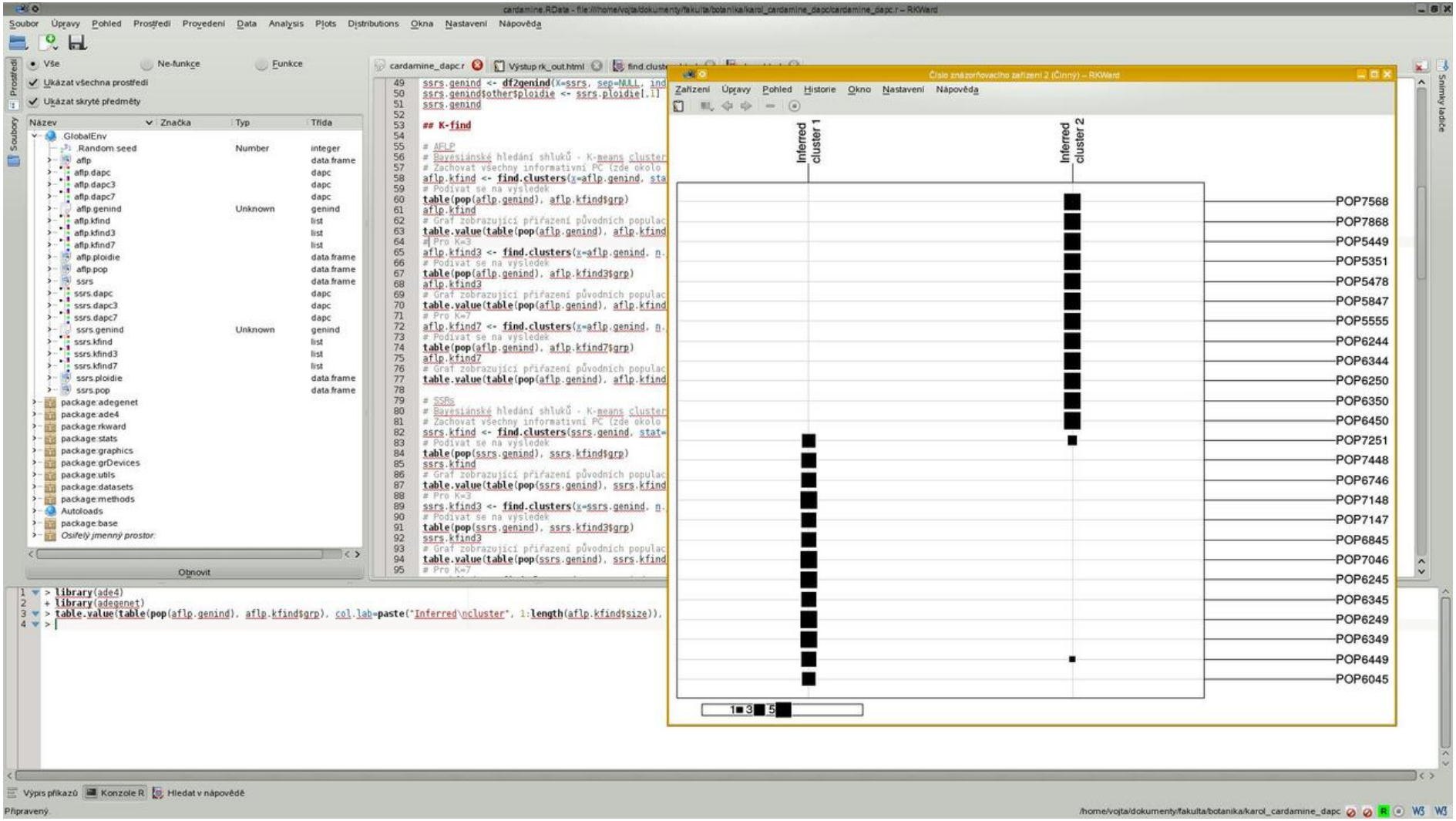
# About R

- Project for Statistical Computing
- Open-source – freely available with source code – anyone can use it and contribute its development
- Development is organized by non-governmental non-profit organization from Vienna
- Thousands of packages extending its functionality are available – all fields of computations in any scientific discipline
- Provides only command line interface – full control over the analysis, easy to rerun and/or modify analysis in the future, easy creation of scripts for batch analysis etc.
- Several projects provide convenient graphical user interfaces (GUI)
- More details: https://www.r-project.org/

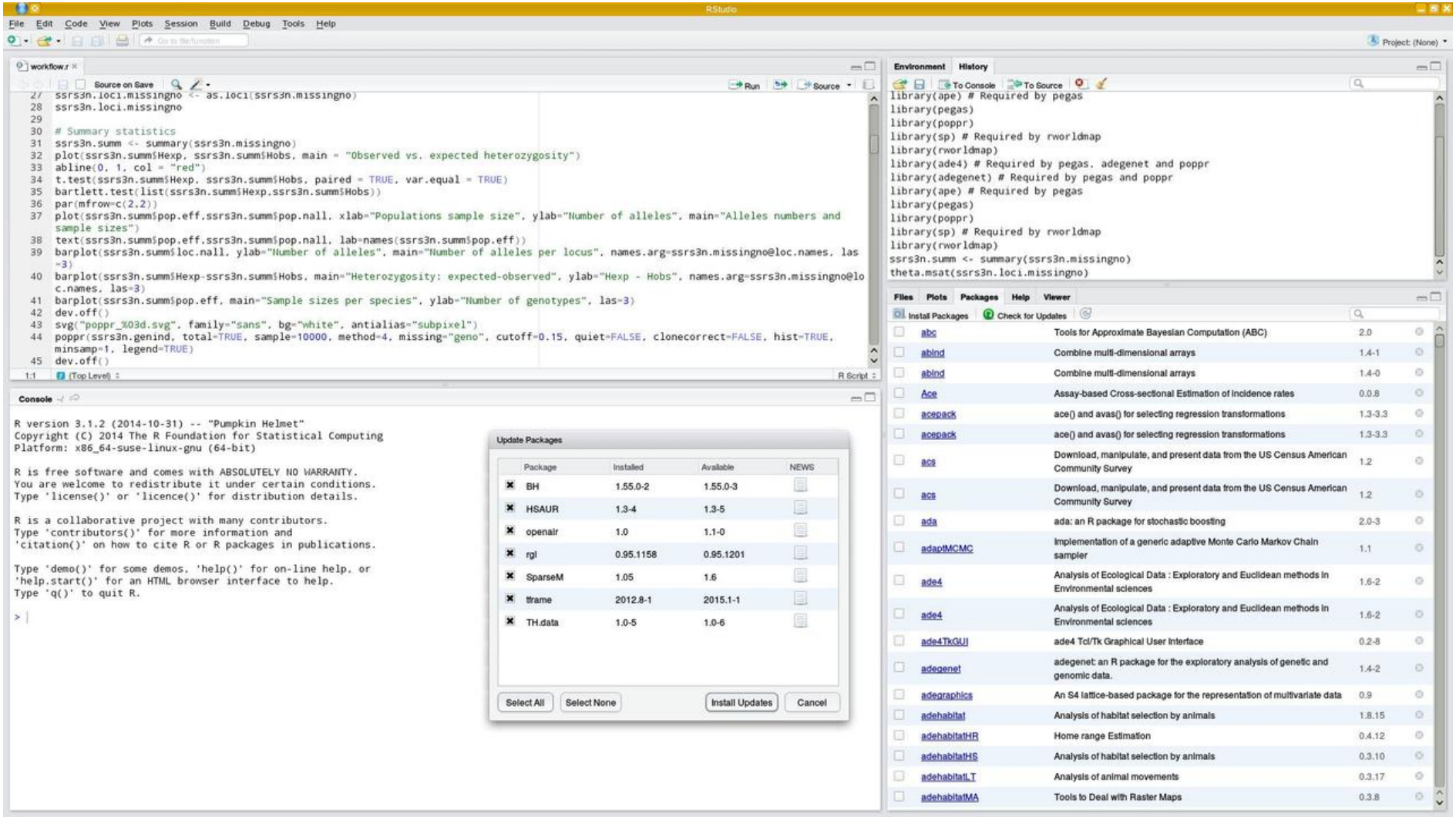
# Graphical user interfaces (GUI)

- Provide more comfortable interface for work with scripts (source code highlight), overview of loaded packages and variables, easier work with figures, ...
- RStudio https://www.rstudio.com/ – probably the most common, multiplatform, very powerful
- RKWard https://rkward.kde.org/ – feature very rich, developed mainly for Linux, available also for another operating systems
- R commander (Rcmdr) http://www.rcommander.com/ – multiplatform, not so rich as previous
- Java GUI for R http://rforge.net/JGR/ – Java (multiplatform, but with all Java issues like memory consumption)
- Tinn-R (Windows only) https://sourceforge.net/projects/tinn-r/ and http://nbcgib.uesc.br/lec/software/editores/tinn-r/en
- Pick one you like (from above list or any else) 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
tiff 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 https://cran.r-project.org/
- Download appropriate version and install as usual
- Download and install selected GUI (not required, but highly recommended)
- Most of packages are available as pre-compiled and can be immediately installed from R – it is 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
- RStudio is available from its download page
- RKWard is also available for Windows and Mac OS X, but it requires some work to install it

# Linux – general

- 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, but install basic Linux packages for building of C, C++, Fortran, ...
- Compilation takes longer time and there are sometimes issues with missing dependencies (tools required by particular packages), but it can then provide higher performance...

# Linux – Debian/Ubuntu and derivatives like Linux Mint or Kali Linux

- Install package `build-essential` (general tools to compile software, including R packages)
- Debian (and derivatives): follow instructions at https://cran.r-project.org/bin/linux/debian/
- Ubuntu (and derivatives): follow instructions at https://cran.r-project.org/bin/linux/ubuntu/
- As `<my.favorite.cran.mirror>` select `https://cran.wu.ac.at/`, see CRAN Mirrors at https://cran.r-project.org/mirrors.html
- Install packages `R-base` (the R), `R-base-dev` (required to compile additional R packages – only some are available in repositories) and optionally `rkward` and/or `rstudio`
- RStudio is also available from its download page

# Linux – openSUSE and SUSE Linux Enterprise

- See instructions at https://cran.r-project.org/bin/linux/suse/
- Add repository/ies for appropriate version of your distribution
  - http://download.opensuse.org/repositories/devel:/languages:/R:/patched/ (daily updated) or/and
  - http://download.opensuse.org/repositories/devel:/languages:/R:/released/ (updated with new R release)
- Install packages `R-base` (the R), `R-base-devel` (required to compile additional R packages – only some are available in repositories) and optionally `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)
- RStudio is also available from its download page

# Linux – RedHat, Fedora and derivatives like CENTOS, Scientific Linux, etc.

- See instructions at https://cran.r-project.org/bin/linux/redhat/README
- Install packages `R-core` (the R), `R-core-devel` (required to compile additional R packages – only some are available in repositories) and optionally `rkward`
- RStudio is available from its download page

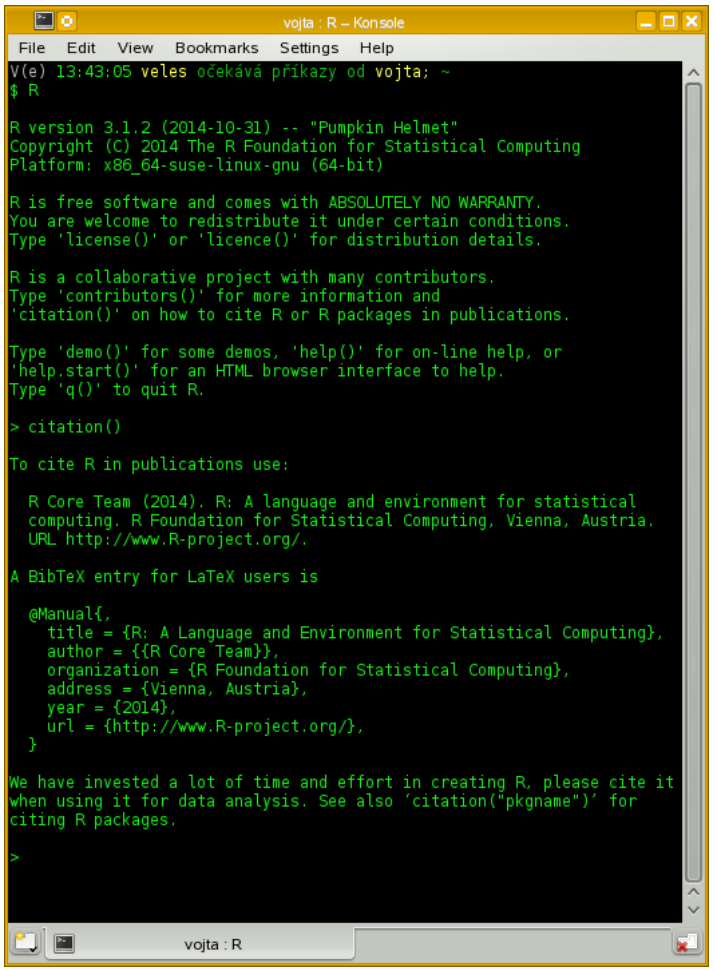
# Sources of R packages

- R CRAN https://cran.r-project.org/ – main and largest source of R packages (almost 10,000 packages + many orphaned and archived – abandoned by developers, might be working)
- Bioconductor https://bioconductor.org/ – mainly bioinformatics packages, genomic data (over 1,400 packages)
- R-Forge https://r-forge.r-project.org/ (over 2,000 packages)
- RForge https://www.rforge.net/ (much smaller)
- And more...
- Some packages are available from more resources
- Same name for function can be used in different packages (there is no central index) – to distinguish them 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...) – see further

# First steps in R

## Recommended is usage of GUI (RKWard or RStudio)

- Linux (UNIX): open any terminal, type `R` and hit Enter
- Windows and Mac: find it as normal application in menu
- Type commands to work...
- Ever wished to be Harry Potter? Secret spells make magic operations :-)
- Use arrows up and down to navigate in history
- `Ctrl+R` works as reverse search – searches text in history



# How it works

- General look of R commands:
- `function(argument1="SomeName", argument2=SomeVariable, argument3=8)`
- `ModifiedObject <- SomeFunction(argument1=MyData, argument2=TRUE)`
- New/modified object (with data, ...) is on the left: "`<-`" says to insert result of the function `SomeFunction` on the right into the object `ModifiedObject` on the left
- 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)`
- Some arguments are required, some optional

# 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.packages("sos") # More comprehensive search from packages
library(sos)
findFn("function") # Search for function name
```

`?` shows help for questioned function (in console type `q` to close it):

- 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/UNIX, R starts in current directory (use `cd` to change it before launching R)
- Set and check working directory in R:

```
setwd("/some/path/") # Or "~/...". In Windows "C:\..."
getwd() # Verifies where we are
dir() # Lists files and folders on the disk
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 file `.Rhistory` file within working directory (by default hidden in Linux/Mac OS X)
- When closing R by `q()` you can save all R data in `.RData` (and command history in `.Rhistory`) file(s) and it/they can be loaded next time
- RStudio and RKWard help with this very much

# Importance of working directory

- Default place to load/save, import/export data/results
  - It changes paths – one of the most common mistakes – something is not found because of wrong path
  - Private folder for particular R project (task) prevents unwanted inferences with another tools/projects
- Without saving and loading the R data next time, it is not possible to do any longer work or to check the work in the future
- Get used that R **always** work in some directory and by default saves/loads files there
- RStudio and RKWard also save session information (list of opened files, ...) – very convenient
- Regularly save your work to prevent looses in case of crash or any other accident

# 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, colors, etc.
- More "advanced" objects to store plots, genetic data, ...
  - Commonly called "**S3**" and "**S4**" objects in R terminology
  - Technically commonly just lists putting together various information
  - We will meet 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 data frame (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 (more data), 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) convertible among each other...

To get information about content of each data type see

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

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(5, 6, 7, 8, 9) # Creates vector (see also ?rep)
x # Print "x" content
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
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 and text with
colnames(y) # Prints column names, use rownames() in very 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 (NA = not available = missing)
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
```

# Repositories

- Repositories (internet directories full of R packages – slide 19) can be set via `options(repos=c())` or as `repos` parameter for each `install.packages()` command (slide 35 and onward)
- Repositories doesn't have to be set as global options, e.g. Bioconductor has its own way to manage packages

## Installation of packages in GUI

- **RStudio:** set repositories by command from slide 35 and in bottom right pane select **Packages** and click on **Install Packages...**
- **RKWard:** go to menu **Settings** | **Configure 'RKWard'** and select **R-Packages**. Add URLs of repositories from slide 35. **OK**. Go to menu **Settings** | **Manage R packages and plugins...**, click to **Install...**, select and install desired packages...

# Theory for packages and their management

- Standalone plain R doesn't have enough tools for most of scientific disciplines – only basic methods and tools for programmers, including for package management
- Users/developers contribute by making extra packages extending computational possibilities – one of biggest R advantages – it then has unlimited possibilities
- R has infrastructure for maintaining (for developers) and installing (for users) packages – the CRAN repository
- For various reasons, some people build their own infrastructures to maintain and install R packages – compatible with R, bud separated
- User has basically two options
  1. Set all repositories in R and use basic commands to install packages (slide 35)
  2. Specify non-CRAN repository every time installing from it (e.g. slide 41) or use special tools (e.g. for Bioconductor – slide 39)

# Set repositories

```
# Basic package installation
install.packages("PackageName") # Case sensitive!
?install.packages # Shows all available parameters (options)
# We will need extra repositories. For Bioconductor keep version for
# your R version (BioC 3.4 for R 3.4, see https://bioconductor.org/).
options(repos=c("https://cran.wu.ac.at/", "https:/rforge.net/",
  "https://r-forge.r-project.org/", "https://bioconductor.
  statistik.tu-dortmund.de/packages/3.4/bioc", "https://bioconductor.
  statistik.tu-dortmund.de/packages/3.4/data/annotation",
  "https://bioconductor.statistik.tu-dortmund.de/packages/3.4/data/
  experiment", "https://bioconductor.statistik.tu-dortmund.de/
  packages/3.4/extra"))
getOption("repos") # Shows actual repositories
options() # Generic function to modify various settings
?options # Gives details
```

- 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

```
# Simplest usage
install.packages("PackageName") # Case sensitive!
# Install packages needed for the course
install.packages(pkgs=c("BiocGenerics", "Biostrings", "Geneland",
  "IRanges", "MASS", "PBSmapping", "ParallelStructure", "RandomFields",
  "RandomFieldsUtils", "RgoogleMaps", "Rmpi", "S4Vectors",
  "TeachingDemos", "XML", "XVector", "ade4", "adegenet", "adephylo",
  "akima", "ape", "brew", "caper", "colorspace", "combinat", "corrplot",
  "diveRsity", "fields", "geiger", "ggplot2", "gplots", "hierfstat",
  "lattice", "mapdata", "mapproj", "maps", "maptools", "muscle",
  "mvtnorm", "nlme", "pegas", "permute", "phangorn", "phylobase",
  "phytools", "picante", "plotrix", "polysat", "poppr", "rworldmap",
  "seqinr", "shiny", "sos", "sp", "spdep", "spam", "vegan"),
  repos=getOption("repos"), dependencies=TRUE)
?install.packages # See for more options
# Installed packages are "inactive" - the mus by loaded to use them:
library(package) # Loads installed package (we will do it on the fly)
update.packages(repos=getOption("repos")) # Updates installed packages
```

# Install packages without setting the repositories

- If repositories from slide 35 are not set (for any reason), it is possible to install in several steps packages from main repository (CRAN) and from another sources (following slides)
- This is the basic and default the most common usage

```
install.packages(pkgs=c("Geneland", "MASS", "PBSmapping",
  "RandomFields", "RandomFieldsUtils", "RgoogleMaps", "Rmpi",
  "TeachingDemos", "XML", "ade4", "adegenet", "adephylo", "akima",
  "ape", "brew", "caper", "colorspace", "combinat", "corrplot",
  "diveRsity", "fields", "geiger", "ggplot2", "gplots", "hierfstat",
  "lattice", "mapdata", "mapproj", "maps", "maptools", "mvtnorm",
  "nlme", "pegas", "permute", "phangorn", "phylobase", "phytools",
  "picante", "plotrix", "polysat", "poppr", "rworldmap", "seqinr",
  "shiny", "sos", "sp", "spdep", "spam", "vegan"), dependencies=TRUE)
update.packages(ask=FALSE) # Update installed (CRAN by default) packages
```

# Install phyloch package

## Example of installation of package not available in any repository

- Check http://www.christophheibl.de/Rpackages.html
- Package phyloch is similar to ips from same author (but some functions behave differently) – both are great for usage of external applications within R

```
# If not done already, install required packages first
install.packages(pkgs=c("ape", "colorspace", "XML"),
  dependencies=TRUE)
# It is possible to specify direct path
# Local or web URL - be careful about correct path) to package source
install.packages(pkgs="http://www.christophheibl.de/
  phyloch_1.5-3.tar.gz", repos=NULL, type="source")
```

# Bioconductor

- Tools for analysis of genomic data, see https://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("https://bioconductor.org/biocLite.R")
?biocLite # Get help how to use it
# Install packages
biocLite(c("package1", "package2", "..."))
biocLite() # Update Bioconductor packages (within same R version)
# Upgrades installed Bioconductor packages to new R release
biocLite("BiocUpgrade") # E.g. from R 3.2 to 3.3
```

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

# Bioconductor packages for the course

- If repositories are not set via command on slide 35, it is possible to use Bioconductor's own installation method using function `biocLite`
- If Bioconductor repositories are set manually, user must select correct version number
- List of available Bioconductor mirrors is at https://bioconductor.org/about/mirrors/

```
# Install needed Bioconductor packages
biocLite(pkgs=c("BiocGenerics", "Biostrings", "IRanges", "S4Vectors",
  "XVector", "muscle"))
# From time to time update Bioconductor packages
biocLite()
# When upgrading to new version of R (e.g. from 3.2 to 3.3) upgrade by
biocLite("BiocUpgrade")
```

# Bioconductor and others – differences from another repositories

```
# Standard installation
install.packages(pkgs=c("adegenet", "poppr", "phytools"))
update.packages(ask=FALSE) # Update packages
# Installation from custom repository (example for our course)
install.packages(pkgs="ParallelStructure",
  repos="https://r-forge.r-project.org")
?install.packages # See help for details
# Bioconductor - if https fails, use http
source("https://bioconductor.org/biocLite.R")
?biocLite
biocLite(pkgs=c("Biostrings", "seqinr")) # Install package(s)
biocLite() # Update Bioconductor packages
```

- Bioconductor has it own installation method
- Its repositories can be set in R and its packages can then be installed as usually with `install.paskages()` — Although Bioconductor developers recommend usage of `biocLite` function...

# Non-R software

- We use several software packages outside R
  - R functions can use this software
  - External software can be used (depending on R package) to create/modify R object, or just as different method for (batch) usage of the software
  - User must install this software manually
- Structure http://pritchardlab.stanford.edu/structure.html (optionally also CLUMPP and distruct – not part of the course)
- MAFFT http://mafft.cbrc.jp/alignment/software/
- MUSCLE http://www.drive5.com/muscle/
- Clustal (W/X; Omega is not used in the course) http://clustal.org/

# Our data... I

- Work with microsatellites is in most cases (except some methods taking advance from microsatellite mutational nature) same as with presence-absence data and methods can handle both data types in nearly same fashion
  - Examples are shown mainly with microsatellites, but AFLP and another presence-absence (1/0) data are used in same way – try it
- Distance-based methods are same regardless input data on the beginning (microsatellites, AFLP, DNA sequences, ...)
- Extraction of SNP from DNA is useful in case of huge datasets – for smaller datasets it is not necessary

**Always save your work!**

We will use data objects during whole course – all the time save your workspace! Use possibilities of your GUI or `save`/`load` functions.

# Brief overview of molecular data types I

## Sorted with respect to usage in R

- Isozymes – forms of proteins differing in electrophoresis by their weight and/or charge
  - Typically coded as presence/absence (1/0) data
- Fragmentation data – length polymorphism
  - Codominant data – e.g. microsatellites (SSRs – Simple Sequence Repeats)
    - Variability in number of short (usually 1-3 bp) oligonucleotide repeats (ATAT vs. ATATAT, typically ca. 25-250 repeats) bordered by unique primer sequences
    - Very variable, fast evolving, species-specific primers needed
    - Mainly for population genetics, relationships among closely related species
    - Similarly ISSRs (Inter Simple Sequence Repeats)
  - Presence/absence (1/0) dominant data

# Brief overview of molecular data types II

## Sorted with respect to usage in R

- The allele **is** or **is not** present – it is impossible to distinguish heterozygots from dominant homozygots
- AFLP (Amplified Fragment Length Polymorphism) – very variable, technically complicated, nowadays bit expensive and outdated
- Simpler methods RAPD (Random Amplified Polymorphic DNA) and PCR-RFLP (Polymerase Chain Reaction-Restriction Fragment Length Polymorphism) are not used anymore at all

- Protein sequences – not used in the course
  - Apart similar usage as with DNA/RNA (sequence analysis) it is possible to work with the structure and conformation of the proteins
  - R has plenty of packages for specialized protein analyses
- Nucleic acid sequences (in nearly any form) – DNA or RNA
  - From "classical" Sanger sequencing – long individual reads
  - From modern HTS (NGS) – 454 pyrosequencing, Illumina, ...

# Brief overview of molecular data types III

## Sorted with respect to usage in R

- Probably most used are RADseq scanning whole genome, HybSeq using specific probes to sequence only single/low-copy nuclear markers, Genome Skimming getting the most abundant part of the genome (plastid and mitchodrial sequences and ITS1-5.8S rRNA-ITS2 region); and their variants
- There are special tools to process raw data from the machines – not part of the course

- Whole sequences (probes/loci or longer assembled regions) or SNPs (Single Nucleotide Polymorphism – only polymorphic sites are retained)

- Most of methods are mathematically well defined for haploids and/or diploids, higher ploidies or mixing of ploidies is not always possible
- Most of methods shown in the course work with more data types – not everything is shown
- For details about the molecular markers check specialized course like Use of molecular markers in plant systematics and population biology

# Notes about paths to import the data I

- Generally, R can accept nearly any local or web location
- If unsure where you are, open any file manager, go to the R working directory (verify with `getwd()` and `dir()`) and verify where everything is
- Web locations start with `https://`, `http://` or `ftp://`, e.g. `FileParameter="https://server.cz/directory/file.txt"`
- Local paths (within one computer) can be absolute or relative
  - Absolute paths start from the top of files hierarchy: on UNIX (Linux, Mac OS X, ...) it use to look like `/home/USER/`, on Windows like `C:\` (e.g. `FileParameter="/path/to/some/file.txt"`)
  - Relative paths start in current directory (so **no** with `/` or `C:`)
    - In the easiest case the input file is in same directory as is R's working directory – verify by `getwd()` and `dir()` – you then need to specify only the filename (e.g. `FileParameter="SomeFile.txt"`)

# Notes about paths to import the data II

- For subdirectory start with its name (**no** with `/` or `C:`), e.g. `FileParameter="subdirectory/another/directory/file.txt"`
- When going directory up, use one `..` for each level, e.g. `FileParameter="../upper/directory/file.txt"`
- On UNIX (Mac OS X, Linux, ...) tilde `~` means user's home directory (e.g. `/home/USER/`), so `FileParameter="~/some/file.txt"` is same as `FileParameter="/home/USER/some/file.txt"`

- If loading data from computer, carefully check the paths or use function `file.choose()` to interactively pick up the file anywhere in the computer – it can replace nearly any filename parameter (e.g. `FileParameter="file.choose()"`

- Some R functions have problems with spaces and special (non-alphanumeric and accented) characters – try to avoid them
- One of the most common source of errors – **when the command fails, double check paths** (and Internet connection, if applicable) – for another common problems see slide 267

# 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 are analyzed in same way
- Most of methods are available for polyploids (although not all)
- Some methods are unavailable for dominant (presence/absence) data
- Mixing of ploidy levels is tricky (but possible) – it doesn't matter when data are encoded as PA, otherwise it is mathematically 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 ...
... ... ...     ...     ...     ...     ...     ...     ...
```

## Loading the data:

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

First line starts with empty cell (if **header** is presented), there can be any extra column, take care about `col.loci`. `row.names` are individual names (first column). Take care about `loci.sep` (here TAB – `\t`) and `allele.sep` (here `/`) 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$pop # "$" separates extra slots within object
```

**Source data:**

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

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

- Coordinates can be in any projection or scale – according to aim
- 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
# Conversion to genpop - for population-level analysis
hauss.genpop <- genind2genpop(hauss.genind, process.other=TRUE)
# See result
hauss.genpop
# Removes missing data - see ?missingno for types of dealing them
# Use with caution! It modifies original data!
hauss.genind.cor <- missingno(pop=hauss.genind, type="mean", cutoff=0.1,
  quiet=FALSE)
# Convert corrected genind to loci
hauss.loci.cor <- genind2loci(hauss.genind.cor)
# Writes loci file to the disk
write.loci(hauss.loci.cor, file="hauss.loci.cor.txt",
  loci.sep="\t", allele.sep="/")
```

# 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 ha/diploid data
read.genepop() # adegenet - reads *.gen files, only ha/diploid data
read.structure() # adegenet - reads *.str files, only ha/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 (e.g. `quote="\""`)? How are columns separated (e.g. `sep="\t"`)? Is there a header with names of populations/loci/whatever (`header=T/F`)? What is decimal separator (e.g. `dec="."`)? Are there row names (used typically as names of individuals; e.g. `row.names=1`)? Check data after import!

# Import of polyploid microsatellites

- `adegent`, `poppr` and related packages can for most of functions handle any ploidy level (including mixing of ploidy levels, but not for all analysis)
- `polysat` package 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 (it can be combined in one file – next slide)

**Triploid microsatellite data:**

```
1         msat58      msat31      msat78      msat61      ...
2  ala1   124/124/124 237/237/237 164/164/172 136/136/138 ...
3  ala2   124/124/124 237/237/237 164/164/172 136/136/138 ...
4  ala4-1 124/124/124 237/237/237 164/164/172 136/136/138 ...
5  ...    ...         ...         ...         ...         ...
```

Triploid species of *Taraxacum* sect. *Taraxacum*

# How to import polyploid microsatellites

```
# Import of table is as usual. Last column contains populations
tarax3n.table <- read.table("http://soubory.trapa.cz/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"]], 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 or any other 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  ...
...   ...   ...
```

AFLP data of *Cardamine amara* group

**Individual's populations:**

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

Just list of populations for respective individuals...

- 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="https://soubory.trapa.cz/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="https://soubory.trapa.cz/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="", ind.names=NULL,
  loc.names=NULL, pop=amara.pop[,1], missing=NA, type="PA")
indNames(aflp.genind) <- amara.aflp[,1] # Add individual names
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 (PA)

```
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
alleles2loci() # pegas - transforms a matrix of alleles into "loci"
# seppop and seploc return lists of genind objects - for further
# analysis using special functions to work on lists (see further)
# read manual pages (?...) of the functions before usage
```

- `alleles2loci()` is very useful when each allele is in separated columns (not like in our case where one column contains one loci with all alleles) – saves time needed to change input file formatting

# 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()` (or read the data with `read.table()`, convert into matrix and use `alleles2loci()`)
- **Do not use** underscores ("`_`") or minuses to name objects in R – only numbers, Latin letters or dots
- `read.loci()` sometimes doesn't work correctly on AFLP or polyploid microsatellites – try `read.table()` instead...
- Genind object (since Adegenet version 2) is able to store mixed ploidy data, but not all analysis are able to handle that...

# Import of DNA sequence data I

```
# Reading FASTA (read.dna() reads also another formats, see ?read.dna)
# Sequences of flu viruses from various years from USA
# (Adegenet toy data)
usflu.dna <- read.dna(file="http://adegenet.r-forge.r-project.org/
  files/usflu.fasta", format="fasta")
class(usflu.dna) # Check the object
usflu.dna # Check the object
# Another possibility (only for FASTA alignments, same result):
usflu.dna2 <- fasta2DNAbin(file="http://adegenet.r-forge.r-project.org
  /files/usflu.fasta") # Normally keeps only SNP - see ?fasta2DNAbin
class(usflu.dna2) # Check the object
usflu.dna2 # Check the object
as.character(usflu.dna2)[1:5,1:10] # Check the object
dim(usflu.dna2) # Does it have correct size?
# Read annotations
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
```

# Import of DNA sequence data II

```
# 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"]])
# read.fasta() from seqinr package reads DNA or AA in FASTA format and
# returns a list (DNAbin is for us now better choice)
usflu.dna3 <- seqinr::read.fasta(file="http://adegenet.r-forge.
  r-project.org/files/usflu.fasta", seqtype="DNA")
class(usflu.dna3)
length(usflu.dna3) # How many sequences we have in the list
usflu.dna3
# Convert into DNAbin class (technically, DNAbin is a list)
class(usflu.dna3) <- "DNAbin"
# Read sequence data in NEXUS
read.nexus.data(file="sequences.nex")
```

- RNA or protein sequences can be handed in same way

# Import sequences from GenBank

- Data from https://www.ncbi.nlm.nih.gov/popset/608602125 (*Meles meles*)

```
# Importing sequences according to sequence ID
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 we use seqinr (next slide)

# Query on-line sequence databases

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

# Importing SNP

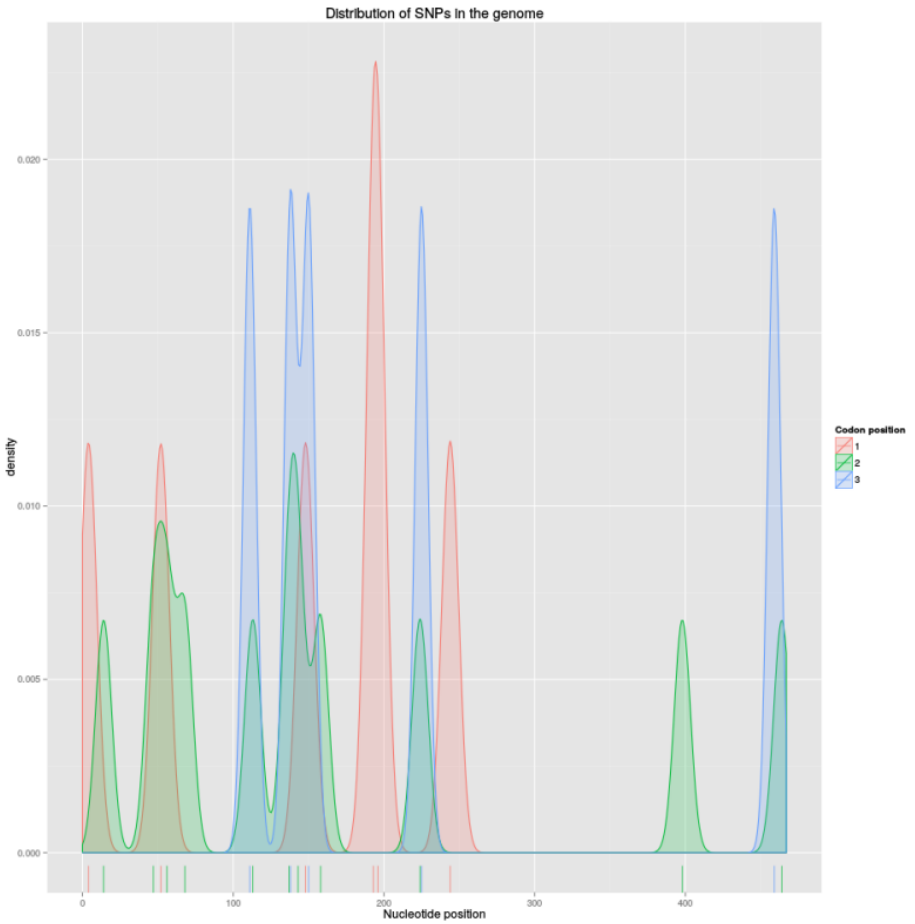
- Import from PLINK requires saving of data with option "-recodeA"
- `read.PLINK(file="PLINKfile", ...) # See ?readPLINK`
- Extracting SNP from alignments reads FASTA alignments and keep only SNPs. The method is relatively efficient even for large data sets with several genomes:

```
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, keep data as genind as it is more information-rich – genlight is more efficient for large data ($>$ ~100,000 SNPs)
- Adegenet has custom format to store SNP as plain text file and function `read.snp` to import it into `genlight` object – check Adegenet tutorial **genomics** first, `?read.snp # See the options`

# 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))
```



# Checking sequences

```
# Nucleotide diversity
pegas::nuc.div(x=meles.dna)
# Base frequencies
ape::base.freq(x=meles.dna)
# GC content
ape::GC.content(x=meles.dna)
# Number of times any dimer/trimer/etc oligomers occur in a sequence
# Note: count() requires single sequence as DNAbin/character
seqinr::count(seq=meles.nogaps[["KJ161328.1"]], wordsize=3)
# View sequences - all must be of the same length
image(x=usflu.dna, c("a", "t", "c" ,"g", "n"), col=rainbow(5))
# Function "image" requires as input matrix
# So that sequences must be of same length
image(x=as.matrix(meles.dna), c("a", "t", "c" ,"g", "n"),
  col=rainbow(5))
# Direct function to display the sequences
image.DNAbin(x=usflu.dna)
image.DNAbin(x=as.matrix(meles.dna))
```

# *Meles* sequences

# 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 (roughly, genind is inefficient since dozens or hundreds thousands of SNPs) – try genind and when work fails because of not enough computer resources, go on with genlight
- 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 optimized 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 as well as genind allow varying ploidy level
- Functions working with genlight use to use parallelisation to speed up operations – this commonly doesn't work properly on MS Windows

# Export data

```
# Convert genind into DF using genind2df()
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 DNA sequnces as NEXUS
write.nexus.data(x=meles.dna, file="meles.nexus", format="dna")
# 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")
```

# Introductory overview of statistics and methods I

- Selected method depends on data type, question to answer, ...
  - Check assumptions and requirements of the methods before usage
  - Think if the method answers your question
- **Population-genetic indices** – from slide 76
  - Huge number...
  - Characterize differences among individuals/groups or genetic variability on various levels (within/among individuals/populations, ...)
  - One number tries to describe whole situation – always very rough
  - Description of heterozygosity, allelic richness, distribution of multi locus genotypes among populations, level of inbreeding, ...
- **Distance-based methods** – from slide 92
  - It is crucial to select appropriate distance method for given data
  - Usually require the distance matrix to be Euclidean
  - Distance matrix has one single number (index) for each pair of comparisons (individuals, populations) – rough

# Introductory overview of statistics and methods II

- Generally, the matrices describe pairwise similarities among the individuals/populations
- Distance-based methods are phenetic
  - Based on similarity (described by the matrix), not on any (evolutionary) model
  - The matrix based on genetic data is supposed to well reflect the genetic similarity, thus real relationships among individuals/populations
- **Hierarchical clustering** – from slide 102
  - Several methods clustering individuals according to their (dis)similarity from top or down into clusters
  - (Un)weighted per-group mean average (**U/WPGMA**) and others
  - Used more in ecology, in genetic data not so much anymore (following methods use to produce better results)
- **Neighbor-Joining (NJ)** – from slide 108
  - A tree starting from the two most similar individuals and connecting in the next steps next and next the most similar individual
  - In some cases artificially chains individuals

# Introductory overview of statistics and methods III

- Several methods try to improve it – slide 120
- **Principal Coordinates Analysis (PCoA)** – from slide 121
  - The most common method of multivariate statistics for genetic data
  - Shows individuals in 2D scatter plot to retain maximum variability (by finding correlations among loci)

- **Minimum Spanning Network (MSN)** – slide 107
  - Simple network connecting the most similar genotypes/haplotypes
  - Useful for clones, cpDNA, mtDNA, ...

- **Multivariate statistics**
  - Two variables are easily displayable in 2D xy-scatter plot (we can calculate correlation, whatever)
  - In molecular data, each locus is more or less independent variable – 1000 bp alignment has 1000 variables: How to display plot with 1000 axes to be able to really see something?

# Introductory overview of statistics and methods IV

- Methods like Principal Component Analysis (**PCA**), Non-Metric Multidimensional Scaling (**NMDS**) or **PCoA** look for correlations between pairs of variables to reduce them into new variables – after many steps new uncorrelated variables retaining maximum of original variability are constructed
- New variables are sorted according amount of variability they show (the decrease is very steep – first 1-4 axes are usually enough) – it is possible to display xy-scatter plot showing most of variability of the data
- Good for data display and creation of hypotheses – not to verify them (there is no statistical test)
- Data are commonly scaled – all variables are in same scale

- **Maximum Parsimony (MP)** – from slide 214
  - Generally, the methods are looking for the most simple solution under given model, e.g. to construct phylogenetic tree requiring the lowest number of changes

# Introductory overview of statistics and methods V

- It is easy to score how good the solution is, but computationally demanding to find it

- **Maximum Likelihood (ML)**
  - Methods look for the most likely (probable) solution of the data under given model, e.g. the most likely tree under given mutational model
  - It is easy to score how good the solution is, but computationally demanding to find it

- **Bayesian statistics**
  - Based on Bayesian theorem – probability of model under given data
  - Methods are looking for the best (e.g. evolutionary) **model** (e.g. phylogenetic tree) **explainig the data** (e.g. DNA sequences)
  - Algorithm exploring possible models and approaching the best runs in steps (generations)
    - After some time it converges to find optimal solution (usually described by logarithms of likelihood of given model)
    - Usually, ∼millions (or even more) of generations are required

# Introductory overview of statistics and methods VI

  - Beginning use to be very unstable – it is discarded as burn-in ("heating" of Markov Chain Monte Carlo (MCMC) doing the exploration and optimization of models), usually ∼10-25% of steps

- MP, ML and Bayesian statistics contain (evolutionary) **models** – they are not based on similarity (as matrix-based methods), so that they are supposed to reveal real structure in the data
- **Permutations**, **bootstraps** and another tests
  - It is necessary to test statistical significance of the obtained results
  - Most common methods somehow shuffle the data (drop one column, ...) and repeat the calculation to see how stable is the result (it might be driven by one or few loci, ...)
  - Whole process is repeated ∼100-1000 times and output is shown as histogram of simulations vs. the observed value, in how many percents the same result was obtained (e.g. bootstrap) or as p-value (what is probability that the pattern was created by random process)
  - $p = 0.05$ means 95% probability that the data are non-random

# 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")
# Bartlett's K-squared of difference
# between observed and expected heterozygosity - not significant
bartlett.test(list(hauss.summ$Hexp, hauss.summ$Hobs))
                Bartlett test of homogeneity of variances
data:  list(hauss.summ$Hexp, hauss.summ$Hobs)
Bartlett's K-squared = 0.069894, df = 1, p-value = 0.7915
```

# Descriptive statistics II

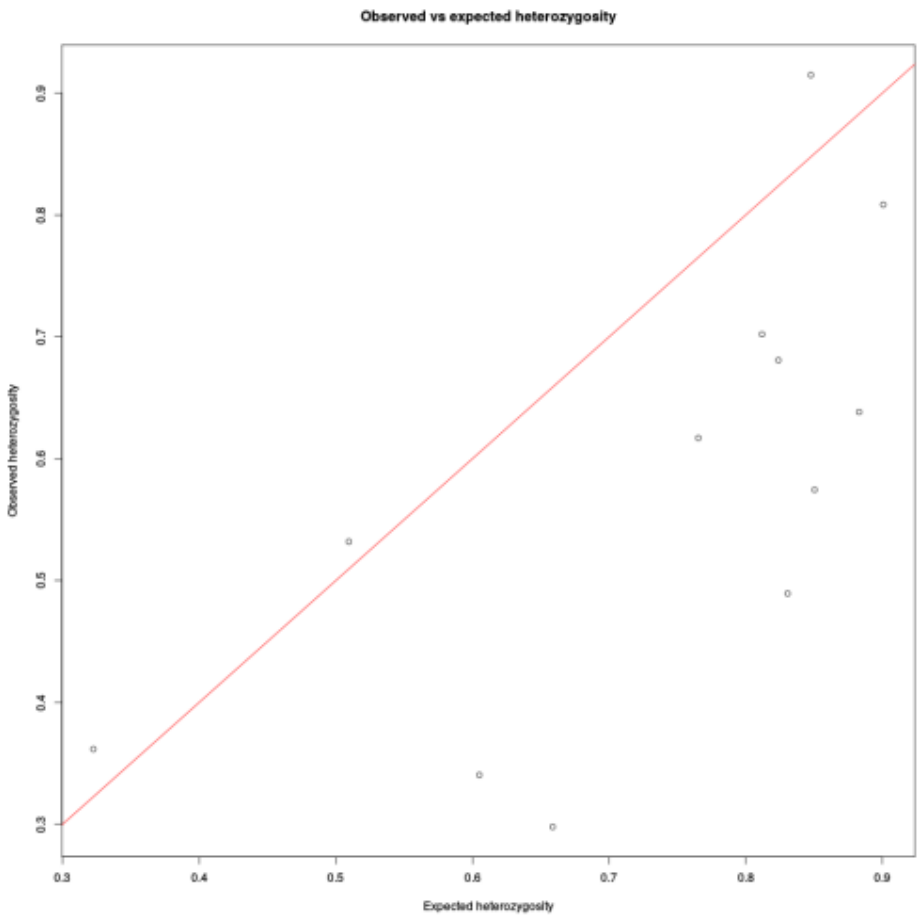
- `t-test` and `bartlett.test` require data to have normal distribution – if the condition is not met, it is necessary to use some weaker non-parametric test (`kruskal.test`, `wilcox.test`, …)
- See respective manual pages for details
- `shapiro.test()` tests the normality of given vector

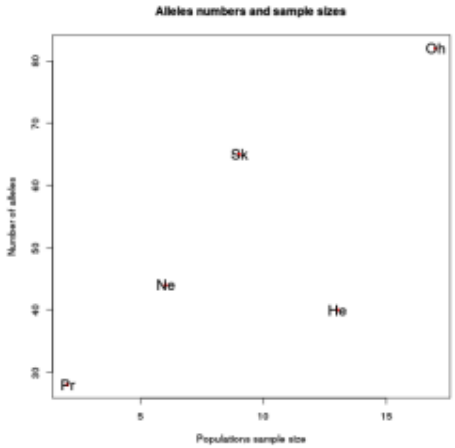
```
# T-test of difference between
# observed and expected heterozygosity - strongly significant
t.test(x=hauss.summ$Hexp, y=hauss.summ$Hobs, paired=TRUE, var.equal=T)
            Paired t-test
data:  hauss.summ$Hexp and hauss.summ$Hobs
t = 3.5622, df = 11, p-value = 0.004456
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.06114303 0.25887357
sample estimates:
mean of the differences
              0.1600083
```

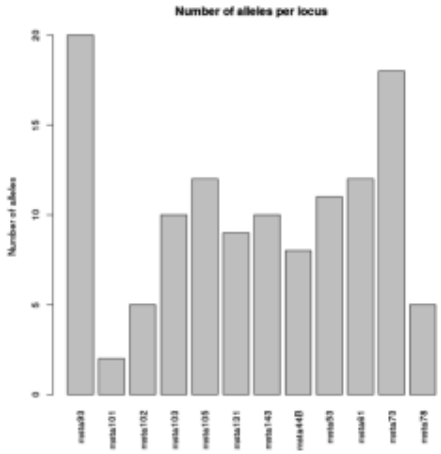
# Descriptive statistics III

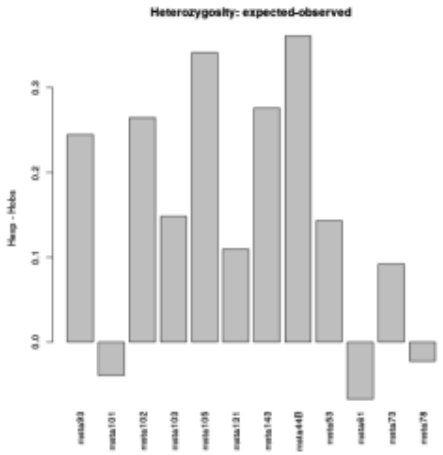
```
# Create pane with some information
par(mfrow=c(2,2)) # Divide graphical devices into 4 smaller spaces
# Plot alleles number vs. population sizes
plot(x=hauss.summ$n.by.pop, y=hauss.summ$pop.nall, xlab="Populations
  sample size", ylab="Number of alleles", main="Alleles numbers and
  sample sizes",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

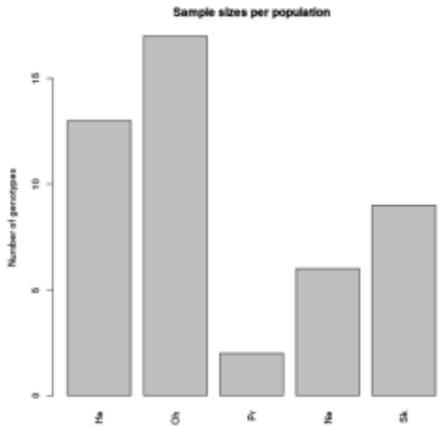
# Graphs from previous slides



# Population statistics by poppr()

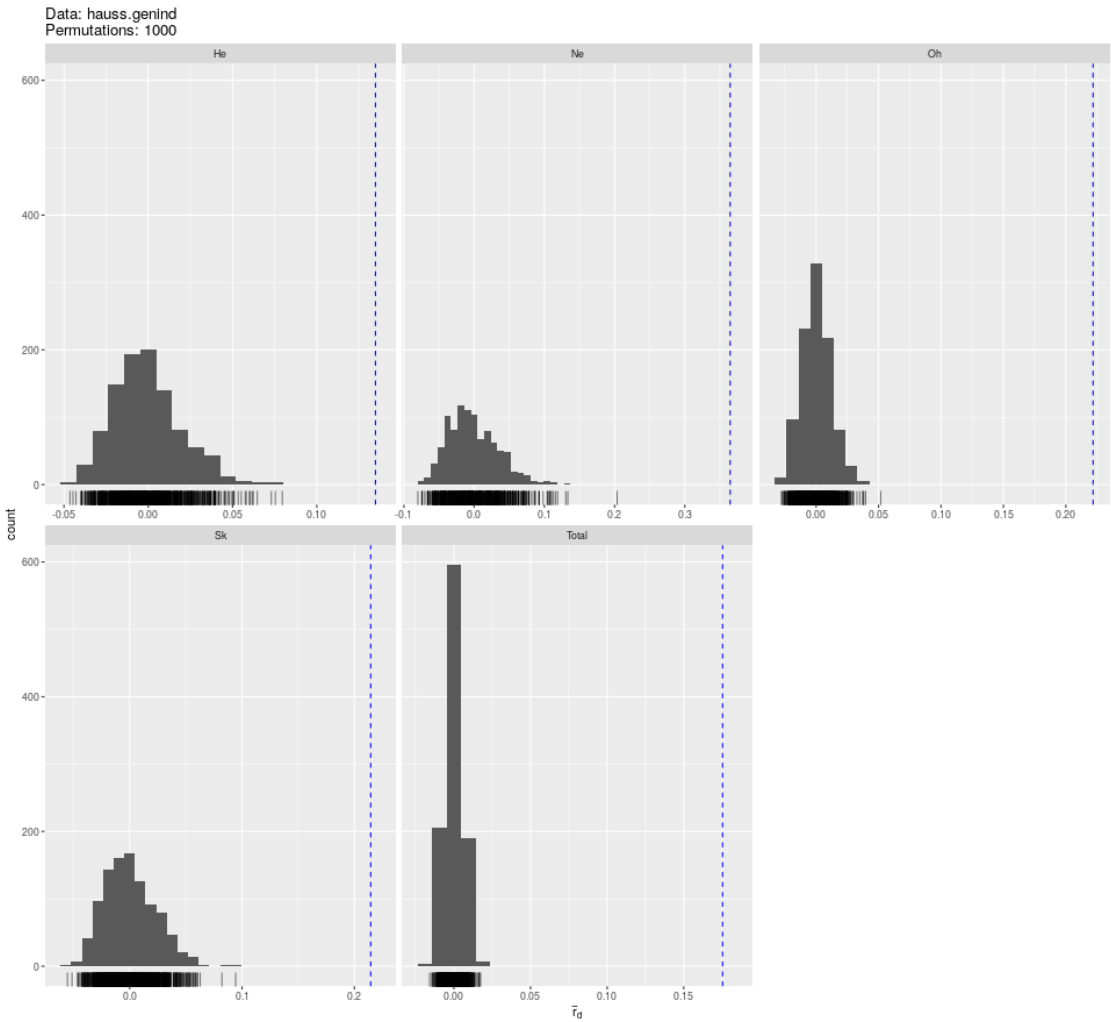
- `poppr()` is central function of `poppr` package calculating plenty of population genetic indices

```
?poppr # See details
poppr(dat=hauss.genind, total=TRUE, sample=1000, method=4,
  missing="geno", cutoff=0.15, quiet=FALSE, clonecorrect=FALSE,
  plot=TRUE, index="rbarD", minsamp=1, legend=TRUE)
... # Output table with indices:
  Pop  N MLG eMLG       SE     H     G lambda   E.5  Hexp   Ia # More...
   He 13  11 1.97 1.58e-01 2.352  9.94  0.899 0.941 0.503 1.42 ...
   Ne  6   5 1.93 2.49e-01 1.561  4.50  0.778 0.930 0.604 3.44 ...
  ..... ...  ...      ...   ...   ...    ...   ...   ...  ... ...
Total 47  43 2.00 6.07e-02 3.732 40.16  0.975 0.961 0.742 1.88 ...
```

- If `plot=TRUE`, histogram of simulations (`sample` must be $> 1$) is plotted for each population for `rbarD` or `Ia` (according to selected `index` – see following slides for details)

# Histograms of simulations of rbarD for each population

## The populations are significantly far from being clonal



# Population statistics returned by poppr() I

## Too much to choose from?

Generally, there are plenty of different population indices (and distances and another statistics) with different assumptions and usage in many packages – it can be complicated to pick the best one... The course shows many examples, but the list is far from being exhaustive...

- `Pop` – Population analyzed
  - If `total=TRUE`, there are also statistics for whole dataset
- `N` – Number of individuals/isolates in the specified population
- `MLG` – Number of multilocus genotypes found in the specified population (see `?mlg`)
- `eMLG` – The expected number of MLG at the lowest common sample size (set by `minsamp`)

# Population statistics returned by poppr() II

- `SE` – The standard error for the rarefaction analysis (assets species richness – how it grows with growing sample size)
  - Big difference between `MLG` and `eMLG` indicate some process lowering/increasing genetic diversity
- `H` – Shannon-Wiener Diversity index – evaluates number of genotypes and their distribution, takes entropy into account, grows with higher richness and diversity, sensitive to uneven sample size
- `G` – Stoddard and Taylor's Index – roughly, similar approach as the previous one, highly enhanced
- `lambda` – Simpson's index $\lambda = 1$ minus the sum of squared genotype frequencies – estimation of the probability that two randomly selected genotypes are different and scales from 0 (no genotypes are different) to 1 (all genotypes are different)

# Population statistics returned by poppr() III

- `E.5` – Evenness – measure of the distribution of genotype abundances, wherein a population with equally abundant genotypes yields a value equal to 1 and a population dominated by a single genotype is closer to 0
- `Hexp` – Nei's gene diversity (expected heterozygosity) – unbiased gene diversity (from 0 = no diversity to 1 = highest diversity)
- `Ia` – Index of Association (`?ia`) – widely used to detect clonal reproduction within populations
  - Populations whose members are undergoing sexual reproduction will produce gametes via meiosis, and thus have a chance to shuffle alleles in the next generation
  - Populations whose members are undergoing clonal reproduction generally do so via mitosis – most likely mechanism for a change in genotype is via mutation – the rate of mutation varies from species to species, but it is rarely sufficiently high to approximate a random shuffling of alleles

# Population statistics returned by poppr() IV

  - The index of association is a calculation based on the ratio of the variance of the raw number of differences between individuals and the sum of those variances over each locus
  - It as the observed variance over the expected variance – if they are the same, then the index is zero (=prevailing clonal reproduction) after subtracting one – it rises with with increasing differences

- `p.Ia` – P-value for `Ia` from the number of reshuffling indicated in `sample`
- `rbarD` – Standardized Index of Association for each population (see `?ia`) – corrected for higher number of loci not to rise so steeply
- `p.rD` – P-value for `rbarD` from the number of reshuffles indicated in `sample`
- See poppr's manual and `vignette("algo", package="poppr")` for details

# Departure from Hardy-Weinberg equilibrium

- In theory, in large panmictic population without evolutionary influence everyone can mate with everyone (it is in equilibrium) and allele frequencies remain stable – in reality, environment, behavior, mutations, genetic drift, etc. are structuring the population

```
# According to loci
hauss.hwe.test <- hw.test(x=hauss.loci, B=1000)
hauss.hwe.test
# According to populations
# Separate genind object into list of genind objects for individual
# populations
hauss.pops <- seppop(hauss.genind)
hauss.pops
# Convert genind back to loci (list of loci objects according to
# populations)
hauss.pops.loci <- lapply(X=hauss.pops, FUN=genind2loci)
# Calculate the results per populations
lapply(X=hauss.pops.loci, FUN=hw.test, B=1000)
```

# Departure from HWE – results per locus

```
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 |
| ... | ... | ... | ... | ... |

- Pr.exact shows significance of the departure (i.e. non-equilibrial distribution of alleles within population – calculated per loci)
- $\chi^2$ test (without or with the permutations) test the departure – if it is significant or not – not how much it is departing

# F-statistics I

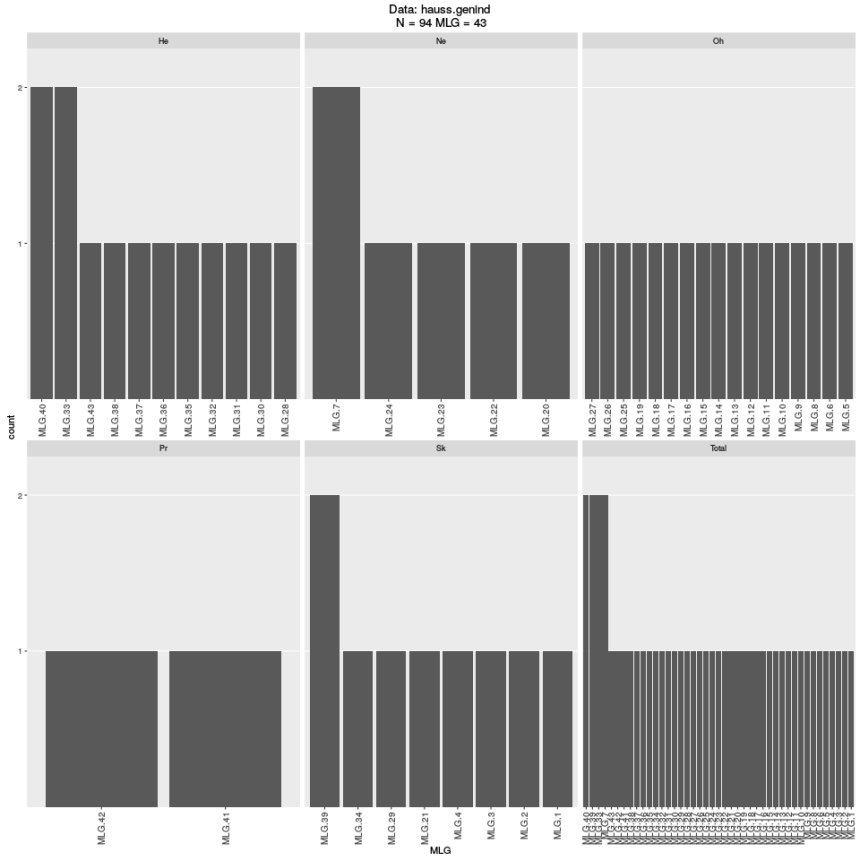
- Functions return tables of F-statistics values for populations/loci (roughly 0 – no structure, 1 – fully structured)
- The different F-statistics look at different levels of population structure. $F_{IT}$ is the inbreeding coefficient of an individual relative to the total population; $F_{IS}$ is the inbreeding coefficient of an individual relative to the subpopulation and averaging them; and $F_{ST}$ is the effect of subpopulations compared to the total population
- For `Fst`, `fstat` and `theta.msat` the loci object **must** contain population column

```
# Fit, Fst and Fis for each locus
Fst(x=hauss.loci, pop=1)
                 Fit        Fst         Fis
msta93    0.31835291 0.17867087  0.17006829
msta101 -0.09968472 0.04064928 -0.14628018
    ...         ...        ...         ...
```

# F-statistics II

```
# Multilocus estimators of variance components and F-statistics,
# alternative to Fst
library(hierfstat)
fstat(x=hauss.genind, pop=NULL, fstonly=FALSE)
            pop       Ind
Total 0.1589501 0.2582641
pop   0.0000000 0.1180834
# Nei's pairwise Fst between all pairs of populations.
# 0 = no structure; 1 = maximal difference
pairwise.fst(x=hauss.genind, pop=NULL, res.type="matrix")
           He         Ne         Oh         Pr         Sk
He 0.00000000 0.19960826 0.11391904 0.09404571 0.11184561
Ne 0.19960826 0.00000000 0.07265306 0.19220430 0.10112859
Oh 0.11391904 0.07265306 0.00000000 0.05302854 0.06287497
Pr 0.09404571 0.19220430 0.05302854 0.00000000 0.10436469
Sk 0.11184561 0.10112859 0.06287497 0.10436469 0.00000000
```

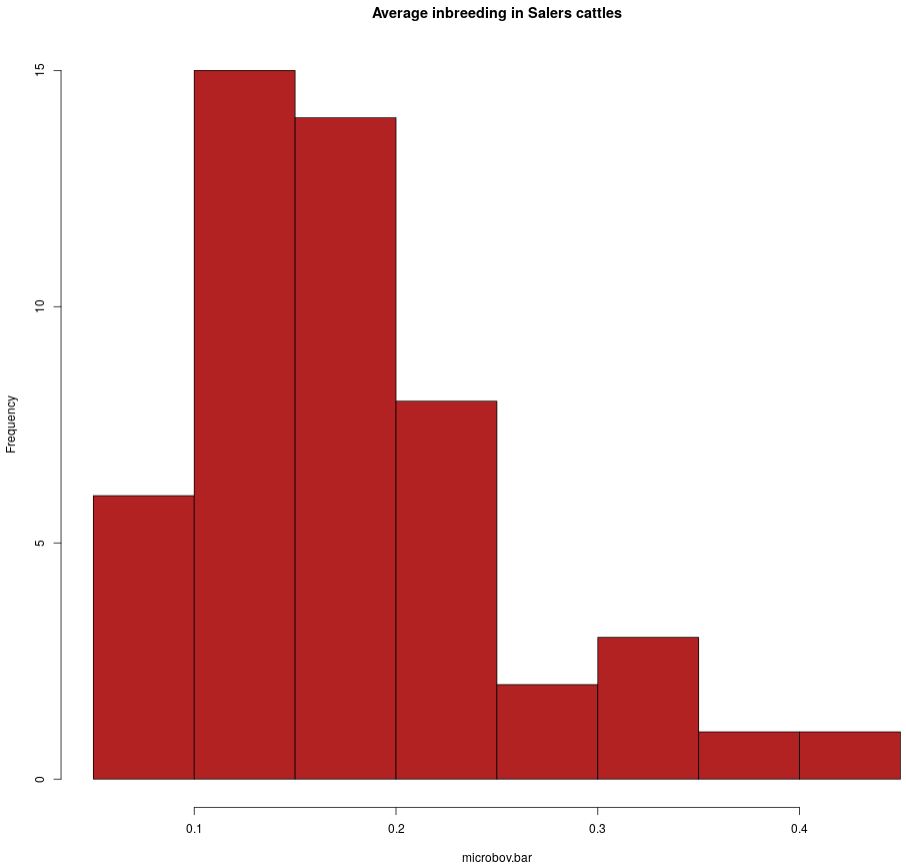
# Multi locus genotypes




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

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

# Inbreeding




```
# Load training data (cattle)
data(microbov)
# Separate populations of Salers
microbov.pops <- seppop(microbov)
  [["Salers"]]
# Calculate the inbreeding
microbov.inbr <- inbreeding(x=
  microbov.pops, N=100)
# Check for more settings
?inbreeding
# population means for plotting
microbov.bar <- sapply(X=
  microbov.inbr, FUN=mean)
# Plot it
hist(x=microbov.bar, col=
  "firebrick", main="Average
  inbreeding in Salers cattles")
```

# Basic distances

```
# See ?dist.gene for details about methods of this distance
hauss.dist.g <- dist.gene(x=hauss.genind@tab, method="pairwise")
# Euclidean distance for individuals (plain ordinary distance matrix)
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
```

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

# Distances reflecting microsatellite repeats

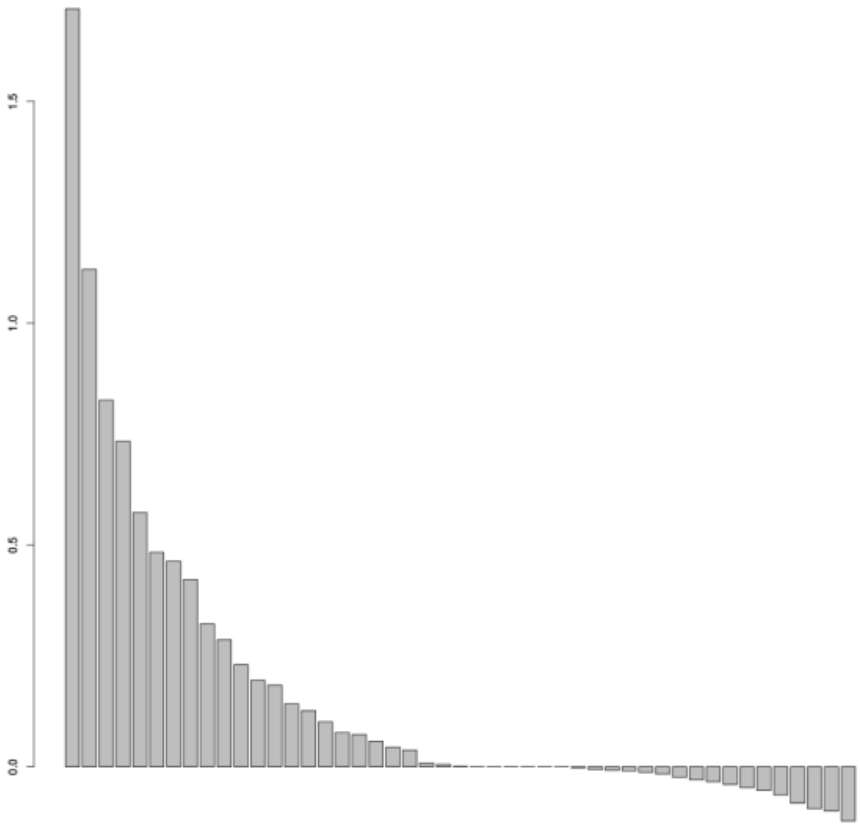
```
# Bruvo's distances weighting SSRs repeats - take care about replen
# parameter - requires repetition length for every SSRs locus
hauss.dist.bruvo <- bruvo.dist(pop=hauss.genind, replen=rep(2, 12),
  loss=TRUE)
# Test if it is Euclidean
is.euclid(hauss.dist.bruvo, plot=TRUE, print=TRUE, tol=1e-10)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

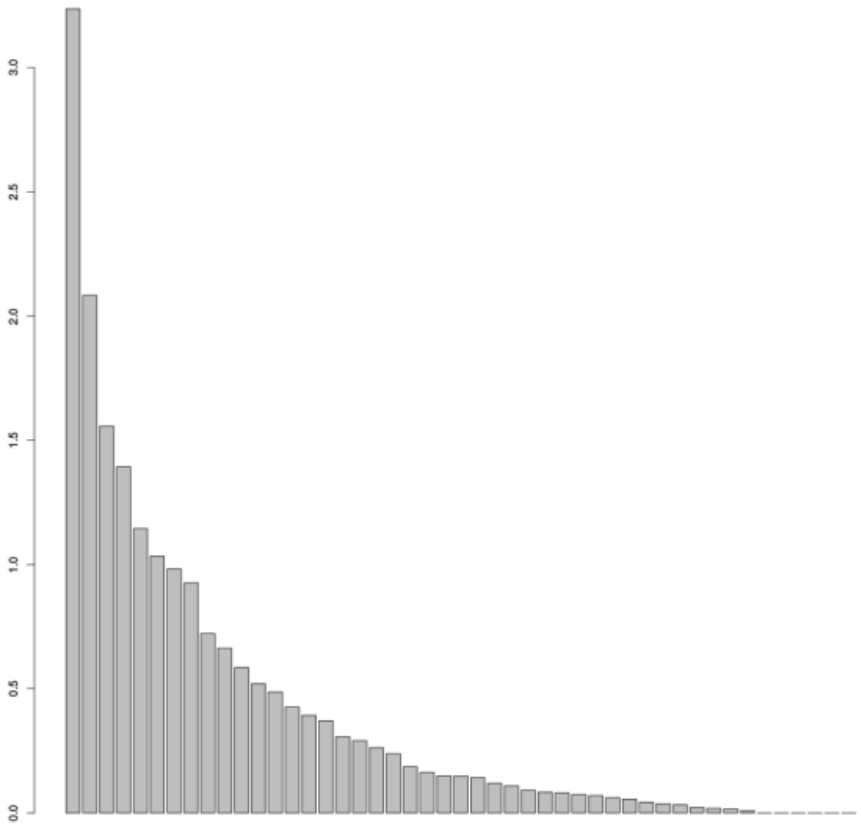
- See poppr's manual and manual pages of the functions for details and different possibilities of settings
- Be careful when changing non-Euclidean distances to Euclidean – the transformation more or less changes meaning of the distances!

# 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
```

# Different distances have different use case and outputs...

| Method | Function | Assumption | Euclidean |
|---|---|---|---|
| Prevosti 1975 | `prevosti.dist`, `diss.dist` | — | No |
| Nei 1972, 1978 | `nei.dist` | Infinite Alleles, Genetic Drift | No |
| Edwards 1971 | `edwards.dist` | Genetic Drift | Yes |
| Reynolds 1983 | `reynolds.dist` | Genetic Drift | Yes |
| Rogers 1972[1] | `rogers.dist` | — | |
| Bruvo 2004 | `bruvo.dist` | Stepwise Mutation | No |

```
# See details of distance methods in package poppr
vignette("algo", package="poppr")
```

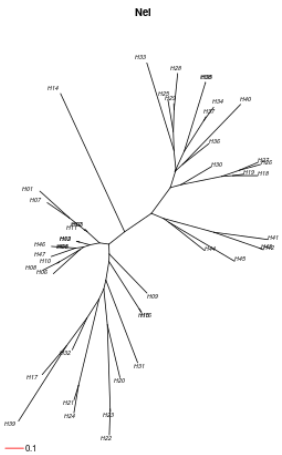
[1] Rogers (1972): Measures of genetic similarity and genetic distances. Pp. 145-153 of Studies in Genetics. University of Texas Publishers

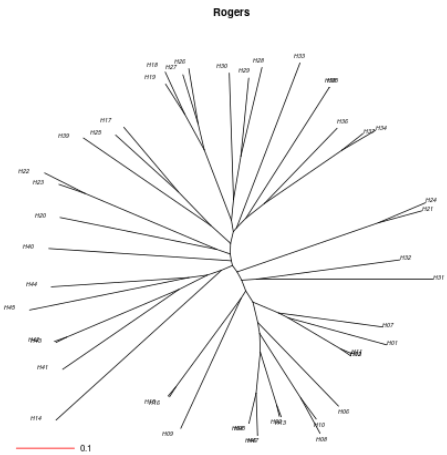
# Comparison of different matrices

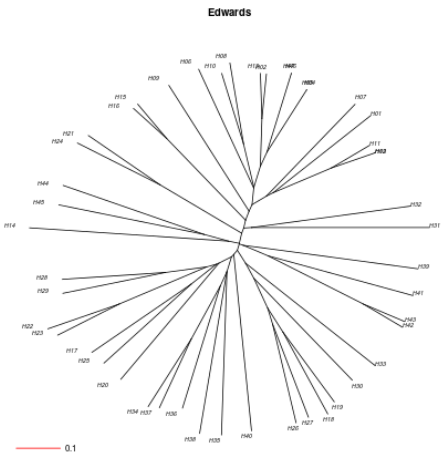
```
# Compare different distance matrices
# List of functions to be parsed to respective dist.* function
distances <- c("Nei", "Rogers", "Edwards", "Reynolds", "Prevosti")
# Calculate the distance matrices
dists <- lapply(distances, function(x) {
  DISTFUN <- match.fun(paste(tolower(x), "dist", sep="."))
  DISTFUN(hauss.genind.cor) })
# Add names for the distance names
names(dists) <- distances
# Add Bruvo distance
dists[["Bruvo"]] <- hauss.dist.bruvo
dists
# Split graphical device into 2 lines, 3 panes each
par(mfrow=c(2, 3))
# Calculate NJ and plot all trees
x <- lapply(names(dists), function(x) {
  plot(njs(dists[[x]]), main=x, type="unrooted")
  add.scale.bar(lcol="red", length=0.1) })
dev.off() # Close graphical device to reset settings
```

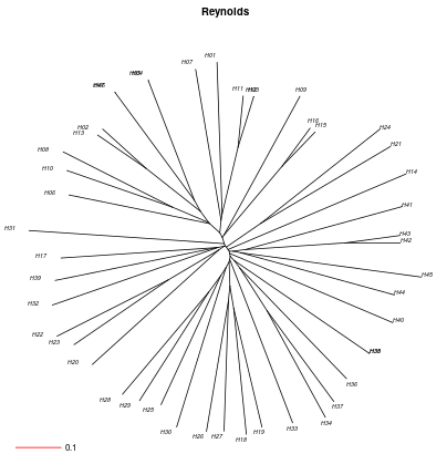
# Neigbor-Joining of same dataset under different matrices

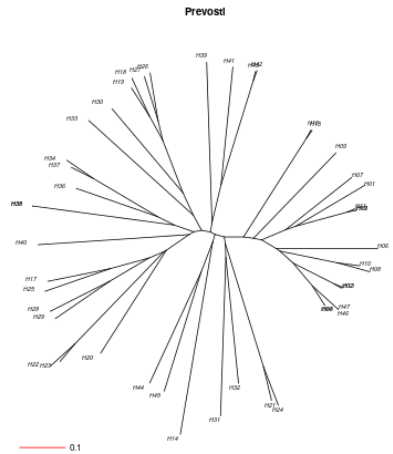
The results are very different...

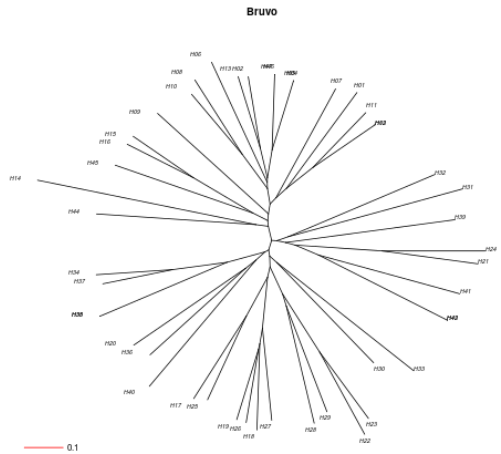


# Distances among DNA sequences

- The sequences must be aligned before calculating distances among them!
- Selection of mutational model has significant impact to results...

```
# There are various models available
?dist.dna
# Create the distance matrix
usflu.dist <- dist.dna(x=usflu.dna, model="TN93")
# Check the resulting distance matrix
usflu.dist
class(usflu.dist)
# Create another distance matrix
dim(as.matrix(usflu.dist))
# Check it
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):

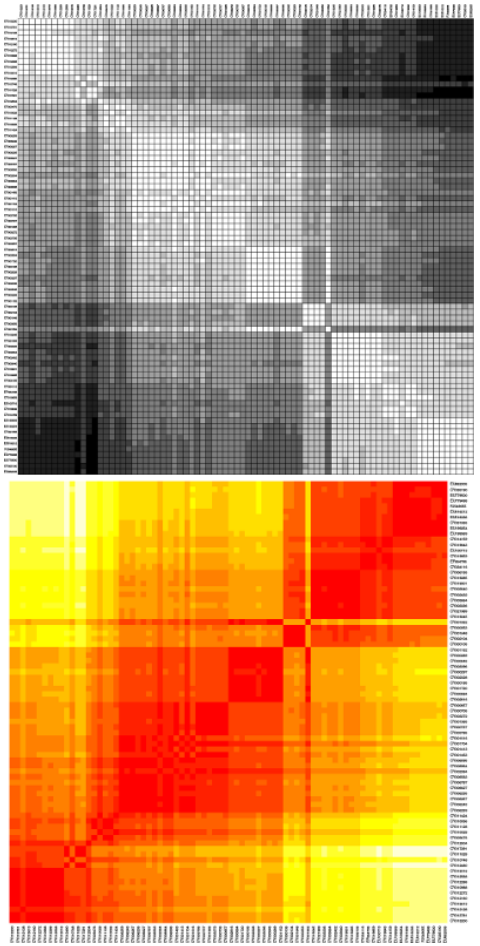
```
usflu.dists.l <- seploc(usflu.genlight, n.block=10, parallel=FALSE)
class(usflu.dists.l)
usflu.dists <- lapply(X=usflu.dists.l, FUN=function(DDD)
  dist(as.matrix(DDD)))
class(usflu.dists)
names(usflu.dists)
class(usflu.dists[[1]])
usflu.distr <- Reduce(f="+", x=usflu.dists)
class(usflu.distr)
usflu.distr
# It is possible to use just basic dist function on whole
# genlight object (might require a lot of RAM)
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, not for all of the cases

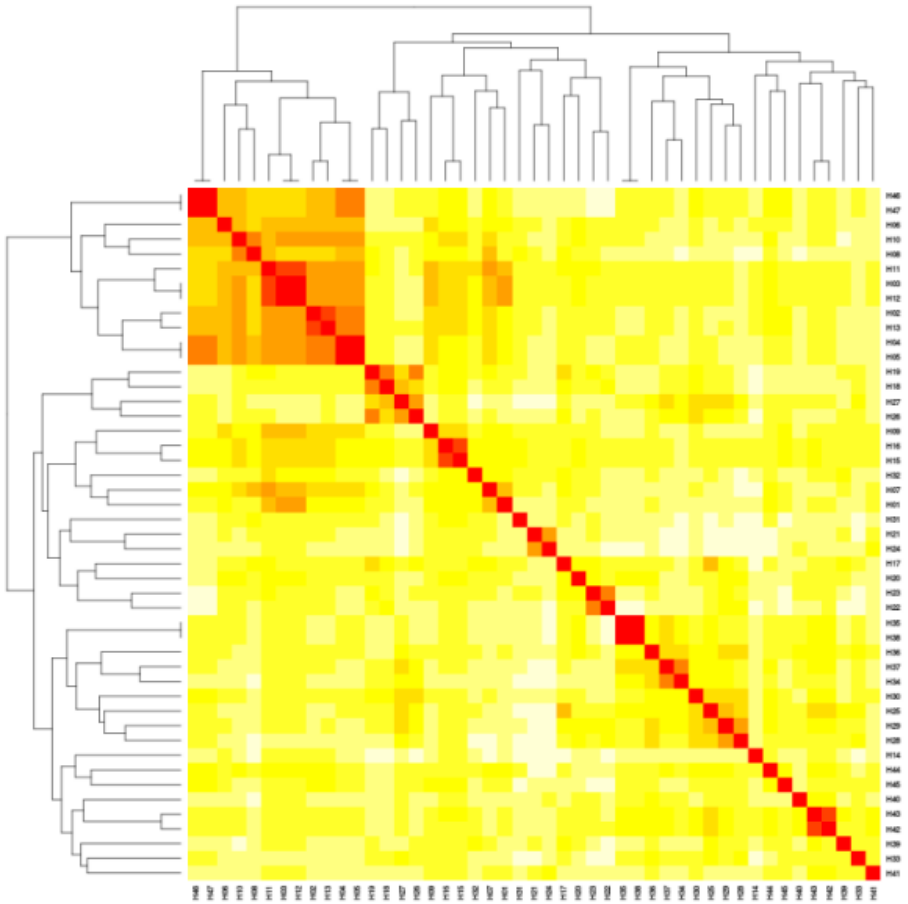
# Visualize pairwise genetic similarities

```
# 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, colored
# 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)
```

- Colored according to value
- Another possibility is to use `corrplot::corrplot()` for correlation plots



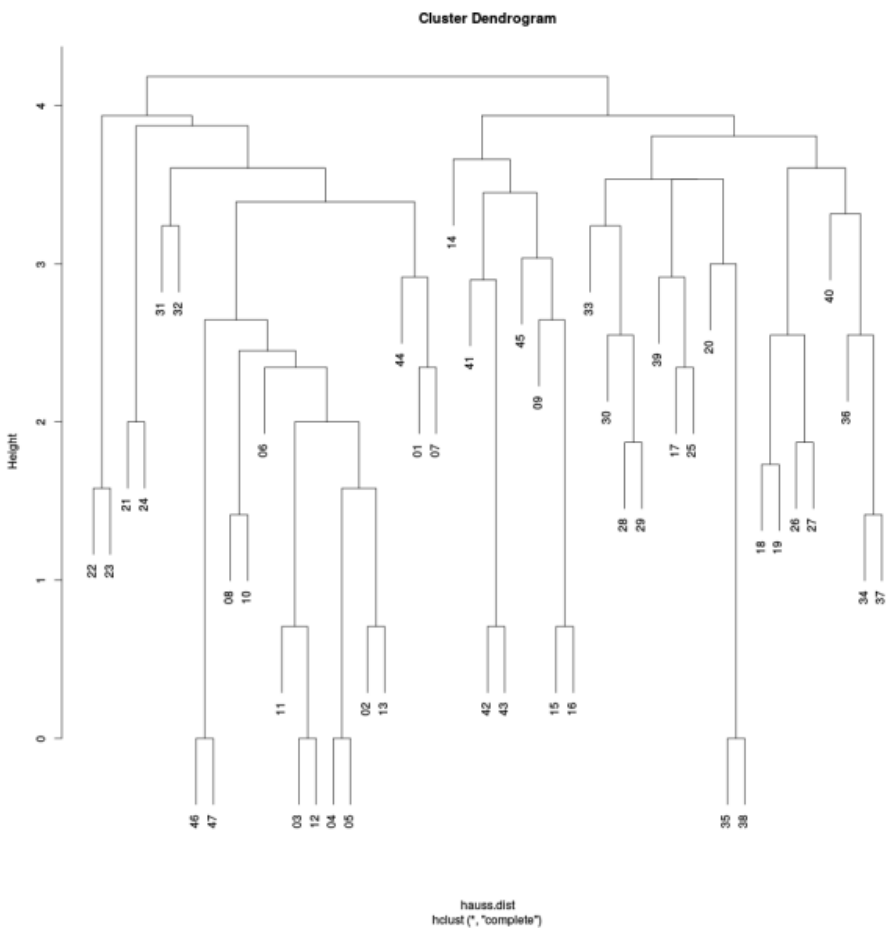
# Heatmaps



```
# Based on various distances
heatmap(as.matrix(hauss.dist),
  symm=TRUE, labRow=rownames(
  as.matrix(hauss.dist.bruvo)),
  labCol=colnames(as.matrix(
  hauss.dist.bruvo)))
  # hauss.dist doesn't contain
  # names of individuals - add here
heatmap(as.matrix(hauss.dist.pop),
  symm=TRUE)
heatmap(as.matrix(hauss.dist.
  bruvo), symm=TRUE)
heatmap(as.matrix(hauss.dist.
  diss), symm=TRUE)
```

There are various settings – colors, dendrogram, ...See `?heatmap`.

# Hierarchical clustering – UPGMA and others



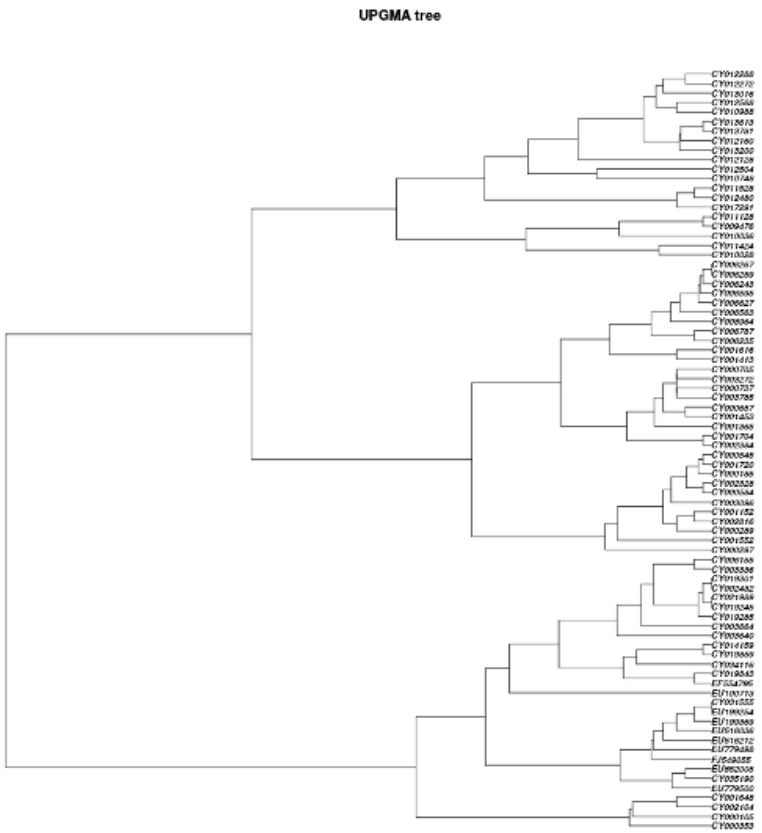

```r
# According to distance used
# see ?hclust for methods
plot(hclust(d=hauss.dist,
  method="complete"))
plot(hclust(d=hauss.dist.pop,
  method="complete"))
plot(hclust(d=hauss.dist.
  bruvo, method="complete"))
```

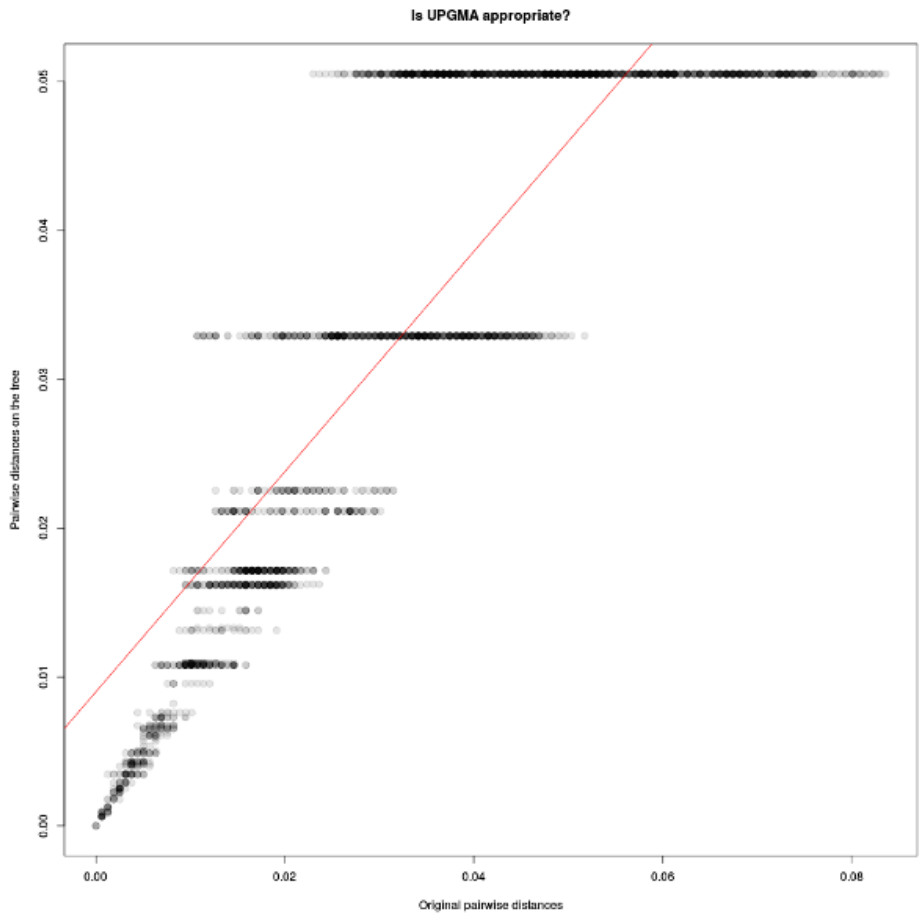
- This is very basic function to make dendrogram
- There are better possibilities (NJ etc – see slide 108 and onward)

# 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)
# Add correlation line
abline(lm(as.vector(as.dist(cophenetic(usflu.upgma)))~
  as.vector(usflu.dist)), col="red")
```

# UPGMA is not the best choice here...




All points in the right graph should be clustered along the red line...

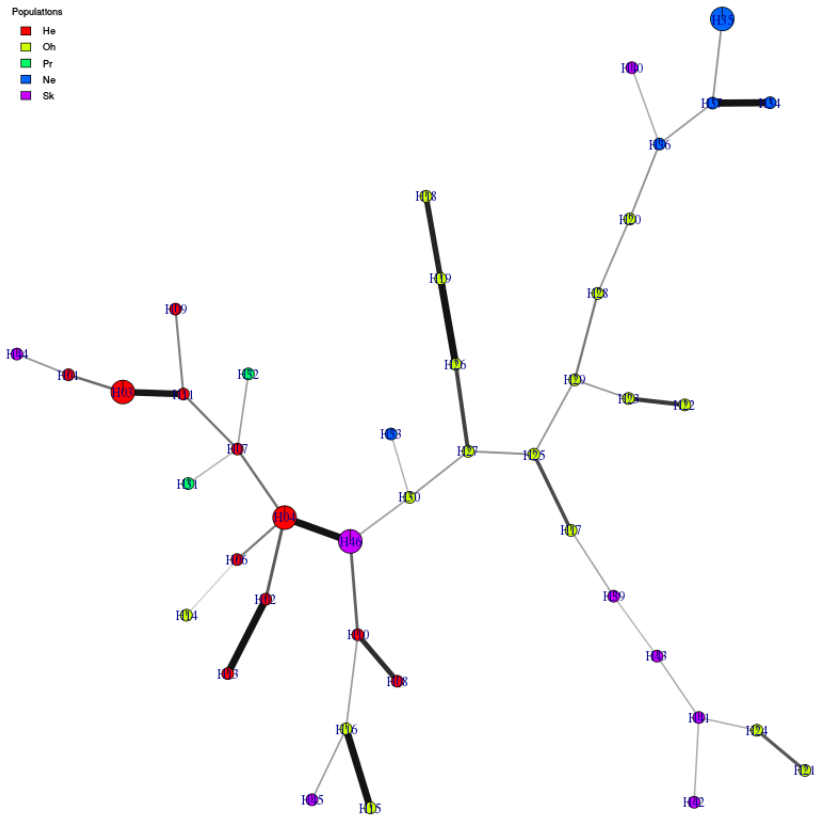
# AMOVA

```
# From package pegas (doesn't directly show percentage of variance)
hauss.pop <- pop(hauss.genind)
hauss.amova <- pegas::amova(hauss.dist~hauss.pop, data=NULL,
  nperm=1000, is.squared=TRUE)
hauss.amova
...
                SSD      MSD df
hauss.pop  30.71923 7.679809  4
Error     119.58100 2.847167 42
Total     150.30023 3.267396 46
...
```

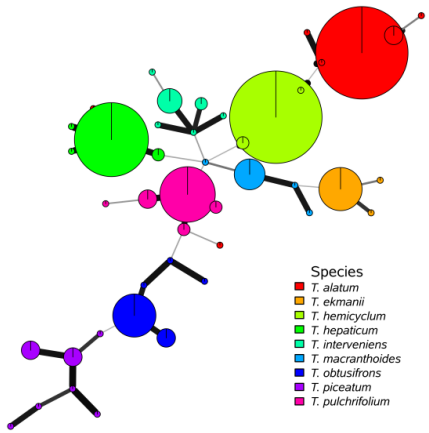
- Analysis of molecular variance tests if there are significant differences among populations (and/or another levels)
- Another possibility is `poppr.amova` – for more complicated hierarchy, see `?poppr.amova`

# Minimum Spanning Network

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




```
bruvo.msn(gid=hauss.genind,
  replen=rep(2, 12), loss=TRUE,
  palette=rainbow, vertex.label
  ="inds", gscale=TRUE,
  wscale=TRUE, showplot=TRUE)
?msn.poppr # For another data types
?imsn # Interactive creation of MSN
```

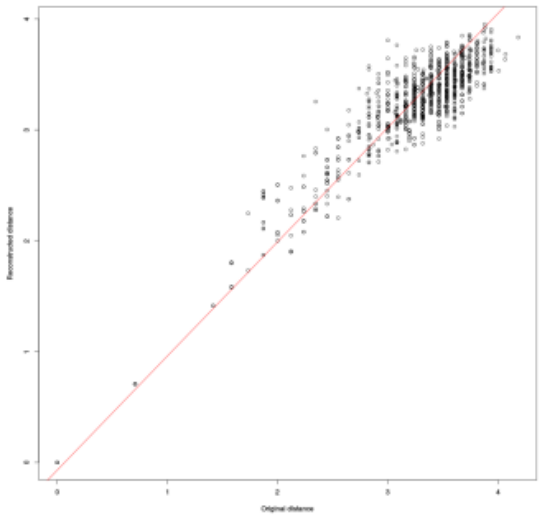
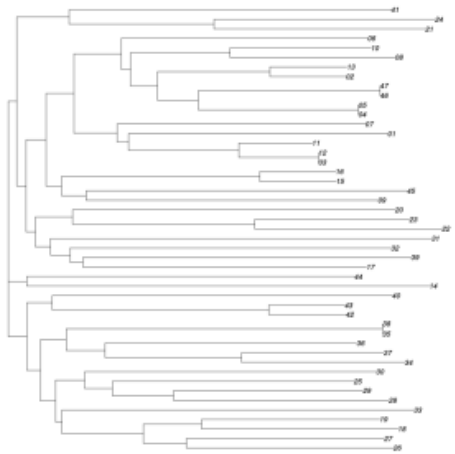
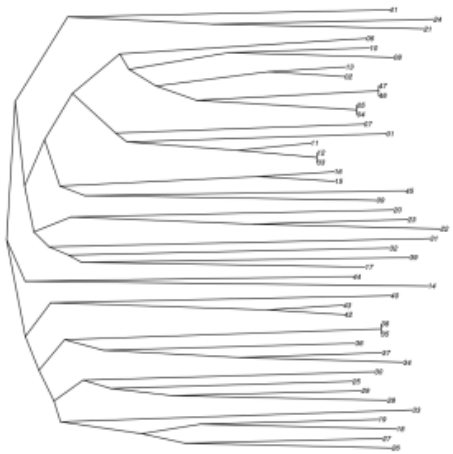


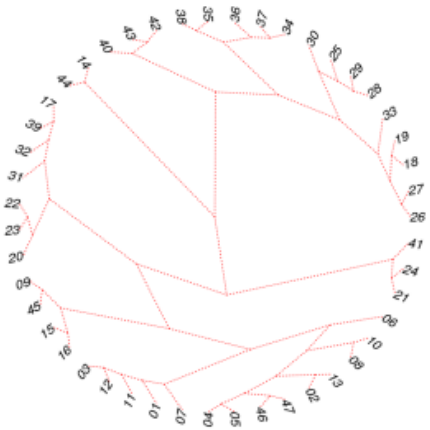
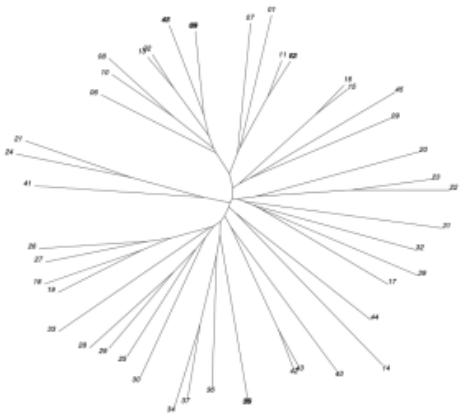

```
?bruvo.msn # See details...
```

# 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")
# Linear model for above graph
summary(lm(as.vector(hauss.dist) ~
  as.vector(as.dist(cophenetic(hauss.nj))))) # Prints summary text
# 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)
# There are enormous graphical possibilities...
```

# Choose your tree...

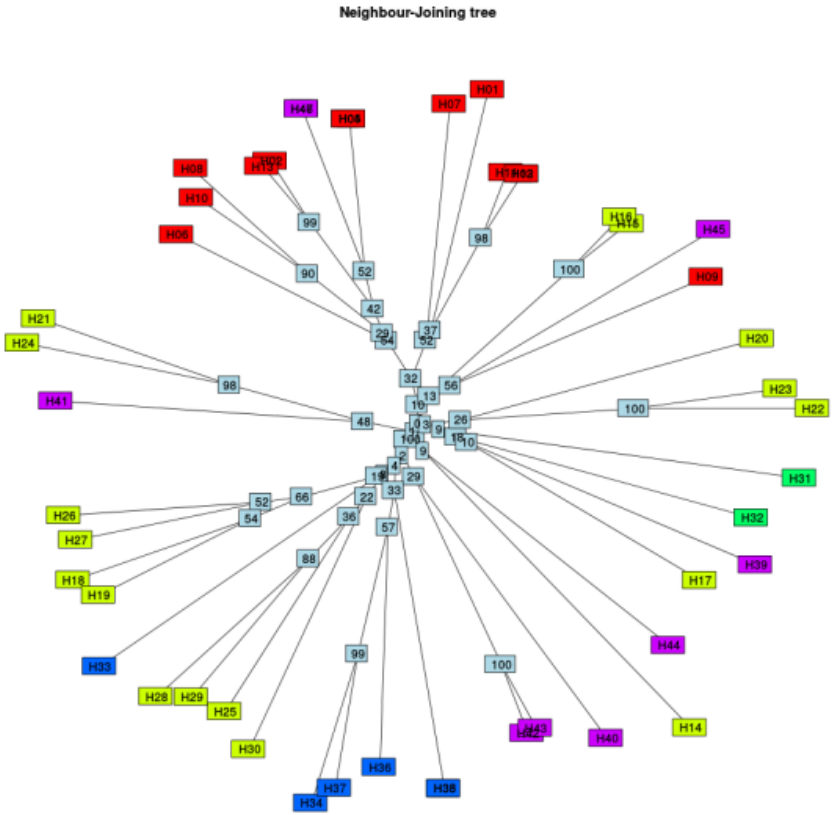
# Bootstrap

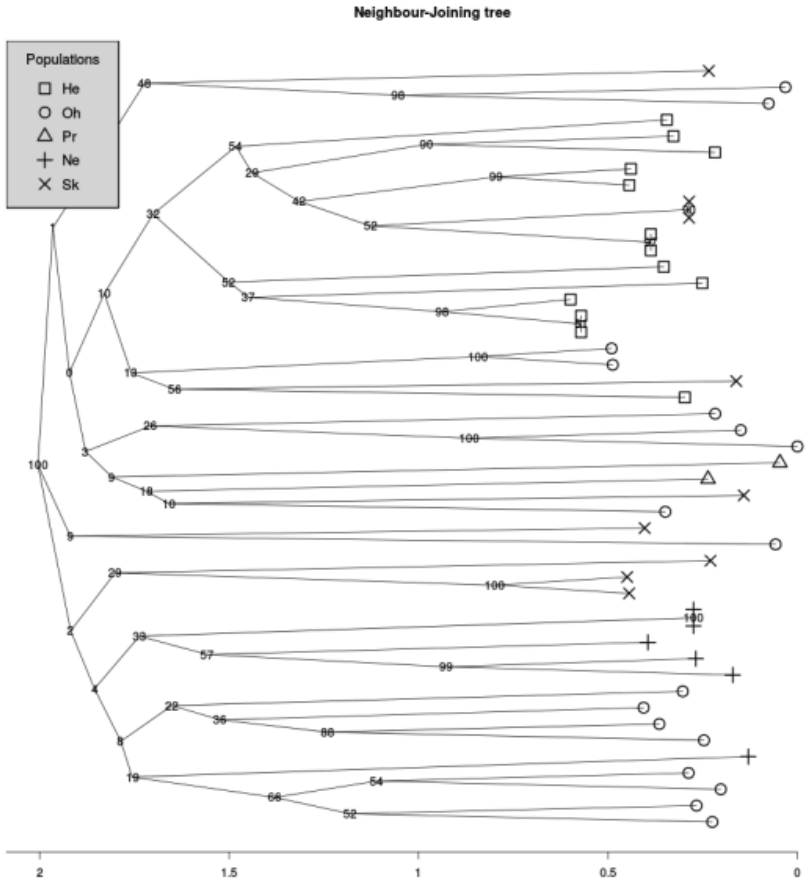
```
# boot.phylo() resamples all columns - remove population column first
hauss.loci.nopop <- hauss.loci
hauss.loci.nopop[["population"]] <- NULL
# Calculate the bootstrap
hauss.boot <- boot.phylo(phy=hauss.nj, x=hauss.loci.nopop,
  FUN=function(XXX) nj(dist(loci2genind(XXX))), B=1000)
# boot.phylo returns NUMBER of replicates - NO PERCENTAGE
# Plot the tree
plot.phylo(x=hauss.nj, type="unrooted", main="Neighbour-Joining tree")
# Labels for nodes - bootstrap - see ?nodelabels for graphical settings
nodelabels(text=round(hauss.boot/10))
?boot.phylo # See details
# Another possibility
hauss.aboot <- aboot(x=hauss.genind, tree="nj", distance=nei.dist,
  sample=100) # Bootstrap values are in slot node.label
# Plot the tree, explicitly display node labels
plot.phylo(x=hauss.aboot, show.node.label=TRUE)
?aboot # Package poppr
```

# 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)) # Colored 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 explaining 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...



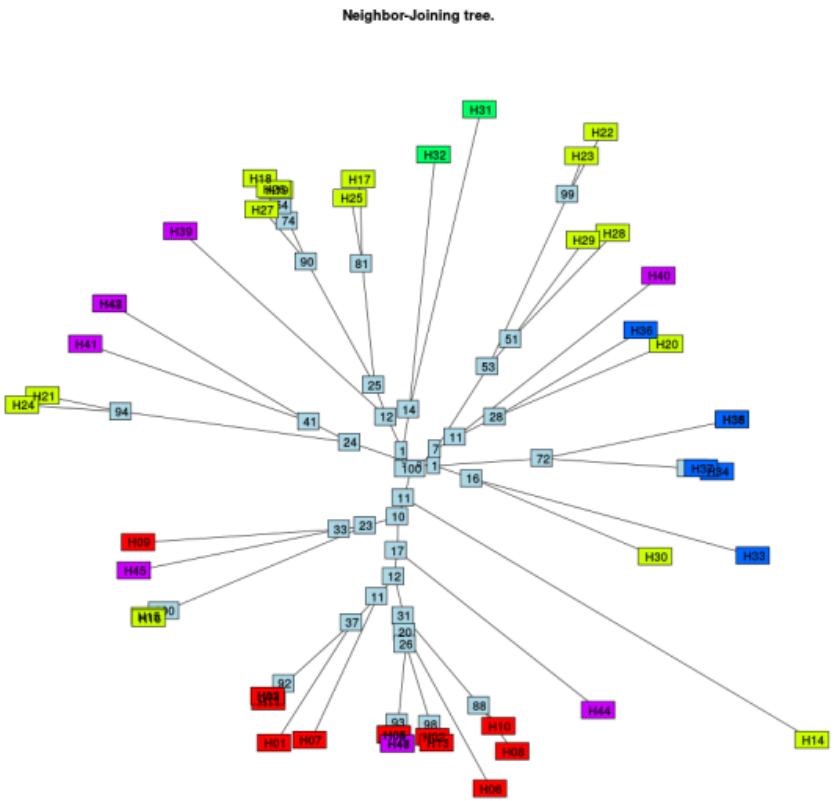



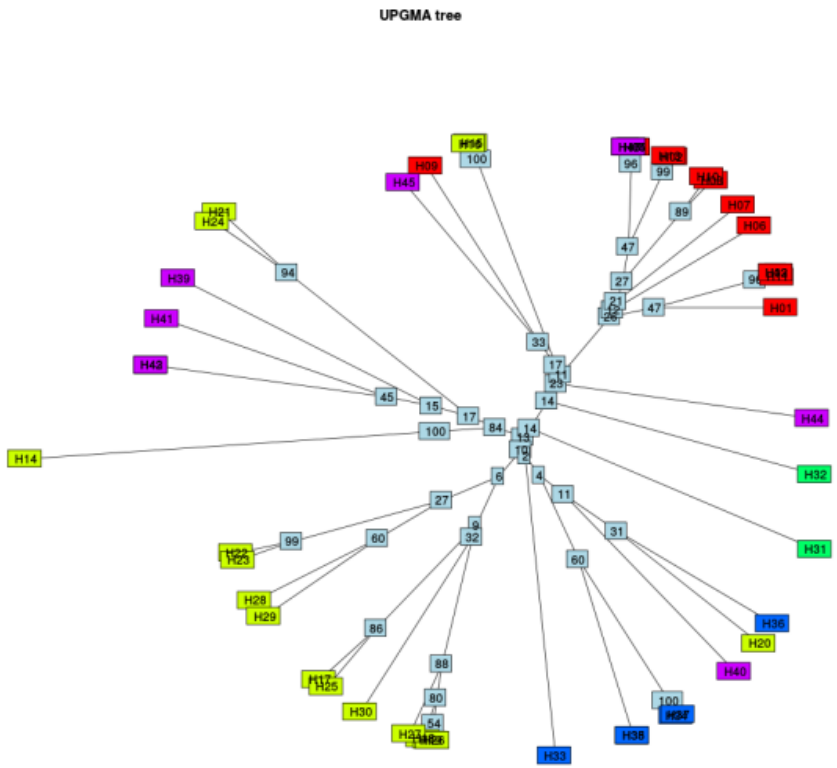
# Trees based on Bruvo's distance

## Package poppr (bootstrap is incorporated within the function)

```
# There are currently problems with compatibility with newest ape...
# NJ
hauss.nj.bruvo <- bruvo.boot(gid=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.")
# 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(gid=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 based on DNA sequences

```
# Calculate the tree
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 colors from color 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=usflu.annot$year, bg=transp(num2col(x=usflu.annot$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 colors from color 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)
```

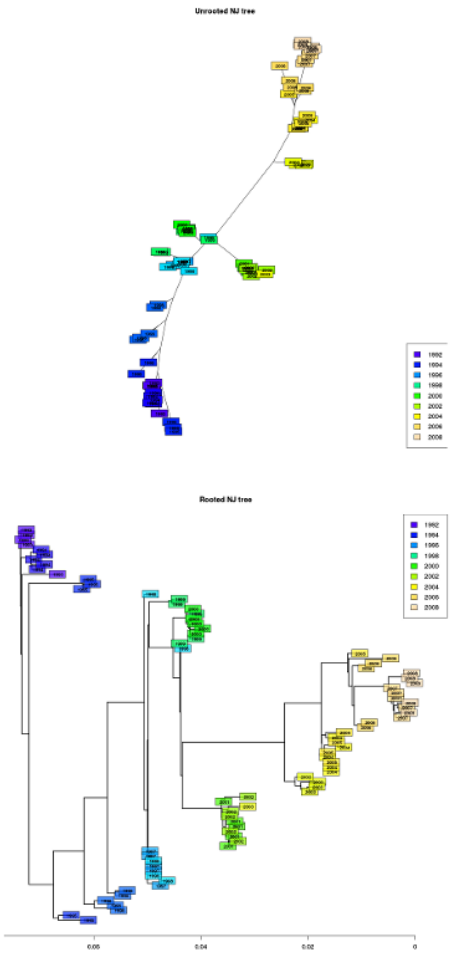
# Bootstrap rooted tree

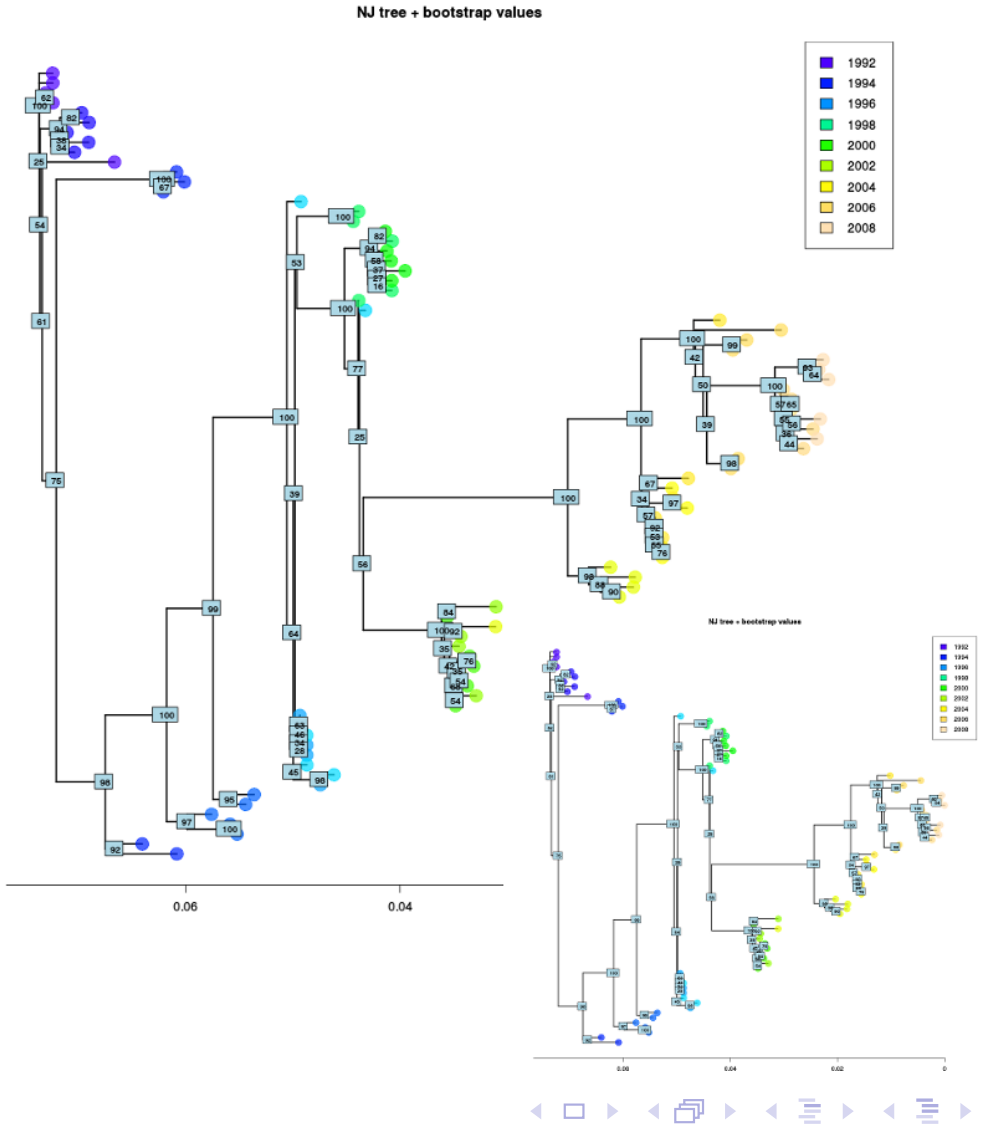
```
# Calculate it
usflu.boot <- boot.phylo(phy=usflu.tree.rooted, x=usflu.dna,
  FUN=function(EEE) root(nj(dist.dna(EEE, model="TN93")),
  outgroup=1), B=1000)
# Plot the tree
plot.phylo(x=usflu.tree.rooted, show.tip=FALSE, edge.width=2)
title("NJ tree + bootstrap values")
tiplabels(frame="none", pch=20,
  col=transp(num2col(x=usflu.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 colors from color 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$edge.length[usflu.tocollapse] <- 0
# Create new tree
usflu.tree.collapsed <- di2multi(phy=usflu.tree.temp, tol=0.00001)
# Plot the consensus tree
plot.phylo(x=usflu.tree.collapsed, show.tip=FALSE, edge.width=2)
title("NJ tree after collapsing weak nodes")
tiplabels(text=usflu.annot$year,
  bg=transp(num2col(x=usflu.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

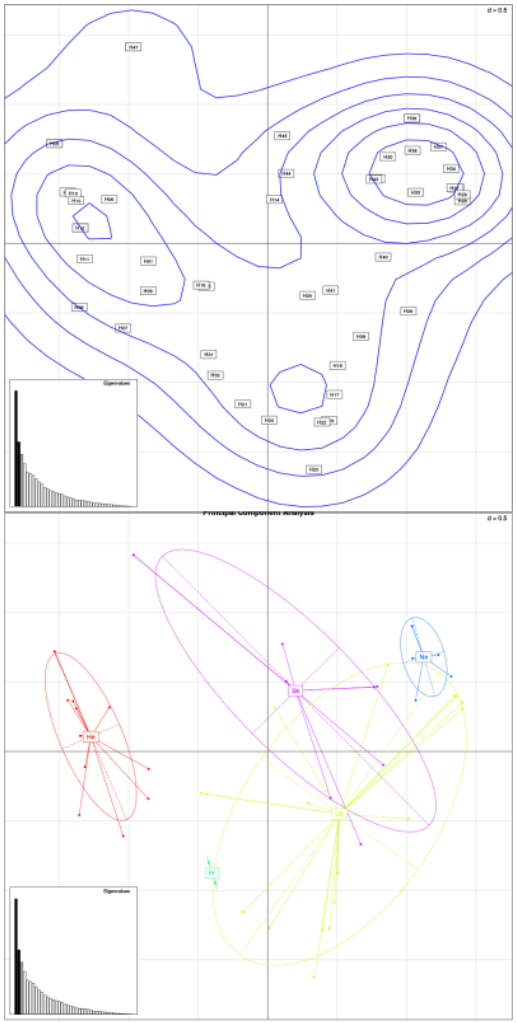
# NJ is death. Long live NJ!

- "Basic" NJ has many limitations (problems with missing data, chaining of individuals, ...) – 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 usage is same as with "classical" `nj()` (read manual pages before using them) – it is also from package `ape`

# PCoA I

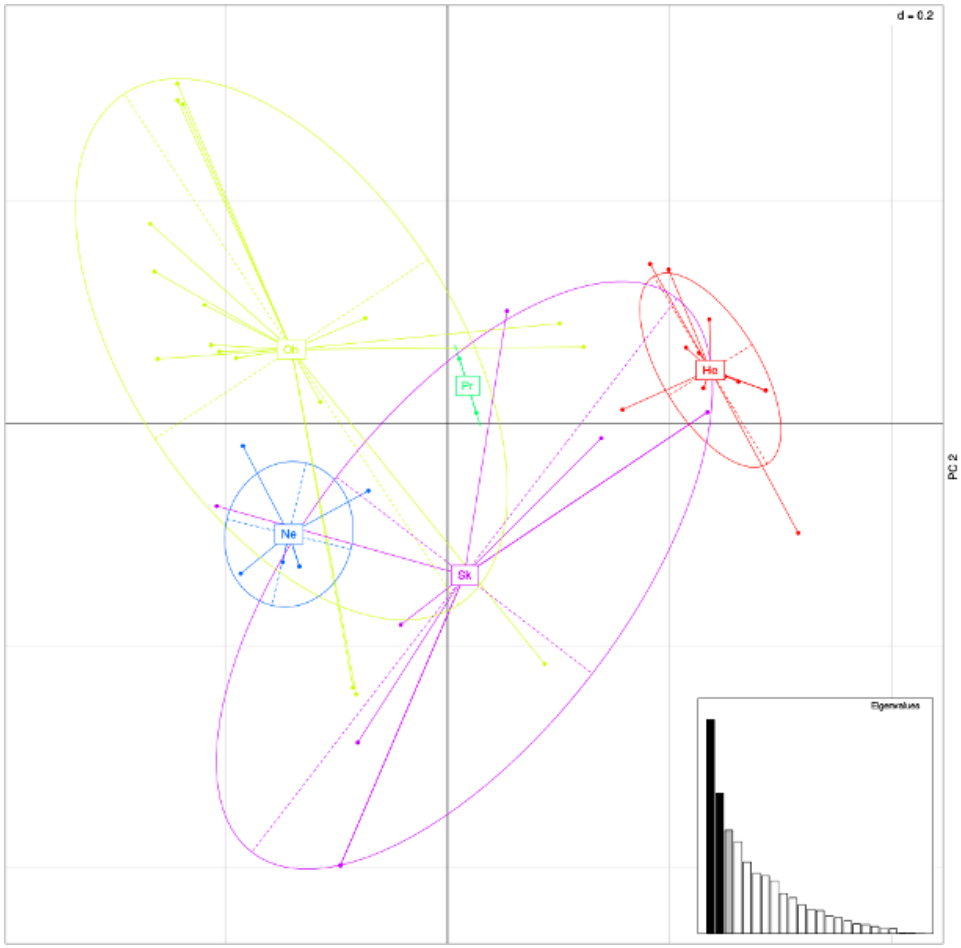
```
hauss.pcoa <- dudi.pco(d=dist(x=scaleGen(x=hauss.genind, center=TRUE,
  scale=FALSE, truenames=TRUE), method="euclidean"), scannf=FALSE,
  nf=3)
# Basic display
s.label(dfxy=hauss.pcoa$li, clabel=0.75)
# To plot different axes use for example dfxy=hauss.pcoa$li[c(2, 3)]
# Add kernel density
s.kde2d(dfxy=hauss.pcoa$li, cpoint=0, add.plot=TRUE)
# Adds histogram of Eigenvalues
add.scatter.eig(w=hauss.pcoa$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
```

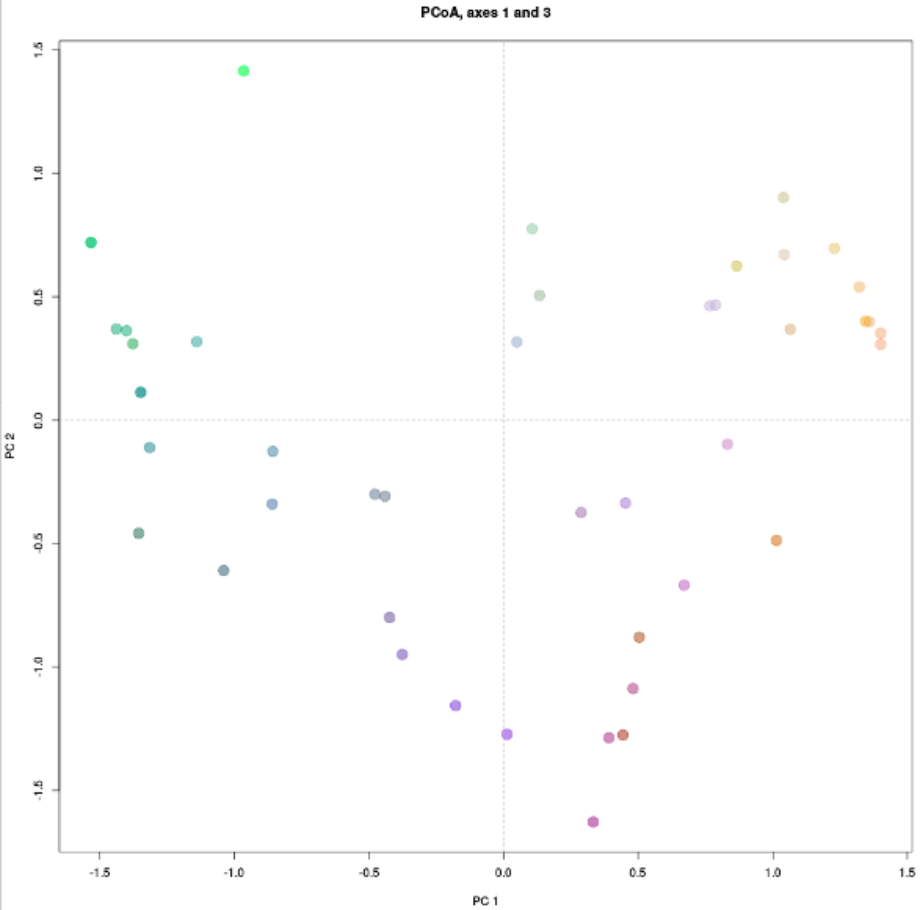
# PCoA II



```
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 colored
# 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



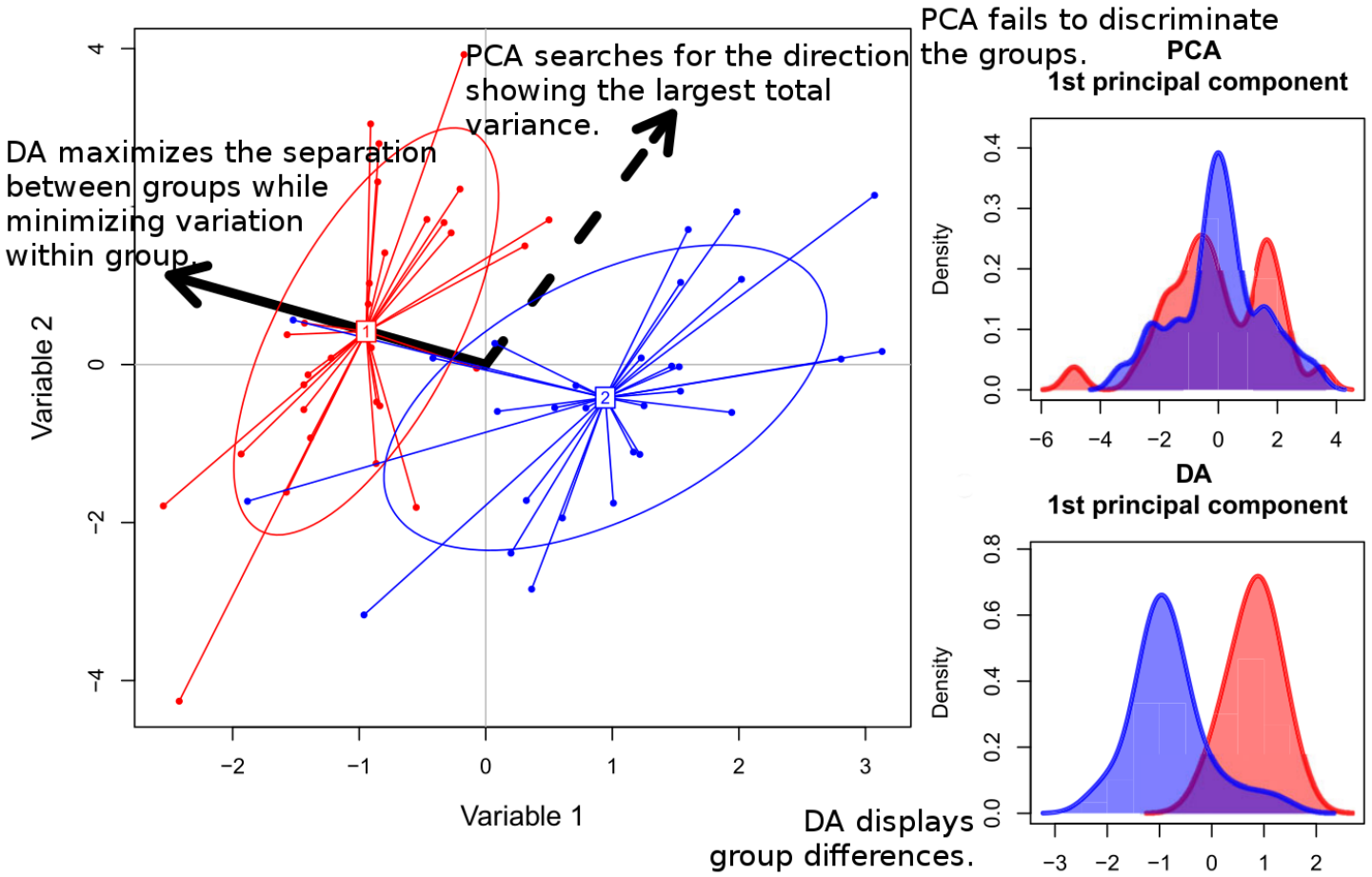



# DAPC

- Discriminant Analysis of Principal components (Jombart et al. 2010)
- Runs K-means Bayesian clustering on data transformed with PCA (reduces number of variables, speeds up process)
- Finally it runs discriminant analysis (DA) to maximize differences among groups
- 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 too complicated to you, try web interface by this command:

```
1 adegenetServer("DAPC") # Recommended to open in Google Chrome/Chromium
```

# Principal difference between PCA and DA

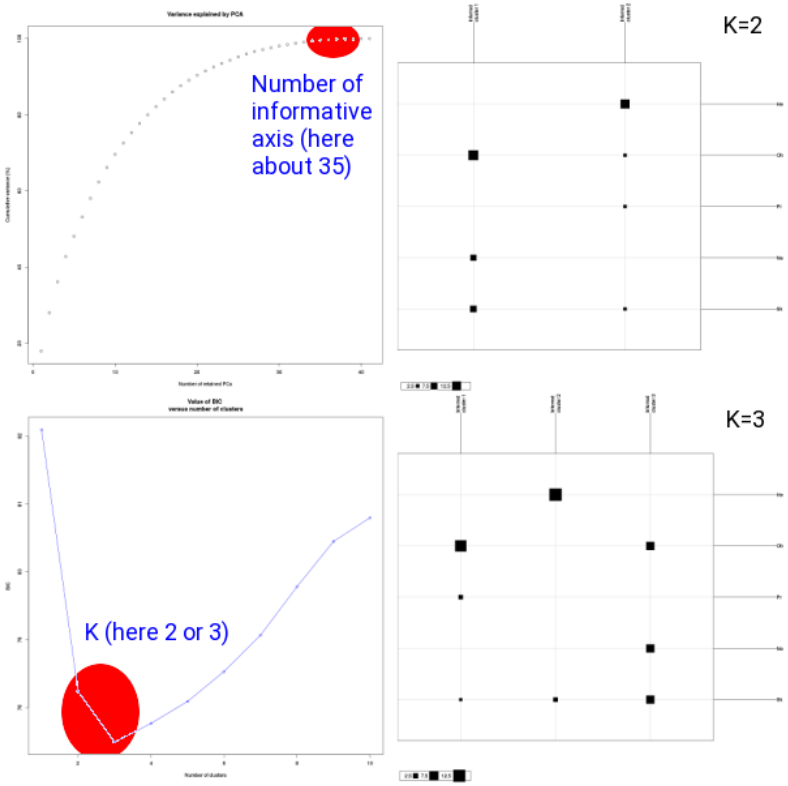


# K-find – Bayesian K-means clustering

```
# Retain all informative PC (here about 35)
# According to second graph select best K (here 2 or 3)
# Now we select K=2 and later rerun the analysis for K=3 (lines 14-18)
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=TRUE)
# 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

- Cumulative variance of axis
- BIC helps to select the best K
- Original and inferred groups

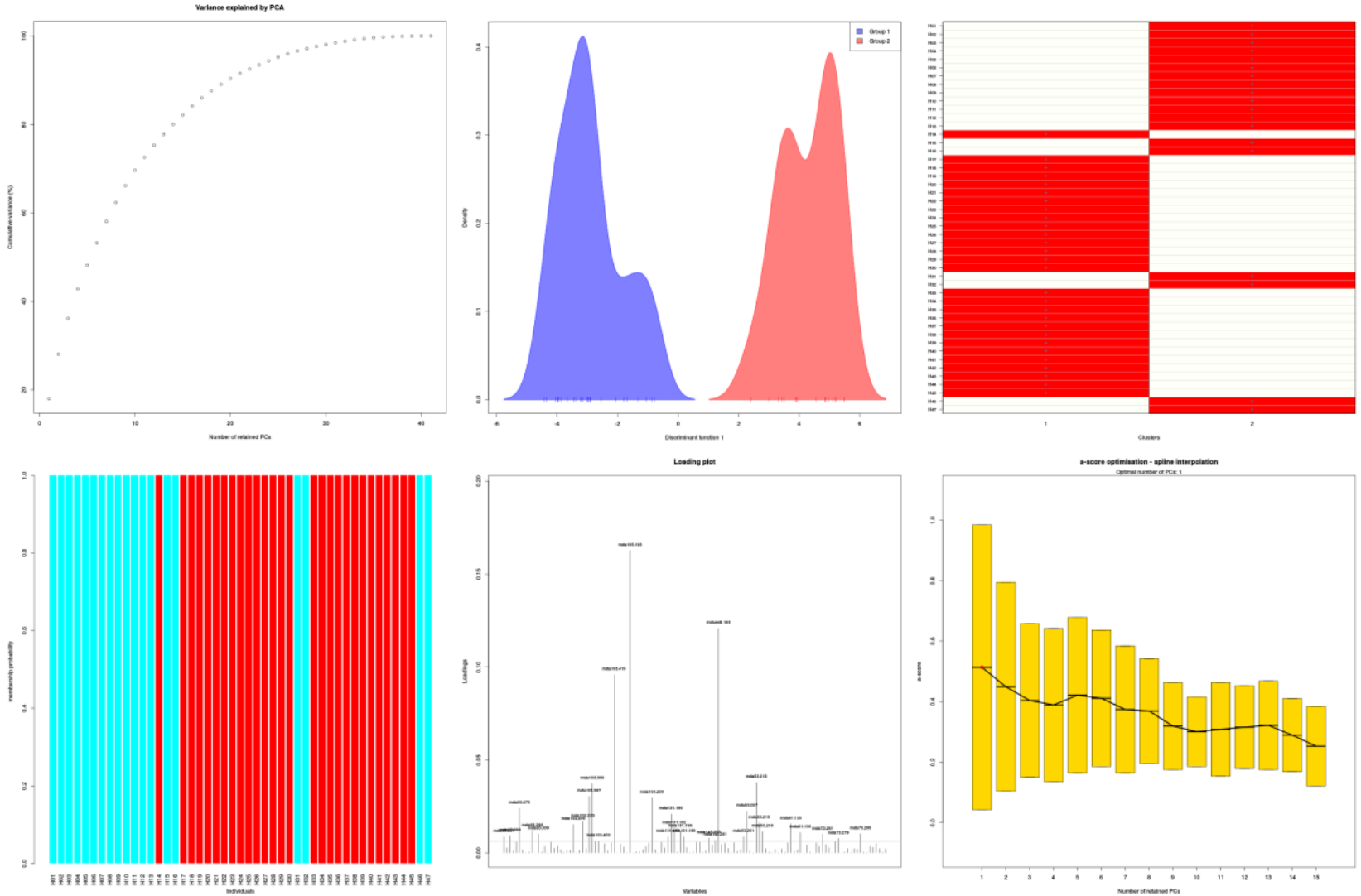



```
# See results as text
table(pop(hauss.genind),
  hauss.kfind3$grp)
hauss.kfind3
# Graph showing table of original
# and inferred populations and
# assignment of individuals
table.value(
  df=table(pop(hauss.
  genind), hauss.kfind3$grp),
  col.lab=paste("Inferred\n
  cluster",
  1:length(hauss.kfind3$size)),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

# DAPC code I

```
## K=2
# Create 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$grp, center=TRUE,
  scale=FALSE, var.contrib=TRUE, pca.info=TRUE, truenames=TRUE)
# Information
hauss.dapc
# Density function - only for first axis here!
scatter(x=hauss.dapc, xax=1, yax=1, main="DAPC", bg="white", solid=0.5,
  leg=TRUE, txt.leg=c("Group 1", "Group 2"), posi.leg="topright")
# Assignment of individuals to clusters
assignplot(x=hauss.dapc)
# Structure-like plot
compoplot(x=hauss.dapc, xlab="Individuals", leg=FALSE)
# Loadingplot - alleles the most adding to separation of individuals
loadingplot(x=hauss.dapc$var.contr)
```

# DAPC for K=2

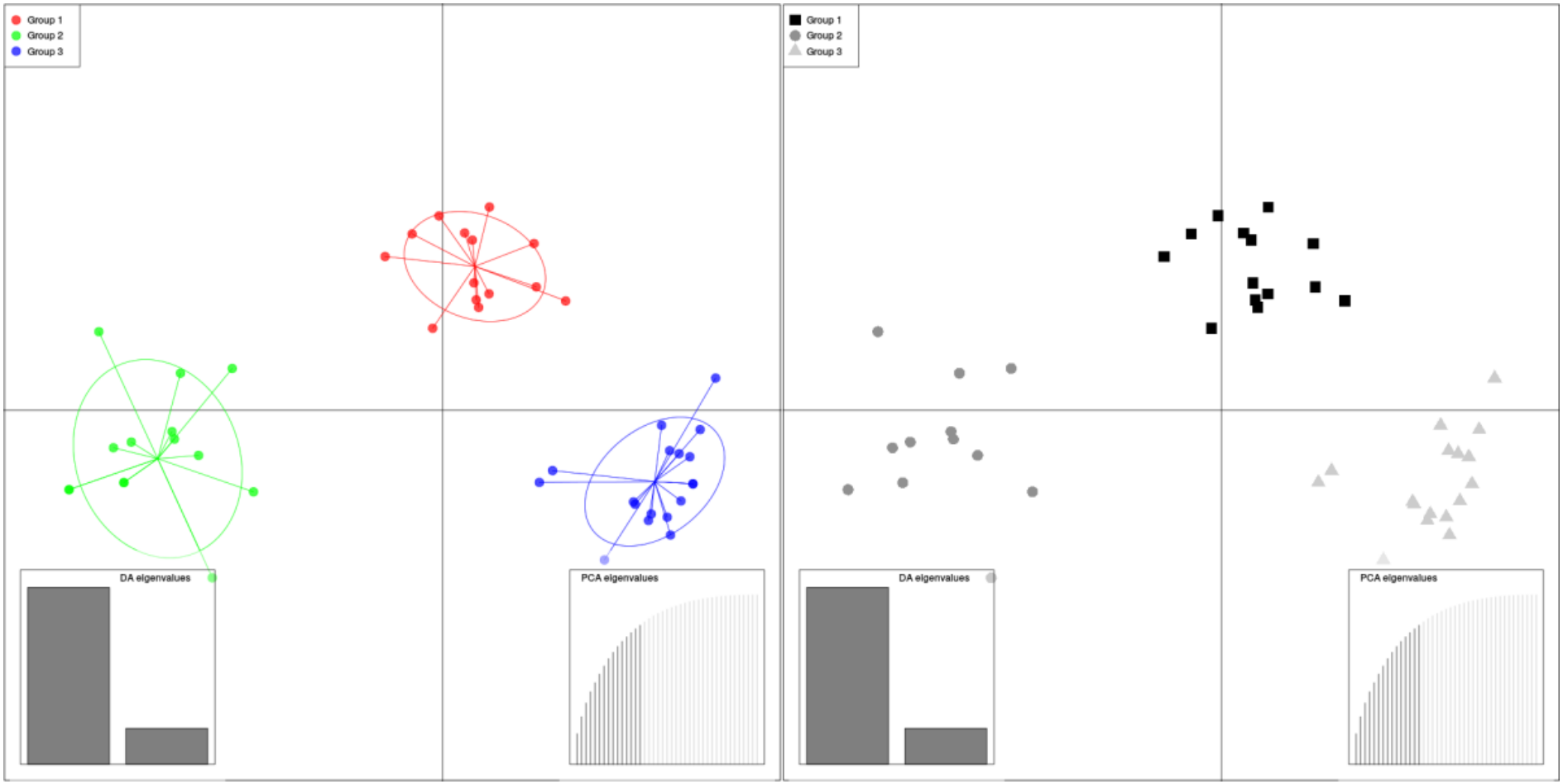

Variance explained by PCA
Cumulative variance (%)
Number of retained PCs
Group 1
Group 2
Density
Discriminant function 1
Clusters
membership probability
Individuals
Loading plot
Loadings
Variables
a-score optimisation - spline interpolation
Optimal number of PCs: 1
a-score
Number of retained PCs

# DAPC code II

```
# alfa-score - according to number of PC axis
optim.a.score(x=hauss.dapc)
## K=3
# Create DAPC
# Number of informative PC (Here 15, adegenet recommends < N/3)
# Select number of informative DA (here 2 - usually keep all of them)
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, Taraxacum haussknechtii",
  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 possibilities...
- See `?scatter` and play with it...

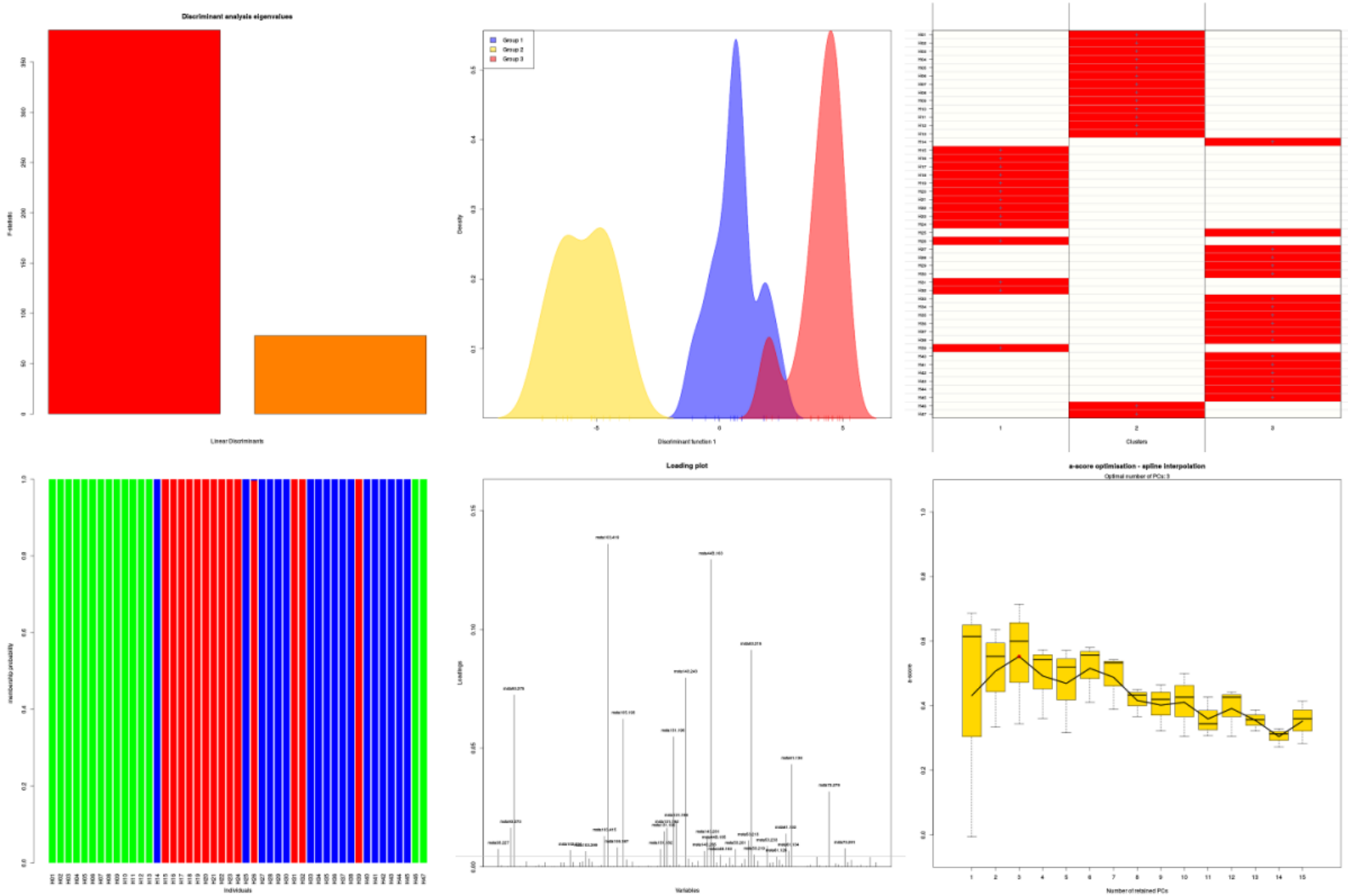
# DAPC for K=3


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

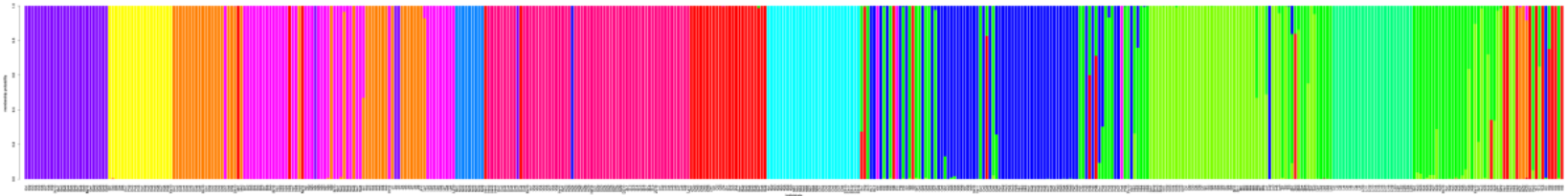
# DAPC code III

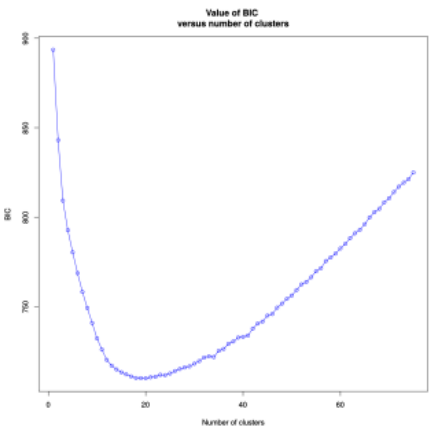
```
# Same in BW
scatter(x=hauss.dapc3, main="DAPC, Taraxacum haussknechtii",
  bg="white", pch=c(15:17), cell=0, cstar=0, solid=1, cex=2.5, clab=0,
  col=grey.colors(3, start=0, end=0.8, gamma=2, alpha=0), posi.da=
  "bottomleft", scree.pca=TRUE, posi.pca="bottomright", leg=TRUE,
  txt.leg=c("Group 1", "Group 2", "Group 3"), posi.leg="topleft")
# Density function - only for first axis here!
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)
```

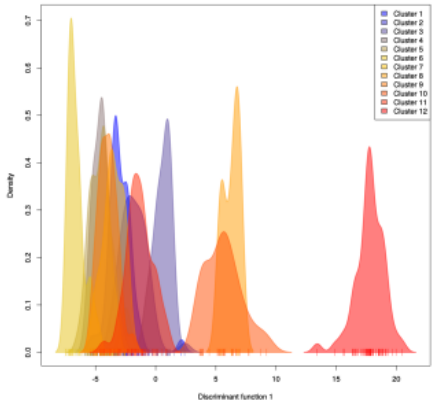
# DAPC for K=3, extra information

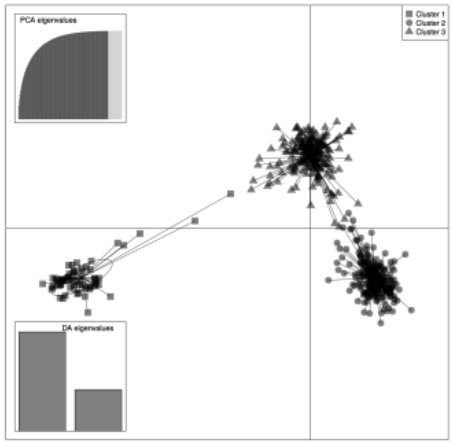


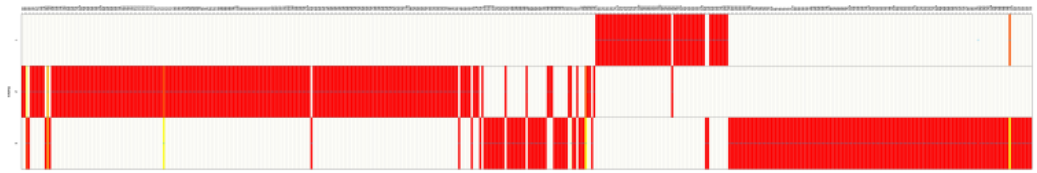
# Another DAPC example

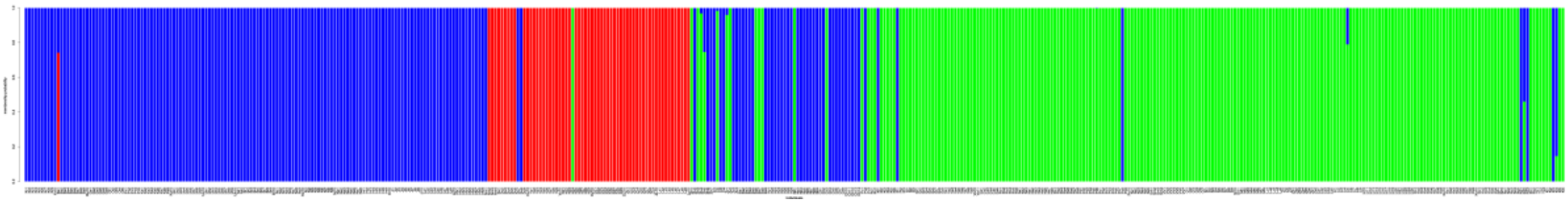






# Special functions to work with huge SNP data sets

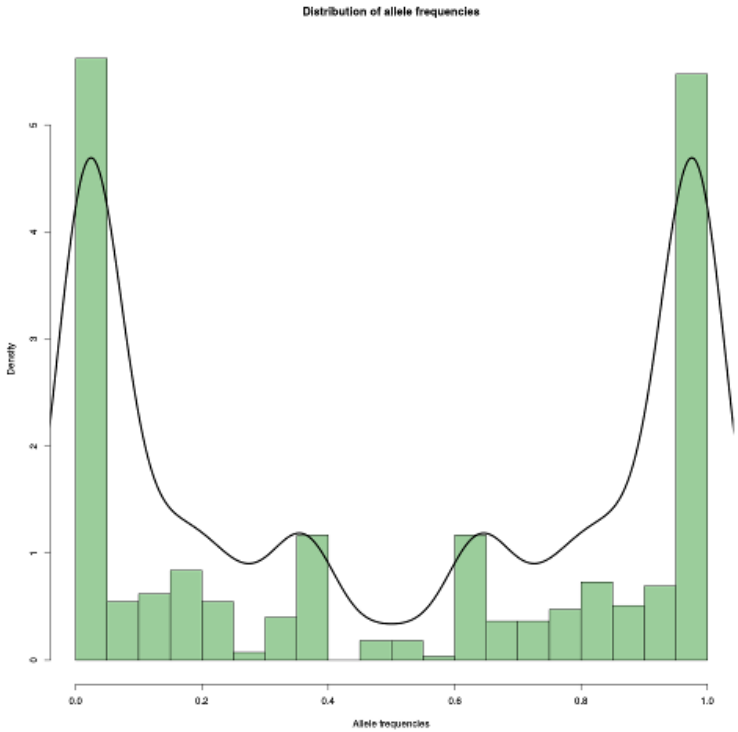
```
# Plot of missing data (white) and number of 2nd alleles
glPlot(x=usflu.genlight, legend=TRUE, posi="topleft")
# Sum of the number of second allele in each SNP
usflu.freq <- glSum(usflu.genlight)
# Plot distribution of (second) allele frequencies
hist(x=usflu.freq, proba=TRUE, col="gold", xlab="Allele
  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)
```

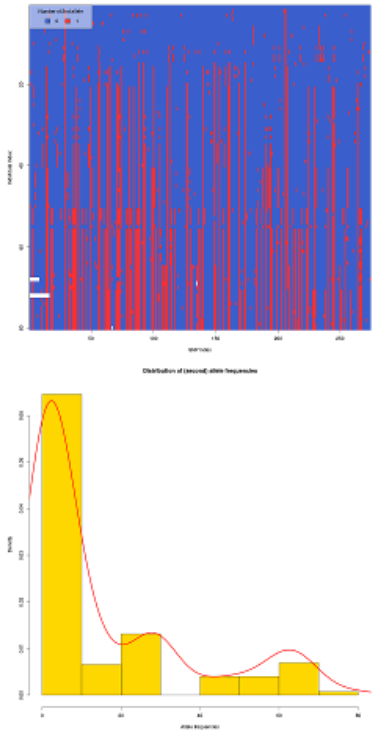
# Number of missing values in each locus

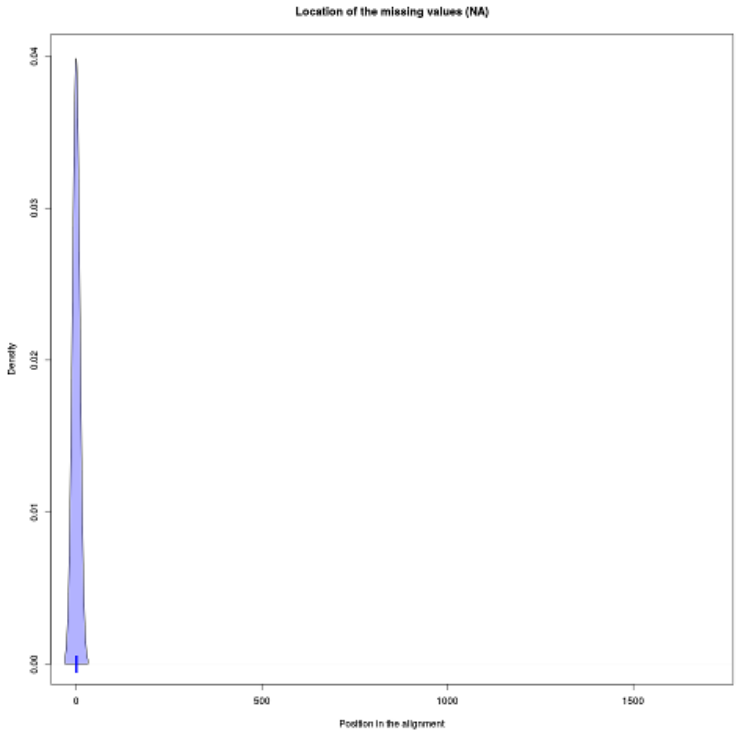
```
# Play with bw parameter to get optimal image
usflu.na.density <- density(glNA(usflu.genlight), bw=10)
# Set range of xlim parameter from 0 to the length
# of original alignment
plot(x=usflu.na.density, type="n", xlab="Position in the alignment",
  main="Location of the missing values (NA)", xlim=c(0, 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 – not needed for smaller (normal) datasets
- Lets keep hoping in fast development of computers...
- Note for Windows users: To speed up the processing, `gl*` functions use parallelisation library, which is not available on Windows – add parameter `parallel=FALSE` to be able to use them on Windows

# 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



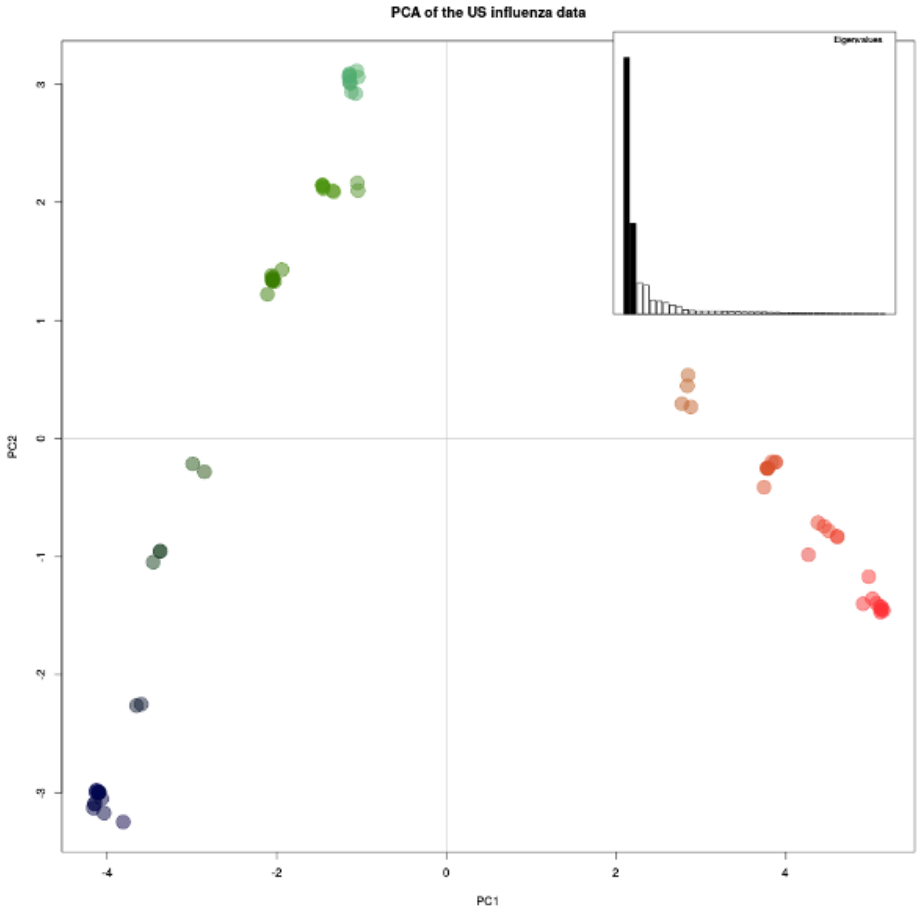



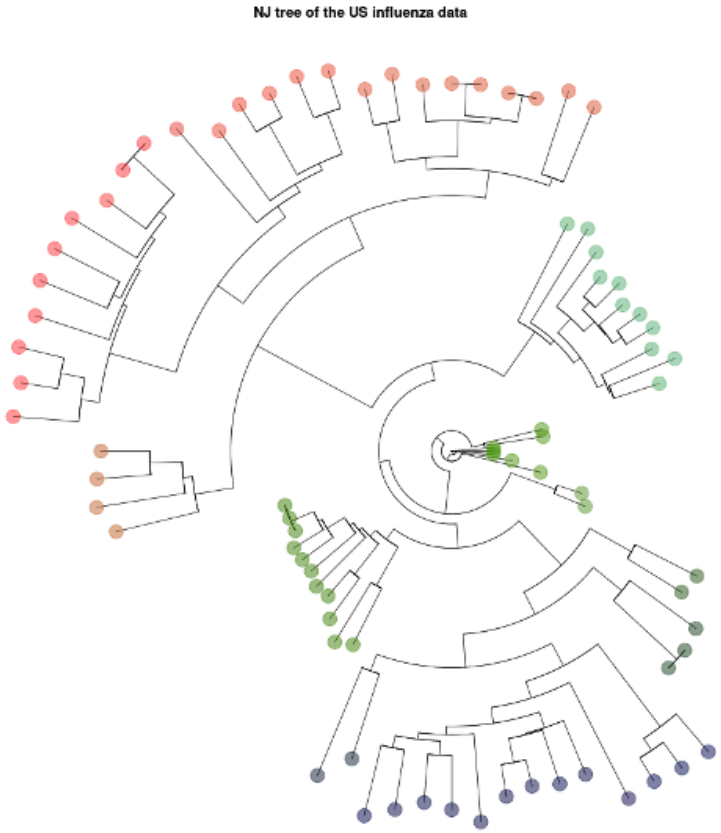


# PCA, NJ and genlight objects

```
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 colored phylogenetic tree
plot.phylo(x=usflu.tree.genlight, typ="fan", show.tip=FALSE)
tiplabels(pch=20, col=num2col(usflu.annot[["year"]],
  col.pal=usflu.pal), cex=4)
title("NJ tree of the US influenza data")
```

# PCA, NJ and genlight objects



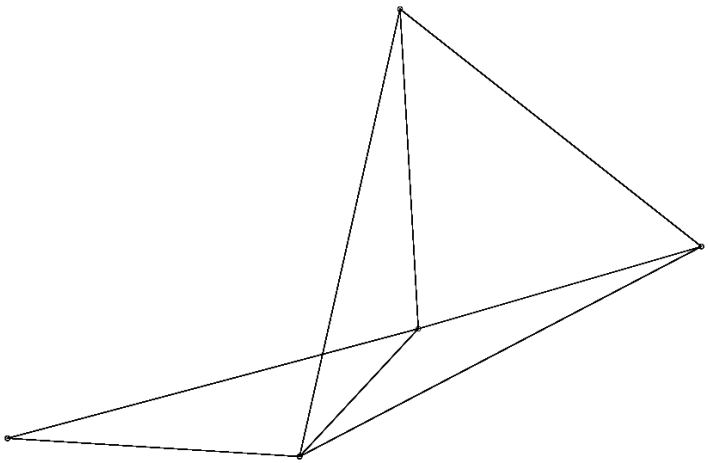
# Short overview of spatial genetics (in R)

## 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 criticized 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 specialized courses and literature

# Moran's *I*

- "Only" autocorrelation index – no genetic/evolutionary model involved – sometimes criticized as biologically irrelevant mechanism
- This (or similar) approach can be used to test correlation between another characteristics (typically used in ecology)
- Calculations are done according to connectivity network connecting individuals/populations (created by `chooseCN`) – carefully check its options and try several parameters
- Pay attention which hypothesis is tested (i.e. if lower, greater or two-sided) – similar to T-test



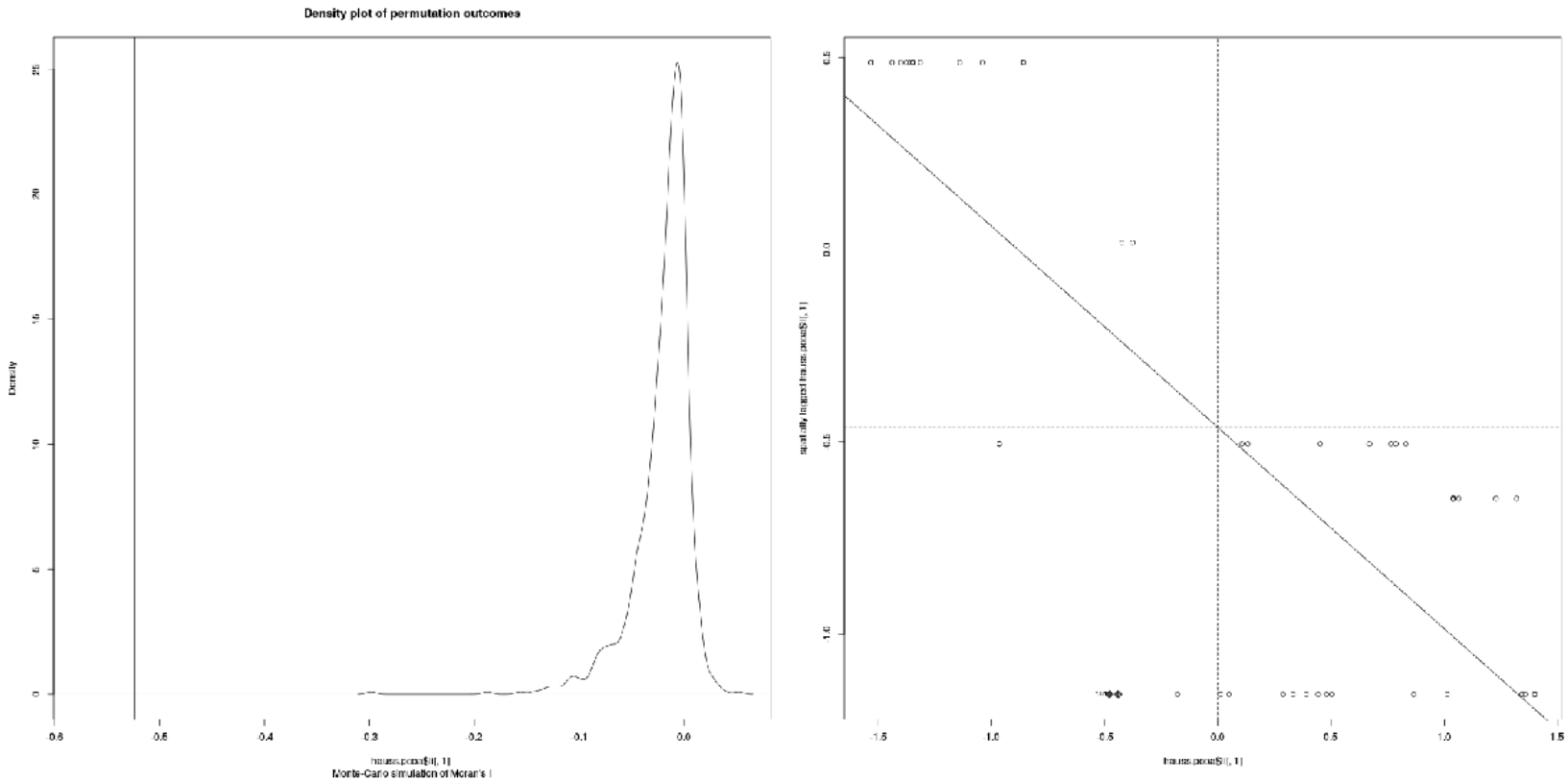
# Calculation of Moran's *I*

```
# Load required library
library(spdep)
# 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
# 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)
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
```

# Calculation of Moran's I II

```
# Test of Moran's I for 1st PCoA axis
hauss.pcoa1.mctest <- moran.mc(x=hauss.pcoa$li[,1],
  listw=hauss.connectivity, alternative="greater", nsim=1000)
hauss.pcoa1.mctest
# Output:
  Monte-Carlo simulation of Moran I
data:  hauss.pcoa$li[, 1]
weights: hauss.connectivity
number of simulations + 1: 1001
statistic = -0.5163, observed rank = 1, p-value = 0.999
alternative hypothesis: greater
# Plot the results
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

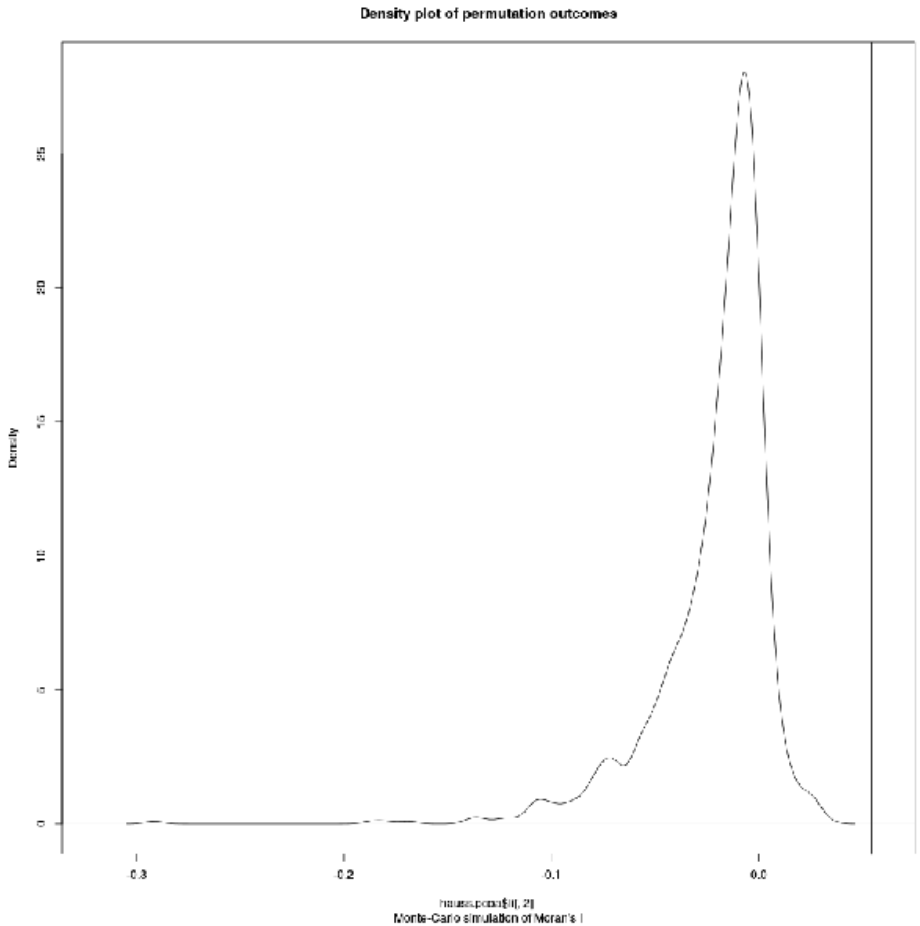


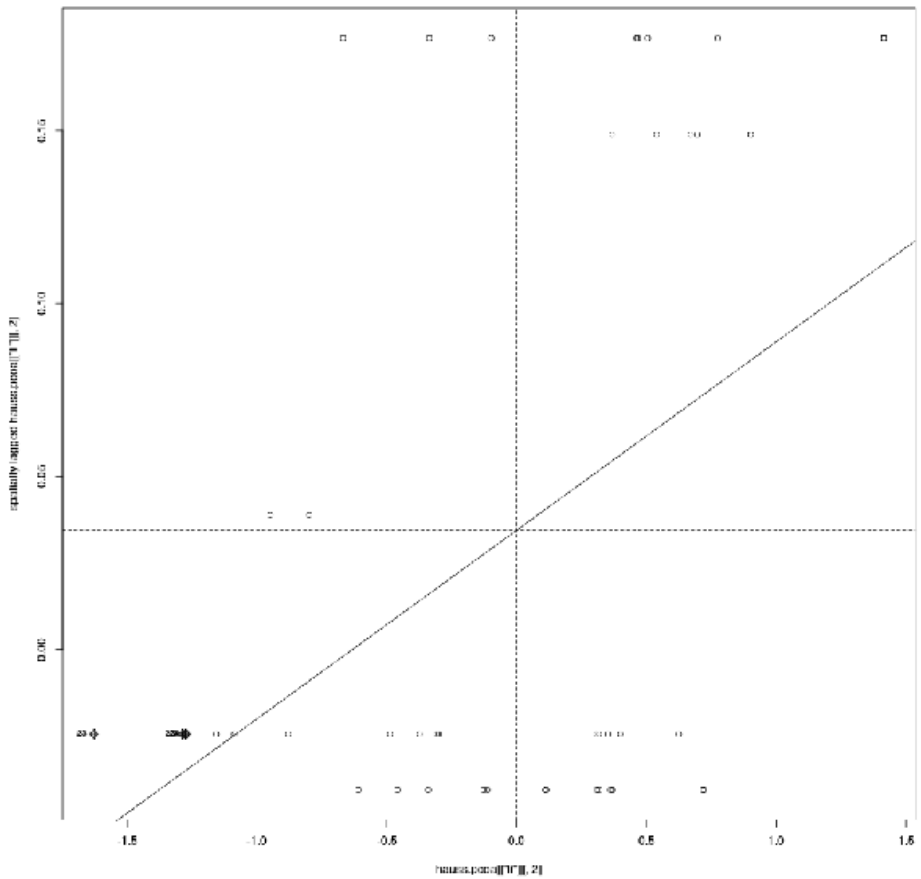

- Tested hypothesis "greater" – **no** significant **positive** autocorrelation
- If testing for hypothesis "less" – significant **negative** autocorrelation – individuals are significantly different

# 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$li[, 2]
weights: hauss.connectivity
number of simulations + 1: 1001
statistic = 0.0545, observed rank = 1001, p-value = 0.000999
alternative hypothesis: greater
# Plot the results
plot(hauss.pcoa2.mctest) # Plot of densitiy of permutations
moran.plot(x=hauss.pcoa$[,2], listw=hauss.connectivity) # PC plot
```

# Second axis is surprisingly significant



- Tested hypothesis "greater" – there **is** significant positive autocorrelation – individuals are genetically similar over space

# Spatial Analysis of Principal Components (sPCA)

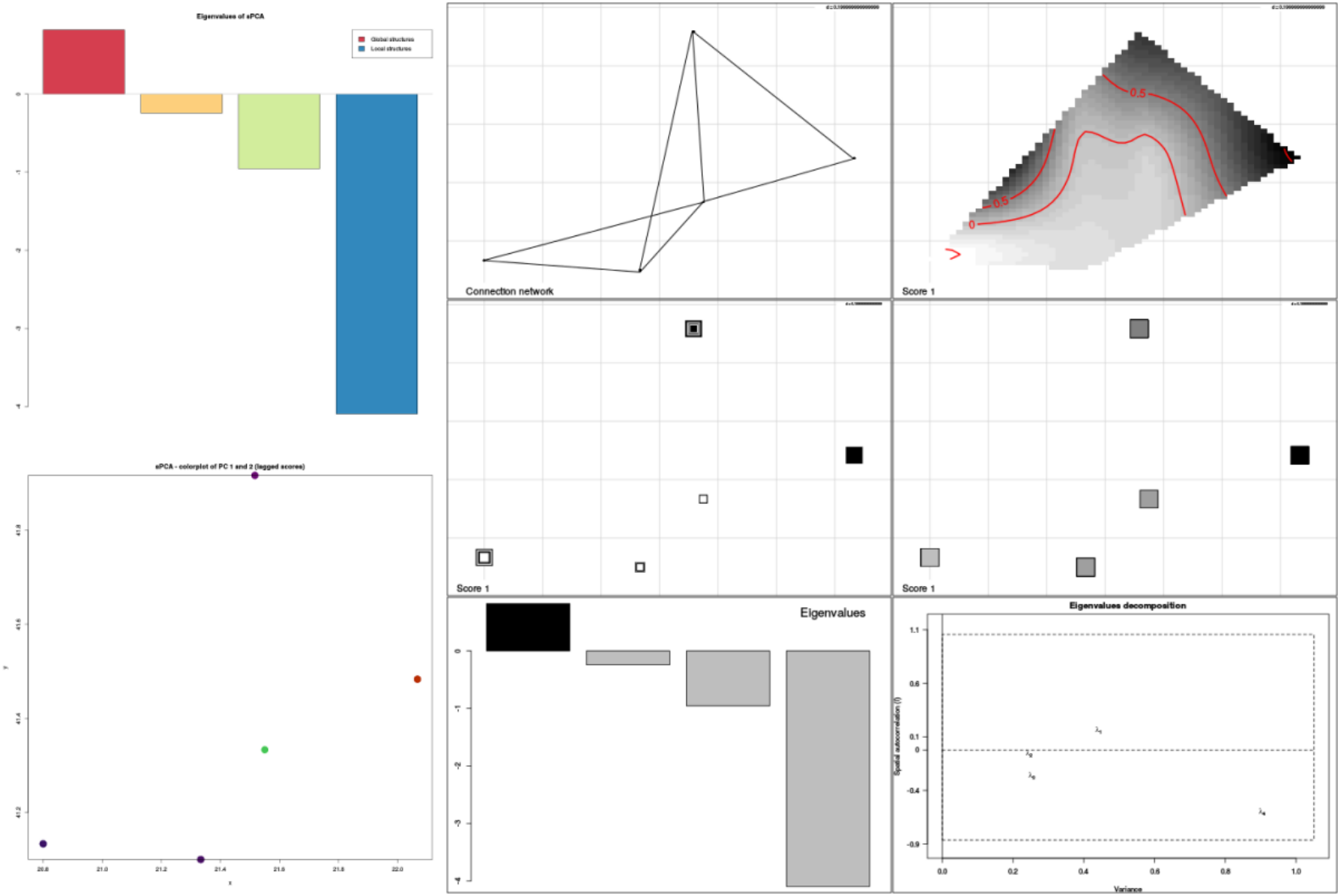
- Implemented in adegenet, see `adegenetTutorial("spca")`
- Analyzes matrix of relative allele frequencies of genotypes/populations and spatial weighting matrix
- The geographical matrix is usually (as for Moran's *I*) created by `chooseCN()` – creates connectivity network among entities (genotypes/populations)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

```
data(rupica) # Loads adegenet's training dataset
             # Rupicapra rupicapra from French Alps
# Try various settings for chooseCN (type=X) - type 1-4 as there
# are identical coordinates (multiple sampling from same locality)
chooseCN(xy=rupica$other$xy, ask=TRUE, type=5/6/7, plot.nb=TRUE,
  edit.nb=FALSE, ...) # Play with settings little bit...
?chooseCN # See for more details - select the best "type" for your data
```

# Calculations of sPCA

```
hauss.spca <- spca(obj=hauss.genind, cn=hauss.connectivity,
  scale=TRUE, scannf=TRUE)
# Plot eigenvalues of sPCA - global 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 I

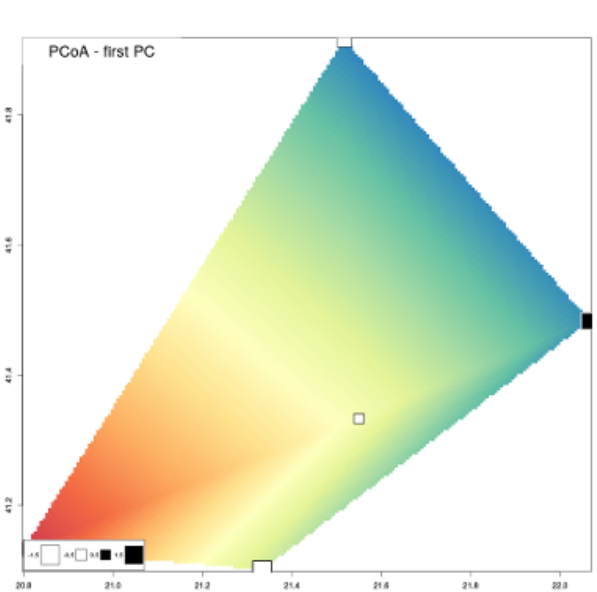


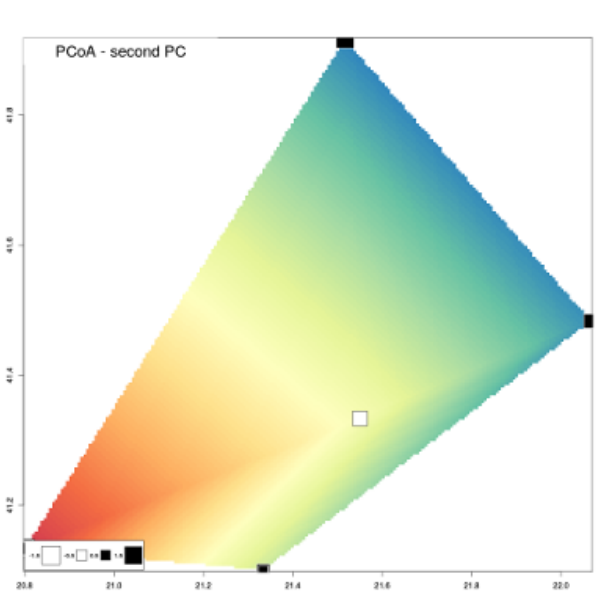
# Map of genetic clines

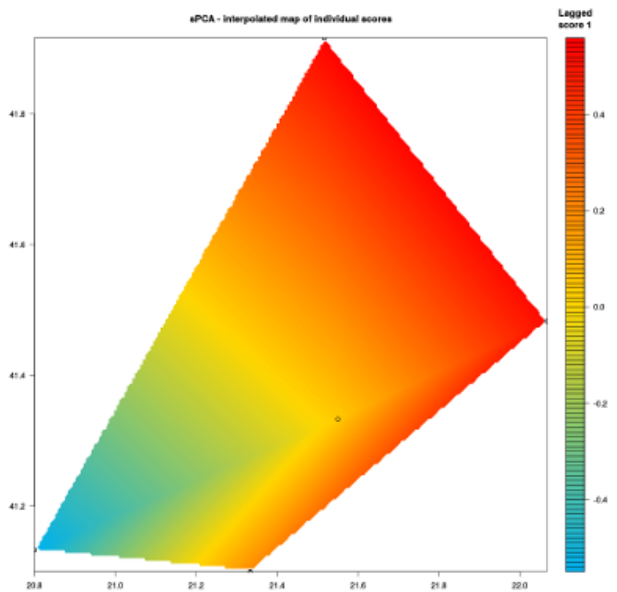
```
library(akima) # It is needed for manipulation with coordinates
# Transform the coordinates
hauss.spca.temp <- interp(other(hauss.genind)$xy[,1],
  other(hauss.genind)$xy[,2], hauss.spca$ls[,1],
  xo=seq(min(other(hauss.genind)$xy[,1]),
  max(other(hauss.genind)$xy[,1]), le=200),
  yo=seq(min(other(hauss.genind)$xy[,2]),
  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 II

```
# 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())
```

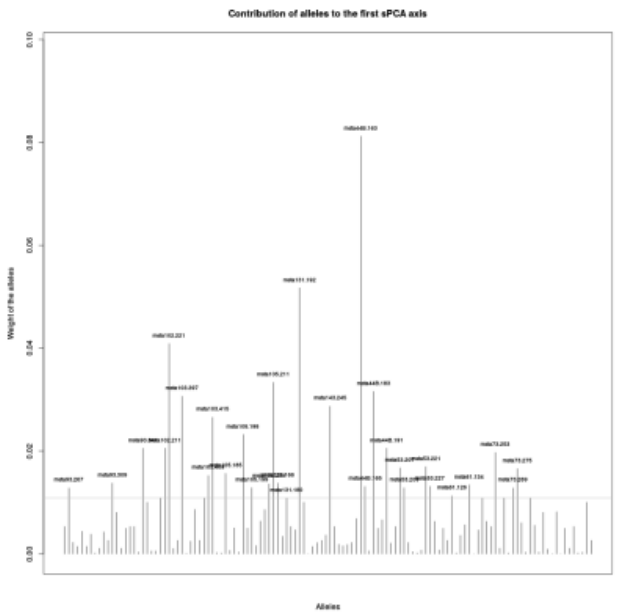


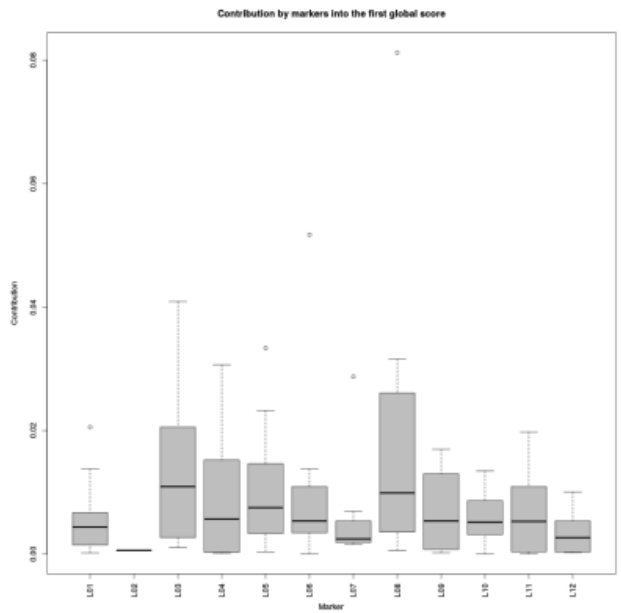



# Loading plots – which alleles contribute the most?

```
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")
```



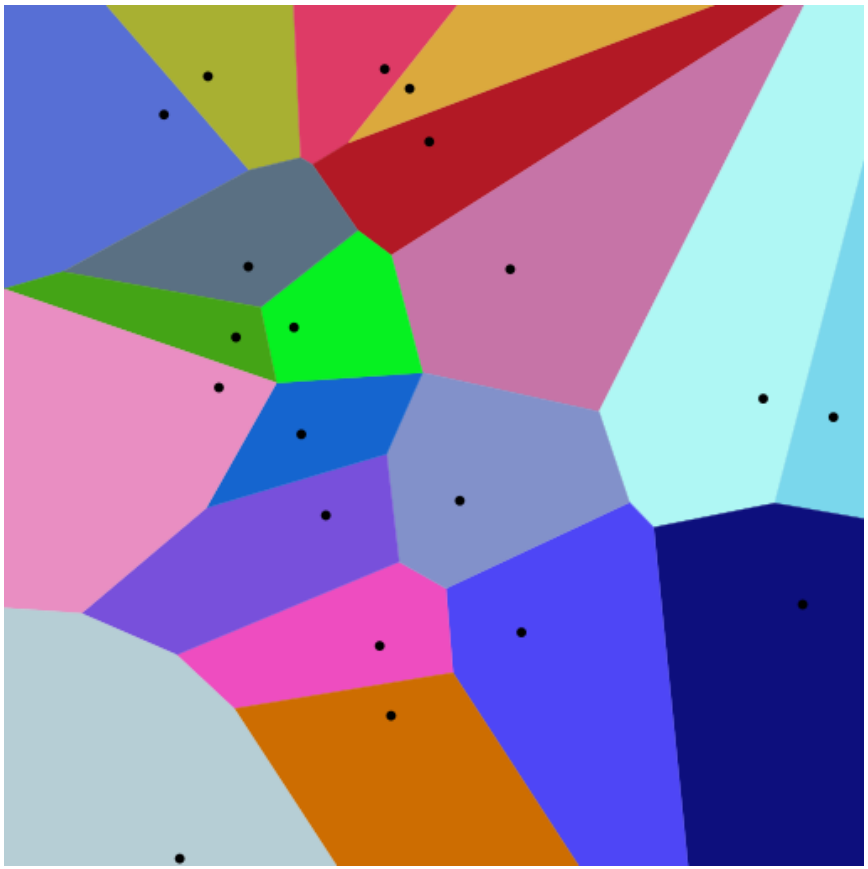



# Monmonier's algorithm – genetic boundaries

- Finds boundaries of maximum differences between contiguous polygons of a tessellation
- Detects genetic boundaries among georeferenced genotypes (or populations)
- For more information see `adegenetTutorial("basics")`
- Requires every point to have unique coordinates – in case of population data it is better to work with populations, not individuals (but it is not ideal)
- It uses Voronoi tessellation

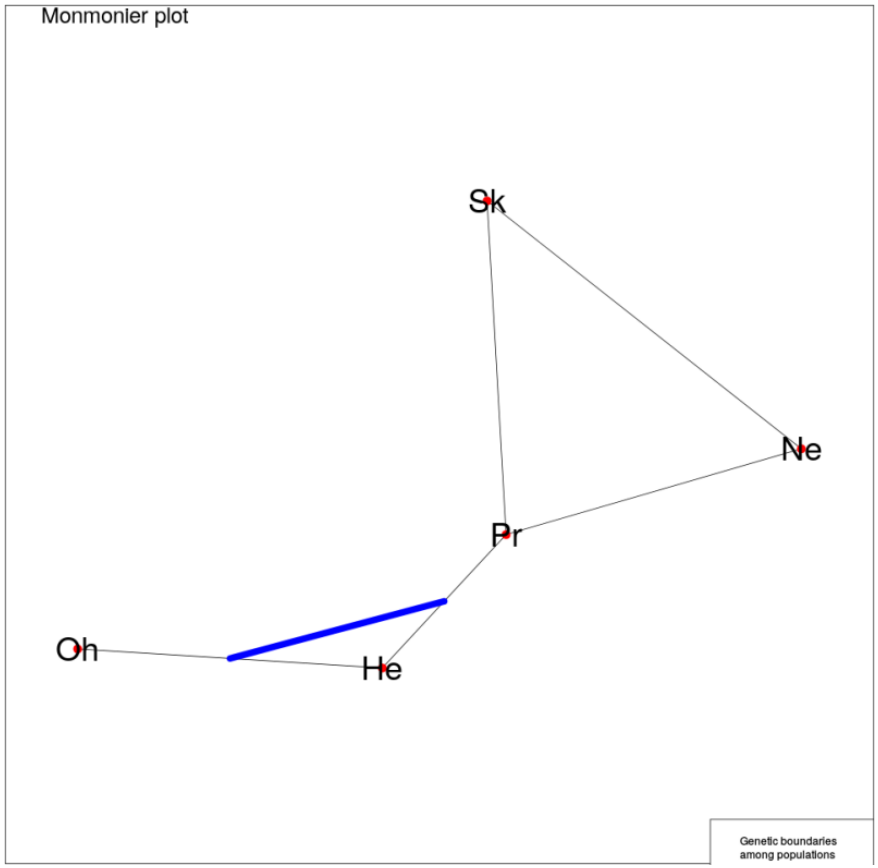
```
# Calculates Monmonier's function (for threshold use 'd')
hauss.monmonier <- monmonier(xy=hauss.genpop$other$xy, dist=
  dist(hauss.genpop@tab), cn=chooseCN(hauss.genpop$other$xy,
  ask=FALSE, type=2, plot.nb=FALSE, edit.nb=FALSE), nrun=1)
coords.monmonier(hauss.monmonier) # See result as text
```

# Voronoi tessellation



- In simplest case, all points have certain area and all points within this area are closer to the respective "main" point than to any other "neighbor" point
- Extreme differences among size of areas make computational problems and results are unstable – this typically occurs when calculations are done on individual level an there are large distances among populations

# 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=popNames(hauss.genpop),
  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 bigger 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 colored symbols of samples and/or `text()` to add text labels

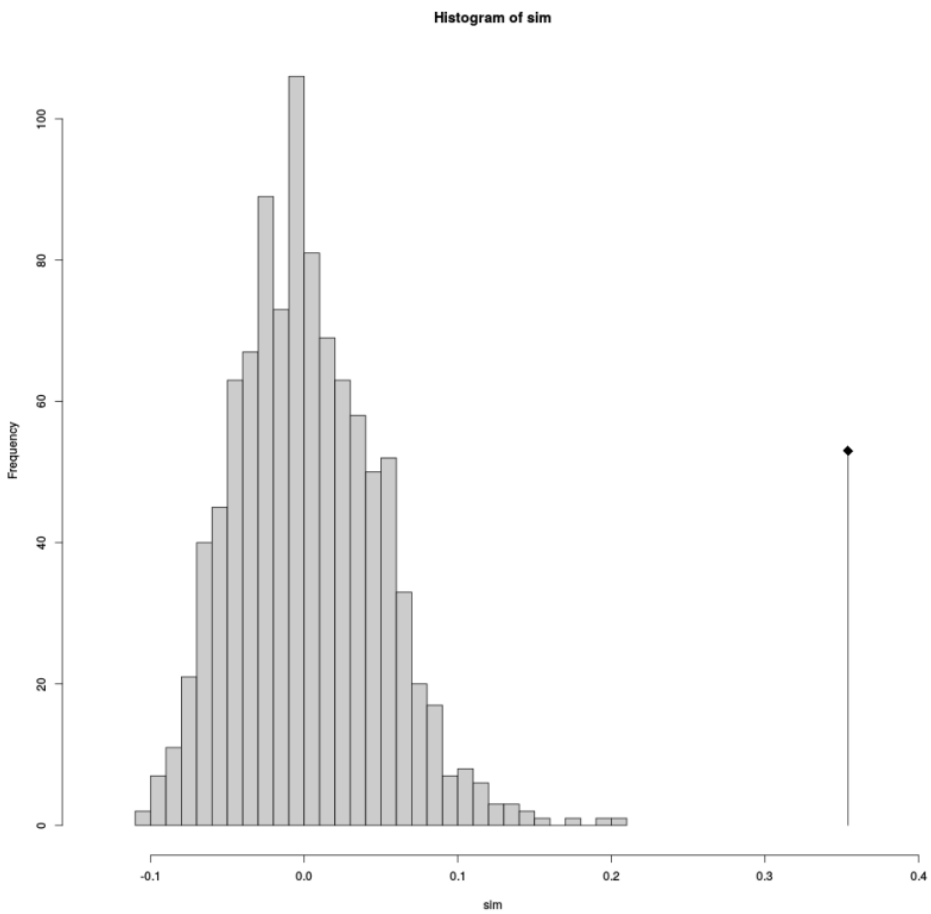
# Mantel test

- Originally created for biomedicine to test correlation between treatment and diseases
- "Only" correlation of two matrices – no biologically relevant underlying model – because of that it is heavily criticized (mainly in ecology)
- It is universal method usable for plenty of tasks
- Test of spatial and genetic relationships is probably one of few biologically relevant applications
- Package vegan (set of ecological tools) has implementation to test genetic similarity in various distance classes – not only overall result – very useful

# 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)
```

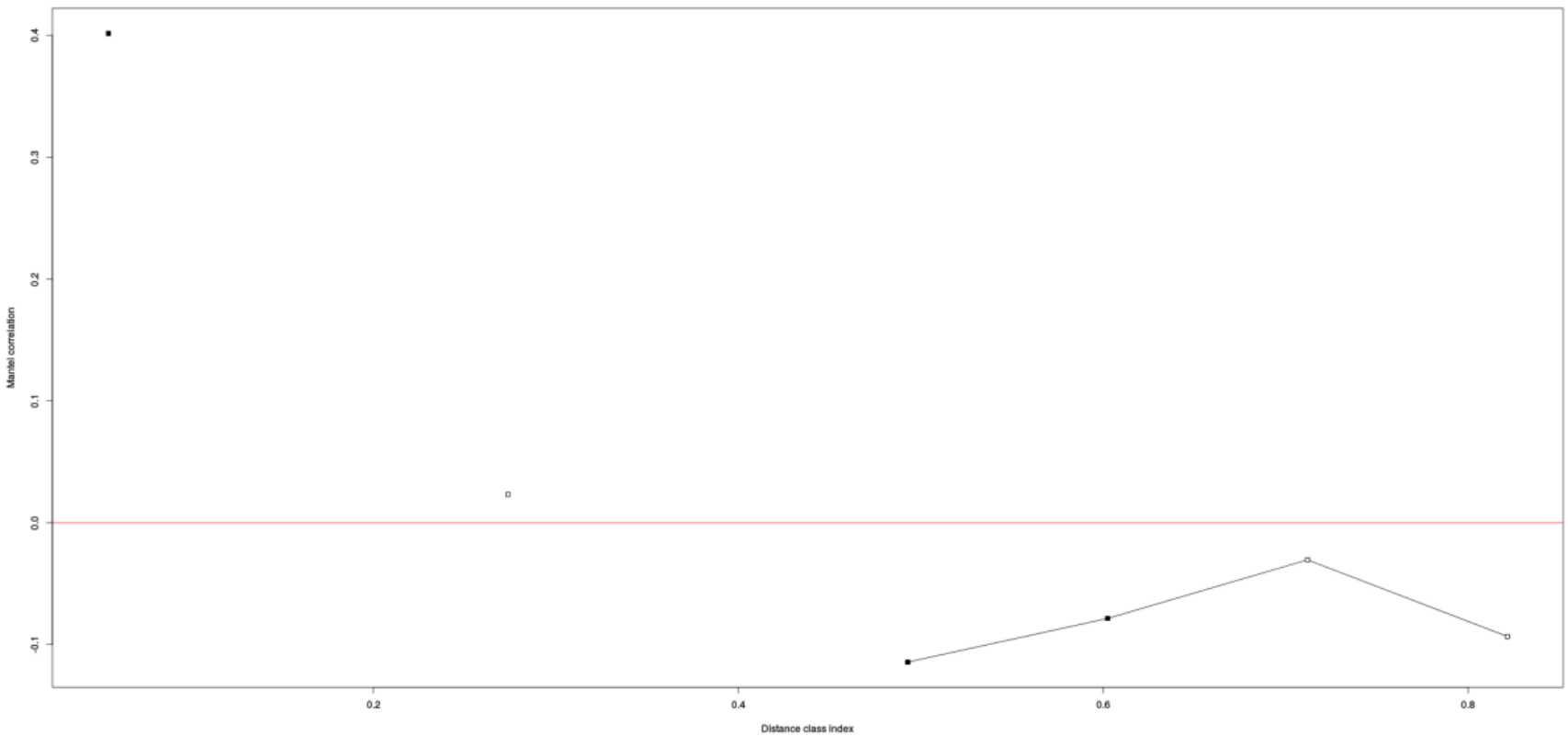
# Mantel test outputs – strongly significant




```
hauss.mantel # See output
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 0.0021389
```

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

# Plot of Mantel Correlogram Analysis




Correlation (genetic similarity) in several distance classes (positive **[up]** in short distance **[left]**, negative **[down]** in long **[right]**; **[full]** – significant, **[empty]** – not significant) – see `?mantel.correlog` for details

# Mantel correlogram – text output

```
hauss.mantel.cor # See the text output:
Mantel Correlogram Analysis
Call:
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)
       class.index     n.dist Mantel.cor Pr(Mantel) Pr(corrected)
D.cl.1    0.054757 532.000000   0.409545     0.0010       0.000999 ***
D.cl.2    0.164271   0.000000         NA         NA             NA
D.cl.3    0.273784  52.000000   0.028055     0.2797       0.279720
D.cl.4    0.383298   0.000000         NA         NA             NA
D.cl.5    0.492812 466.000000  -0.097214     0.0160       0.031968 *
D.cl.6    0.602325  36.000000  -0.086288     0.0140       0.041958 *
D.cl.7    0.711839 108.000000  -0.044109     0.1568       0.313686
D.cl.8    0.821353 458.000000  -0.095780     0.0100       0.049950 *
...            ...        ...        ...        ...            ...
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# About Geneland

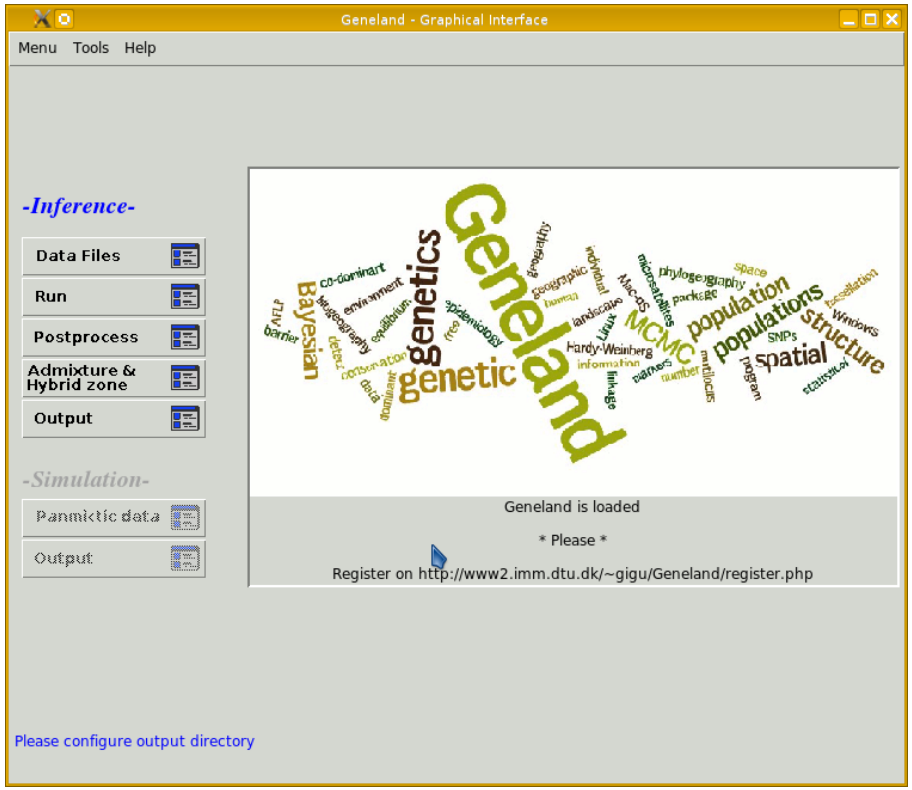
- Works with haploid and diploid codominant markers (microsatellites or SNPs)
- Spatially explicit Bayesian clustering
- Produces maps of distribution of inferred genetic clusters
- Relative complicated tool with various modeling options

```
# 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 https://www2.imm.dtu.dk/~gigu/Geneland/

# Geneland GUI

- Some tasks are easier in GUI, some in command line...
- Command line is great for its repeatability...
- Always read manual! It is not the simplest tool...



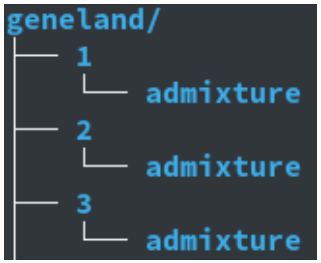
# Loading and conversions of coordinates

```
# 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=
  "https://soubory.trapa.cz/rcourse/haussknechtii_geneland.txt",
  na.string="-999", header=FALSE, sep="\t")
dim(hauss.geneland.data)
hauss.geneland.data
```

# Before running MCMC

- Monte Carlo Markov Chains (MCMC) require usually millions of generations (iterations, `nit`) to find optimal solution
- Beginning (∼10-20%) of the steps (`burnin`) use to be very unstable and useless for following analyses and it is discarded
- Geneland allows to set density of sampling among generations (`thinning`) – it is not necessary to sample every generation
- Within millions of generations we can sample every 1000-10000$^{th}$ generation
- Denser sampling produces smoother data, but can consume too much disk space...

Directory structure for Geneland:



# 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
  # Inference - 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
# In practice set much higher number of iterations (nit),
# appropriate sampling (thinning) and longer burnin
# 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 - tessellations
  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 in output directories
- Not every graphical software can handle PS (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 (the others are empty)

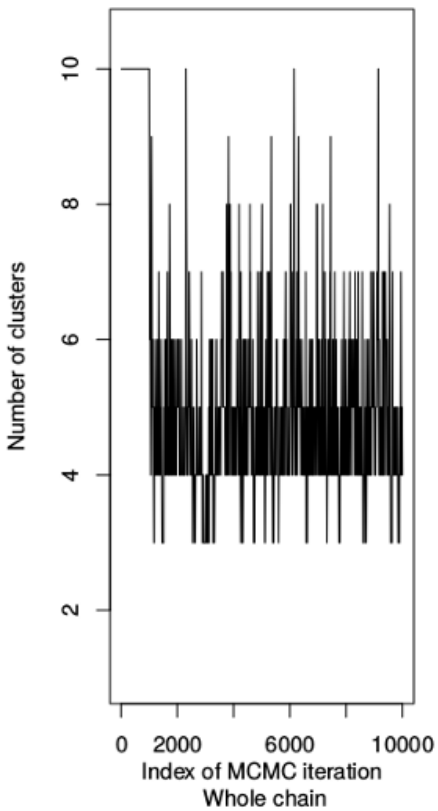
# Determine which run is the best

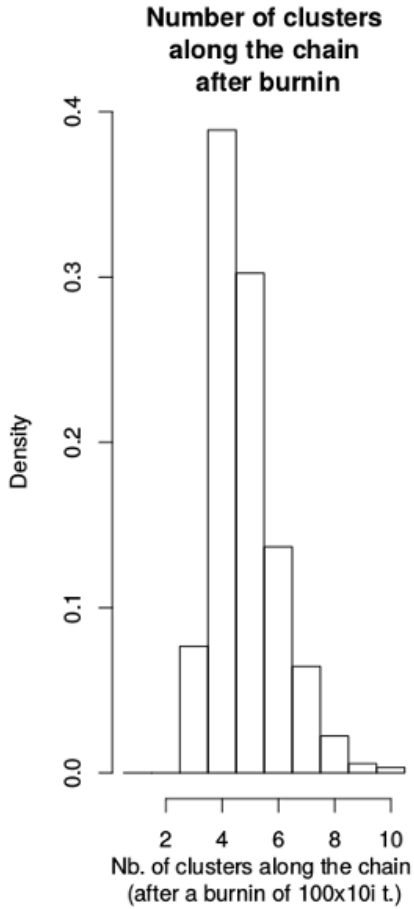
```
# Calculate average posterior probability
hauss.geneland.lpd <- rep(NA, hauss.geneland.nrun)
for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
  hauss.geneland.path.mcmc <- paste("geneland/",
    hauss.geneland.irun, "/", sep="")
  hauss.geneland.path.lpd <- paste(hauss.geneland.path.mcmc,
    "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)
[1] 5 1 4 3 2 # Run 5 is the best here
hauss.geneland.lpd # Here the runs are unsorted
[1] -645.0238 -782.7912 -676.9559 -664.9947 -601.7902 # Run 5 wins
```

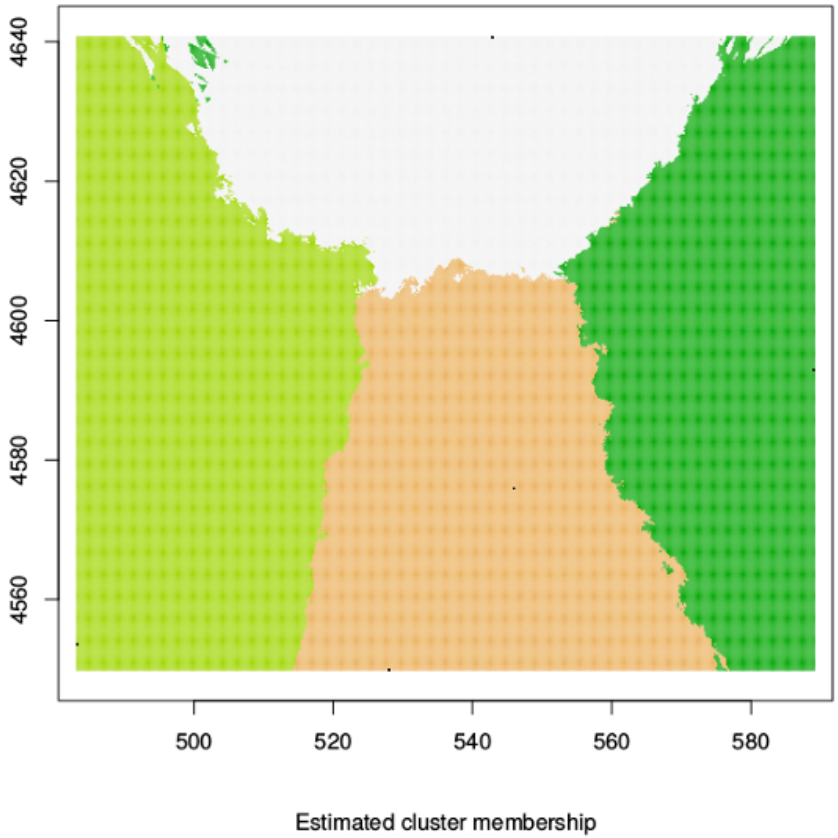
- We will use figures and $F_{ST}$ outputs only from the best run
- It is useful to keep all runs especially for comparison if there are different solutions with similar posterior probability

# MCMC chain, number of clusters and their map

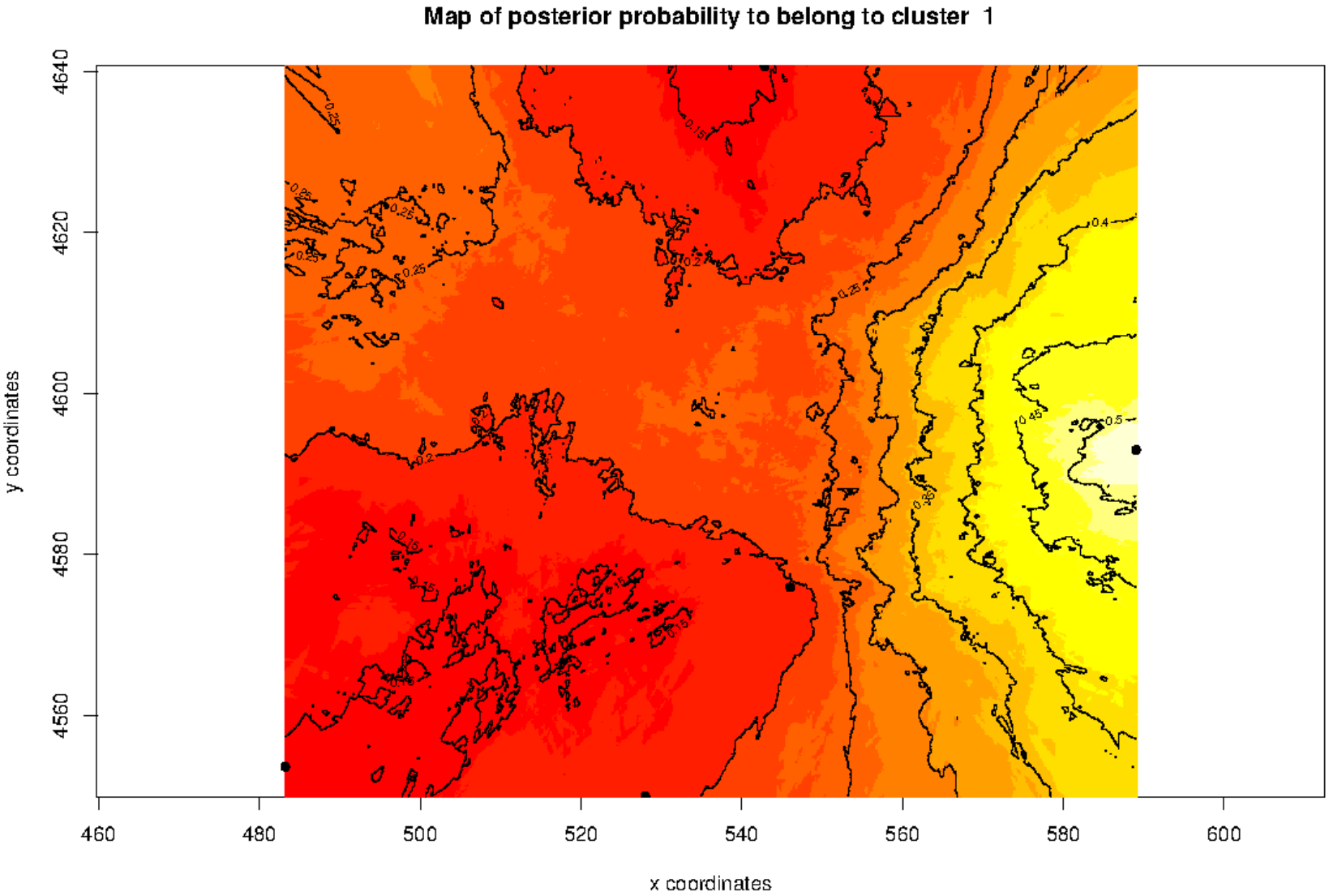
MCMC did not converge yet – too few generations, the most likely solution is K=4 followed by K=5. Final product is map of distribution of genetic clusters.



# Map of posterior probability of belonging into cluster 1

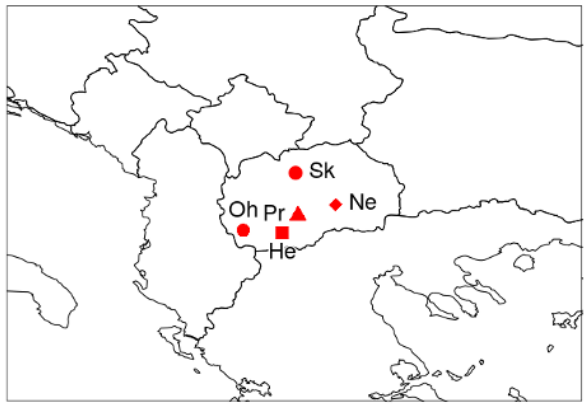


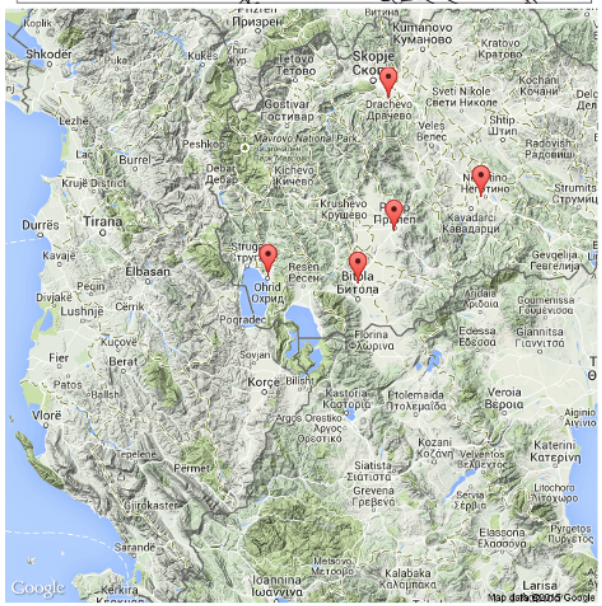
# Very basic mapping in R

```
# 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(popNames(hauss.genind)), 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. Parameter markers contain
# data frame with coordinates and
# possibly with more information
hauss.gmap <- GetMap(center=
  c(lat=41, lon=21), size=c(640,
  640), destfile="gmap.png",
  zoom=8, markers=hauss.coord,
  maptype="terrain") # Plot saved map:
PlotOnStaticMap(MyMap=hauss.gmap)
```



# OpenStreetMaps and datasets from mapproj

```
# Plot on OpenStreetMaps - server is commonly overloaded and does not
# respond correctly - in fact, it rarely works...
GetOsmMap(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
      # outdated, see https://cran.r-project.org/web/packages/mapdata/
library(mapproj) # Convert 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=3)
```

# Plotting on SHP files I

Get SHP files from https://soubory.trapa.cz/rcourse/macedonia.zip

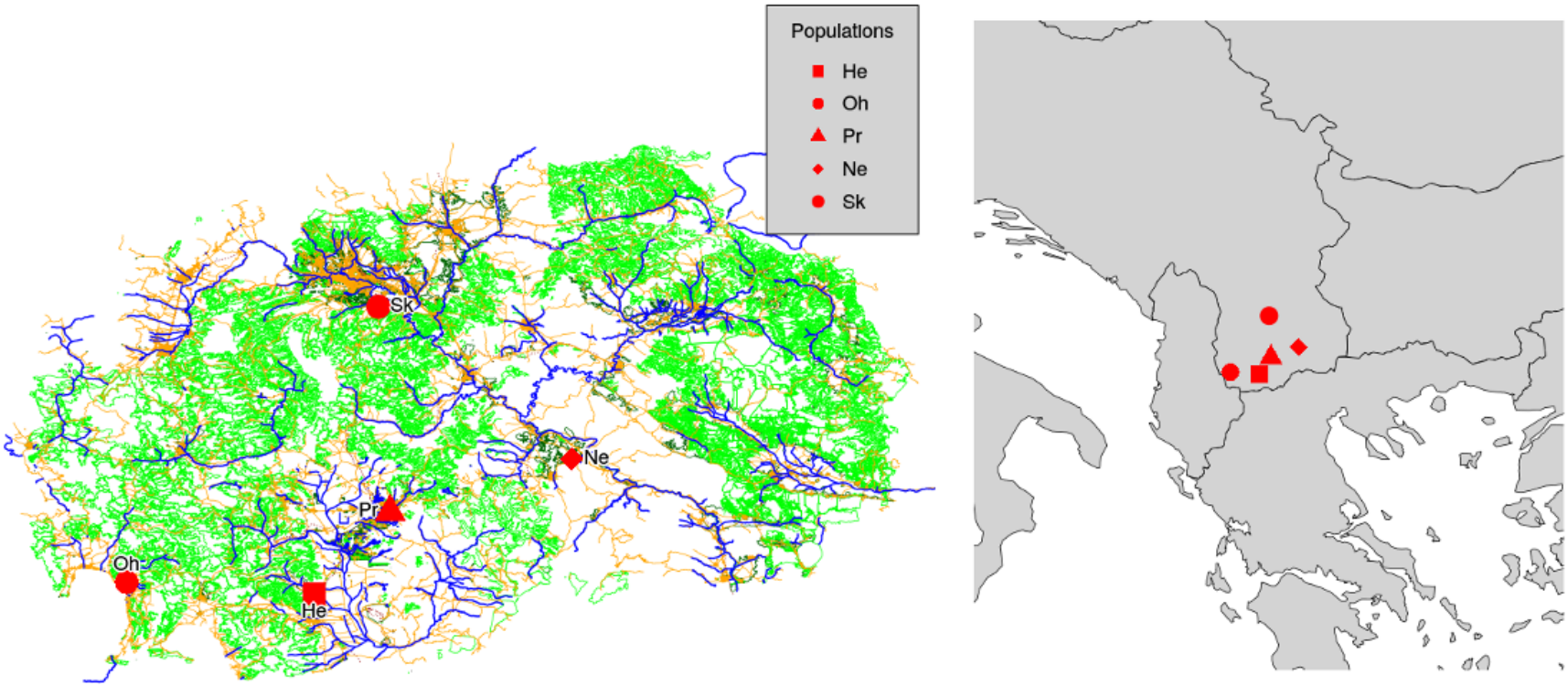
```
library(maptools)
# Load SHP file
# Data from http://download.geofabrik.de/europe/macedonia.html
# Directory has to contain also respective DBF and SHX files
# (same name, only different extension)
# There are several functions readShape* - select appropriate
# according to data stored in respective SHP file
macedonia_building <- readShapeLines(fn="macedonia_buildings.shp")
plot(macedonia_building)
macedonia_landuse <- readShapeLines(fn="macedonia_landuse.shp")
plot(macedonia_landuse)
macedonia_natural <- readShapeLines(fn="macedonia_natural.shp")
plot(macedonia_natural)
macedonia_railways <- readShapeLines(fn="macedonia_railways.shp")
plot(macedonia_railways)
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 sampling points
shadowtext(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$
  xy[["lat"]], labels=as.vector(popNames(hauss.genind)), 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")
```



# Structure

- Population genetic software for Bayesian clustering, http://pritchardlab.stanford.edu/structure.html
- Uses Bayesian algorithm to find optimal distribution of individuals into the most natural number of groups (K)
- One Structure run tests one selected K
  - User must run it repeatedly for several Ks to find the best division
  - User must run it repeatedly for each K to see if the result is stable (because of stochasticity of the computational algorithm)
  - Finally there use to be hundreds of runs...
  - Structure has Java GUI to set up repeated runs – another possibility is to use R (or possibly any other scripting language like BASH, Perl or Python)
- Full procedure (parallel running of Structure using R and post-processing results with R and BASH) for Linux/UNIX computers is described at https://trapa.cz/en/structure-r-linux

# Structure work flow

1. Run multiple runs of Structure
   - User must test several numbers of genetic clusters (K) and test each several times
   - ParallelStructure (see later) can do this
2. Decide which K is the best – explore outputs
   - Plain command line Structure does not help with this
   - There is need for external application reading Structure output, summing them and helping decide which K is the best
   - Structure Harvester or Structure-sum R script (see later) can do this
3. Post process Structure outputs to prepare them for plotting
   - Sort and align Structure outputs
   - Probably most commonly done in CLUMPP
4. Plot the final graphs
   - Can be done in nearly any enough advanced graphical tool (including some R functions), probably the most commonly used is distruct

# Running Structure in parallel with R

- Let's use modern multi-core CPUs and plenty of RAM in current computers – parallelisation saves time
- ParallelStructure R 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
- Authors recommend to run it without GUI and not on Windows...
- For this chapter start new R project in new working directory

```
# Prepare special new empty directory and set working directory
setwd("~/dokumenty/fakulta/vyuka/r_mol_data/examples/structure/")
install.packages("ParallelStructure",
  repos="https://r-forge.r-project.org") # Install the package
library(ParallelStructure) # Load the library
# It takes more or less same parameter as normal Structure
?parallel_structure # See Structure manual and function's documentation
```

# 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 (in practice use much higher number than for the example – also for length of burnin chain)
  - Columns are separated by spaces (or TABs)
- Data input file (see Structure manual)
  - Make it as simple as possible – remove all unneeded columns
  - For population names use subsequent numbers from 1 to number of populations
  - For individual names use only alphanumerical characters

# 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=3, 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 toy input file https://soubory.trapa.cz/rcourse/hauss_stru.in and joblist https://soubory.trapa.cz/rcourse/joblist.txt

# ParallelStructure and Windows

- Authors do not recommend to run ParallelStructure on Windows...
- `parallel_structure()` uses for parallelisation library which is not available on Windows, instead try

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

- It may help to set `n_cpu=1`, but it is only for testing then, not for real work...
- If this fails, look for some UNIX machine (Linux, Mac OS X, BSD, ...)...

# Post process Structure results – select the best K

- Using Structure-sum-2011 R script by Dorothee Ehrich

```
# Load the script
source("https://soubory.trapa.cz/rcourse/structure-sum-2011.r")
# Create new directory with result files results_job_*_f and set
# working directory accordingly
setwd("/home/vojta/dokumenty/fakulta/vyuka/r_mol_data/examples/
  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), see manual
- Prepare **list_k.txt** containing on each line K and name of output file
- Get list K (example of the joblist below) from https://soubory.trapa.cz/rcourse/list_k.txt

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

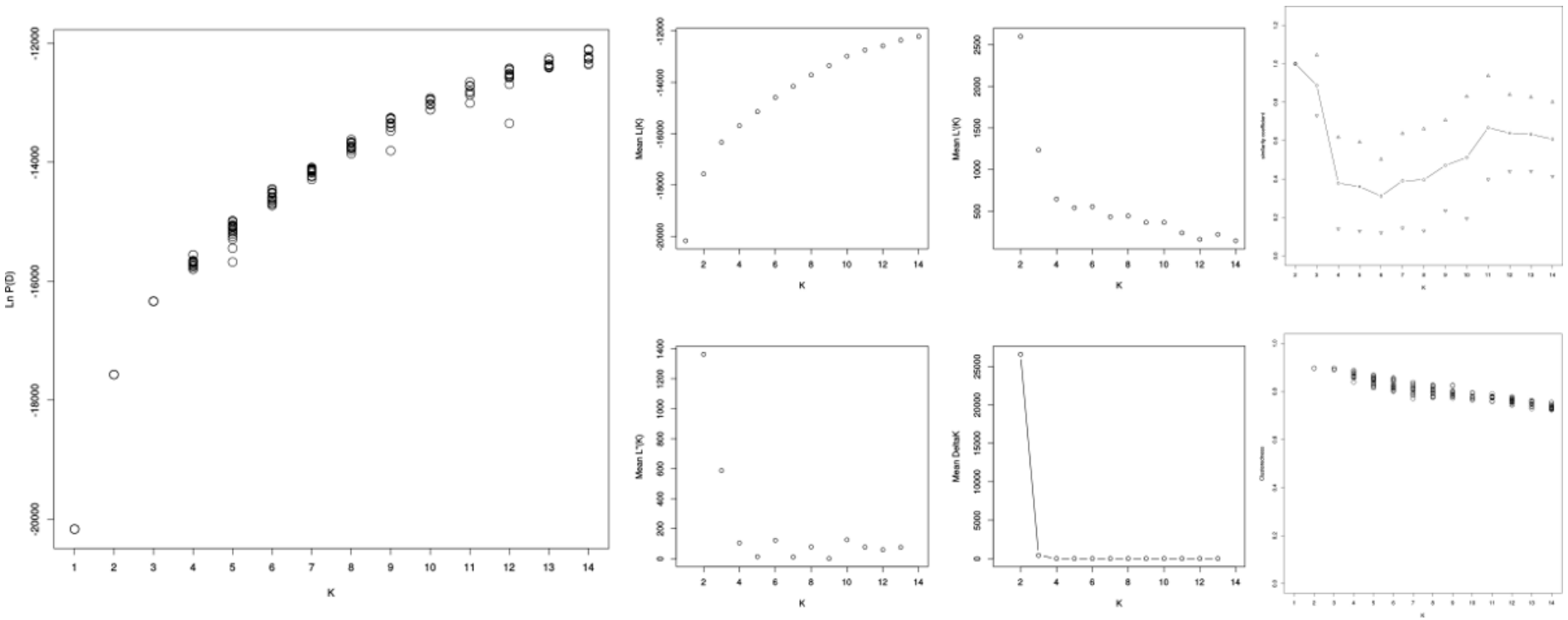
# Run Structure-sum

```
# 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

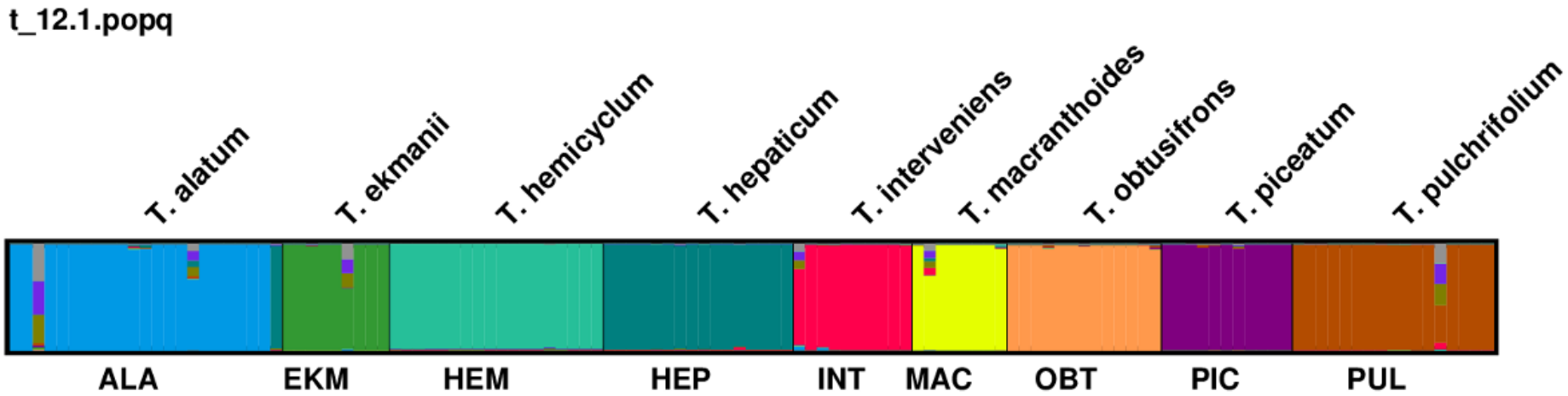
Results from different data set, not from our toy



# Example of final Structure plot

## Drawn by distruct after alignment of Structure outputs by CLUMPP

- Each bar is one individual
- Each color is one genetic cluster (here is shown clustering for K=12)
- Individuals with columns composing of more colors are genetically mixture of more clusters
- Details: https://trapa.cz/en/structure-r-linux



# Multiple sequence alignment

- Good alignment is basic condition for any analysis of DNA sequences
- R doesn't have any possibility for visual editing (use rather software like Geneious, CLC Sequence Viewer or BioEdit)
- R can automatically (in batch) run multiple sequence alignments of multiple genes (there are several possibilities)
  - Simple scripts for this task can be written in any scripting language like BASH, Perl or Python – only matters what user likes, knows and wish to do with the results...
- R packages use common alignment software: MAFFT, MUSCLE, Clustal, ...
  - User must install this software manually – R is just using external applications (in the examples shown)

# Multiple sequence alignment with MAFFT

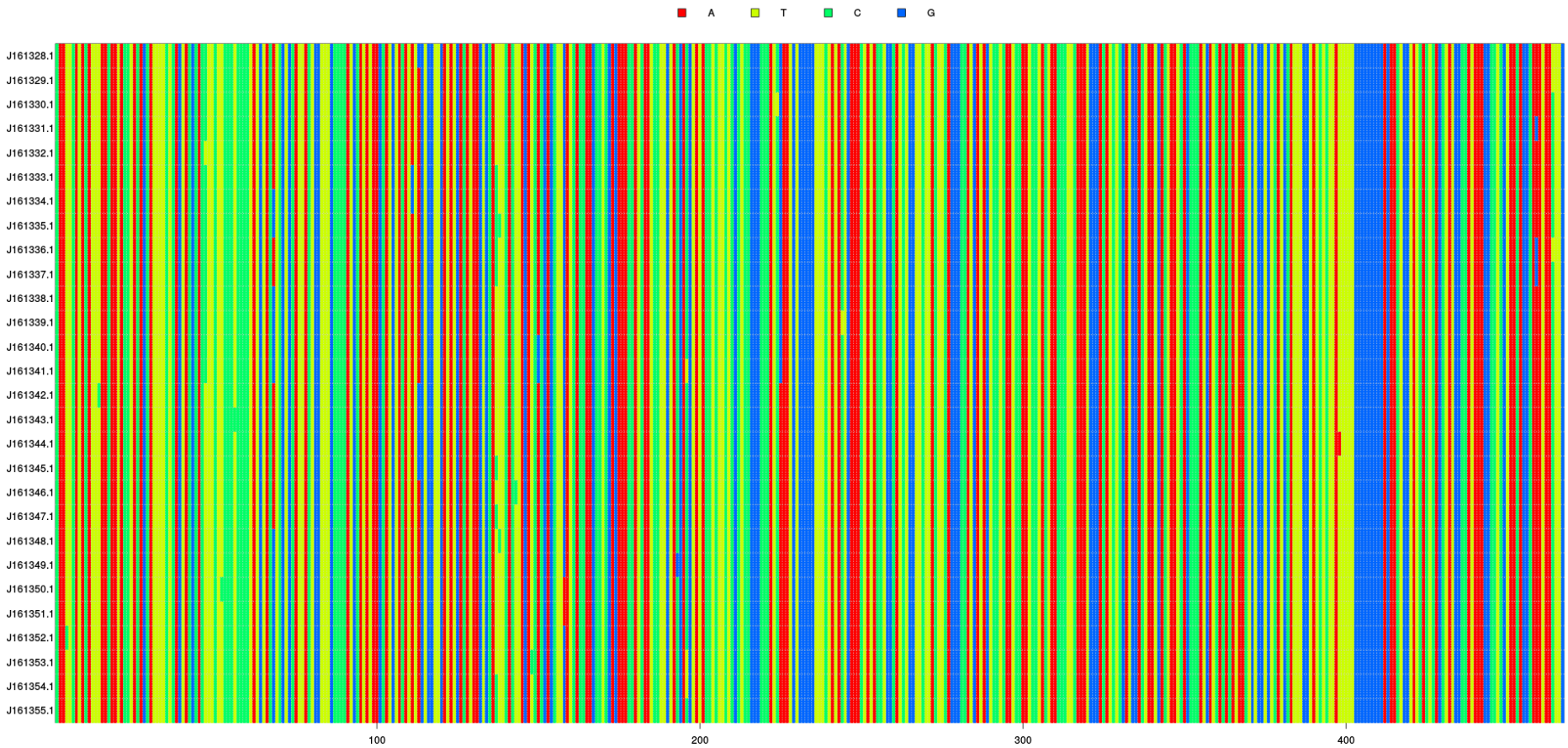
- MUSCLE is available in packages `muscle` and `ape` – first one reads "`*StringSet`" class R objects and writes "`*MultipleAlignment`" R objects; the latter reads and writes object of class "`DNAbin`"
- `ape` also contains functions to use Clustal and T-Coffee – both read and write `DNAbin`
- MAFFT 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 path to MAFFT binary - set it according to your installation
5 # read ?mafft and mafft's documentation
6 meles.mafft <- mafft(x=meles.dna, method="localpair", maxiterate=100,
7   path="/usr/bin/mafft") # Change "path" to fit your path to mafft!
```

# Clustal, MUSCLE and T-Coffee from ape

```
meles.mafft
class(meles.mafft)
# read ?clustal and documentation of Clustal, Muscle and T-Coffee
# when using them to set correct parameters
meles.clustal <- ape::clustal(x=meles.dna, pw.gapopen=10, pw.gapext=0.1,
  gapopen=10, gapext=0.2, exec="/usr/bin/clustalw2", quiet=FALSE,
  original.ordering=TRUE) # Change "exec" to fit your path to clustal!
meles.muscle <- muscle(x=meles.dna, exec="muscle", quiet=FALSE,
  original.ordering=TRUE) # Change "exec" to fit your path to muscle!
meles.muscle
class(meles.muscle)
# Plot the alignment - you can select which bases to plot
# and/or modify colors
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) # See ?del.gaps for details!
```

# Multiple sequence alignment with MUSCLE



# Cleaning the alignment

```
# Shortcut for plotting alignment
image.DNAbin(x=meles.mafft)
# Display aligned sequences with gaps
image.DNAbin(x=usflu.dna)
# Delete all columns containing any gap
library(ips)
usflu.dna.ng <- deleteGaps(x=usflu.dna, nmax=0)
# See of settings of "nmax" value - threshold for gap deletion
?deleteGaps # "nmax=0" deletes all columns with any gap
# Do not confuse with function delete.gaps() from phyloch package
# Display the result
image.DNAbin(x=usflu.dna.ng)
# Delete positions in alignment containing only missing data/N
?deleteEmptyCells # See help page for details
```

# Read and write tree and drop tips

```
# Read trees in NEWICK format - single or multiple tree(s)
oxalis.trees <- read.tree
  ("https://soubory.trapa.cz/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)
nodelabels() # See node labels (numbers) - needed for some tasks
# 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=0.4)
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)
nodelabels()
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)
# resolve.root=TRUE ensures root will be bifurcating (needed here)
# (without this parameter it soemtimes doesn't work)
hauss.nj.rooted <- root(phy=hauss.nj, resolve.root=TRUE, 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() # unroot the tree
# 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 without 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 (afterward use `class(x) <- "multiPhylo"` to ensure other functions will see it as `multiPhylo` object)

# Topographical distances among matrices I – implementations

- Robinsons-Foulds distance in `phytools::multiRF`
  - The index adds 1 for each difference between pair of trees
  - Well defined only for fully bifurcating trees – if not fulfilled, some results might be misleading
  - Allow comparison of trees created by different methods
- Methods implemented in `ape::dist.topo` allow comparison of trees with polytomies (`method="PH85"`) or use of squared lengths of internal branches (`method="score"`)
- Final matrices are commonly not Euclidean – may be problematic for usage in methods like PCA
  - Test it with `ade4::is.euclid`, can be scaled (forced to became Euclidean) by functions like `quasieuclid` or `cailliez` in `ade4` – carefully, it can damage meaning of the data

# Topographical distances among trees II

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 } # dist.topo can compare only two trees in one step... :-(
15 # Basic information about the distance matrix
16 dim(oxalis.trees.d)
17 head.matrix(oxalis.trees.d)
```

# Topographical distances among trees III

## Post process 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

```
colnames(oxalis.trees.d) <- names(oxalis.trees) # Add names of
rownames(oxalis.trees.d) <- names(oxalis.trees) # columns and rows
# Make matrix symmetric
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))
```

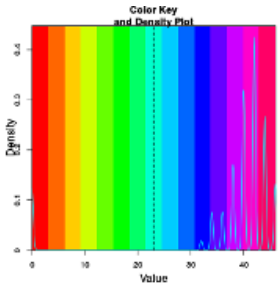
# Topographical distances among trees IV

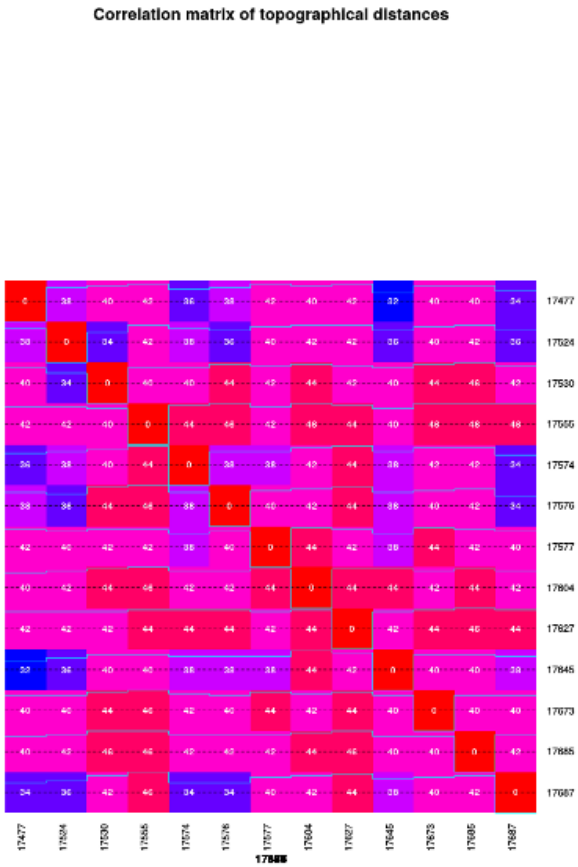
## Calculate Robinsons-Foulds distance matrix among trees and plot it

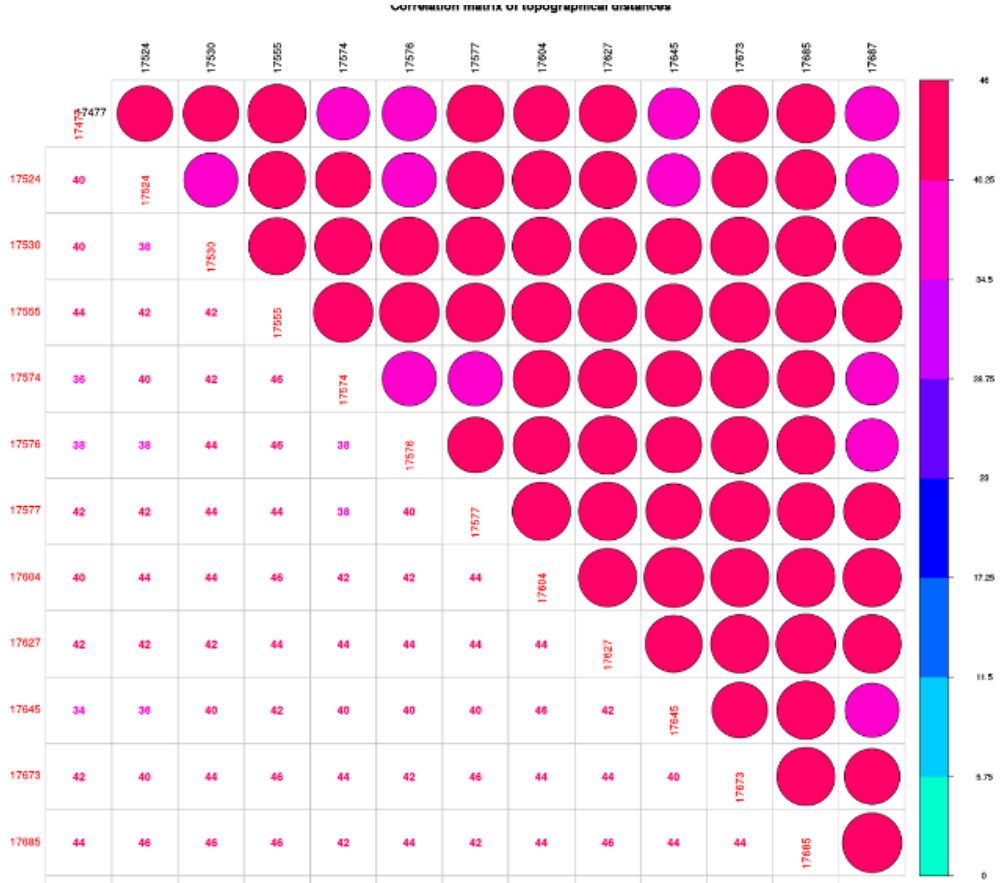
- `phytools::multiRF` can handle `multiPhylo` objects and directly create matrices (no need to create loops)

```
# 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 V – the matrices



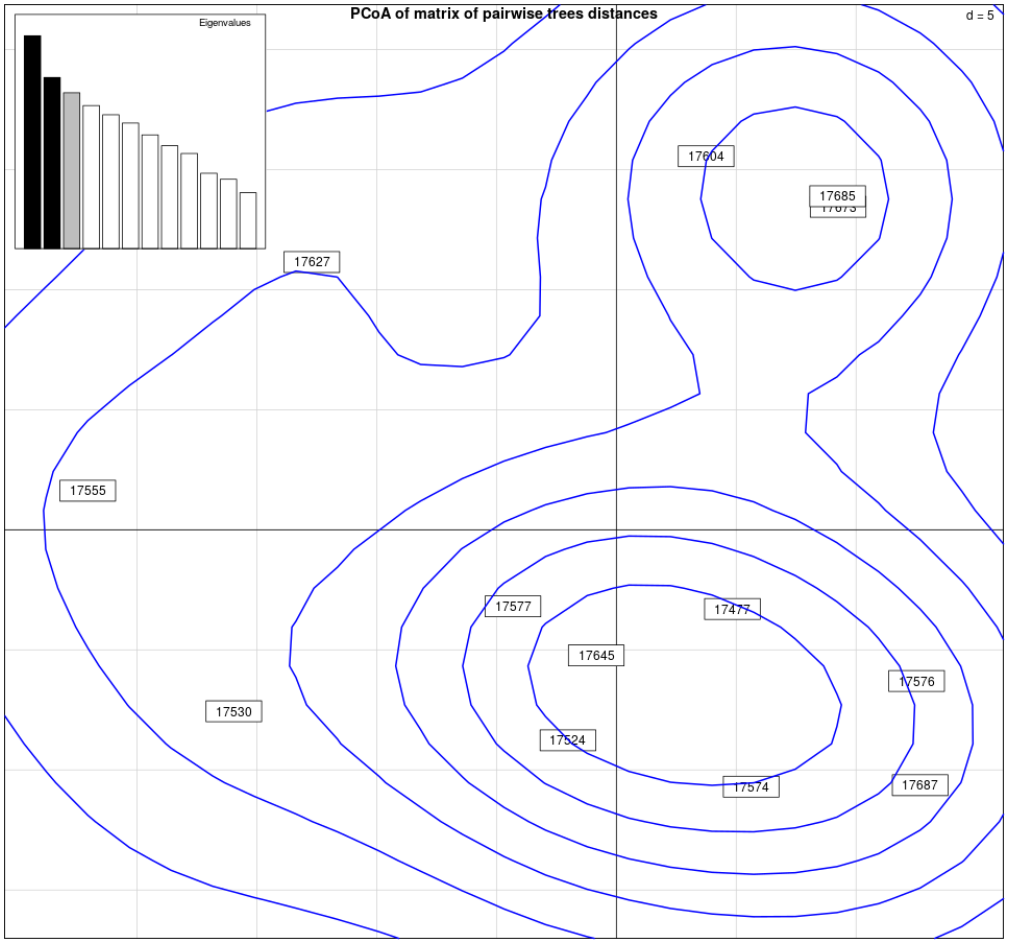



# PCoA from distance matrices of topographical differences among trees – the code

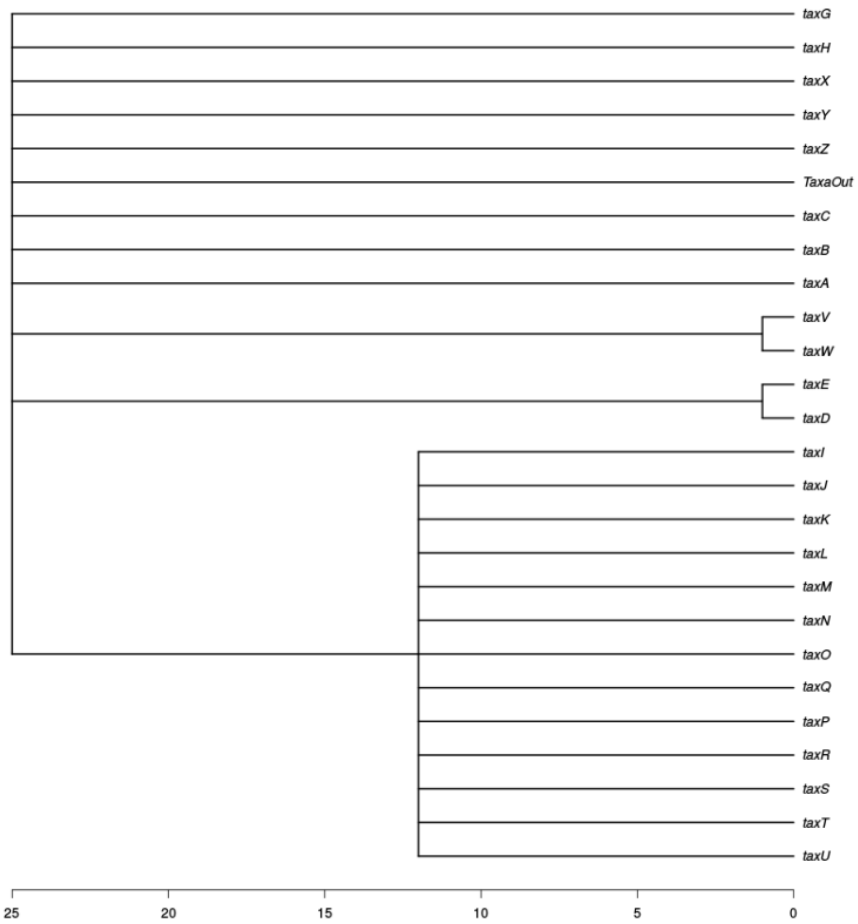
PC plots help to identify outliers – trees with noticeably different topology

```
# Test if the distance matrix is Euclidean or not
is.euclid(distmat=as.dist(oxalis.trees.d), plot=TRUE)
[1] TRUE # OK. If it wouldn't be, we could use e.g. quasieuclid()
# Calculate the PCoA
oxalis.trees.pcoa <- dudi.pco(d=as.dist(oxalis.trees.d), scannf=TRUE,
  full=TRUE)
# Plot PCoA
s.label(dfxy=oxalis.trees.pcoa$li)
# Add kernel densities
s.kde2d(dfxy=oxalis.trees.pcoa$li, cpoint=0, add.plot=TRUE)
# Add histogram of eigenvalues
add.scatter.eig(oxalis.trees.pcoa[["eig"]], 3,1,2, posi="topleft")
# Add title to the plot
title("\nPCoA of matrix of pairwise trees distances")
# Alternative function to plot PCA plot
scatter(x=oxalis.trees.pcoa, posieig="topleft")
```

# PCoA from distance matrices of topographical differences among trees – the plot

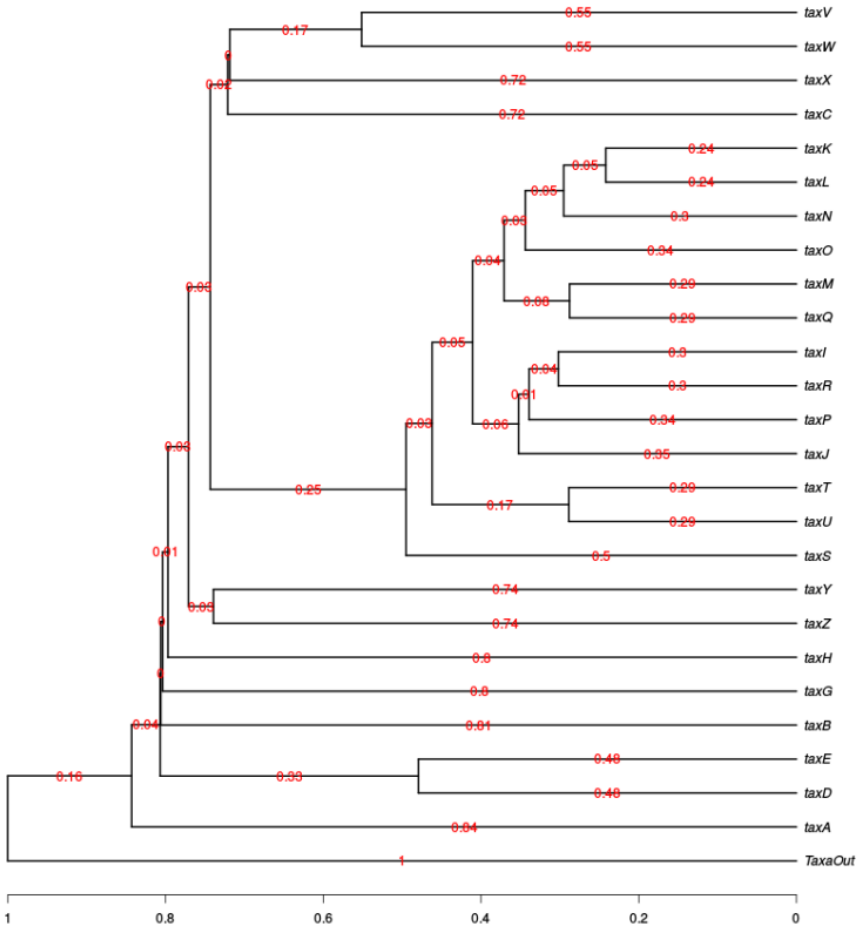


# Consensus tree



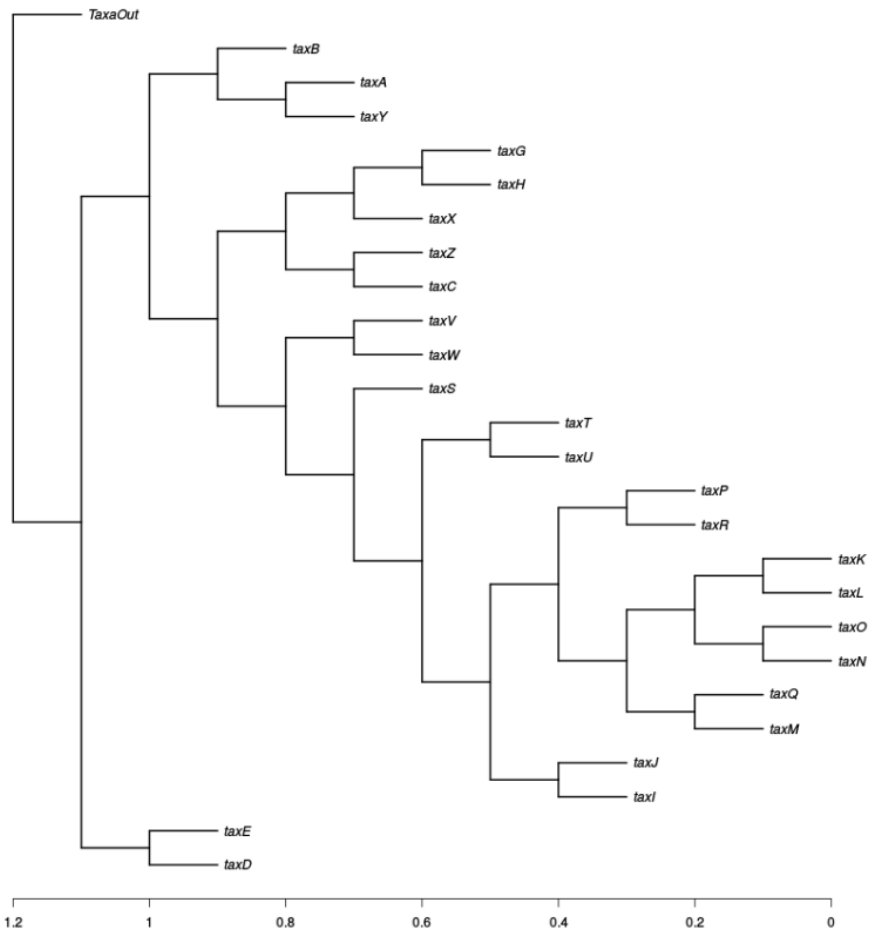

```
1  # Root all trees
2  oxalis.trees.rooted <- lapply
3    (X=oxalis.trees, FUN=root,
4    "TaxaOut")
5  class(oxalis.trees.rooted) <-
6    "multiPhylo"
7  # Consensus tree (50 % rule)
8  oxalis.tree.con <- consensus
9    (oxalis.trees.rooted, p=0.5,
10   check.labels=TRUE)
11 print.phylo(oxalis.tree.con)
12 # Plot the tree
13 plot.phylo(oxalis.tree.con,
14   edge.width=2, label.offset=0.3)
15 axisPhylo(side=1)
16 # What a nice tree... :-P
```

# Species tree – all trees must be ultrametric




```
1  # Chronos scale trees
2  oxalis.trees.ultra <- lapply
3    (X=oxalis.trees.rooted,
4    FUN=chronos, model="correlated")
5  class(oxalis.trees.ultra) <-
6    "multiPhylo"
7  # Mean distances
8  oxalis.tree.sp.mean <- speciesTree
9    (oxalis.trees.ultra, mean)
10 # Plot the tree
11 plot.phylo(oxalis.tree.sp.mean,
12   edge.width=2, label.offset=0.01)
13 edgelabels(text=round(oxalis.
14   tree.sp.mean[["edge.length"]],
15   digits=2), frame="none",
16   col="red", bg="none")
17 axisPhylo(side=1)
```

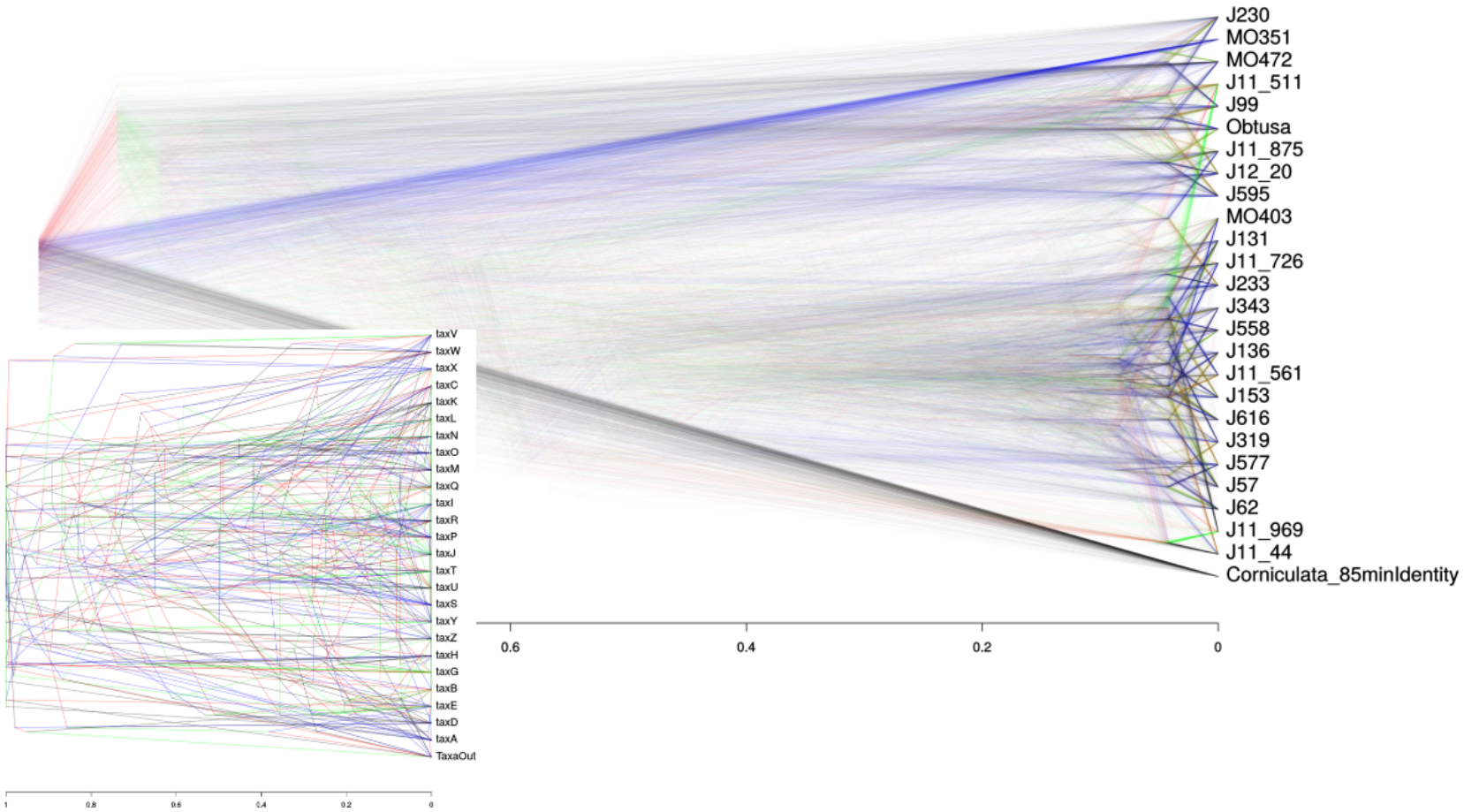
# Parsimony super tree




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

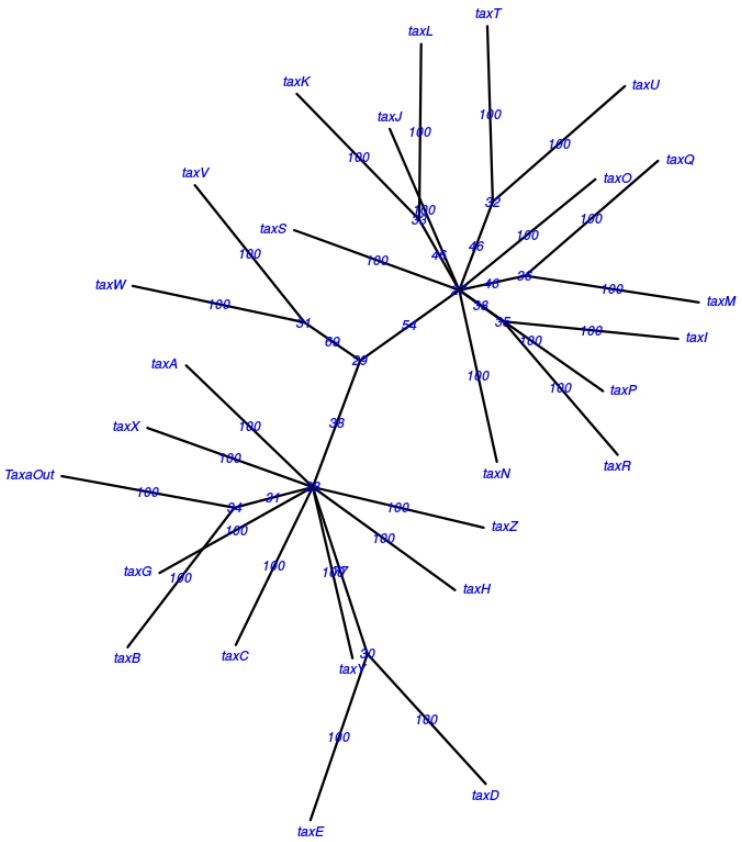
# Density tree

```
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") # 2D - left
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") # 3D
```

# Kronoviz – see all trees on same scale

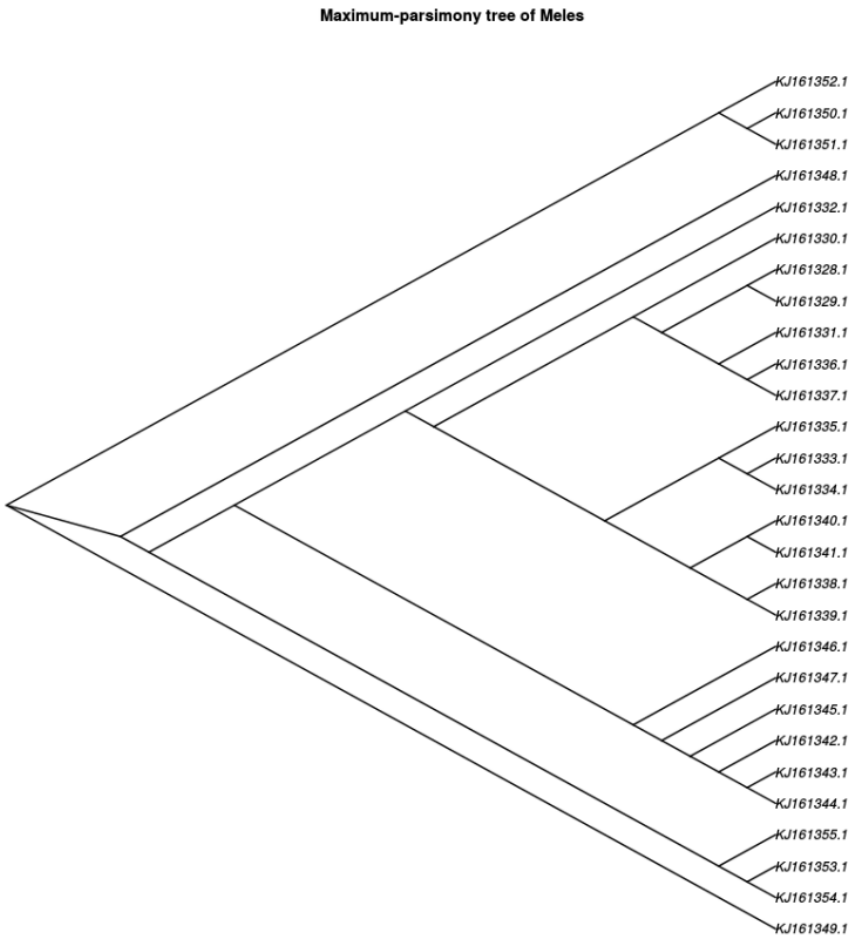

```
kronoviz(x=oxalis.trees.rooted,
  layout=length(oxalis.trees.
  rooted), horiz=TRUE)
# Close graphical device to
# cancel division of plotting
# device
dev.off()
```

- 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 – theory

- Maximum parsimony finds optimal topology of the phylogenetic tree by minimizing of the total number of character-state changes
- It minimizes homoplasy (convergent evolution, parallel evolution, evolutionary reversals)
- Very simple criterion, easy to score the tree, but not to find it – exhaustive search to explore all possible trees is realistic until ∼9 taxa, branch-and-bound swapping (guaranteeing finding the best tree) until ∼20 taxa, for more heuristic search is needed – it doesn't always guarantee to find the most probable tree
- To speed up calculations, initial tree (usually NJ – slide 108) is used to start the search
- With rising performance of computers, it use to replaced my maximum likelihood or Bayesian methods

# Maximum parsimony – code and result



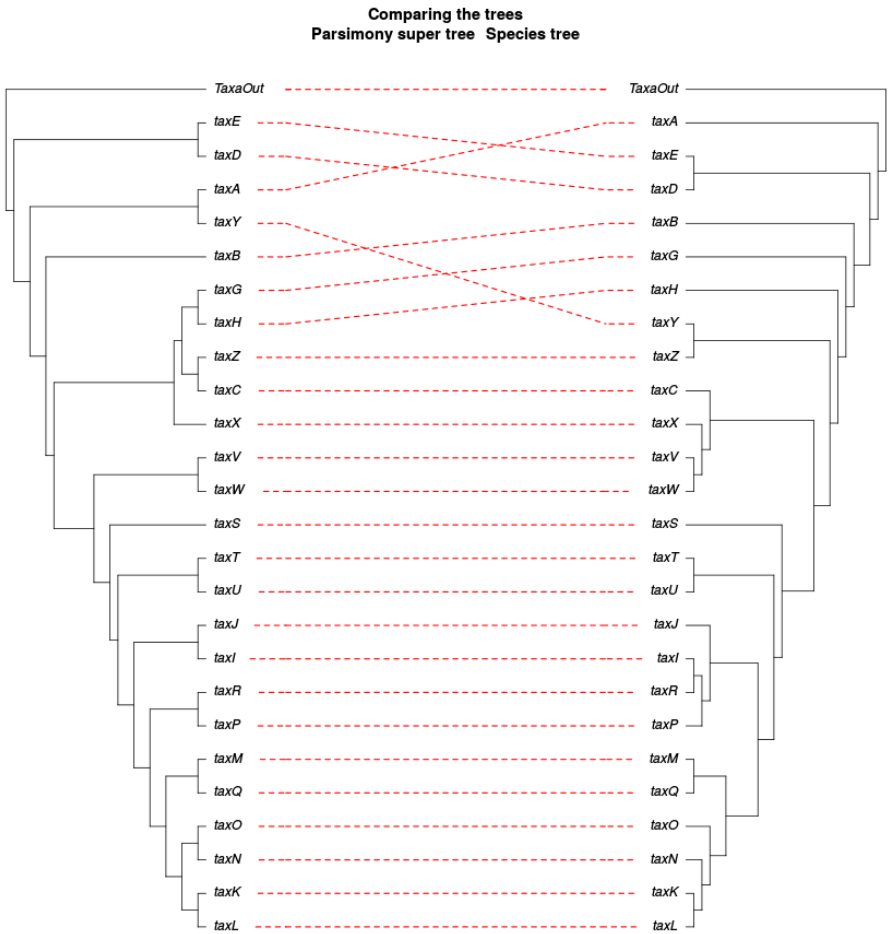

```
?parsimony # Parsimony details
```

```
# Conversion to phyDat
meles.phydat <-
  as.phyDat(meles.dna)
# Prepare starting tree
meles.tre.ini <- nj(dist.
  dna(x=meles.dna,model="raw"))
# Maximum parsimony score
parsimony(tree=meles.tre.ini,
  data=meles.phydat)
# Optimization
# 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)
```

# Compare two trees

```
# Compare topology of the species trees - basically outputs TRUE/FALSE
all.equal.phylo(oxalis.tree.sp, oxalis.tree.sp.mean,
  use.edge.length=FALSE)
?all.equal.phylo # Use to see comparison possibilities
# Plot two trees with connecting lines
# We need 2 column matrix with tip labels
tips.labels <- matrix(data=c(sort(oxalis.tree.sp[["tip.label"]]),
  sort(oxalis.tree.sp.mean[["tip.label"]])),
  nrow=length(oxalis.tree.sp[["tip.label"]]), ncol=2)
# Draw a tree - play with graphical parameters and use rotate=TRUE
# to be able to adjust fit manually
cophyloplot(x=ladderize(oxalis.tree.sp),
  y=ladderize(oxalis.tree.sp.mean),  assoc=tips.labels,
  use.edge.length=FALSE, space=60, length.line=1, gap=2,
  type="phylogram", rotate=TRUE, col="red", lwd=1.5, lty=2)
title("Comparing the trees\nParsimony super tree\tSpecies tree")
legend("topleft", legend="Red lines\nconnect tips", text.col="red",
  cex=0.75, bty="n", x.intersp=-2, y.intersp=-2)
```

# Cophyloplot comparing two trees

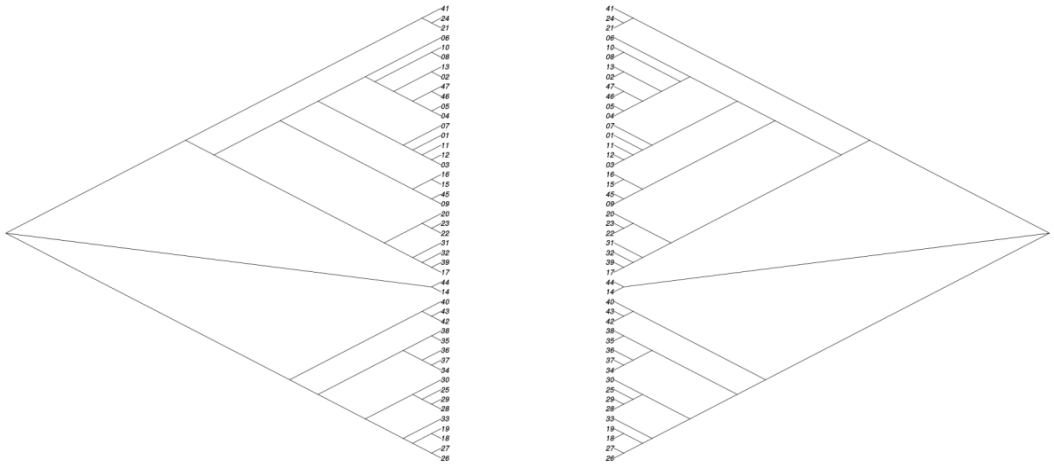



- `ladderize()` pre-sorts tips in the tree – it helps to `cophyloplot()` to create better plot
- Automatic plot is usually not perfect – there use to be unneeded crossing lines – `rotate=TRUE` is recommended to can fix this manually by clicking to the nodes
- `cophyloplot()` has similar parameters like `plot.phylo()` – play with it and adjust in graphical editor

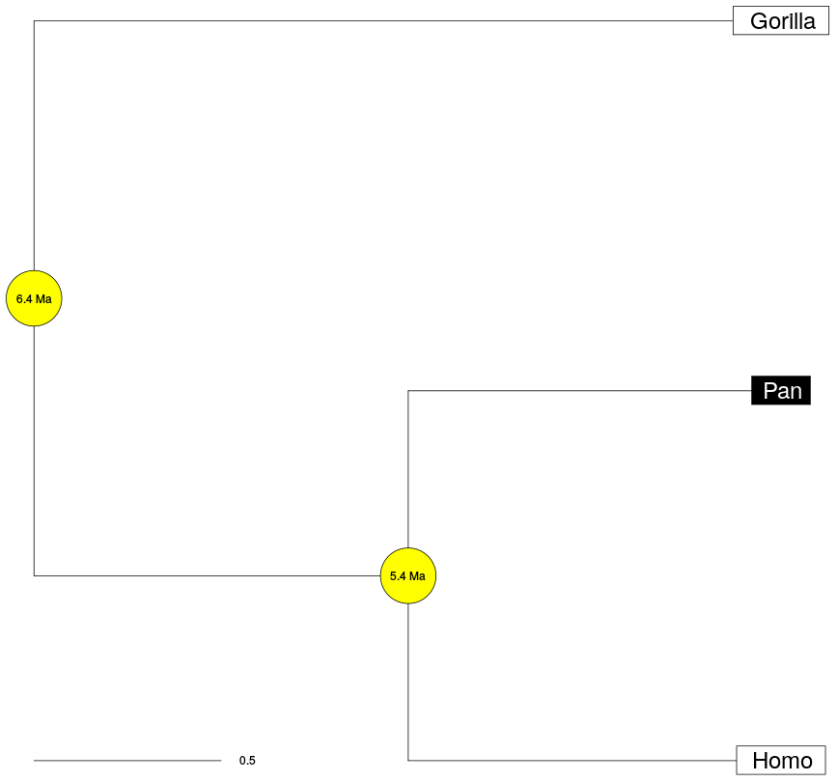
# Change orientation of plots

- `plot.phylo()` has plenty of possibilities to influence – check `?plot.phylo`, `?par`, `?points`, ...

```
?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




```
# Load tree in text format
trape <- read.tree(text=
  "((Homo, Pan), Gorilla);")
# Plot the tree
plot.phylo(x=trape,
  show.tip.label=FALSE)
# Add colored tip labels
tiplabels(trape[["tip.label"]],
  bg=c("white", "black",
  "white"), col=c("black",
  "white", "black"), cex=2)
# Add colored node labels
nodelabels(text=c("6.4 Ma",
  "5.4 Ma"), frame="circle",
  bg="yellow")
add.scale.bar() # Add scale bar
# Note vectors for tip/nodelabels
```

# Phylogenetic independent contrast

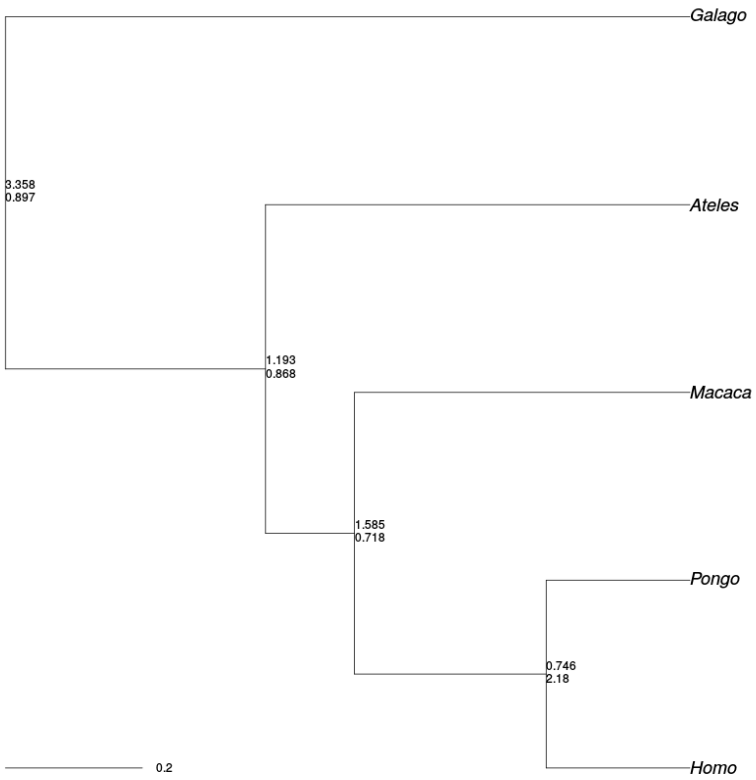
- When analyzing comparative data takes phylogeny into account
- If we assume that a continuous trait evolves randomly in any direction (i.e. the Brownian motion model), then the "contrast" between two species is expected to have a normal distribution with mean zero, and variance proportional to the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

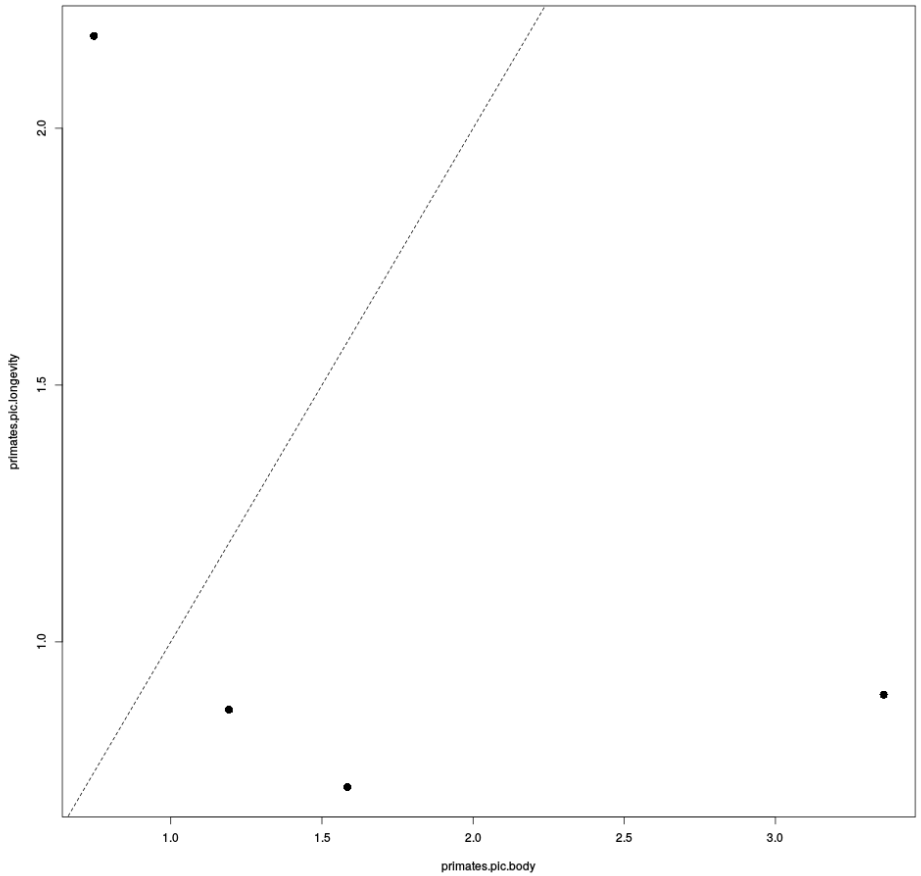
```
# Prepare the data # Body mass of primates
primates.body <- c(4.09434, 3.61092, 2.37024, 2.02815, 1.46968)
# Longevity of primates
primates.longevity <- c(4.74493, 3.3322, 3.3673, 2.89037, 2.30259)
# Add names to the values
names(primates.body) <- names(primates.longevity) <- c("Homo", "Pongo",
  "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)
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
```

# Intraspecific variation

- `pic.ortho()` requires list of measurements (vectors) for all taxa – their lengths can differ
- If we have sets of measurements in separated vectors (each vector has measurements for all taxa), we must for each `list` item use `cbind` to join columns and select appropriate line (from 1 to number of taxa)
- In example below, `jitter()` adds random noise
- Other usage is same in previous case...

```
primates.pic.ortho <- pic.ortho(x=list(cbind(primates.body,
  jitter(primates.body), jitter(primates.body))[1,],cbind(primates.
  body, jitter(primates.body), jitter(primates.body))[2,],
  cbind(primates.body, jitter(primates.body), jitter(primates.body))
  [3,], cbind(primates.body, jitter(primates.body), jitter(primates.
  body))[4,], cbind(primates.body, jitter(primates.body),
  jitter(primates.body))[5,]), phy=primates.tree, var.contrasts=FALSE,
  intra=FALSE)
```

# Explanation of the cbind trick

```
cbind(primates.body, jitter(primates.body), jitter(primates.body))
Homo          4.09434 4.113074 4.038092
Pongo         3.61092 3.671217 3.558953
Macaca        2.37024 2.426757 2.430310
Ateles        2.02815 1.986006 2.091402
Galago        1.46968 1.494281 1.496831
cbind(primates.body, jitter(primates.body), jitter(primates.body))[1,]
  4.094340      4.072501      4.035324
cbind(primates.body, jitter(primates.body), jitter(primates.body))[2,]
  3.610920      3.572728      3.664654
class(cbind(primates.body, jitter(primates.body),
  jitter(primates.body)))
[1] "matrix"
class(cbind(primates.body, jitter(primates.body),
  jitter(primates.body))[1,])
[1] "numeric"
# jitter() adds random noise every time, so that the values differ
```

# 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
```

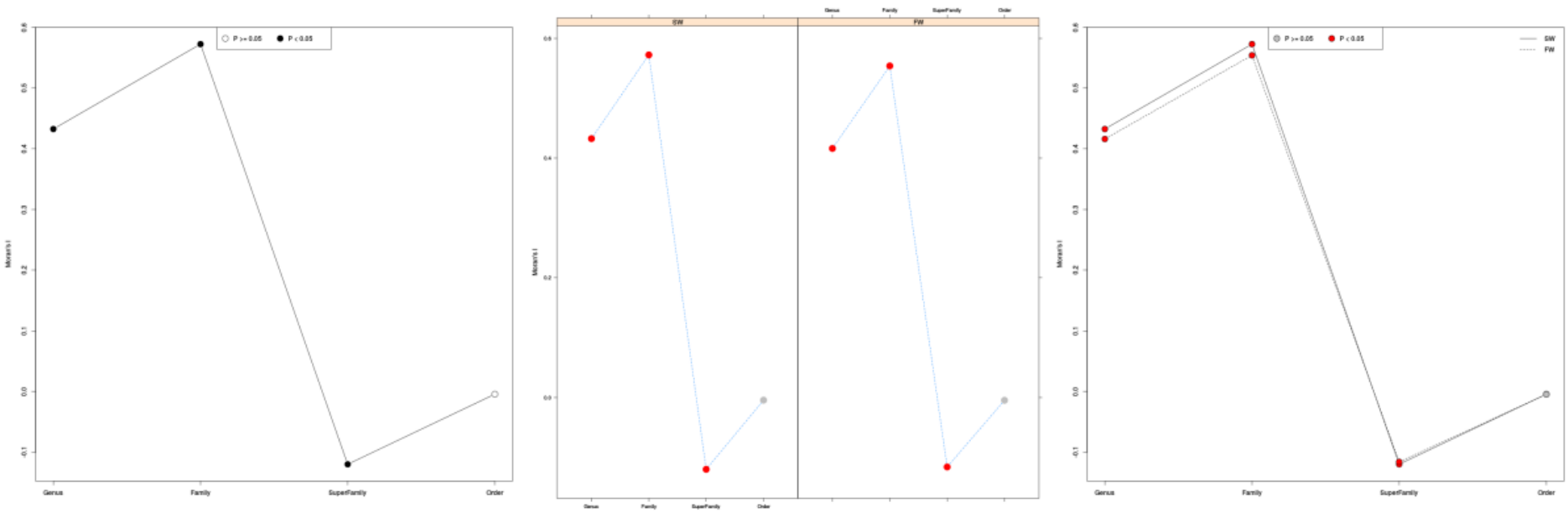
# Testing of Moran's *I*

```
# Calculate Moran's I
# Slightly 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 randomization 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", nrepet=1000,
  alter="greater")
```

# Correlogram to visualize results of phylogenetic autocorrelation 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)
```

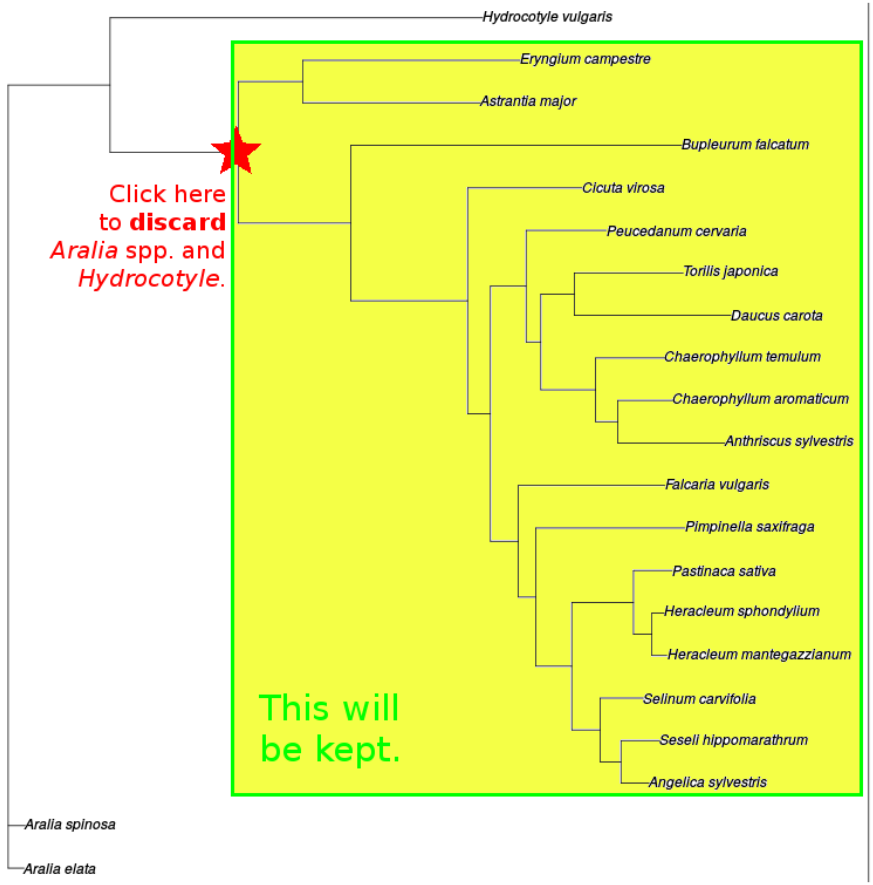
# Correlograms of SW and SW+FW (in one or two graphs) depending on taxonomical level with marked significance

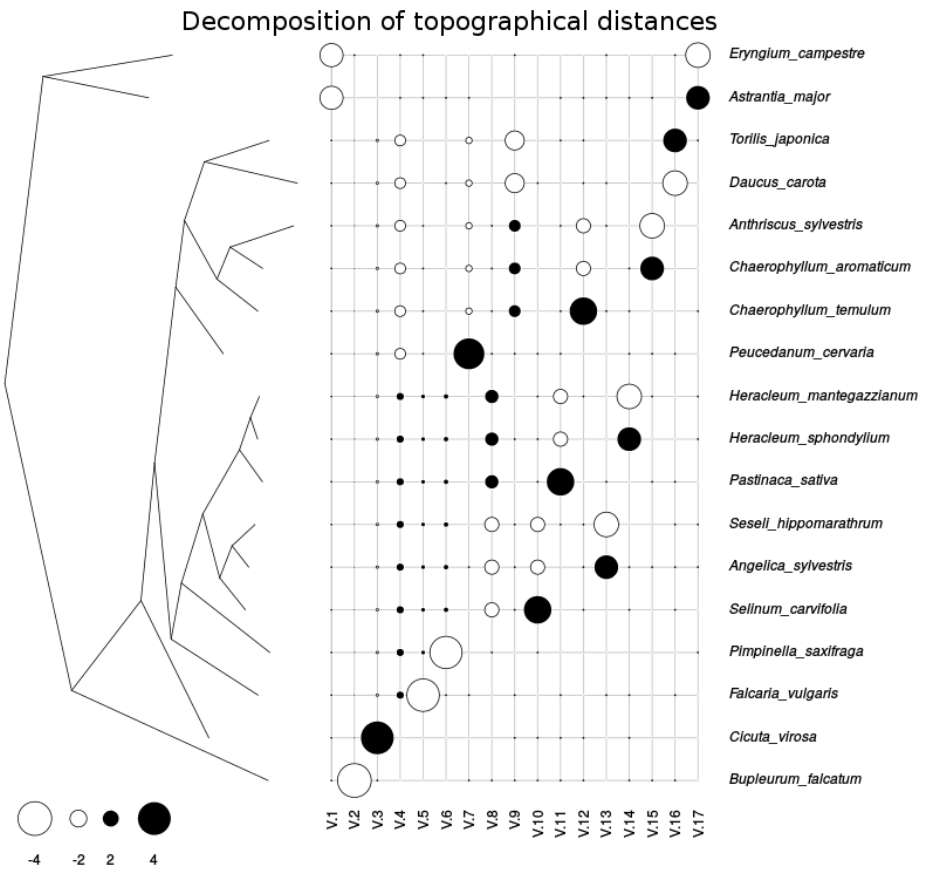


# Prepare toy data set (tree)

```
# Load MrBayes tree in NEXUS format
apiaceae.tree <- read.nexus
  ("https://soubory.trapa.cz/rcourse/apiaceae_mrbayes.nexus")
print.phylo(apiaceae.tree) # See it
plot.phylo(apiaceae.tree) # See it
# Root the tree
apiaceae.tree <- root(apiaceae.tree, "Aralia_elata")
# Remove "_" from taxa names
# plot.phylo() by default omits "_" from tip names
apiaceae.tree$tip.label <- gsub(pattern="_", replacement=" ",
  x=apiaceae.tree$tip.label)
# Drop outgroup (Aralia and Hydrocotyle)
# Click on last common ancestor of ingroup desired to be kept
plot.phylo(apiaceae.tree)
apiaceae.tree <- extract.clade(apiaceae.tree, interactive=TRUE)
plot.phylo(apiaceae.tree)
library(adephylo)
library(phylobase)
```

# Modified tree




```
# Decomposition of topographical distances (right plot)
table.phylo4d(x=phylo4d(x=apiaceae.tree, tip.data=treePart
  (x=apiaceae.tree, result="orthobasis")), treetype="cladogram")
```

# Prepare toy data set (the variable)

```
# Generate some random variable
library(geiger)
apiaceae.eco <- sim.char(phy=apiaceae.tree, par=0.1, nsim=1,
  model="BM")[,,1]
?sim.char # See it for another possibilities to simulate data
# Names for the values
names(apiaceae.eco) <- apiaceae.tree[["tip.label"]]
apiaceae.eco # See it
```

- `sim.char()` creates an array (we keep only numeric vector of $1^{st}$ simulation – `[,,1]`) of simulated characters, with `model="BM"` under Brownian motion
- Many methods compare **names** of character values with `tip.label` slot of the tree to pair character values with correct taxa
  - Otherwise values must be ordered in same way as in `tip.label` slot
  - Always check manual for respective function and all data!

# Orthonormal decomposition - phylogenetic eigenvector regression

```
anova(lm(apiaceae.eco ~ as.matrix(orthobasis.phylo(x=apiaceae.tree,
  method="patristic")[,1:2])))
```

- Significant result – significant phylogenetic inertia (phylogenetic effect) – the tendency for traits to resist evolutionary change despite environmental perturbations
- `orthobasis.phylo()` return matrix, which is linear transformation of cophenetic distances – columns 1 and 2 can be used to calculate phylogenetic variance – it can be used to calculate linear regression

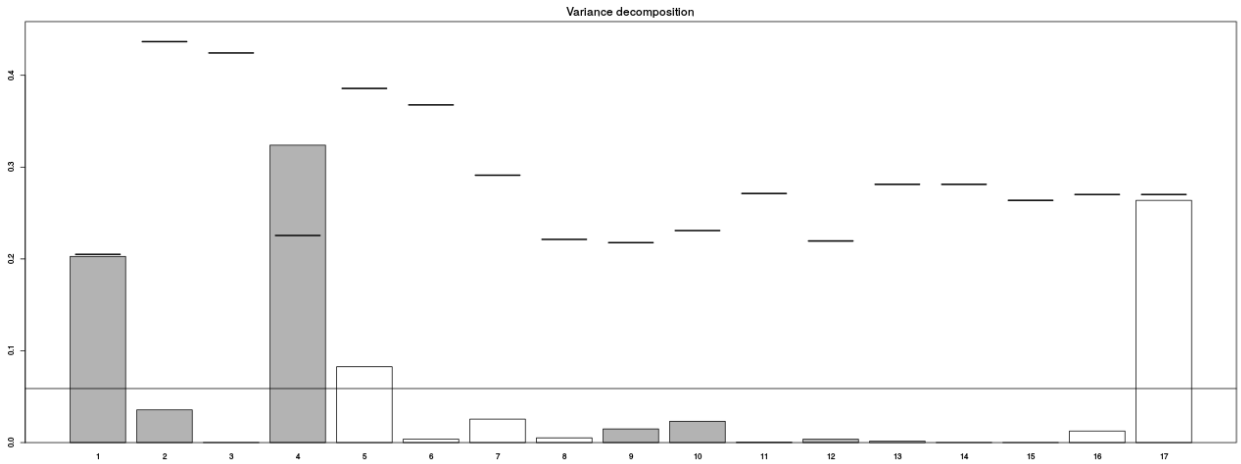
```
              Df    Sum Sq  Mean Sq F value Pr(>F)
as.matrix...   2 0.063689 0.031845  2.2275 0.1422
Residuals     15 0.214443 0.014296
```

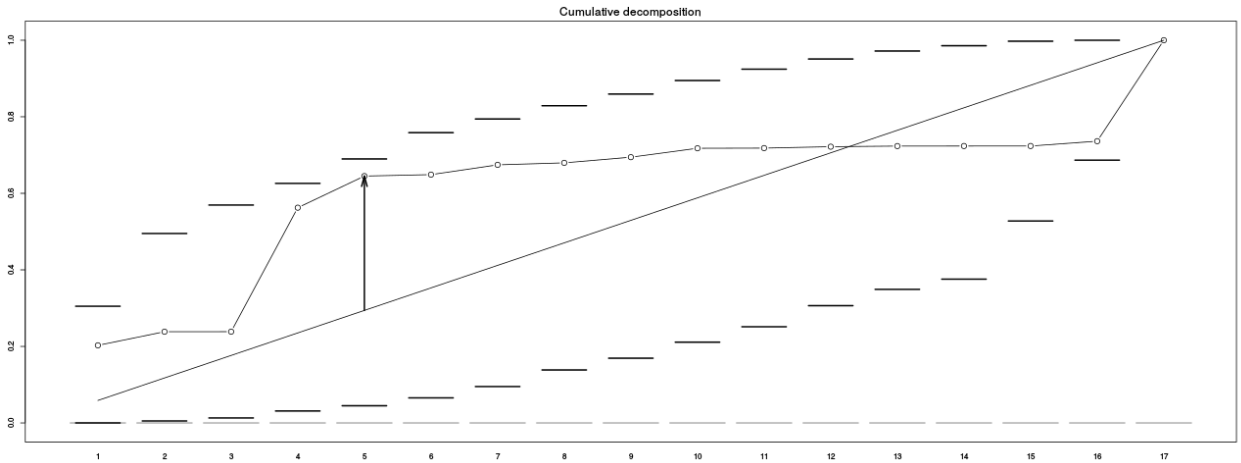
# Orthonormal decomposition of variance of a quantitative variable on an orthonormal basis

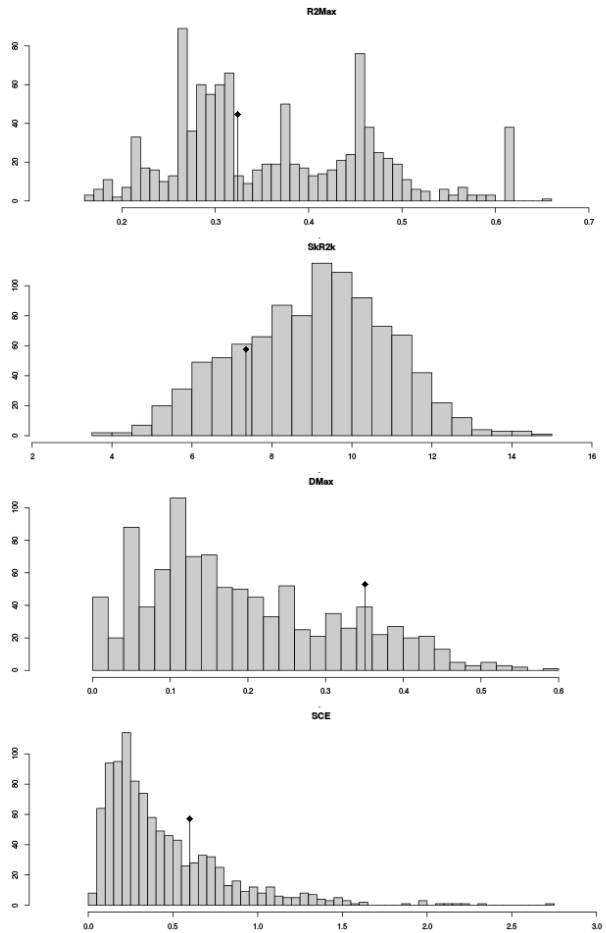
```
orthogram(x=apiaceae.eco, tre=apiaceae.tree, nrepet=1000,
  alter="two-sided")
?orthogram # See another calculation possibilities
```

- Analyses one quantitative trait
- Do not confuse with `ade4::orthogram` – similar, but require data in little bit different form, marked as deprecated and replaced by the `adephylo` version
- It returns results of 5 non-parametric tests associated to the variance decomposition
- Procedure decomposes data matrix to separate phylogeny and phenotype to see if there is significant signal

# Orthogram




- Observed value is within permutations – no significant phylogenetic signal...

# Phylogenetic Generalized Least Squares

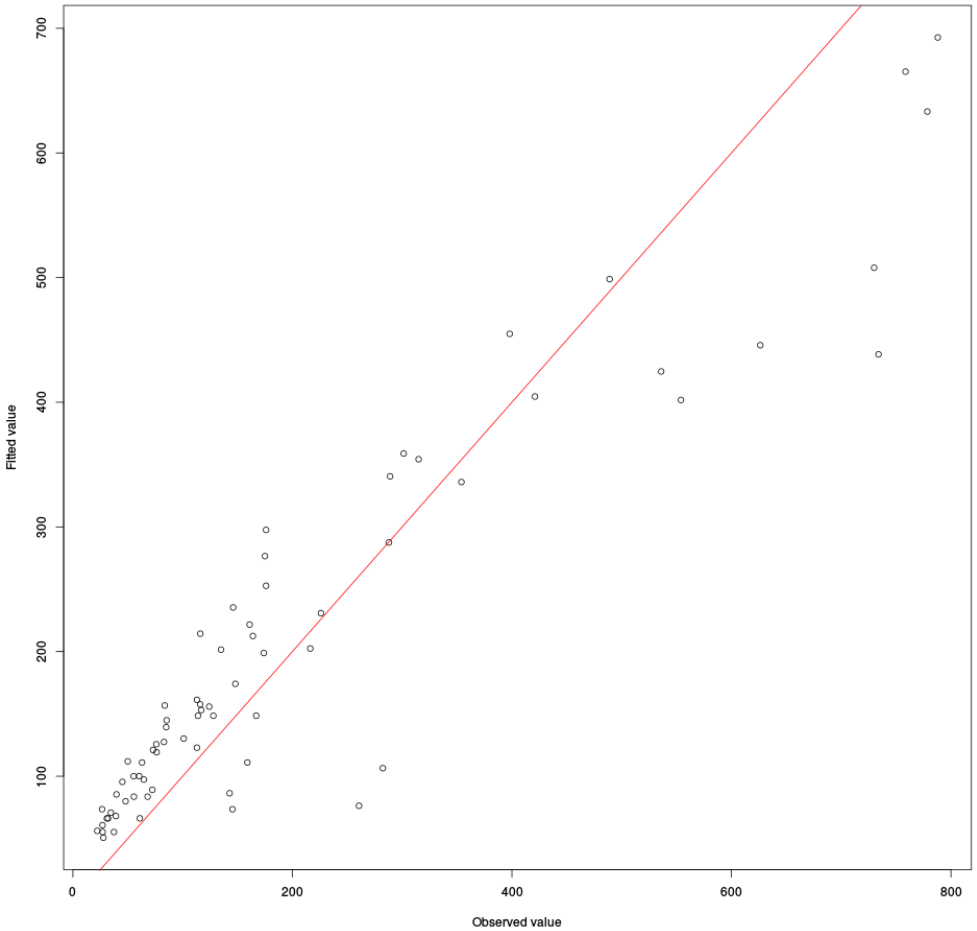
- Model-based testing if there is significant correlation between two traits (after removing the phylogenetic component)
- `nlme::gls` fits a linear model using generalized least squares
- Functions `corBlomberg`, `corBrownian`, `corMartins` and `corPagel` from `ape` package create correlation matrix of evolution of continuous character according to the given tree

```
1 library(nlme)
2 library(ape)
3 summary(gls(model=primates.longevity ~ primates.body,
4   data=as.data.frame(cbind(primates.longevity, primates.body)),
5   correlation=corBrownian(value=1, phy=primates.tree)))
```

# Implementation in caper package

```
library(caper) # Load needed library
data(shorebird) # Load training data, see ?shorebird.data
# Calculate the model
shorebird.pgls <- pgls(formula=shorebird.data[["F.Mass"]] ~
  shorebird.data[["Egg.Mass"]], data=comparative.data(phy=
  shorebird.tree, data=as.data.frame(cbind(shorebird.data[["F.Mass"]],
  shorebird.data[["Egg.Mass"]], shorebird.data[["Species"]])),
  names.col=V3, vcv=TRUE))
# See the result
summary(shorebird.pgls)
# See the plot of observer and fitted values
plot(shorebird.pgls)
abline(a=0, b=1, col="red")
# ANOVA view of the model
anova(shorebird.pgls)
# Akaike's information criterion (smaller = better)
AIC(shorebird.pgls)
```

# Results of PGLS




- `pgls()` uses maximum likelihood to test for phylogenetic signal
- The signal is clearly presented
- Usually, tuning the model (possible data transformations and or changing model parameters) is necessary to find the best model – AIC helps
- See caper manual for details

# Generalized Estimating Equations

- Extension of GLM for correlated data – usage is similar
- It is possible to use phylogeny or correlation matrix (typically based on phylogeny)

```
# Calculate the model
compar.gee(formula=primates.longevity ~ primates.body,
  phy=primates.tree)
# or with correlation matrix:
compar.gee(formula=primates.longevity ~ primates.body,
  corStruct=corMartins(value=1, phy=primates.tree, fixed=TRUE))
# for corStruct there are similar functions corBlomberg, corMartins,
# corPagel, corBrownian - see manuals for differences
```

# Not significant in this case...

```
Call: compar.gee(formula = primates.longevity ~
  primates.body, phy = primates.tree)
Number of observations:  5
Model:
                     Link: identity
Variance to Mean Relation: gaussian
QIC: 7.310142
Summary of Residuals:
       Min          1Q      Median          3Q         Max
-0.8031302 -0.0132754  0.0999588  0.1988258  0.2862064
Coefficients:
                Estimate       S.E.        t Pr(T > |t|)
(Intercept)    1.0670417 0.5838429 1.827618   0.2695894
primates.body 0.8497249 0.2157006 3.939372   0.1101432
```

# Phylogenetic signal

- Direct consequence of the evolution of trait depends on evolution – if trait variation is driven by environment, phylogenetic signal is 0

```
library(picante)
# Test for Bloomberg's K statistics
Kcalc(x=apiaceae.eco, phy=apiaceae.tree, checkdata=TRUE)
# Test with permutations
phylosignal(x=apiaceae.eco, phy=apiaceae.tree, reps=1000,
  checkdata=TRUE)
```

- If Blomberg's values of 1 correspond to a Brownian motion process, which implies some degree of phylogenetic signal or conservatism
- K values closer to zero correspond to a random or convergent pattern of evolution, while K values greater than 1 indicate strong phylogenetic signal and conservatism of traits
- Blomberg's K statistic of phylogenetic signal

# Analyze multiple traits in once

```
# sapply performs analysis on list of variables (numeric vectors)
sapply(X=list(body=primates.body, longevity=primates.longevity),
  FUN=Kcalc, phy=primates.tree, checkdata=FALSE)
sapply(X=list(body=primates.body, longevity=primates.longevity),
  FUN=phylosignal, phy=primates.tree, reps=1000)
# Alternative to use multiPhylosignal instead of sapply
multiPhylosignal(x=as.data.frame(cbind(primates.body,
  primates.longevity)), phy=primates.tree, reps=1000)
# Note sapply() and multiPhylosignal() return same data, but the
# matrices are transposed - use t() to transpose one to look like
# the other:
t(multiPhylosignal(x=as.data.frame(cbind(primates.body,
  primates.longevity)), phy=primates.tree, reps=1000))
```

# When there are vectors with standard errors of measurements

- Functions for testing of phylogenetic signal do not work with more measurements per taxon
  - Currently, the only possibility is `phylosig()` which is able to work with SE (user must prepare this vector from the data manually)
- `phylosig()` can be used as an alternative to `phylosignal()` – the functions are similar in basic usage

```
library(phytools)
?phylosig # See for details
# Test for phylogenetic signal (here without SE)
phylosig(tree=apiaceae.tree, x=apiaceae.eco, method="K", test=TRUE,
  nsim=1000)
phylosig(tree=primates.tree, x=primates.body, method="lambda",
  test=TRUE)
```

# Alternative testing for phylogenetic signal with GLM

- It is possible to use intercept-only (`model/formula` will be something like `variable ~ 1`, not `variable1 ~ variable2`) GLM to quantify phylogenetic signal in trait
- It is tricky to select the best correlation structure – AIC can help with selections

```
# Examples of usage of GLS for testing of phylogenetic signal
summary(gls(model=primates.longevity ~ 1, data=as.data.frame
  (primates.longevity), correlation=corBrownian(value=1,
  phy=primates.tree)))
summary(pgls(formula=shorebird.data[["M.Mass"]] ~ 1,
  data=comparative.data(phy=shorebird.tree, data=as.data.frame
  (cbind(shorebird.data[["M.Mass"]], shorebird.data[["Species"]])),
  names.col=V2, vcv=TRUE)))
```

# Phylogenetic principal component analysis

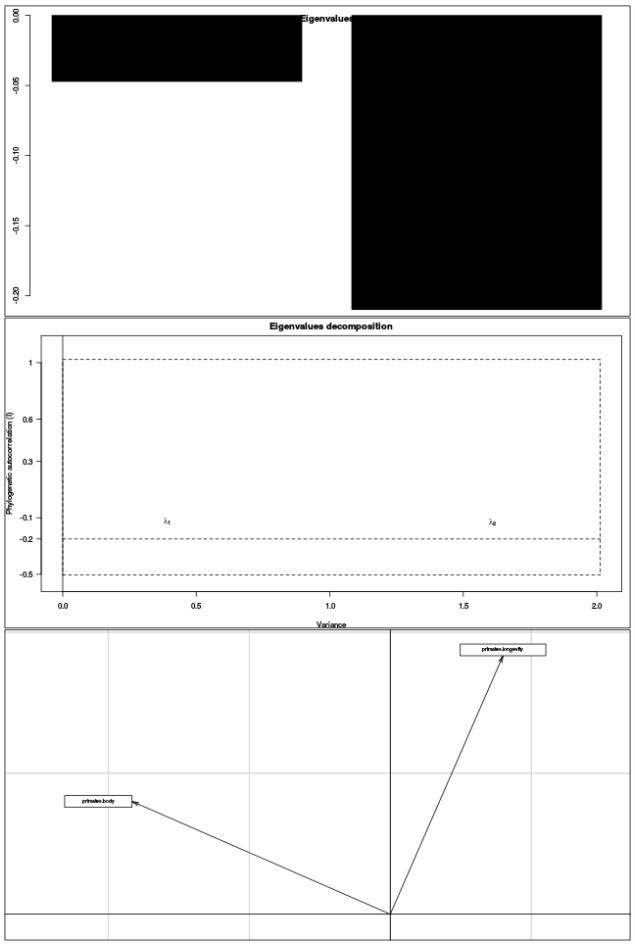
## PCA corrected for phylogeny

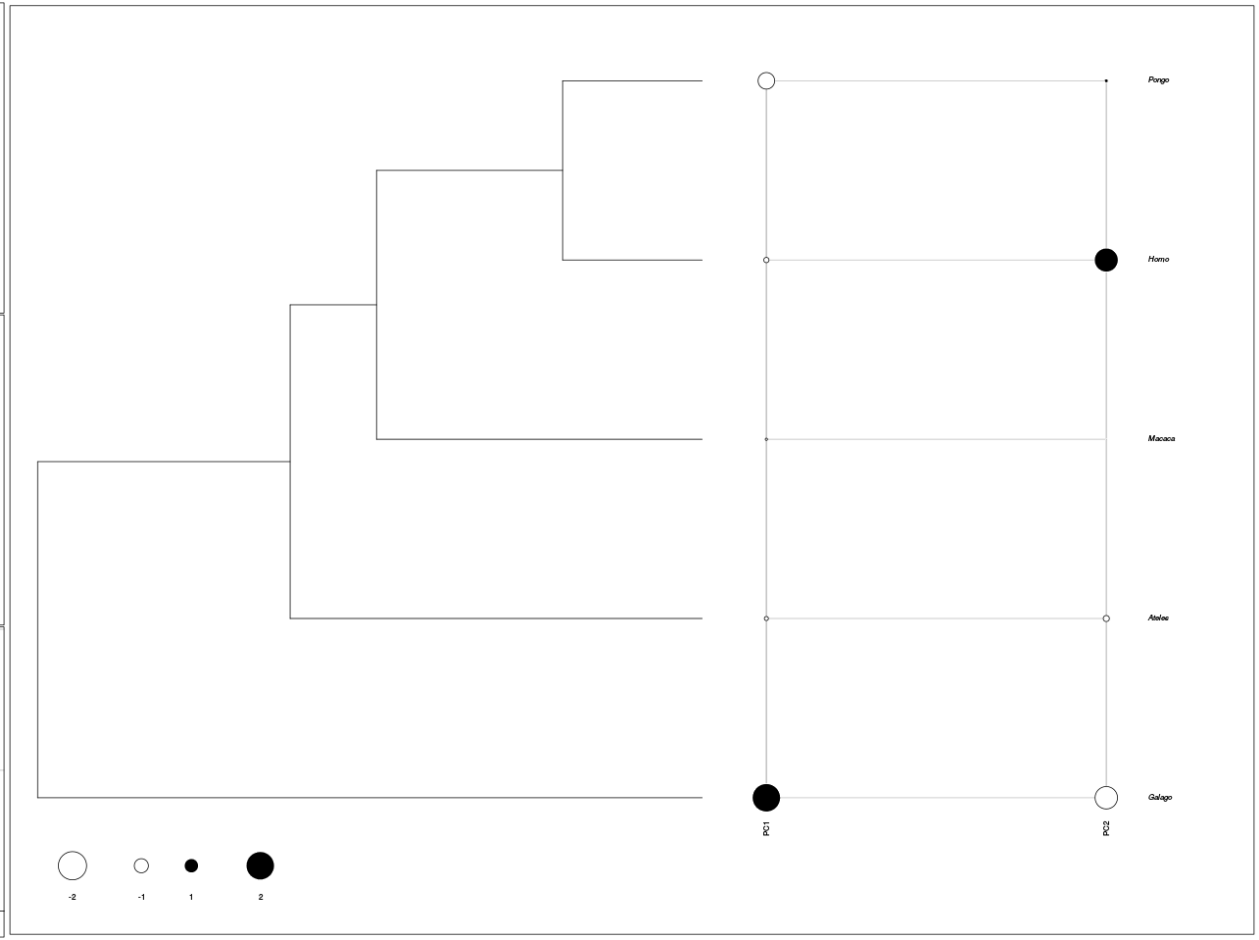
- It requires as input phylogenetic tree and respective comparative data
- Phylogenetic component is removed from the data, then classical PCA is calculated
- Together with nodes (taxa), PCA scores for PC axes are plotted – not the taxa – it shows trends of character evolution on the tree, not positions of taxa in PC space
- Other graphs show global vs. local structure, eigenvalues decomposition and positions of characters in virtual space (if they correlate or not)
- From package adephylo by Jombart et al. 2010
- It doesn't contain any test, it is more method of data exploration or dealing with big data sets, it is not for verifying hypothesis

# Phylogenetic principal component analysis – the code

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



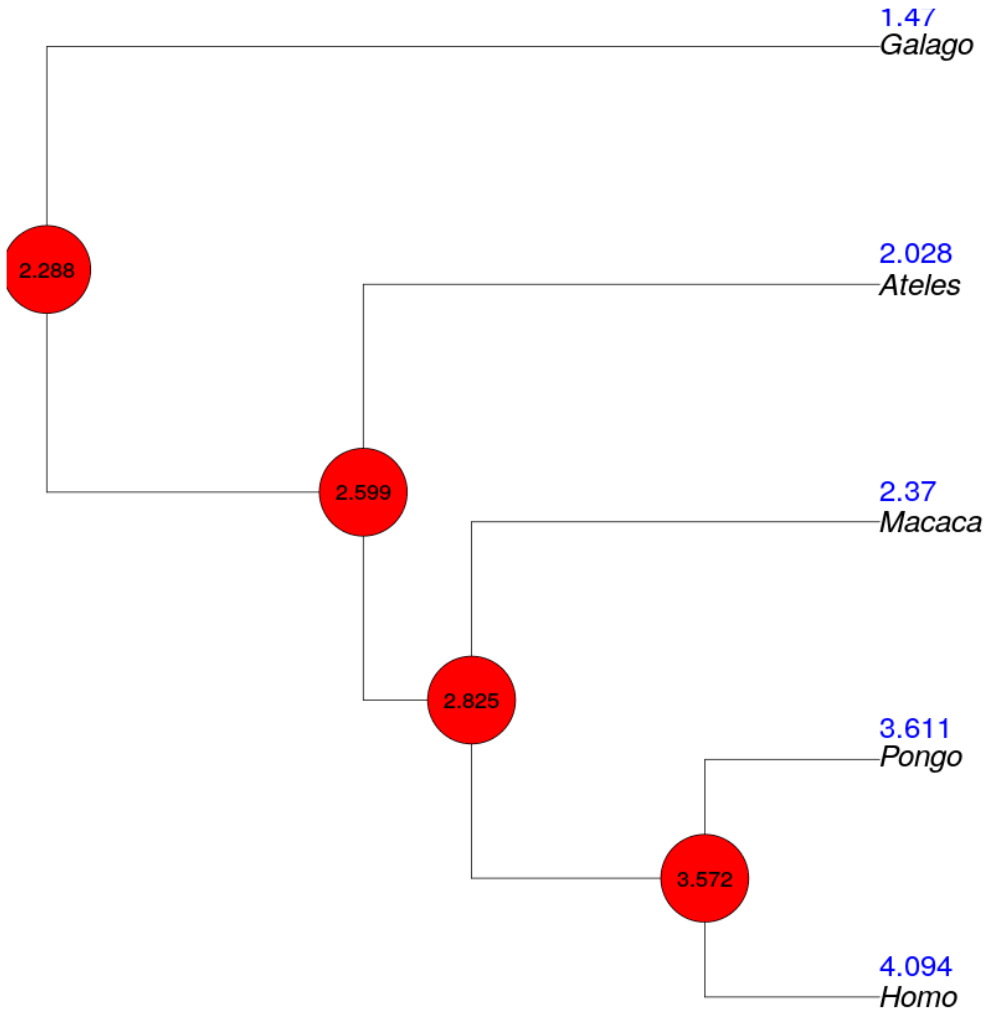



# Ancestral state reconstruction

- By default `ape::ace()` performs estimation for continuous characters assuming a Brownian motion model fit by maximum likelihood
- `ace()` can handle continuous as well as discrete data

```
# See ?ace for possible settings and estimations
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 $ace slot
# To be plotted on nodes - 1st column are node numbers
primates.body.ace
# Plot it
plot.phylo(primates.tree, lwd=2, cex=2)
tiplabels(round(primates.body, digits=3), adj=c(0, -1),
  frame="none", col="blue", cex=2)
nodelabels(round(primates.body.ace$ace, digits=3),
  frame="circle", bg="red", cex=1.5)
```

# Ancestral state reconstructions of primates body weights



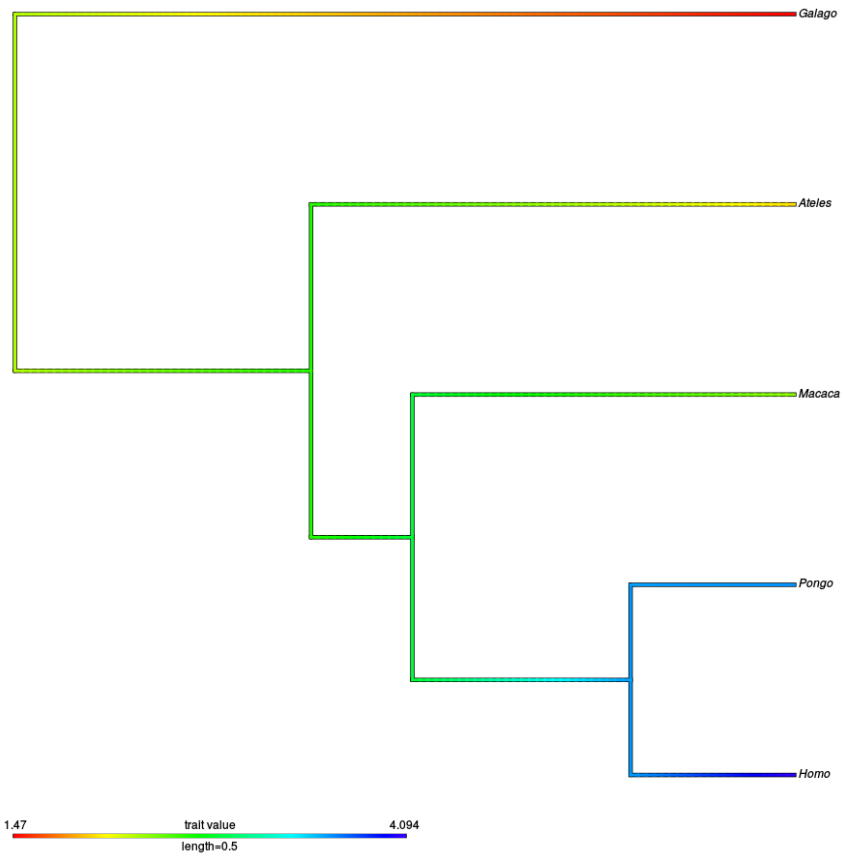
# Another possibilities (package phytools)

```
plot.phylo(primates.tree, lwd=2, cex=2)
# ML estimation of a continuous trait, can compute confidence interval
nodelabels(fastAnc(tree=primates.tree, x=primates.body))
# ACE for Brownian evolution with directional trend
nodelabels(anc.trend(tree=primates.tree, x=primates.body,
  maxit=1000)$ace)
# ACE for Brownian evolution using likelihood
nodelabels(round(anc.ML(tree=primates.tree, x=primates.body,
  maxit=1000, model="BM")$ace))
# Bayesian ancestral character estimation (next slide)
primates.body.ace.bayes <- anc.Bayes(tree=primates.tree,
  x=primates.body, ngen=1000) # Use more MCMC generations
primates.body.ace.bayes
nodelabels(primates.body.ace.bayes[11,3:6]) # See next slide
# ACE returns long numbers - truncate them by e.g.
round(x=..., digits=3) # "x" is vector with ACE values
# Another possibility for ancestral character reconstruction
?phangorn::ancestral.pml
```

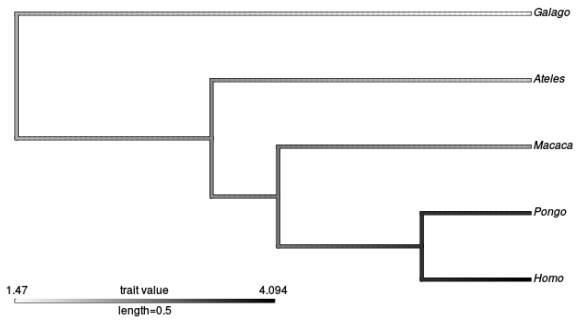
# Bayesian ancestral character estimation

```
primates.body.ace.bayes # Print the output object and check it
       gen     sig2       '6       7       8       9'    logLik
 [1,]    0 1.391013 2.288170 2.288170 2.288170 2.288170 -13.238593
 [2,]  100 1.394484 1.742455 2.248855 2.648808 3.105533  -7.552295
 [3,]  200 1.280646 1.501700 2.514334 2.524295 3.251273  -7.565108
 [4,]  300 1.230536 1.433547 2.242559 2.593056 2.725938 -10.204376
 [5,]  400 1.414644 1.648370 2.178676 2.573255 3.381587  -6.949438
 [6,]  500 2.069528 2.205779 2.111277 2.966358 3.361720  -8.461222
 [7,]  600 2.314460 2.361329 3.006070 2.995382 3.885636  -8.059215
 [8,]  700 2.808398 3.119423 3.621859 3.504098 3.736082  -9.159853
 [9,]  800 3.101082 2.281787 3.497516 2.526587 3.146811 -11.130158
[10,]  900 3.110501 2.971506 2.649267 2.913260 4.132872  -9.346940
[11,] 1000 2.361981'1.819626 2.674728 2.814608 3.412479' -7.964549
# We need node labels (nodes are numbered - here columns "6",
# "7", "8" and "9") from the last Bayes generation (here line 11)
primates.body.ace.bayes[11,3:6] # Use it for nodelabels()
       6        7        8        9
1.819626 2.674728 2.814608 3.412479
```

# Continuous map

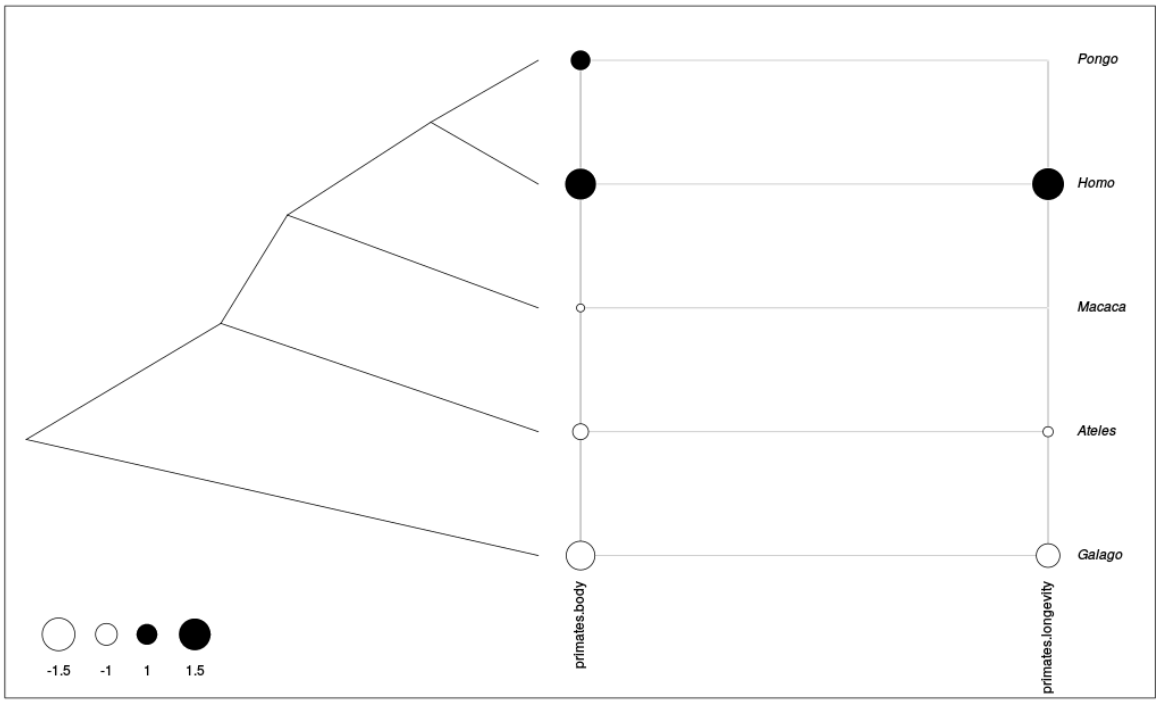



```
library(phytools)
contMap(tree=primates.tree,
  x=primates.body)
# Change colors with setMap()
primates.contmap <- setMap(x=
  contMap(primates.tree,
  primates.body),
  colors=c("white", "black"))
plot(primates.contmap)
# See ?par for more settings
```



# Display more characters on a tree in a table

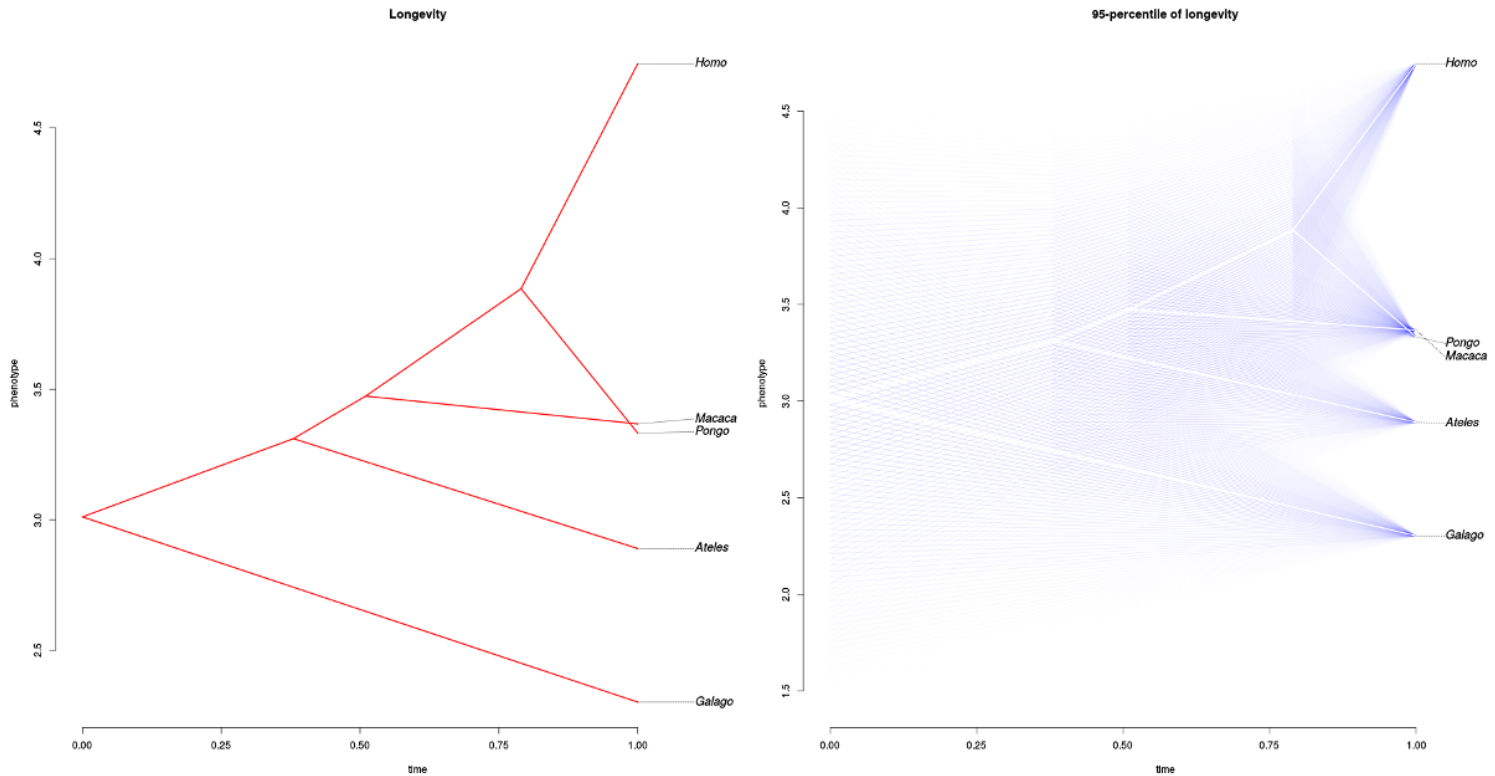
```
library(adephylo)
table.phylo4d(x=phylo4d(x=primates.tree, tip.data=as.data.frame
  (cbind(primates.body, primates.longevity))), treetype="cladogram",
  symbol="circles", scale=FALSE, ratio.tree=0.5)
table.phylo4d(x=phylo4d(x=shorebird.tree, tip.data=shorebird.data),
  treetype="cladogram", symbol="circles", scale=FALSE, ratio.tree=0.5)
```



# Phenogram

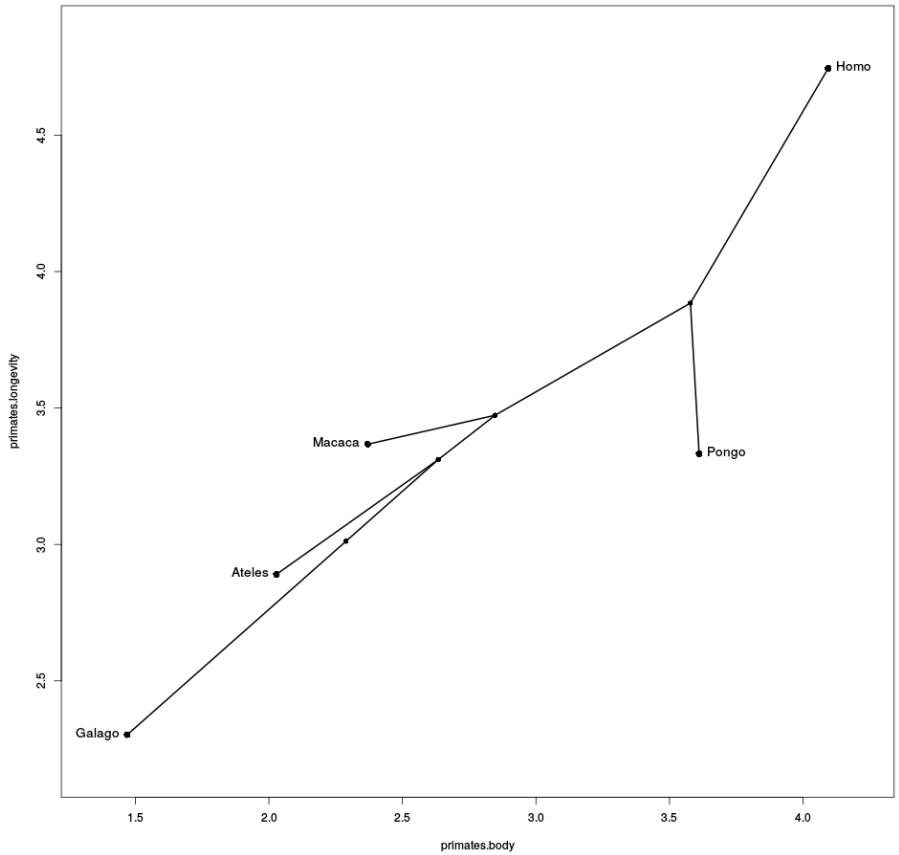
## Vertical axis shows character values

```
phenogram(tree=primates.tree, x=primates.longevity, fsize=1.2,
  ftype="i", colors="red", main="Longevity")
fancyTree(tree=primates.tree, type="phenogram95", x=primates.longevity,
  fsize=1.2, ftype="i", main="95-percentile of longevity")
```



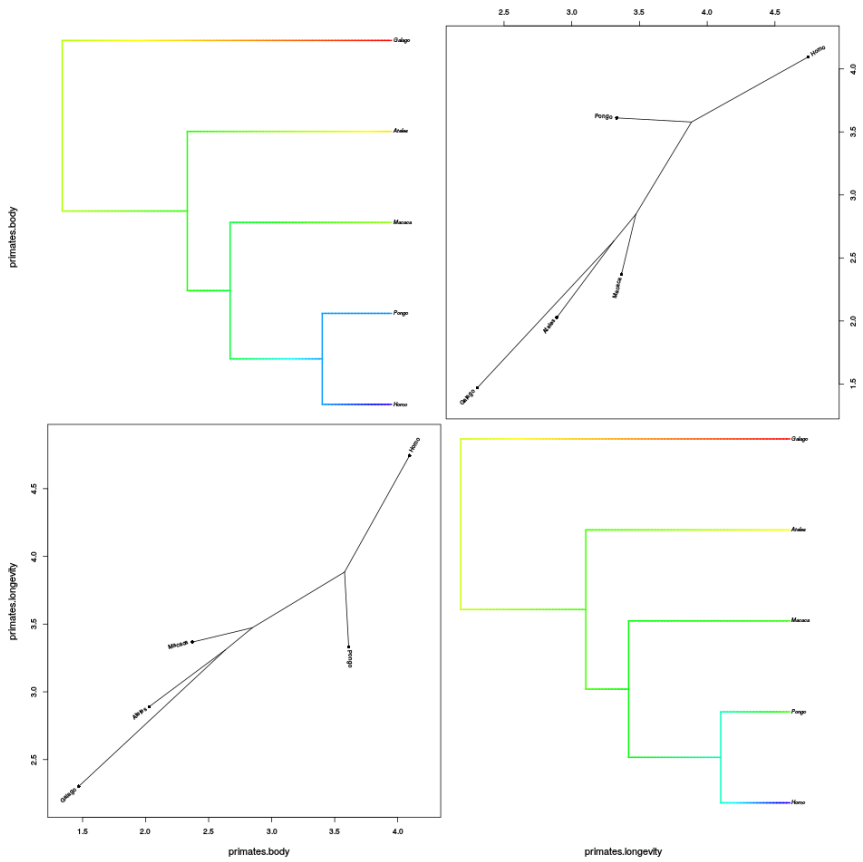
# Display 2 continuous characters in space and 3D tree connecting them

```
# 2 characters on 2 axis
phylomorphospace(tree=
  primates.tree, X=cbind
  (primates.body,
  primates.longevity),
  label="horizontal",
  lwd=2, fsize=1.5)
# 3D (3rd character is fake here)
# 3 characters it a rotating cube
phylomorphospace3d(tree=
  primates.tree, X=cbind
  (primates.body,
  primates.longevity,
  abs(primates.body-
  primates.longevity)),
  label=TRUE)
```



# Combine phenograms and ancestral state reconstructions

```
# 2 characters on 2 axis
fancyTree(tree=primates.tree,
  type="scattergram",
  X=cbind(primates.body,
  primates.longevity),
  res=500, ftype="i")
# See manuals for more settings
?fancyTree
?phenogram
?phylomorphospace
?phylomorphospace3d
?contMap
?setMap
?par
```



# Direct saving of plots to disk

Useful e.g. if plot should be bigger than screen, requires special settings, if done in batch, script, etc.

```
# Output figure will be saved to the disk as OutputFile.png
png(filename="OutputFile.png", width=720, height=720, bg="white")
# Here can go any number of functions making plots...
plot(...) # Whatever...
# When using plotting commands, nothing is shown on the screen
# The final plot(s) will be saved by:
dev.off() # Closes graphical device - needed after use of plotting
          # functions png(), svg(), pdf(), ... followed by any
          # function like plot() to write the file(s) to the disk
filename="OutFiles_%03d.png" # Returns list of files named
                             # OutFiles_001.png, OutFiles_002.png, ...
                             # Useful for functions returning more
                             # graphs.
?png # These functions have various possibilities to set size, whatever.
?svg # Exact possibilities of all 3 functions vary from system to system
?pdf # according to graphical libraries available in the computer.
```

# Graphical packages

- Basic plotting functions in R are very limited...
  - The usage is simple, but anything more complicated requires extensive coding (plenty of examples were shown in the course)...
  - It can be tricky to get desired figure – some magic use to be needed...
- There are plenty of graphical packages
- Advanced functions we used internally by used packages are lattice (web), gplot and ggplot2 (web1, web2)
  - They have enormous possibilities, but it is large topic for another long course...
- `par()` sets graphical parameters for following plots (splitting into panes, style of lines, points, text – see `pch`, `lwd`, `lty`, `cex`, `mai`, `mar`, `mfcol`, `mfrow`, ...) – see help pages...
- Most important low-level functions are `points`, `lines`, `text`, `abline`, `legend`, `axis`, `axes`, `arrows`, `box` – see help pages...

# Install package from GitHub

- GitHub is currently probably the most popular platform to host development of open-source projects – plenty of R packages are there
- Git is version controlling system – it traces changes among all versions – absolutely crucial for any software development
- Normal stable version of package is installed from repository as usual, but sometimes it can be useful to get latest developmental version (e.g. when it fixes some bug and new release is not available yet)

```
# Needed library
install.packages("devtools")
library(devtools)
dev_mode(on=TRUE)
# Install selected package from GitHub (user/project)
install_github("thibautjombart/adegenet")
# when finished go back to normal version
dev_mode(on=FALSE)
```

# R script and its running from command line

- R script is just plain `TXT` file with `.r` (e.g. `myscript.r`) extension with list of R commands
- Mark all user comments with `#` on the beginning
- In command line (Linux/Mac OS X/Windows/...) use
  - `Rscript myscript.r` to work **interactively** – all output is written to the terminal (as usual), user can be asked for some values, ...
  - `R CMD BATCH myscript.r` to let it run **non-interactively** – all output is written into `myscript.Rout`, terminal is clean and user can not influence the script anyhow – e.g. on MetaCentrum
- Script ends when there is some error or on the end of the file
- When working on both Windows and Mac OS X/Linux, take care about end of lines
  - Windows and UNIX (Linux, Mac OS X, ...) have different internal symbol for new line
  - Use UNIX command line utilities `dos2unix myscript.r` or `unix2dos myscript.r` to get correct ends of lines for target system

# Simple function

- Functions pack sets of commands for more comfortable repeated usage
- People more interested in R programming need to check special courses and/or documentation

```
# General syntax:
MyFunction <- function (x, y) {
  # Any commands can be here...
  x + y
  }
# Use as usually:
MyFunction(5, 8)
MyFunction(1, 4)
MyFunction(x=4, y=7)
MF <- MyFunction(9, 15)
MF # See it works
```

# Simple loop – for cycles

- Loops repeat one task given number of times
- Variable `i` has changing value for every repetition – useful for working with indexes (within lists, matrices, ...)
- It is possible to use variables or numeric output of functions in `from:to` expression – this is very variable
- In `for` loop we know in advance the number of repetitions (cycles), in `while` loop (next slide) we don't

```
# Simplest loop - print value of "i" in each step
# "i" is commonly used for various indexing
for (i in 1:5) { print(i) }
[1] 1 # This is the value of "i"...
[1] 2
[1] 3
[1] 4
[1] 5
```

# For and while loops

```
# In every step modify value of variable "X" (add 1 to previous value)
X <- 0 # Set initial value
for (i in 10:1) {
  # Any commands can be here...
  print("Loop turn") # Some message for user
  print(i) # Print number of turn - note it is decreasing
  X <- X+i # Rise value of "X" by current value of "i" (previous line)
  print(paste("Variable value:", X)) # Print current value of "X"
  }
# Work on each item of a list object
# Print length of each sequence in nothofagus.sequences
for (L in 1:length(nothofagus.sequences)) {
  print(length(nothofagus.sequences[[L]]))
  }
# While loop - it is done while the condition is valid
# While value of "Q" is < 5 (starting from 0), print it and add 1
Q <- 0
while (Q < 5) { print(Q <- Q+1) }
```

# If-else branching I

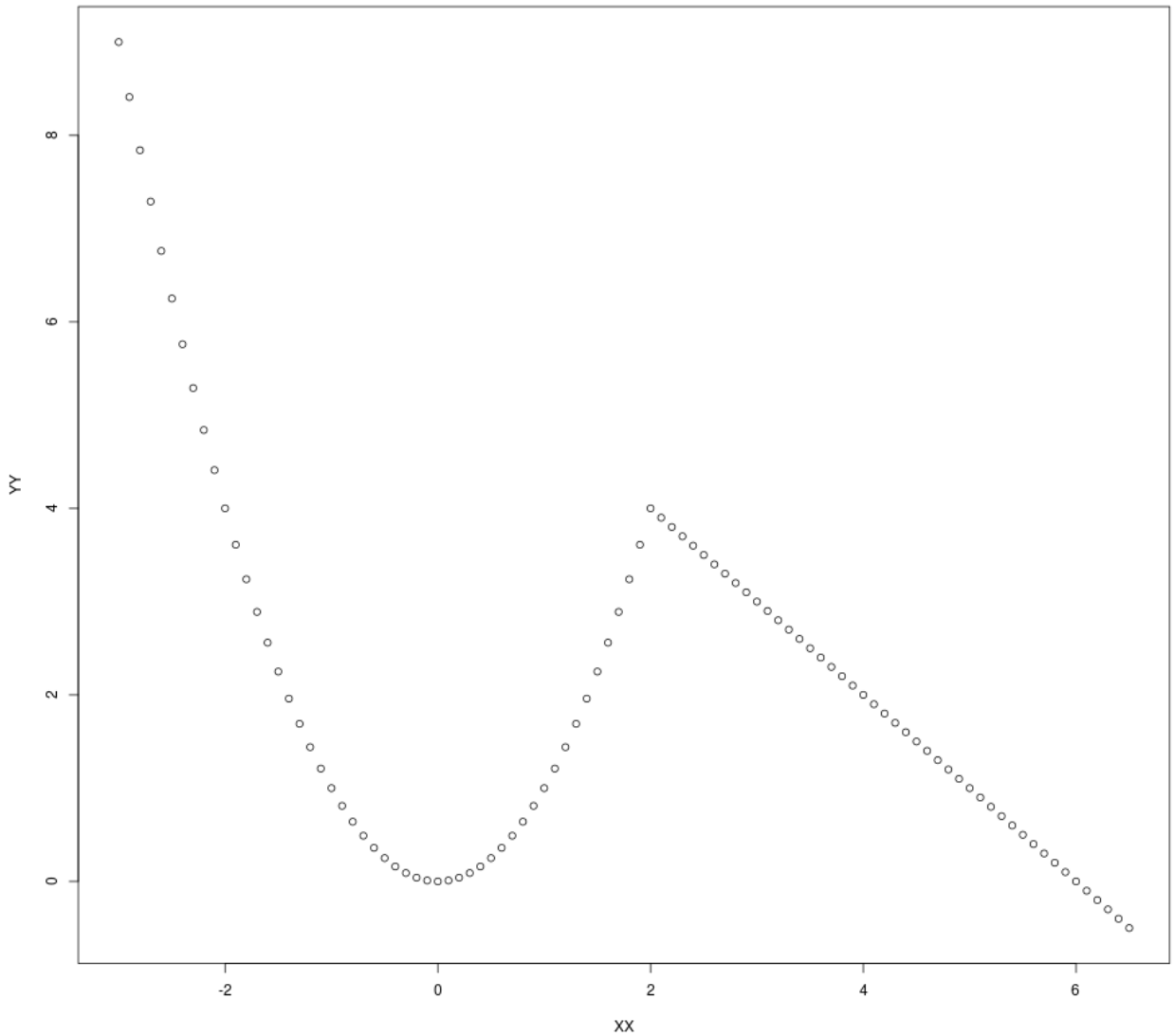
- Basic method of branching the code – **if** the condition is met, one branch is followed, **else** – in any other case – the other branch of the code is executed
- `else` part can be missing – the code is executed **only if** the condition is met

```
1  XX <- seq(from=-3, to=6.5, by=0.1)
2  XX
3  YY <- c()
4  for (II in 1:length(XX)) {
5    if(XX[II] <= 2) { # Executed for XX <= 2
6      YY[II] <- XX[II]^2
7      } else if(XX[II] > 2) { # Executed for XX > 2
8        YY[II] <- 6-XX[II]
9        }
10   }
11 YY # See next two slides for the end of the example
```

# If-else branching II

```
plot(XX, YY) # See the result
# Or (different possibility to get very same result)
# Note "XX" is reused from the previous slide
CC <- function(AA) {
  if(AA <= 2) { # Executed for XX <= 2
    BB <- AA^2
  } else { # Executed for XX > 2
    BB <- 6-AA
    }
  return(BB) # The output value
  }
 CC # Previously, "YY" contained values to plot made by the for loop,
    # here "CC" contains function to by used by sapply() when plotting
 plot(sapply(XX, CC)) # See the result
# The plot (same for both ways how to do it) is on next slide
```

# Output of the if-else branching example



# Most common problems and their solutions I

- Something was not found (object, function file, ...)
  - Check spelling of all methods, parameters, etc.
  - Check all paths (slide 47)
  - Check if all required objects were correctly created in previous steps
  - Check if all required libraries are loaded
- Unknown parameter, method, etc.
  - Check spelling of all parameters, consult manual pages
  - Check if all required libraries are loaded
- Graphics is not plotted correctly
  - Graphical window is too small (common problem with RStudio on screen with low resolution) – try to enlarge plotting window/pane
  - Reset graphical settings from some previous plot(s) by (repeated) calling of `dev.off()`
- R does nothing (but CPU is not extensively used)

# Most common problems and their solutions II

- R is waiting for some user input
- If command line starts with `+`, previous line was not completed correctly (e.g. missing closing bracket `)`) – check syntax, add it and hit `Enter`
- Some functions show plots and ask user for decision what to do (e.g DAPC, slide 124) – write the answer into command line or special window and hit `Enter`

- Some functions are not (without extra work) usable on all operating systems, some don't work correctly in GUI
  - Check manual and/or some on-line forum (slide 273 and onward)
- R and packages are more or less changing from version to version
  - Old methods can became outdated and not working anymore
  - Check release notes and change logs for new versions, manual pages and on-line forums (slide 273 and onward)
  - Generally, follow news for your topic (appropriate mailing list, ...)

# How to ask for help I

- Never ever ask simple silly lazy questions you can quickly find in manual or web
- People on mailing lists and forums respond volunteerly in their spare free time – do not waste it – be polite, brief and informative
- Be as specific and exact as possible
  - Write exactly what you did ("It doesn't work!" is useless...)
  - Copy/paste your commands and their output, especially error messages – they are keys to solve the problem
  - Try to search web for the error messages (or their parts)
  - Try to provide minimal working example – add at least part of your data (if applicable) so that the problem is reproducible
  - Specify version(s) of R/packages, operating system and/or another important details – authors will commonly insist on newest versions: add outputs of `sessionInfo()` and `packageVersion("PackageName")`

# How to ask for help II

- **R is free as freedom of speech – not as free beer!**
  - As soon as you don't pay for support, you can't blame anyone for lack of responses
  - There are plenty of reasons some package/function doesn't work – usage/data author didn't expect, unsupported operating system, author's mistake, user's mistake, ...
  - Authors wish their software to be useful – constructive feedback, reporting bugs and wishes is welcomed, but it must be provided in the way useful for the developer
- R functions commonly lack control of input data – error messages are returned by internal functions
  - They are not straightforward
  - It requires some training and experience to be quickly able to find what is going on
  - Always carefully read error messages and think about them
- Imagine you should answer – which information do you need?

# 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

# Further reading

## The most important books for our topics

Emmanuel Paradis

Analysis of Phylogenetics and Evolution with R, second edition

Springer, 2012

http://ape-package.ird.fr/APER.html

Michael J. Crawley

The R Book, second edition

Wiley, 2012

Paurush Praveen Sinha

Bioinformatics with R Cookbook

Packt Publishing, 2014

# Where to look for the help I

Before asking, ensure your question is in answerable form – slide 269.

- R homepage https://www.r-project.org/ and packages https://cran.r-project.org/web/packages/ (with documentation and links)
- List of R documentation https://cran.r-project.org/manuals.html
- 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 home page https://bioconductor.org/

# Where to look for the help II

- Bioconductor support forum https://support.bioconductor.org/ and Bioconductor help pages https://master.bioconductor.org/help/
- R phylo wiki http://www.r-phylo.org/wiki/Main_Page
- R phylogenetics at CRAN https://cran.r-project.org/web/views/Phylogenetics.html
- Integrated documentation search http://www.rdocumentation.org/
- RForge package repository https://r-forge.r-project.org/ (with documentation)
- Little Book of R for Bioinformatics https://a-little-book-of-r-for-bioinformatics.readthedocs.org/en/latest/

# Where to look for the help III

- Little Book of R for Multivariate Analysis https://little-book-of-r-for-multivariate-analysis.readthedocs.org/en/latest/
- Little Book of R for Biomedical Statistics https://a-little-book-of-r-for-biomedical-statistics.readthedocs.org/en/latest/
- Little Book of R for Time Series https://a-little-book-of-r-for-time-series.readthedocs.org/en/latest/
- Adegenet web http://adegenet.r-forge.r-project.org/, help mailing list https://lists.r-forge.r-project.org/cgi-bin/mailman/listinfo/adegenet-forum and GitHub page https://github.com/thibautjombart/adegenet/wiki
- APE home page http://ape-package.ird.fr/

# Where to look for the help IV

- Information and manual about pegas http://ape-package.ird.fr/pegas.html
- Phytools http://phytools.org/, its blog http://blog.phytools.org/ and GitHub page https://github.com/liamrevell/phytools
- Poppr documentation http://grunwaldlab.cgrb.oregonstate.edu/primer-population-genetic-analyses-r/installation and forum https://groups.google.com/forum/#!forum/poppr
- ade4 home page http://pbil.univ-lyon1.fr/ADE-4/ade4-html/00Index.html?lang=eng and documentation http://pbil.univ-lyon1.fr/ade4/home.php?lang=eng
- Phangorn resources http://cran.r-project.org/web/packages/phangorn/index.html

# Where to look for the help V

- 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 ecology mailing list https://stat.ethz.ch/mailman/listinfo/r-sig-ecology
- Books about R https://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 https://journal.r-project.org/

# Where to look for the help VI

- R-bloggers – aggregation of R blogs http://www.r-bloggers.com/
- 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/
- R for open big data http://ropensci.org/
- Statistics with R http://zoonek2.free.fr/UNIX/48_R/all.html
- The R Inferno book http://www.burns-stat.com/documents/books/the-r-inferno/
- Springer R series https://www.springer.com/series/6991?detailsPage=titles

# Where to look for the help VII

- ggplot2 (the most powerful graphical library used by many packages) information http://ggplot2.org/
- plyr documentation http://plyr.had.co.nz/ – manipulation with data
- Learning R https://learnr.wordpress.com/
- R for Community Ecologists http://ecology.msu.montana.edu/labdsv/R/
- Try R on-line course full of adventure anf heroic quests http://tryr.codeschool.com/
- Quick-R learning resource http://statmethods.net/
- R manual and help search http://finzi.psych.upenn.edu/
- Biostars – general bioinformatics forum https://www.biostars.org/

# Where to look for the help VIII

- Biology – general forum about biology at StackExchange https://biology.stackexchange.com/
- 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)`

# Packages we used... I

We used following packages – but not all functions – explore them for more possibilities

- adegenet: exploration of genetic and genomic data
- adephylo: multivariate tools to analyze comparative data
- ade4: multivariate data analysis and graphical display (enhancements: ade4TkGUI, adegraphics)
- akima: cubic spline interpolation methods for irregular and regular gridded data
- ape: Analyses of Phylogenetics and Evolution
- Biostrings: string matching algorithms, and other utilities, for fast manipulation of large biological sequences or sets of sequences
- caper: phylogenetic comparative analyses
- corrplot: graphical display of a correlation or general matrix
- fields: tools for spatial data

# Packages we used... II

We used following packages – but not all functions – explore them for more possibilities

- geiger: fitting macroevolutionary models to phylogenetic trees
- Geneland: stochastic simulation and MCMC inference of structure from genetic data
- ggplot2: data visualisations using the Grammar of Graphics
- gplots: plotting data
- hierfstat: estimation and tests of hierarchical F-statistics
- ips: interfaces to phylogenetic software
- IRanges: infrastructure for manipulating intervals on sequences
- lattice: Trellis graphics, with an emphasis on multivariate data
- mapdata: supplement to maps, larger and/or higher-resolution databases
- mapproj: converts latitude/longitude into projected coordinates

# Packages we used... III

We used following packages – but not all functions – explore them for more possibilities

- maps: draws geographical maps
- maptools: manipulating and reading geographic data
- MASS: functions and datasets for venables and ripley's MASS
- MUSCLE: Multiple Sequence Alignment with MUSCLE
- mvtnorm: multivariate normal and t probabilities
- nlme: fits and compares Gaussian linear and nonlinear mixed-effects models
- ParallelStructure: R framework when running analysis in the population genetics software STRUCTURE
- PBSmapping: spatial analysis tools
- pegas: population and evolutionary genetics analysis
- permute: restricted permutation designs

# Packages we used... IV

We used following packages – but not all functions – explore them for more possibilities

- phangorn: phylogenetic analysis
- phylobase: phylogenetic structures and comparative data
- phyloch: interfaces and graphic tools for phylogenetic data
- phytools: phylogenetic analysis, comparative biology
- picante: integrates phylogenies and ecology
- plotrix: various labeling, axis and color scaling functions
- poppr: genetic analysis of populations with mixed reproduction
- RandomFields: simulation of Gaussian fields (+ RandomFieldsUtils)
- RgoogleMaps: interface to query the Google server for static maps and uses the map as a background image to overlay plots
- rworldmap: mapping global data

# Packages we used... V

We used following packages – but not all functions – explore them for more possibilities

- seqinr: exploratory data analysis and data visualization for biological sequence
- seqLogo: sequence logos for DNA sequence alignments
- sos: searches contributed R packages
- sp: classes and methods for spatial data
- spdep: spatial dependence: weighting schemes, statistics and models
- TeachingDemos: demonstrations for teaching and learning
- vegan: community ecology
- XVector: representation and manpulation of external sequences

# Another interesting packages (we did not use)... I

For your own explorations...

- adhoc: ad hoc distance thresholds for DNA barcoding identification
- apex: analysis of multiple gene data
- apTreeshape: analysis of phylogenetic tree topologies
- BAMMtools: analyzing and visualizing complex macroevolutionary dynamics on phylogenetic trees
- bayou: Bayesian fitting of Ornstein-Uhlenbeck models to phylogenies
- betapart: partitioning beta diversity into turnover and nestedness components
- Biodem: biodemography
- BioGeoBEARS: probabilistic inference of both historical biogeography as well as comparison of different models of range evolution

# Another interesting packages (we did not use)... II

For your own explorations...

- cati: community assembly processes using trait values of individuals or populations
- convevol: Quantifies and assesses the significance of convergent evolution
- corHMM: analysis of binary character evolution
- DAMOCLES: maximum likelihood of a dynamical model of community assembly
- DDD: diversity-dependent diversification
- dendextend: extending dendrogram objects
- DiscML: estimating evolutionary rates of discrete characters using maximum likelihood
- distory: geodesic distance between phylogenetic trees

# Another interesting packages (we did not use)... III

For your own explorations...

- diversitree: comparative phylogenetic analyses of diversification
- diveRsity: calculation of both genetic diversity partition statistics, genetic differentiation statistics, and locus informativeness for ancestry assignment
- evobiR: comparative and population genetic analyses
- expands: expanding ploidy and allele-frequency on nested subpopulations
- expoTree: calculates the density dependent likelihood of a phylogenetic tree
- gee: generalized estimation equation solver
- genetics: population genetics
- geomorph: geometric morphometric analyses of 2D/3D landmark data

# Another interesting packages (we did not use)... IV

For your own explorations...

- ggtree: visualization and annotation of phylogenetic trees
- HardyWeinberg: statistical tests and graphics for HWE
- HMPTrees: models, compares, and visualizes populations of taxonomic tree objects
- hwde: models and tests for departure from HWE and independence between loci
- HWxtest: tests whether a set of genotype counts fits the HW expectations
- HyPhy: macroevolutionary phylogentic analysis of species trees and gene trees
- iteRates: iterates through a phylogenetic tree to identify regions of rate variation

# Another interesting packages (we did not use)... V

For your own explorations...

- jaatha: simulation-based maximum likelihood parameter estimation
- kdetrees: non-parametric method for identifying potential outlying observations in a collection of phylogenetic trees
- knitr: general-purpose tool for dynamic report generation
- LDheatmap: graphical display, as a heat map, of measures of pairwise linkage disequilibria between SNPs
- Linarius: dominant marker analysis with mixed ploidy levels
- markophylo: markov chain models for phylogenetic trees
- MCMCglmm: MCMC generalised linear mixed models
- MINOTAUR: multivariate visualisation and outlier analysis
- MonoPhy: Visualization of monophyletic clades on a tree
- MPSEM: modeling phylogenetic signals using eigenvector maps

# Another interesting packages (we did not use)... VI

For your own explorations...

- mvMORPH: multivariate comparative tools for fitting evolutionary models to morphometric data
- ouch: Ornstein-Uhlenbeck models for evolution along a phylogenetic tree
- OUwie: analysis of evolutionary rates in an OU framework
- paleoPhylo: assess how speciation, extinction and character change contribute to biodiversity
- paleotree: paleontological and phylogenetic analyses of evolution
- paleoTS: analyze paleontological time-series
- pastis: phylogenetic assembly with soft taxonomic inferences
- PBD: protracted birth-death model of diversification
- PCPS: principal coordinates of phylogenetic structure

# Another interesting packages (we did not use)... VII

For your own explorations...

- phyclust: phylogenetic clustering
- phyloclim: integrating phylogenetics and climatic niche modeling
- PHYLOGR: manipulation and analysis of phylogenetically simulated data sets and phylogenetically based analyses using GLS
- phyloland: models a space colonization process mapped onto a phylogeny
- phylolm: phylogenetic linear models and phylogenetic generalized linear models
- phyloTop: calculating and viewing topological properties of phylogenetic trees
- phylotools: supermatrix for DNA barcodes using different genes
- plyr: splitting, applying and combining data

# Another interesting packages (we did not use)... VIII

For your own explorations...

- pmc: phylogenetic Monte Carlo
- polyfreqs: Gibbs sampling algorithm to perform Bayesian inference on biallelic SNP frequencies, genotypes and heterozygosity in a population of autopolyploids
- polysat: polyploid microsatellite analysis
- RADami: phylogenetic analysis of RADseq data
- RColorBrewer: ColorBrewer palettes
- rdryad: access for Dryad web services
- Reol: interface to the Encyclopedia of Life
- rphast: interface to PHAST software for comparative genomics
- RMesquite: interoperability with Mesquite
- Rphylip: interface for PHYLIP

# Another interesting packages (we did not use)... IX

For your own explorations...

- SigTree: identify and visualize significantly responsive branches in a phylogenetic tree
- spatstat: spatial point pattern analysis
- spider: analysis of species limits and DNA barcoding
- splits: delimiting species and automated taxonomy at many levels of biological organization
- strap: stratigraphic analysis of phylogenetic trees, palaeontology
- strataG: analyzing stratified population genetic data
- surface: fitting Hansen models to investigate convergent evolution
- SYNCSA: analysis of metacommunities based on functional traits and phylogeny of the community components
- taxize: taxonomic information from around the web

# Another interesting packages (we did not use)... X

For your own explorations...

- TESS: simulation of reconstructed phylogenetic trees under tree-wide time-heterogeneous birth-death processes and estimation of diversification parameters under the same model
- treebase: discovery, access and manipulation of TreeBASE phylogenies
- TreePar: estimating birth and death rates based on phylogenies
- TreeSim: simulating phylogenetic trees

And more... R is continuously evolving and new packages are arising...

# Orientation in so many packages...

- ...is not easy...
- Many methods are implemented in more packages
  - Quality and richness of implementations may vary a lot...
  - Same methods in different packages may require data in different formats/R classes (conversion use to be simple – but always see respective documentation)
- Anyone can create and submit R package...
  - Plenty of packages to choose from...
  - No restrictions (apart basic technical requirements in repositories) – quality may be variable...
- Follow news on R sites, mailing lists, journal articles introducing new packages, etc.
- Be open for new tools, explore, try, share your experience

# The methods are over

- We went in more or less details through plenty of methods to work with molecular data to analyze phylogeny, population genetics, evolution and so on in R
- There are many more methods to try...
- It is nearly impossible to go in reasonable time through all relevant R tools – a lot of space for you

# The end

## Our course is over...

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

...any feedback is welcomed...

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

... any final questions?

Typesetting using XeLaTeX on openSUSE GNU/Linux, January 20, 2017.