

# 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](mailto:zeisek@natur.cuni.cz)

February 19 to 22, 2018



# Outline I

## 1 Introduction

## 2 R

- Installation

- Let's start with R

- Basic operations in R

- Packages for our work

## 3 Data

- Overview of data and data types

- Microsatellites

- AFLP

- Notes about data

- DNA sequences, SNP

- VCF

- Export

## 4 Basic analysis

# Outline II

First look at the data

Statistics

Genetic distances

Hierarchical clustering

AMOVA

MSN

NJ (and UPGMA) tree

PCoA

## 5 DAPC

Bayesian clustering

Discriminant analysis and visualization

## 6 SNP

PCA and NJ

## 7 Spatial analysis

Moran's I

# Outline III

sPCA

Monmonier

Mantel test

Geneland

Plotting maps

## 8 Structure

Running Structure from R

ParallelStructure on Windows

Post processing

## 9 Alignment

Overview and MAFFT

MAFFT, Clustal, MUSCLE and T-Coffee

Display and cleaning

## 10 Trees

Manipulations

# Outline IV

Seeing trees in the forest

MP

Comparisons

Notes about plotting the trees

## 11 Evolution

PIC

Autocorrelation

Decomposition

PGLS

GEE

Phylosignal

pPCA

Ancestral state

Phenogram

## 12 The end

# Outline V

Graphics

GitHub

Scripts

Functions

Loops

If-else branching

Solving problems

Resources

Summary

The end

# The course information

- The course page:  
<https://trapa.cz/en/course-molecular-data-r-2018>
  - Český: <https://trapa.cz/cs/kurz-molekularni-data-r-2018>
- Subject in SIS: <https://is.cuni.cz/studium/eng/predmety/index.php?do=predmet&kod=MB120C16>
  - Český: <https://is.cuni.cz/studium/predmety/index.php?do=predmet&kod=MB120C16>
  - For students having subscribed the subject, requirements are on next slide
- Working version is available at <https://github.com/V-Z/course-r-mol-data> — feel free to contribute, request new parts or report bugs

# Requirements to exam (“zápočet”)

- 1 Be present whole course.
- 2 Be active — ask and answer questions.
- 3 Process some data. This will be very variable and individual. Everyone should be able take some data (according to her/his interest) and do several simple analysis (according to her/his interest). Students can of course use manual, internet, discuss with anyone. The aim is to repeat part the most interesting/important for the student and edit introduced commands to fit her/his needs. Students can thus bring their data (if they are not too large), download any data from the internet or I can give them some toy data.
- 4 Write at least one page (can be split into multiple articles) on Wikipedia about any method or related topic discussed during the course. Again, this is very open, students can write about any topic they like. I prefer native language of the student (typically to make larger non-English Wikipedia).



# Materials to help you...

- Download the presentation from [https://soubory.trapa.cz/rcourse/r\\_mol\\_data\\_phylogen.pdf](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](https://soubory.trapa.cz/rcourse/course_commands.html)
- Download the script from [https://soubory.trapa.cz/rcourse/course\\_commands.r](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** (next slide) — 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

# Importance of good text editor

Can your text editor...?

- Show syntax highlight
- Show line numbers
- Show space between brackets
- Open any encoding and EOL
- Fold source code
- Show line breaks
- Mark lines
- Open multiple files
- [Kate](#)
- [KWrite](#)
- [Vim](#)
- [GNU Emacs](#)
- [Geany](#)
- [Bluefish](#)
- Advanced search and replace
- Use regular expressions
- Make projects, add notes
- Use command line
- Check spelling
- Debug source code
- [Gedit](#)
- [Notepad++](#)
- [Sublime](#)
- [Atom](#)
- [Nano](#)
- And [more...](#)
- The best option is to use text editor of selected R GUI...

# Think before you type (and hit Enter)...

- Commands from file `course_commands.r` can be mostly directly launched without editing them, but before you do so...
  - ① Read the command and all comments around, do not blindly launch it
  - ② Ensure you understand, what is **aim** of the command (what it is supposed to do)
  - ③ Ensure you understand all **limitations** of the method (when you can use it and when not)
  - ④ Ensure you understand **syntax** of the command (what it technically does)
- Some commands from `course_commands.r` *do require* to be edited according to particular user's computer — it is described in the comments around the command
- Learning R is effective only if you learn R syntax (language grammar), otherwise you only memorize commands without understanding them or blindly repeat someone's code — in such case, you wouldn't be able to solve any issue with your workflow

# 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, VCF, ...
  - 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, ...)

# The R

Basic introduction to work with R, installation of all required software

## 2 R

Installation

Let's start with R

Basic operations in R

Packages for our work

# 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, multi-platform, very powerful
- RKWard <https://rkwart.kde.org/> — feature very rich, developed mainly for Linux, available also for another operating systems
- R commander (Rcmdr) <http://www.rcommander.com/> — multi-platform, not so rich as previous
- Java GUI for R <https://rforge.net/JGR/> — Java (multi-platform, 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

The screenshot displays the RKward interface with two main windows:

- Left Window (R Script Editor):** Shows R code for performing a k-means clustering analysis. The code includes:
 

```

            # K-means
            # Správna hierarchická informácia PC (ale stále)
            srs_kmind2 <- find_clusters(srs_genind, 2)
            srs_kmind <- find_clusters(srs_genind, 2)
            srs_kmind3 <- find_clusters(srs_genind, 3)
            srs_kmind4 <- find_clusters(srs_genind, 4)
            srs_kmind5 <- find_clusters(srs_genind, 5)
            srs_kmind6 <- find_clusters(srs_genind, 6)
            srs_kmind7 <- find_clusters(srs_genind, 7)
            srs_kmind8 <- find_clusters(srs_genind, 8)
            srs_kmind9 <- find_clusters(srs_genind, 9)
            srs_kmind10 <- find_clusters(srs_genind, 10)
            srs_kmind11 <- find_clusters(srs_genind, 11)
            srs_kmind12 <- find_clusters(srs_genind, 12)
            srs_kmind13 <- find_clusters(srs_genind, 13)
            srs_kmind14 <- find_clusters(srs_genind, 14)
            srs_kmind15 <- find_clusters(srs_genind, 15)
            srs_kmind16 <- find_clusters(srs_genind, 16)
            srs_kmind17 <- find_clusters(srs_genind, 17)
            srs_kmind18 <- find_clusters(srs_genind, 18)
            srs_kmind19 <- find_clusters(srs_genind, 19)
            srs_kmind20 <- find_clusters(srs_genind, 20)
            srs_kmind21 <- find_clusters(srs_genind, 21)
            srs_kmind22 <- find_clusters(srs_genind, 22)
            srs_kmind23 <- find_clusters(srs_genind, 23)
            srs_kmind24 <- find_clusters(srs_genind, 24)
            srs_kmind25 <- find_clusters(srs_genind, 25)
            srs_kmind26 <- find_clusters(srs_genind, 26)
            srs_kmind27 <- find_clusters(srs_genind, 27)
            srs_kmind28 <- find_clusters(srs_genind, 28)
            srs_kmind29 <- find_clusters(srs_genind, 29)
            srs_kmind30 <- find_clusters(srs_genind, 30)
            srs_kmind31 <- find_clusters(srs_genind, 31)
            srs_kmind32 <- find_clusters(srs_genind, 32)
            srs_kmind33 <- find_clusters(srs_genind, 33)
            srs_kmind34 <- find_clusters(srs_genind, 34)
            srs_kmind35 <- find_clusters(srs_genind, 35)
            srs_kmind36 <- find_clusters(srs_genind, 36)
            srs_kmind37 <- find_clusters(srs_genind, 37)
            srs_kmind38 <- find_clusters(srs_genind, 38)
            srs_kmind39 <- find_clusters(srs_genind, 39)
            srs_kmind40 <- find_clusters(srs_genind, 40)
            srs_kmind41 <- find_clusters(srs_genind, 41)
            srs_kmind42 <- find_clusters(srs_genind, 42)
            srs_kmind43 <- find_clusters(srs_genind, 43)
            srs_kmind44 <- find_clusters(srs_genind, 44)
            srs_kmind45 <- find_clusters(srs_genind, 45)
            srs_kmind46 <- find_clusters(srs_genind, 46)
            srs_kmind47 <- find_clusters(srs_genind, 47)
            srs_kmind48 <- find_clusters(srs_genind, 48)
            srs_kmind49 <- find_clusters(srs_genind, 49)
            srs_kmind50 <- find_clusters(srs_genind, 50)
            srs_kmind51 <- find_clusters(srs_genind, 51)
            srs_kmind52 <- find_clusters(srs_genind, 52)
            srs_kmind53 <- find_clusters(srs_genind, 53)
            srs_kmind54 <- find_clusters(srs_genind, 54)
            srs_kmind55 <- find_clusters(srs_genind, 55)
            srs_kmind56 <- find_clusters(srs_genind, 56)
            srs_kmind57 <- find_clusters(srs_genind, 57)
            srs_kmind58 <- find_clusters(srs_genind, 58)
            srs_kmind59 <- find_clusters(srs_genind, 59)
            srs_kmind60 <- find_clusters(srs_genind, 60)
            srs_kmind61 <- find_clusters(srs_genind, 61)
            srs_kmind62 <- find_clusters(srs_genind, 62)
            srs_kmind63 <- find_clusters(srs_genind, 63)
            srs_kmind64 <- find_clusters(srs_genind, 64)
            srs_kmind65 <- find_clusters(srs_genind, 65)
            srs_kmind66 <- find_clusters(srs_genind, 66)
            srs_kmind67 <- find_clusters(srs_genind, 67)
            srs_kmind68 <- find_clusters(srs_genind, 68)
            srs_kmind69 <- find_clusters(srs_genind, 69)
            srs_kmind70 <- find_clusters(srs_genind, 70)
            srs_kmind71 <- find_clusters(srs_genind, 71)
            srs_kmind72 <- find_clusters(srs_genind, 72)
            srs_kmind73 <- find_clusters(srs_genind, 73)
            srs_kmind74 <- find_clusters(srs_genind, 74)
            srs_kmind75 <- find_clusters(srs_genind, 75)
            srs_kmind76 <- find_clusters(srs_genind, 76)
            srs_kmind77 <- find_clusters(srs_genind, 77)
            srs_kmind78 <- find_clusters(srs_genind, 78)
            srs_kmind79 <- find_clusters(srs_genind, 79)
            srs_kmind80 <- find_clusters(srs_genind, 80)
            srs_kmind81 <- find_clusters(srs_genind, 81)
            srs_kmind82 <- find_clusters(srs_genind, 82)
            srs_kmind83 <- find_clusters(srs_genind, 83)
            srs_kmind84 <- find_clusters(srs_genind, 84)
            srs_kmind85 <- find_clusters(srs_genind, 85)
            srs_kmind86 <- find_clusters(srs_genind, 86)
            srs_kmind87 <- find_clusters(srs_genind, 87)
            srs_kmind88 <- find_clusters(srs_genind, 88)
            srs_kmind89 <- find_clusters(srs_genind, 89)
            srs_kmind90 <- find_clusters(srs_genind, 90)
            srs_kmind91 <- find_clusters(srs_genind, 91)
            srs_kmind92 <- find_clusters(srs_genind, 92)
            srs_kmind93 <- find_clusters(srs_genind, 93)
            srs_kmind94 <- find_clusters(srs_genind, 94)
            srs_kmind95 <- find_clusters(srs_genind, 95)
            
```
- Right Window (Dendrogram):** Displays a dendrogram showing hierarchical clustering of 60 samples. The samples are labeled on the right as POP7568 through POP6045. Two vertical lines indicate the inferred clusters: 'Inferred cluster 1' and 'Inferred cluster 2'. The dendrogram shows that cluster 1 contains samples POP7568 to POP6244, and cluster 2 contains samples POP6344 to POP6045.



# When using RKWard, consider change of settings of text editor for more comfortable work

The screenshot illustrates the process of configuring RKWard's text editor settings. The main window shows an R script with code for sequence analysis. The 'Settings' menu is open, and the 'Configure Editor...' option is selected. The 'Configure - RKWard' dialog box is open, showing the 'Appearance' tab with 'Dynamic Word Wrap' and 'Whitespacing' options. Another 'Configure - RKWard' dialog box is open, showing the 'Font & Color Schemas' tab with a 'Vim (dark)' schema selected.

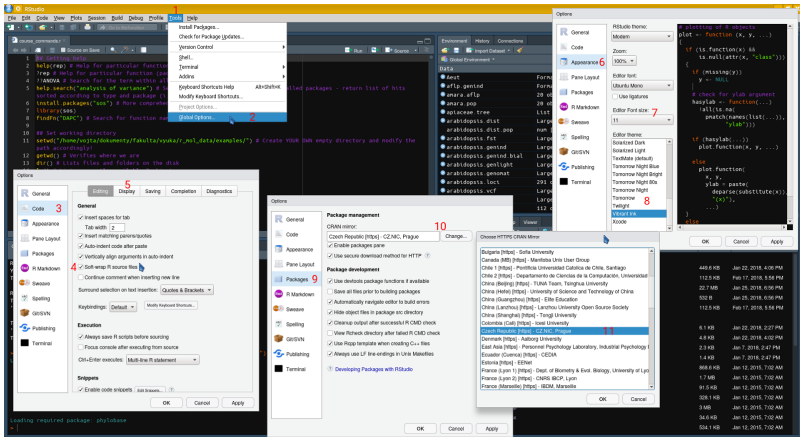
# RStudio

The screenshot displays the RStudio environment with the following components:

- Script Editor:** Contains R code for data analysis, including loading packages, summarizing statistics, and plotting results.
- Console:** Shows the R version (3.1.2) and copyright information for the R Foundation.
- Update Packages Dialog:** A window listing available updates for installed packages.
 

Package	Installed	Available	NEWS
BH	1.55.0-2	1.55.0-3	[icon]
HSAUR	1.3-4	1.3-5	[icon]
openair	1.0	1.1-0	[icon]
rgl	0.95.1158	0.95.1201	[icon]
Spatstat	1.05	1.6	[icon]
strata	2012.9-1	2015.1-1	[icon]
THdata	1.0-5	1.0-6	[icon]
- Environment/History:** Lists loaded packages such as library(ape), library(pegas), and library(poppr).
- Files/Plots/Packages/Help/Viewer:** A sidebar showing a list of available packages with their descriptions and versions.

# When using RStudio, turn on soft line wrap, select Czech mirror to download packages and consider change of appearance for more comfortable work



# 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://mirrors.nic.cz/R/>, 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
  - <https://download.opensuse.org/repositories/devel:/languages:/R:/patched/> (daily updated) or/and
  - <https://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](#)



# Important note about names of directories

- There must not be any spaces or accented characters in the path to R working directory or local R library, otherwise some R functions can fail (and there is no other solution)
- If the user has e.g. on Windows path like `C:/Documents and Settings/Šílen úpějící koníček/R kurzíček`, change it to something like `C:/Users/username/rcourse`, otherwise user can experience a lot of problems
- It might be required to make a new user on the computer...
- Similarly on Mac OS X and Linux, avoid directory names with spaces and accented characters

# Sources of R packages

- R CRAN <https://cran.r-project.org/> — main and largest source of R packages (over 12,000 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

```

R version 2.1.2 (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.

> citation()

To cite R in publications use:

  R Core Team (2014). R: A language and environment for statistical
  computing. R Foundation for Statistical Computing, Vienna, Austria.
  URL http://www.R-project.org/.

A BibTeX entry for LaTeX users is

  @Manual{
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2014},
    url = {http://www.R-project.org/},
  }

We have invested a lot of time and effort in creating R, please cite it
when using it for data analysis. See also 'citation("pkgname")' for
citing R packages.
  
```

# How it works

- General look of R commands:

```
1 function(argument1="SomeName", argument2=SomeVariable, argument3=8)
2 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:

```
1 SomeFunction(MyData, TRUE, 123, "SomeName")
```

- When only some of the arguments are in use, use the names (order doesn't matter any more)

```
1 SomeFunction(argument2=TRUE, argument3=123, argument1=MyData)
```

- Some arguments are required, some optional

# Get help in R

```

1 # "#" marks comments - notes within code which are not executed
2 help(function) # Help for particular function (package must be loaded)
3 ?function # Help for particular function (package must be loaded)
4 ??SearchedTerm # Search for the term within all installed packages
5 help.search("searched phrase") # Search for the phrase within all
6 # installed packages - return list of hits sorted according to
7 # type and package (i.e. package::function)
8 install.packages("sos") # More comprehensive search from packages
9 library(sos)
10 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:

```

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

```

- In Windows (**File | working directory**) or in RStudio (**Session | Set working directory**) set it in menu or by above command
- R saves history of commands into 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 losses 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)
- `vcfR` — imported (and possibly edited) VCF
- `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

# Popular object classes (we are going to use) II

- `treeshape` — derived from `hclust`
- `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	<code>as(x, "phylo4")</code>	phylobase
phylo	matching	<code>as.matching(x)</code>	ape
phylo	treeshape	<code>as.treeshape(x)</code>	apTreeshape
phylo	hclust	<code>as.hclust(x)</code>	ape
phylo	prop.part	<code>prop.part(x)</code>	ape
phylo	splits	<code>as.splits(x)</code>	phangorn
phylo	evonet	<code>evonet(x, from, to)</code>	ape
phylo	network	<code>as.network(x)</code>	ape
phylo	igraph	<code>as.igraph(x)</code>	ape
phylo4	phylo	<code>as(x, "phylo")</code>	phylobase
matching	phylo	<code>as.phylo(x)</code>	ape
treeshape	phylo	<code>as.phylo(x)</code>	apTreeshape
splits	phylo	<code>as.phylo(x)</code>	phangorn

# Conversions among data types II

From	To	Command	Package
splits	network	<code>as.network(x)</code>	phangorn
evonet	phylo	<code>as.phylo(x)</code>	ape
evonet	network	<code>as.network(x)</code>	ape
evonet	network	<code>as.network(x)</code>	ape
evonet	igraph	<code>as.igraph(x)</code>	ape
haploNet	network	<code>as.network(x)</code>	pegas
haploNet	igraph	<code>as.igraph(x)</code>	pegas
hclust	phylo	<code>as.phylo(x)</code>	ape
hclust	dendrogram	<code>as.dendrogram(x)</code>	stats
DNABin	character	<code>as.character(x)</code>	ape
DNABin	alignment	<code>as.alignment(x)</code>	ape
DNABin	phyDat	<code>as.phyDat(x)</code>	phangorn
DNABin	genind	<code>DNABin2genind(x)</code>	adegenet

# Conversions among data types III

From	To	Command	Package
character	DNABin	<code>as.DNABin(x)</code>	ape
character	loci	<code>as.loci(x)</code>	pegas
alignment	DNABin	<code>as.DNABin(x)</code>	ape
alignment	phyDat	<code>as.phyDat(x)</code>	phangorn
alignment	character	<code>as.matrix(x)</code>	seqinr
alignment	genind	<code>alignment2genind(x)</code>	adegenet
phyDat	DNABin	<code>as.DNABin(x)</code>	phangorn
phyDat	character	<code>as.character(x)</code>	phangorn
loci	genind	<code>loci2genind(x)</code>	pegas
loci	data frame	<code>class(x) &lt;- "data.frame"</code>	—
genind	loci	<code>genind2loci(x)</code>	pegas
genind	genpop	<code>genind2genpop(x)</code>	adegenet
genind	df	<code>genind2df(x)</code>	adegenet

# Conversions among data types IV

From	To	Command	Package
data frame	phyDat	<code>as.phyDat(x)</code>	phangorn
data frame	loci	<code>as.loci(x)</code>	pegas
data frame	genind	<code>df2genind(x)</code>	adegenet
matrix	phyDat	<code>as.phyDat(x)</code>	phangorn
vcfR	chromR	<code>vcfR2chromR(x)</code>	vcfR
vcfR	genind	<code>vcfR2genind(x)</code>	vcfR
vcfR	migrate	<code>vcfR2migrate(x)</code>	vcfR
vcfR	loci	<code>vcfR2loci(x)</code>	vcfR
vcfR	tidy	<code>vcfR2tidy(x)</code>	vcfR
vcfR	DNABin	<code>vcfR2DNABin(x)</code>	vcfR
vcfR	genlight	<code>vcfR2genlight(x)</code>	vcfR

# Basic operations with data I

```

1 x <- c(5, 6, 7, 8, 9) # Creates vector (see also ?rep)
2 x # Print "x" content
3 c() # Is generic function to concatenate objects into new one
4 length(x) # Length of the object - for matrices and DF use dim()
5 str(x) # Information about structure of the object
6 mode(x) # Gets type of storage mode of the object
7 class(x) # Shows class of the object
8 x[2] # Shows second element of the object
9 x <- x[-5] # Removes fifth element
10 y <- matrix(data=5:20, nrow=4, ncol=4) # Creates a matrix
11 is.matrix(y) # Is it matrix? Try is.<TAB><TAB>
12 # TAB key shows available functions and objects starting by typed text
13 y # Prints the matrix
14 y[,2] # Prints second column
15 y[3,] # Prints third row
16 y[4,3] # Prints element from fourth row and third column
17 x <- y[2,] # Replaces "x" by second row of "y" (no warning)
18 # R doesn't ask neither notifies when overwriting objects! Be careful!

```

# Basic operations with data II

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



# Repositories

- Repositories (internet directories full of R packages — slide 26) can be set via `options(repos=c())` or as `repos` parameter for each `install.packages()` command (slide 43 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 43 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 43. **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, but separated
- User has basically two options
  - ① Set all repositories in R and use basic commands to install packages (slide 43)
  - ② Specify non-CRAN repository every time installing from it (e.g. slide 48) or use special tools (e.g. for **Bioconductor** — slide 49)

# Set repositories

```

1 # Basic package installation
2 install.packages("PackageName") # Case sensitive!
3 ?install.packages # Shows all available parameters (options)
4 # We will need extra repositories. For Bioconductor keep version for
5 # your R version (BioC 3.6 for R 3.4, see https://bioconductor.org/).
6 options(repos=c("https://mirrors.nic.cz/R/", "https://bioconductor.
7   statistik.tu-dortmund.de/packages/3.6/bioc", "https://bioconductor.
8   statistik.tu-dortmund.de/packages/3.6/data/annotation", "https://
9   bioconductor.statistik.tu-dortmund.de/packages/3.6/data/experiment",
10  "https://r-forge.r-project.org/", "https://rforge.net/"))
11 getOption("repos") # Shows actual repositories
12 options() # Generic function to modify various settings
13 ?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

- If repositories from slide 43 are not set (for any reason), it is possible to install in several steps packages from main repository (CRAN) and from another sources (next slide)
- This is the basic and default the most common usage

```
1 # Simplest usage
2 install.packages("PackageName") # Case sensitive!
3 ?install.packages # See for more options
4 install.packages(pkgs=c("pkg1", "pkg2", "pkg3", ...),
5   repos=getOption("repos"), dependencies=TRUE)
6 # Installed packages are "inactive" - the mus by loaded to use them:
7 library(package) # Loads installed package (we will do it on the fly)
8 update.packages(repos=getOption("repos")) # Updates installed packages
9 update.packages(ask=FALSE) # Update installed (CRAN by default) packages
```

# Install packages without setting the repositories

```

1 install.packages(pkgs=c("ade4", "adegenet", "adephylo", "akima",
2   "animation", "ape", "boot", "brew", "caTools", "caper", "cluster",
3   "clusterGeneration", "coda", "colorspace", "combinat", "corrplot",
4   "deSolve", "deldir", "devtools", "digest", "doParallel", "dplyr",
5   "expm", "fastmatch", "fields", "foreach", "foreign", "gdata",
6   "geiger", "ggplot2", "git2r", "gmodels", "gplots", "gtable", "gtools",
7   "hierfstat", "httr", "igraph", "ips", "jsonlite", "KernSmooth",
8   "lattice", "lazyeval", "LearnBayes", "magrittr", "mapdata",
9   "mapplots", "mapproj", "maps", "maptools", "MASS", "Matrix",
10  "memoise", "memuse", "mgcv", "mnormt", "msm", "mvtnorm", "ncbit",
11  "nlme", "numDeriv", "PBSmapping", "pegas", "permute", "phangorn",
12  "philentropy", "phylobase", "phytools", "picante", "plotrix", "plyr",
13  "png", "poppr", "RandomFields", "RandomFieldsUtils", "Rcpp",
14  "reshape2", "rgdal", "rgeos", "RgoogleMaps", "rlang", "Rmpi", "rncl",
15  "RNeXML", "rworldmap", "scales", "scatterplot3d", "segmented",
16  "seqinr", "shapefiles", "shiny", "snow", "sos", "sp", "spData",
17  "spam", "spdep", "StAMPP", "stringr", "subplex", "TeachingDemos",
18  "tibble", "tidyr", "tripack", "vcfR", "vegan", "viridisLite",
19  "whisker", "withr", "XML"))

```



# 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

```
1 # If not done already, install required packages first
2 install.packages(pkgs=c("ape", "colorspace", "XML"),
3   dependencies=TRUE)
4 # It is possible to specify direct path
5 # Local or web URL - be careful about correct path) to package source
6 install.packages(pkgs="http://www.christophheibl.de/
7   phyloch_1.5-3.tar.gz", repos=NULL, type="source")
8 # If above command fails on Windows, try
9 install.packages(pkgs="http://www.christophheibl.de/phyloch_1.5-3.zip",
10   repos=NULL)
```

# Install Geneland package

- Check <https://www2.imm.dtu.dk/~gigu/Geneland/> and manual
- Since version 4, Geneland is not available in CRAN anymore

```

1 # Install package Geneland (since version 4 not available in CRAN)
2 # It is possible to specify direct path to package source
3 install.packages("https://www2.imm.dtu.dk/~gigu/Geneland/distrib/
4   Geneland_4.0.8.tar.gz", repos=NULL, type="source")
5 # If above command fails on Windows, try
6 install.packages("https://www2.imm.dtu.dk/~gigu/Geneland/distrib/
7   Geneland_4.0.8.zip", repos=NULL)
8 # Other packages used when using Geneland
9 # Needed is PBSmapping or mapproj for conversion of coordinates
10 # GUI uses for parallelisation snow and Rmpi
11 # RgoogleMaps (requires rgdal) can be used to plot Geneland output
12 # on top of Google map, mapproj (requires rgeos and sp), shapefiles
13 # (requires foreign) and tripack on GIS layer
14 install.packages(pkgs=c("PBSmapping", "RgoogleMaps", "Rmpi", "foreign",
15   "mapproj", "mapproj", "mapproj", "mapproj", "mapproj", "mapproj",
16   "tripack"), dependencies=TRUE)

```

# Bioconductor and others — differences from another repositories

```

1 # Standard installation
2 install.packages(pkgs=c("adegenet", "poppr", "phytools"))
3 update.packages(ask=FALSE) # Update packages
4 # Installation from custom repository (example for our course)
5 install.packages(pkgs="ParallelStructure",
6   repos="https://r-forge.r-project.org/")
7 ?install.packages # See help for details
8 # Bioconductor - if https fails, use http
9 source("https://bioconductor.org/biocLite.R")
10 ?biocLite
11 biocLite(pkgs=c("Biostrings", "muscle")) # Install package(s)
12 biocLite() # Update Bioconductor packages

```

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



# Bioconductor

- Tools for analysis of genomic data, see <https://bioconductor.org/>
- To install it, add appropriate repositories (repository has its own version number not strictly following R — change them accordingly when upgrading R, see [web](#)) or **use Bioconductor's special function** (recommended by Bioconductor):

```

1 # Load basic Bioconductor function
2 source("https://bioconductor.org/biocLite.R")
3 ?biocLite # Get help how to use it
4 # Install packages
5 biocLite(c("package1", "package2", "..."))
6 biocLite() # Update Bioconductor packages (within same R version)
7 # Upgrades installed Bioconductor packages to new R release
8 biocLite("BiocUpgrade") # E.g. from R 3.3 to 3.4

```

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

# Bioconductor packages for the course

- If repositories are not set via command on slide 43, 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/> — for us, **Technische Universität Dortmund** is the closest (fastest)

```
1 # Install needed Bioconductor packages
2 biocLite(pkgs=c("BiocGenerics", "Biostrings", "IRanges", "muscle",
3   "S4Vectors", "XVector"), ask=FALSE)
4 # From time to time update Bioconductor packages
5 biocLite()
6 # When upgrading to new version of R (e.g. from 3.3 to 3.4) upgrade by
7 biocLite("BiocUpgrade")
```

# Non-R software I

- 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
- Clustal (W/X; Omega is not used in the course)  
<http://clustal.org/>
  - Aligner of sequences (from slide 249)
- Gdal <https://trac.osgeo.org/gdal/wiki/DownloadingGdalBinaries>
  - Required for conversion of coordinates for Geneland (from slide 209)
- GIMP <https://www.gimp.org/>
  - Image manipulation, free open-source alternative to products of Adobe or Corel
  - Optional, one of possibilities to view and edit output graphics from R

# Non-R software II

- Inkscape <https://inkscape.org/>
  - Vector drawing, free open-source alternative to products of Adobe or Corel
  - Optional, one of possibilities to view and edit output graphics from R
- MAFFT <https://mafft.cbrc.jp/alignment/software/>
  - Aligner of sequences (from slide 249)
- MPI <http://www.stats.uwo.ca/faculty/yu/Rmpi/>
  - Library for parallelisation used by Rmpi package (optional, recommended)
  - If it is not available, respective function can sometimes use different parallelisation backend or user can turn off parallelisation for respective function
- MUSCLE <https://www.drive5.com/muscle/>
  - Aligner of sequences (from slide 249)
- proj <http://proj4.org/download.html>
  - Required for conversion of coordinates for Geneland (from slide 209)

# Non-R software III

- Structure <https://web.stanford.edu/group/pritchardlab/structure.html> (optionally also [CLUMPP](#) and [distruct](#) — not part of the course)
  - De facto standard for analysis of population genetic structure (from slide [236](#))
  - Its outputs are usually post-processed in other software (not part of the course, see [web](#))

# Our data

Import and export of data we will use through the course, data types

## 3 Data

Overview of data and data types

Microsatellites

AFLP

Notes about data

DNA sequences, SNP

VCF

Export

# 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
    - The allele **is** or **is not** present — it is impossible to distinguish heterozygots from dominant homozygots

# Brief overview of molecular data types II

Sorted with respect to usage in R

- 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, ...
    - 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 mitochondrial sequences and ITS1-5.8S rRNA-ITS2 region); and their variants



# Brief overview of molecular data types III

Sorted with respect to usage in R

- 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 every variant is shown
- For details about the molecular markers check specialized course like [Use of molecular markers in plant systematics and population biology \(česky\)](#)

# R training data

- R packages commonly contain training data to illustrate its abilities
- We will use some of them during the course
- We will also use data provided by the teacher and/or his colleagues

```

1 data() # List data available in currently loaded packages
2 # List data available in all installed packages (can be very long)
3 data(package=rownames(installed.packages()))
4 # Load selected data set
5 # For example phylogeny and species traits of shorebirds from package
6 # caper (we will use it much later)
7 data(shorebird, package="caper")
8 # Optional method (load respective library and then data)
9 library(caper) # Library containing (among others) desired data
10 data(shorebird) # Loading the data
11 ?shorebird # See content of the dataset # or
12 ?caper::shorebird

```

# Our data... I

- We will use
  - Modified subset of diploid microsatellite data of *Taraxacum haussknechtii* (Asteraceae) from Macedonia by [Zeisek et al 2015](#)
  - Modified subset of triploid microsatellite data of several species of *Taraxacum* sect. *Taraxacum* (Asteraceae) — *T. alatum*, *T. ekmanii*, *T. hemicyclum* and *T. hepaticum* from central and northern Europe by [Kirschner et al 2016](#)
  - Small subset of population AFLP data of *Cardamine amara* (Brassicaceae) from Europe by [Karol Marhold](#) and his team
  - Small subset of non-synonymous SNPs from [ASY3](#) gene (required for meiosis) from diploids and tetraploids of *Arabidopsis arenosa* (Brassicaceae) from central and northern Europe by [Magdalena Holcová](#) and her colleagues
  - Small subset of trees and taxa from phylogeny of *Oxalis* spp. (Oxalidaceae) from South Africa (the Cape region) by [Schmickl et al. 2016](#)

## Our data... II

- Partial mitochondrial sequence of mm18 control region of *Meles meles* (Mustelidae) from [Frantz et al 2014](#) downloaded from [GenBank](#)
- Maturase K (matK) plastid sequences of *Nothofagus* (Nothofagaceae) downloaded from [EMBL-EBI](#) (various authors)
- Phylogeny of selected species of Apiaceae reconstructed from several concatenated plastid genes downloaded from [GenBank](#) and simulated trait
- Training data from packages **Adegenet** (sequence of influenza from USA sampled in several years; SSRs genotypes of cattle breeds, `?adegenet::microbov`; and *Rupicapra rupicapra* (Bovidae) SSRs genotypes from French Bauges mountains, `?adegenet::rupica`), **ape** (morphological traits of Carnivora, `?ape::carnivora`) and **caper** (phylogeny and morphological traits of shorebirds, `?cape::shorebird`)
- 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

## Our data... III

- 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 data sets — for smaller data sets 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.

# Input/output data formats I

## Representative selection

- **BAM** (Binary Alignment Map)
  - Compressed version of SAM (see below)
  - Suffix `*.bam`
- **CSV** (Comma Separated Values)
  - “One sheet of Excel”
  - Common format to store data (traits, coordinates, ...), similar to TSV
  - Columns (cells) are separated by commas, cells are commonly bordered by quotation marks — it is important to check structure before import into R (and verify it after import)
  - Suffix usually `*.csv`
- **FASTA**
  - Simple and popular text format to store genetic sequences (DNA, RNA, proteins)

# Input/output data formats II

## Representative selection

- **Line 1** of every records starts with `>` and contain name/description of the sequence (e.g. `> Seq 1`), **line 2+** contain(s) the sequence (ATCG...) — until end of file or next line starting with `>`
- Suffix usually `*.fasta` (generic, also `*.fas`, `*.fa`, `*.seq`, `*.fsa`), `*.fna` (nucleic acid), `*.ffn` (nucleotides, coding regions), `*.faa` (amino acids), `*.frn` (non-coding RNA)
- **FASTQ**
  - Text based format to store sequences (mainly nucleotides)
  - Every record consists of **4 lines**: (1) sequence ID (with possible description) starting with `@`, (2) the sequence (ATCG...), (3) `+` optionally followed by the same ID as line 1, and (4) quality values for nucleotides from line 2
  - Common format for output of modern high throughput sequencing machines (e.g. Illumina)
  - Commonly compressed by `gzip` (`*.gz`), sometimes by other compression application

# Input/output data formats III

## Representative selection

- Suffix usually `*.fastq`, `*.fq`, `*.fastq.gz`, `*.fq.gz`, ...
- **NEWICK**
  - Every line contains one tree represented by brackets, optionally with numbers (separated by `:`) labeling nodes and/or branches (bootstrap supports, likelihoods, branch lengths, ages, ...)
  - E.g. `(A,B,(C,D)E)F`; or `(A:0.1,B:0.2,(C:0.3,D:0.4):0.5)`;
  - Suffix usually `*.newick`, `*.nwk`, `*.tre`, ...
- **NEXUS**
  - Popular plain text format used by software like [Mesquite](#), [MrBayes](#), [PAUP\\*](#), [SplitsTree](#), ...
  - Structure can be complex, is divided into blocks containing e.g. sequences, trees (in NEWICK format), distance matrix, fragmentation data, networks (e.g. for SplitsTree), MrBayes commands, traits, ...
  - Several variants, sometimes problems with interoperability
  - Suffix usually `*.nexus`, `*.nex`, `*.nxs`



# Input/output data formats IV

## Representative selection

- **SAM** (Sequence Alignment Map)
  - Text-based format for storing biological sequences aligned to a reference sequence
  - Structure is relatively complex
  - Used by applications like **bamtools** or **SAMtools**
  - Suffix usually **\*.sam**
- **TSV** (TAB separated values)
  - “One sheet of Excel”
  - Common format to store data (traits, coordinates, ...), similar to CSV
  - Columns (cells) are separated by tabs (**\t**) — it is important to check structure before import into R (and very it after import)
  - Suffix usually **\*.tsv**, **\*.tab**
- **TXT**
  - Plain text file can contain content of all listed file formats (except binary BAM) — suffix (**\*.txt**) is not really reliable...
  - Technically, all listed plain text formats belong also to this category

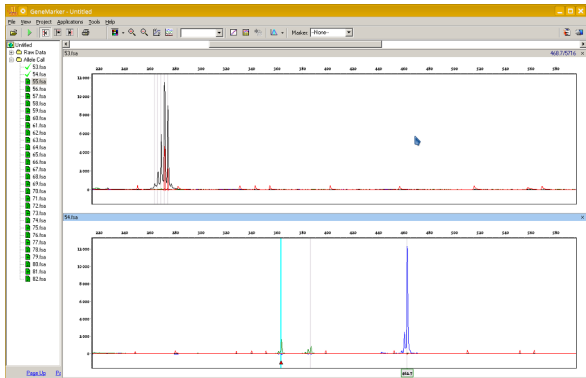
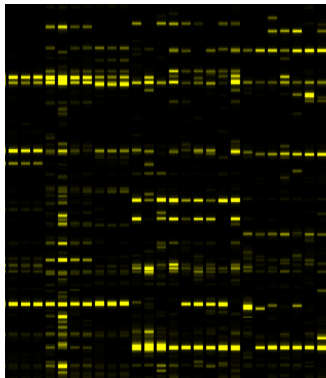
# Input/output data formats V

## Representative selection

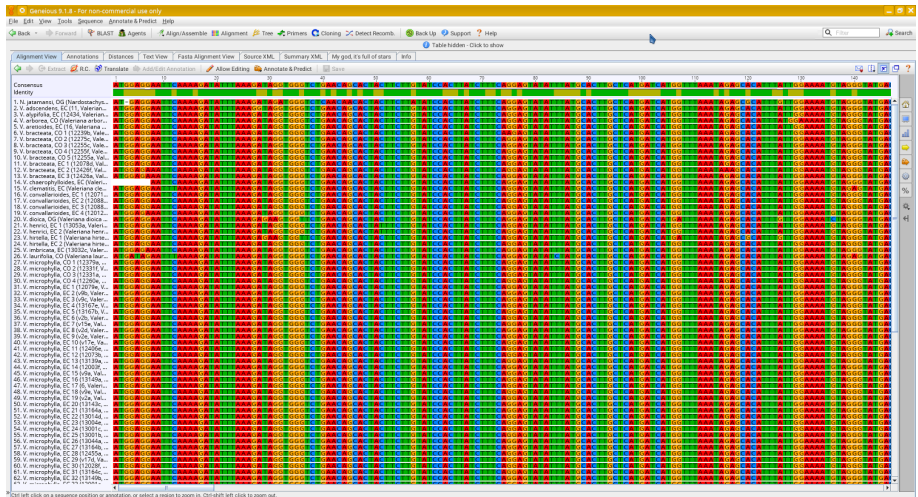
- **VCF** (Variant Call Format)
  - Do not confuse with **vCard** (`*.vcf`, `*.vcard`) storing virtual business cards and address books
  - Bioinformatics plain text format storing gene variants, annotations, quality data and more information
  - Used by software like **Bcftools**, **GATK**, **Picard**, **VCFtools**, ...
  - Complex structure, sequences are not stored as in FASTA, but as SNP variants on respective positions — useful to store processed NGS/HTS data (e.g. from Illumina machines)
  - Commonly compressed by **gzip** (`*.gz`), sometimes by other compression application
  - Several versions and variants (including binary BCF, `*.bcf`), sometimes there are problems with interoperability
  - Suffix usually `*.vcf` or `*.vcf.gz`

# Examples of data and formats I — AFLP

(presence/absence) gel and microsatellites from sequencing machine



# Examples of data and formats II — Aligned DNA sequences displayed in Geneious



# Examples of data and formats III — FASTA and FASTQ sequences in text view

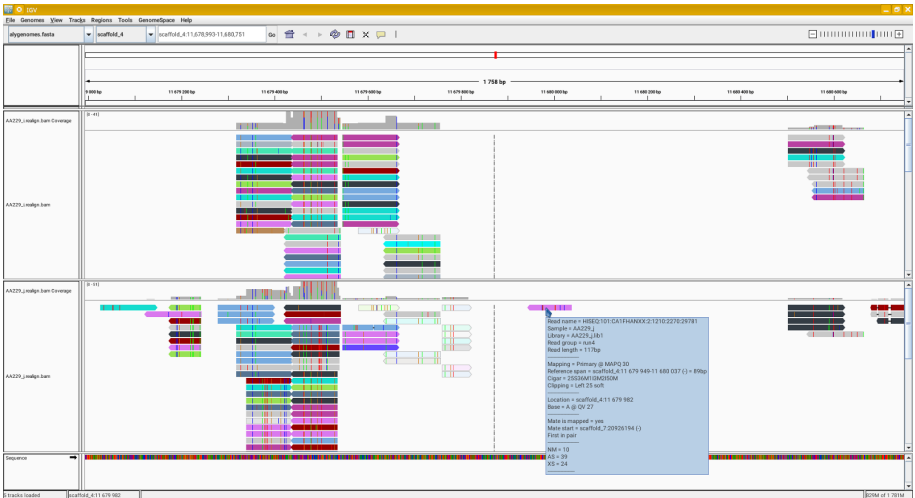
```

1 > CY013200
2 atgaagactatcattgctttgagctacattttatgtctggttttcgctcaaaaacttcccctagtgaaaaca
3 ggaaatgacaacagcacagcaacgctgtgcctgggacaccatgcagtgccaaacggaacgatcacgaatgat
4 > CY013781
5 atgaagactatcattgctttgagctacattttatgtctggttttcgctcaaaaacttccccaaattgaagt
6 ggaaatgacaacagcacagcaacgctgtgcctgggacaccatgcagtgccaaacggaacgactaatgctact
7 ... # FASTA continues...
8 @7001425F:141:CAV4JANXX:3:1104:1496:1976 1:N:0:GTGTCC
9 NCGATTCATTTAGCAATTAGACGTGAAGTCTCTTGATGAAAGACACTAACGAACTCTTTCCTTGGACACC
10 +
11 #<<BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
12 @7001425F:141:CAV4JANXX:3:1104:1250:1987 1:N:0:GTGTCC
13 TCGATTCACTGAACTGAATGTCCGACAACTTTAGTTTGTCTGTTTCTACCTACCAAAGTTCGGAGCTTCGA
14 +
15 BBBBFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
16 ... # FASTQ continues...

```

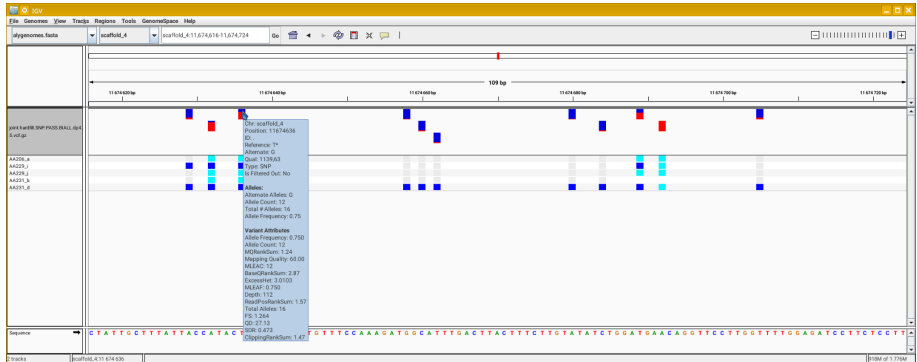
# Examples of data and formats IV — BAM displayed in IGV

It contains Illumina short reads mapped to reference



# Examples of data and formats V — VCF with multiple samples displayed in IGV

Variants of alleles and depth of coverage for each sample, mapped to reference



# Examples of data and formats VI — Parts of VCF in text view

```

1 ... # Header
2 ##ALT=<ID=NON_REF,Description="Represents any possible alternative al...
3 ##FILTER=<ID=DP_4,Description="DP < 4">
4 ... # Information about data stored
5 ##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in geno...
6 ##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency, for ...
7 ... # Information about chromosomes etc.
8 ##contig=<ID=scaffold_1,length=33132539>
9 ##contig=<ID=scaffold_2,length=19320864>
10 ... # The data (variants of nucleotides)
11 #CHROM POS ID REF ALT QUAL FILTER INFO FORMA...
12 scaffold_4 22846289 . C A 91.52 PASS ...
13 scaffold_4 22846291 . C T 325.58 PASS ...
14 ... AC=1; AF=1.057e-03; AN=946; BaseQRankSum=0.825; ClippingRankSum=0.118; ...
15 ... DP=4046; ExcessHet=3.0302; FS=0.000; InbreedingCoeff=-0.0096; MQ=79.24...
16 ... # More data...

```



# 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 — 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 342

# 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:

```
1   pop  msta93 msta101 msta102 msta103 msta105 msta131 ...
2 H01  He  269/269 198/198 221/223 419/419 197/197 196/196 ...
3 H02  He  275/283 198/198 221/223 419/419 193/193 168/190 ...
4 ...  ...      ...      ...      ...      ...      ...      ...
```

## Loading the data:

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

First line starts with empty cell (if `header` is presented), there can be any extra column, 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

```

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

```

## Source data:

```

1 Ind      lon    lat
2 H01    21.3333  41.1
3 ...      ...    ...

```

## Read coordinates:

```

1 hauss.coord <- read.csv("https://soubory.trapa.cz/rcourse/
2   haussknechtii_coordinates.csv", header=TRUE, sep="\t",
3   quote="", dec=".", row.names=1)
4 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

```
1 # Add coordinates - note identification of slots within object
2 hauss.genind$other$xy <- hauss.coord
3 # See result
4 hauss.genind$other$xy
5 hauss.genind
6 # Conversion to genpop - for population-level analysis
7 hauss.genpop <- genind2genpop(hauss.genind, process.other=TRUE)
8 # See result
9 hauss.genpop
10 # Removes missing data - see ?missingno for types of dealing them
11 # Use with caution! It modifies original data!
12 hauss.genind.cor <- missingno(pop=hauss.genind, type="mean",
13   cutoff=0.1, quiet=FALSE)
14 ?missingno # See other options of handling missing data
15 # Convert corrected genind to loci
16 hauss.loci.cor <- genind2loci(hauss.genind.cor)
17 # Writes loci file to the disk
18 write.loci(hauss.loci.cor, file="hauss.loci.cor.txt",
19   loci.sep="\t", allele.sep="/")
```

# Import existing data set from popular software

```
1 read.genalex() # poppr - reads *.csv file
2 read.fstat() # adegenet - reads *.dat files, only haploid/diploid data
3 read.genetix() # adegenet - reads *.gtx files, only ha/diploid data
4 read.genepop() # adegenet - reads *.gen files, only ha/diploid data
5 read.structure() # adegenet - reads *.str files, only ha/diploids
6 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

- `adegenet`, `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 124/124/124 237/237/237 164/164/172 136/136/138 ...  
5  ...           ...           ...           ...           ...
```

Triploid species of *Taraxacum* sect. *Taraxacum*



# How to import polyploid microsatellites

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

# Import of AFLP data — background

Two files — AFLP data with individual names, and populations

## AFLP or any other presence/absence data:

```
1 L1 L2 L3 L4 L5 L6 L7 L8 L9 ...  
2 Ind1 0 0 1 1 1 0 0 0 1 ...  
3 IndG 0 0 1 1 0 0 0 0 0 ...  
4 ... .. ...
```

AFLP data of *Cardamine amara* group

## Individual's populations:

```
1 POP  
2 pop1  
3 popZ  
4 ...
```

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

```
1 amara.aflp <- read.table(file="https://soubory.trapa.cz/rcourse/  
2   amara_aflp.txt", header=TRUE, sep="\t", quote="")  
3 amara.aflp  
4 dim(amara.aflp)  
5 class(amara.aflp) # Must be matrix or data frame  
6 # Populations - just one column with population names for all inds  
7 amara.pop <- read.table(file="https://soubory.trapa.cz/rcourse/  
8   amara_pop.txt", header=TRUE, sep="\t", quote="")  
9 amara.pop  
10 # You can use just one file, where populations are in last column and  
11 # in df2genind() use for example X=aflp[,1:XXX] and pop=aflp[,YYY]  
12 # Create genind object - ind.names and loc.names are taken from X  
13 aflp.genind <- df2genind(X=amara.aflp, sep="", ind.names=NULL,  
14   loc.names=NULL, pop=amara.pop[,1], missing=NA, type="PA")  
15 indNames(aflp.genind) <- amara.aflp[,1] # Add individual names  
16 aflp.genind  
17 summary(amara.aflp)  
18 # 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)

```
1 genind2df() # adegenet - export into data frame
2 genind2genalex() # poppr - export for genalex
3 splitcombine() # poppr - edits population hierarchy
4 popsub() # poppr - extracts only selected population(s)
5 clonecorrect() # poppr - corrects for clones
6 informloci() # poppr - removes uninformative loci
7 seppop() # adegenet - separates populations from genind or genlight
8 seploc() # adegenet - splits genind, genpop or genlight by markers
9 alleles2loci() # pegas - transforms a matrix of alleles into "loci"
10 # seppop and seploc return lists of genind objects - for further
11 # analysis using special functions to work on lists (see further)
12 # 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 using genind are able to handle that...

# Import of DNA sequence data I

```
1 # Reading FASTA (read.dna() reads also another formats, see ?read.dna)
2 # Sequences of flu viruses from various years from USA
3 # (Adegenet toy data)
4 usflu.dna <- read.dna(file="http://adegenet.r-forge.r-project.org/
5   files/usflu.fasta", format="fasta")
6 class(usflu.dna) # Check the object
7 usflu.dna # Check the object
8 # Another possibility (only for FASTA alignments, same result):
9 usflu.dna2 <- fasta2DNABin(file="http://adegenet.r-forge.r-project.org
10   /files/usflu.fasta") # Normally keeps only SNP - see ?fasta2DNABin
11 class(usflu.dna2) # Check the object
12 usflu.dna2 # Check the object
13 as.character(usflu.dna2)[1:5,1:10] # Check the object
14 dim(usflu.dna2) # Does it have correct size?
15 # Read annotations
16 usflu.annot <- read.csv("http://adegenet.r-forge.r-project.org/files/
17   usflu.annot.csv", header=TRUE, row.names=1)
18 head(usflu.annot) # See result
```

# Import of DNA sequence data II

```
1 # Convert DNABin to genind - only polymorphic loci (SNPs) are retained
2 # When converting DNABin to genind, the sequences must be aligned!
3 usflu.genind <- DNABin2genind(x=usflu.dna, pop=usflu.annot[["year"]])
4 usflu.genind # Check it
5 # read.fasta() from seqinr package reads DNA or AA in FASTA format and
6 # returns a list (DNABin is for us now better choice)
7 usflu.dna3 <- seqinr::read.fasta(file="http://adegenet.r-forge.
8   r-project.org/files/usflu.fasta", seqtype="DNA")
9 class(usflu.dna3)
10 length(usflu.dna3) # How many sequences we have in the list
11 usflu.dna3 # Check it
12 # Convert into DNABin class (technically, DNABin is a list)
13 class(usflu.dna3) <- "DNABin"
14 # Read sequence data in NEXUS
15 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*, phylogeographical study of Frantz et al. 2014)

```
1 # Importing sequences according to sequence ID
2 meles.dna <- read.GenBank(c("KJ161355.1", "KJ161354.1", "KJ161353.1",
3 "KJ161352.1", "KJ161351.1", "KJ161350.1", "KJ161349.1", "KJ161348.1",
4 "KJ161347.1", "KJ161346.1", "KJ161345.1", "KJ161344.1", "KJ161343.1",
5 "KJ161342.1", "KJ161341.1", "KJ161340.1", "KJ161339.1", "KJ161338.1",
6 "KJ161337.1", "KJ161336.1", "KJ161335.1", "KJ161334.1", "KJ161333.1",
7 "KJ161332.1", "KJ161331.1", "KJ161330.1", "KJ161329.1", "KJ161328.1"))
8 meles.dna
9 class(meles.dna)
10 # Converts DNABin to genind - extracts SNP - for large data sets can be
11 # computationally very intensive
12 meles.genind <- DNABin2genind(meles.dna)
13 meles.genind # Check it
```

- To query on-line database as through web we use [seqinr](#) (next slide)



# Query on-line sequence databases

```
1 library(seqinr)
2 choosebank() # List genetic banks available for seqinr
3 choosebank("embl") # Choose some bank
4 ?query # See how to construct the query
5 # Query selected database - there are a lot of possibilities
6 nothofagus <- query(listname="nothofagus",
7   query="SP=Nothofagus AND K=rbcl", verbose=TRUE)
8 nothofagus$req # See the sequences information
9 # Get the sequences as a list
10 nothofagus.sequences <- getSequence(nothofagus$req)
11 nothofagus.sequences # See sequences
12 # Get annotations
13 nothofagus.annot <- getAnnot(nothofagus[["req"]])
14 nothofagus.annot
15 closebank() # Close the bank when work is over
16 # Convert sequences from a list to DNABin (functions as.DNABin*)
17 nothofagus.dna <- as.DNABin.list(nothofagus.sequences)
18 nothofagus.dna # See it
```

# Importing SNP

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

```
1 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:

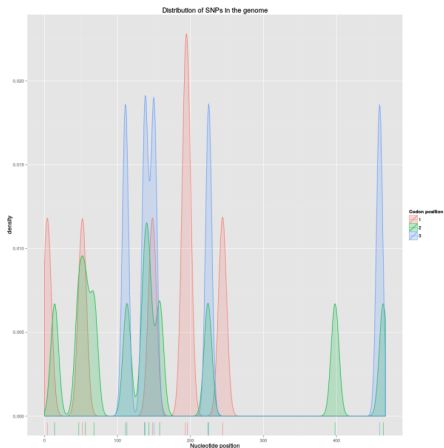
```
1 usflu.genlight <- fasta2genlight  
2 (file="http://adegenet.r-forge.r-project.org/files/usflu.fasta",  
3 quiet=FALSE, saveNbAlleles=TRUE)  
4 ?fasta2genlight # Function has several options to speed up reading  
5 # If it crashes (on Windows), try add parameter "parallel=FALSE"
```

- For small data sets, 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

```

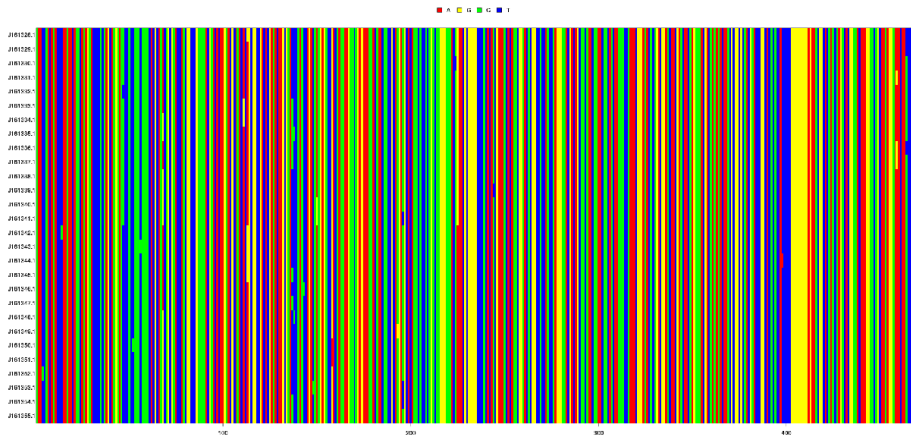
1 # Position of polymorphism within
2 # alignment - snpossi.plot requi-
3 # res input data in form of matrix
4 snpossi.plot(x=as.matrix(
5   meles.dna), codon=FALSE)
6 # Position of polymorphism within
7 # alignment-differentiating codons
8 snpossi.plot(as.matrix(meles.dna))
9 # When converting DNAbin to genind
10 # only polymorphic loci are kept -
11 # threshold for polymorphism can
12 # be arbitrary (polyThres=...)
13 meles.genind <- DNAbin2genind(x=
14   meles.dna, polyThres=0.01)
15 meles.genind # See it
16 # Test is distribution of SNPs
17 # is random (1000 permutations)
18 snpossi.test(as.matrix(meles.dna))
    
```



# Checking sequences

```
1 # Nucleotide diversity
2 pegas::nuc.div(x=meles.dna)
3 # Base frequencies
4 ape::base.freq(x=meles.dna)
5 # GC content
6 ape::GC.content(x=meles.dna)
7 # Number of times any dimer/trimer/etc oligomers occur in a sequence
8 # Note: count() requires single sequence as DNABin/character
9 seqinr::count(seq=as.character.DNABin(meles.dna[["KJ161328.1"]]),
10  wordsize=3)
11 # View sequences - all must be of the same length
12 image(x=usflu.dna, c("a", "t", "c", "g", "n"), col=rainbow(5))
13 # Function "image" requires as input matrix
14 # So that sequences must be of same length
15 image(x=as.matrix(meles.dna), c("a", "t", "c", "g", "n"),
16  col=rainbow(5))
17 # Direct function to display the sequences
18 image.DNABin(x=usflu.dna)
19 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

# Reading VCF

- Download input file from <https://soubory.trapa.cz/course/arabidopsis.vcf.gz> (`read.vcfR` does not work well yet with network addresses)
- Non-synonymous SNPs from **ASY3** gene (required for meiosis) from diploids and tetraploids of *Arabidopsis arenosa* from central and northern Europe
- Package `vcfR` has functions to manipulate and explore VCF

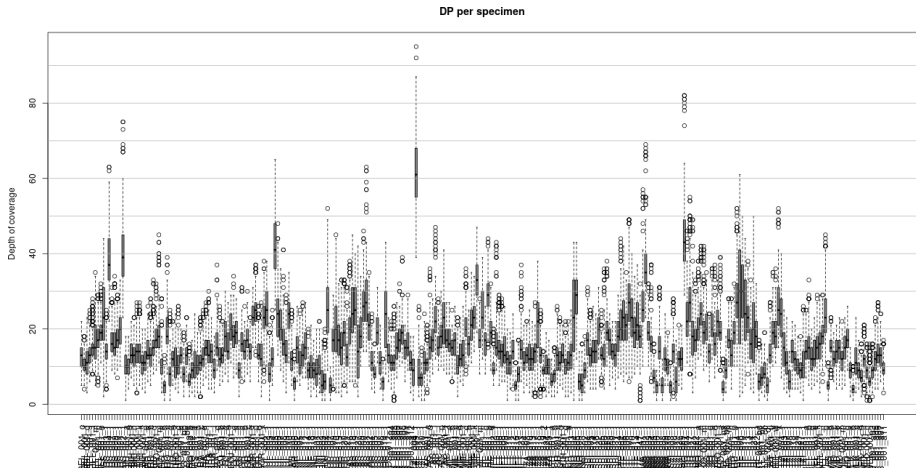
```
1 library(vcfR) # Required library
2 # Pick up downloaded file 'arabidopsis.vcf.gz' from the disc
3 arabidopsis.vcf <- read.vcfR(file=file.choose())
4 # It returns object of class vcfR-class
5 ?read.vcfR # See more import options
6 # Another option returns list of objects loci and data.frame
7 ?pegas::read.vcf
```

# Checking VCF I

```
1 arabidopsis.vcf
2 head(arabidopsis.vcf)
3 arabidopsis.vcf@fix[1:10,1:5]
4 # Get information about depth of coverage (DP)
5 # Se description of DP slot
6 strwrap(x=grep(pattern="ID=DP,", x=arabidopsis.vcf@meta, value=TRUE))
7 # Extract the DP
8 arabidopsis.vcf.dp <- extract.gt(x=arabidopsis.vcf, element="DP",
9   as.numeric=TRUE)
10 # See it
11 dim(arabidopsis.vcf.dp)
12 head(arabidopsis.vcf.dp)
13 # Boxplot of DP (next slide)
14 boxplot(x=arabidopsis.vcf.dp, col="#808080", ylab="Depth of coverage",
15   las=3)
16 title("DP per specimen")
17 abline(h=seq(from=0, to=90, by=10), col="#b3b3b3")
```



# DP per specimen



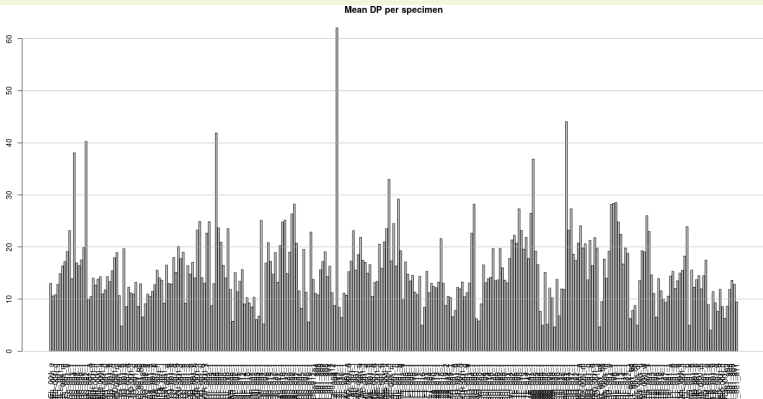
# Checking VCF II

## Bar plot of mean DP

```

1 barplot(apply(X=arabidopsis.vcf.dp, MARGIN=2, FUN=mean, na.rm=TRUE),
2         las=3)
3 title("Mean DP per specimen")
4 abline(h=seq(from=0, to=60, by=10), col="#b3b3b3")

```



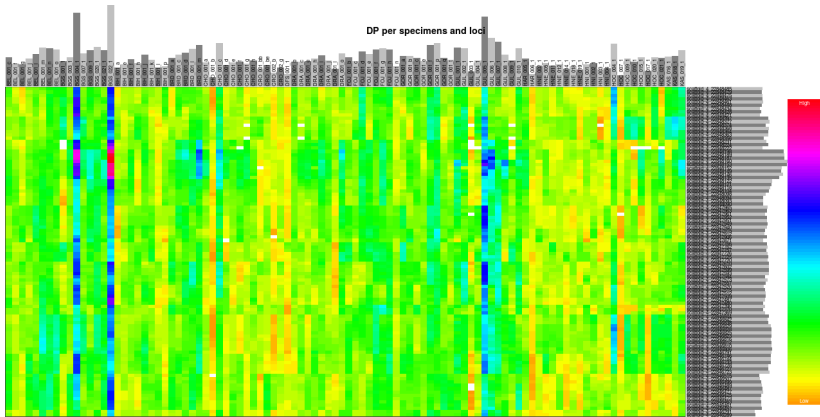
# Checking VCF III

## Heat map of DB

```

1 heatmap.bp(x=arabidopsis.vcf.dp[1:100,1:100], col.ramp=rainbow(n=100,
2   start=0.1)) # Subset - only first 100 loci and individuals
3 title("DP per specimens and loci")

```



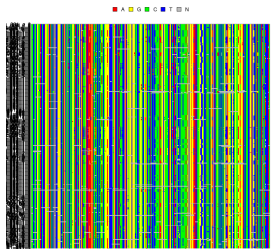
# Convert VCF into genind, genlight and loci

```
1 # Genind
2 arabidopsis.genind <- vcfR2genind(x=arabidopsis.vcf)
3 # Check it
4 arabidopsis.genind
5 nInd(arabidopsis.genind)
6 indNames(arabidopsis.genind)
7 nLoc(arabidopsis.genind)
8 locNames(arabidopsis.genind)
9 # Genlight (suitable for huge data, not required now)
10 # On Linux/Mac and with large data use higher n.cores
11 arabidopsis.genlight <- vcfR2genlight(x=arabidopsis.vcf, n.cores=1)
12 # Check it
13 arabidopsis.genlight
14 # Loci
15 arabidopsis.loci <- vcfR2loci(x=arabidopsis.vcf)
16 # Check it
17 arabidopsis.loci
18 print(x=arabidopsis.loci, details=TRUE)
```

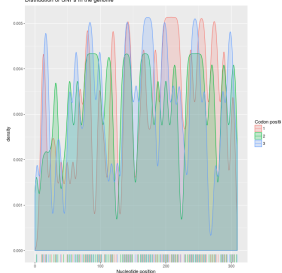
# Convert VCF into DNABin

```

1 # There are various options how
2 # to process variants in VCF
3 ?vcfR2DNABin
4 arabidopsis.dnabin <- vcfR2DNABin
5   (x=arabidopsis.vcf, consensus=
6   FALSE, extract.haps=TRUE,
7   unphased_as_NA=FALSE)
8 # Check it
9 arabidopsis.dnabin
10 dim(arabidopsis.dnabin)
11 as.character.DNABin
12   (arabidopsis.dnabin[1:15,1:12])
13 image.DNABin(arabidopsis.dnabin)
14 snpposi.plot.DNABin
15   (arabidopsis.dnabin)
16 snpposi.test.DNABin
17   (arabidopsis.dnabin)
    
```



Distribution of SNPs in the genome



# Remove non-biallelic loci

- This is commonly done with SNPs as many downstream analyses are well defined only for biallelic loci

```
1 # Create vector of loci to keep
2 arabidopsis.genind.keep <- nAll(arabidopsis.genind) < 3
3 # See the vector of loci
4 arabidopsis.genind.keep
5 # Keep only passing loci
6 arabidopsis.genind.bial <-
7   arabidopsis.genind[loc=arabidopsis.genind.keep]
8 # See result
9 arabidopsis.genind.bial
10 # Filtration of missing data
11 ?poppr::missingno # Various options...
```

# Export data

```
1 # Convert genind into DF using genind2df()
2 hauss.df <- genind2df(x=hauss.genind, pop=NULL, sep="/",
3   usepop=TRUE, oneColPerAll=FALSE)
4 # Save microsatellites to disk - check settings of write.table
5 write.table(x=hauss.df, file="haussdata.txt", quote=FALSE,
6   sep="\t", na="NA", dec=".", row.names=TRUE, col.names=TRUE)
7 # Export of DNA sequences into FASTA format
8 write.dna(x=usflu.dna, file="usflu.fasta", format="fasta",
9   append=FALSE, nbc=6)
10 seqinr::write.fasta(sequences=meles.dna, names=names(meles.dna),
11   file.out="meles.fasta", open="w")
12 # Export DNA sequences as NEXUS
13 write.nexus.data(x=meles.dna, file="meles.nexus", format="dna")
14 # Export VCF
15 write.vcf(x=arabidopsis.vcf, file="arabidopsis.vcf.gz")
16 # Export tree(s) (objects of class phylo)
17 write.tree(phy=hauss.nj.bruvo, file="haussknechtii.nwk") # In NEWICK
18 write.nexus(hauss.nj.bruvo, file="haussknechtii.nexus") # In NEXUS
```

# Import your data

## Tasks

- 1 Prior to import into R, **ensure your data are correct** — same decimal separator everywhere, consistent structure of CSV/TSV, no syntactic problems in FASTA/NEXUS/NEWICK, ...
- 2 Import some of your data into R
  - Be inspired by previous slides — edit commands to fit your needs and process your data
  - R is extremely flexible, but not everything is figured out within one minute...
- 3 **Check your data after import** to ensure they were correctly read
  - Upcoming chapters can serve like inspiration (not exhaustive) how to process your data in R, what is possible to do with them...
  - Previous examples are not covering all possibilities...



# Basic analysis

## 4 Basic analysis

First look at the data

Statistics

Genetic distances

Hierarchical clustering

AMOVA

MSN

NJ (and UPGMA) tree

PCoA

# 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 112
  - 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 129
  - **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 142
  - 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, for genetic data not so much anymore (following methods use to produce better results)
- **Neighbor-Joining (NJ)** — from slide 149
  - 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
  - Several methods try to improve it — slide 162

# Introductory overview of statistics and methods III

- **Principal Coordinates Analysis (PCoA)** — from slide 163
  - 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 148
  - 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 279
  - 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 the best one
- **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 the best one
- **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) **explaining 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  $\sim 10\text{--}25\%$  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  $\sim 100\text{--}1000$  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 of *Taraxacum haussknechtii*, you can try other data examples by yourselves

```
1 # Get summary - names and sizes of populations,  
2 # heterozygosity, some info about loci  
3 hauss.summ <- summary(hauss.genind)  
4 # Plot expected vs. observed heterozygosity  
5 # it looks like big difference  
6 plot(x=hauss.summ$Hexp, y=hauss.summ$Hobs,  
7      main="Observed vs expected heterozygosity",  
8      xlab="Expected heterozygosity", ylab="Observed heterozygosity")  
9 abline(0, 1, col="red")  
10 # Bartlett's K-squared test of difference  
11 # between observed and expected heterozygosity - not significant  
12 bartlett.test(list(hauss.summ$Hexp, hauss.summ$Hobs))  
13           Bartlett test of homogeneity of variances  
14 data:  list(hauss.summ$Hexp, hauss.summ$Hobs)  
15 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

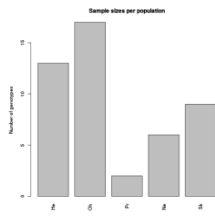
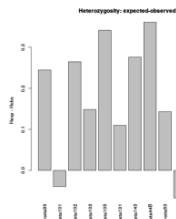
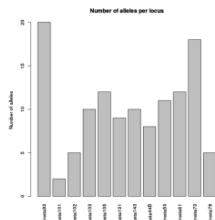
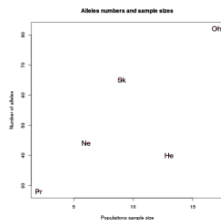
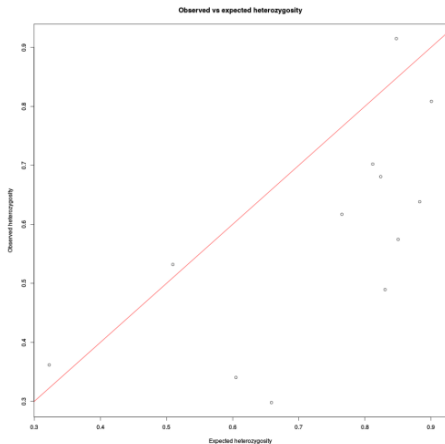
```
1 # T-test of difference between
2 # observed and expected heterozygosity - strongly significant
3 t.test(x=hauss.summ$Hexp, y=hauss.summ$Hobs, paired=TRUE, var.equal=T)
4           Paired t-test
5 data:  hauss.summ$Hexp and hauss.summ$Hobs
6 t = 3.5622, df = 11, p-value = 0.004456
7 alternative hypothesis: true difference in means is not equal to 0
8 95 percent confidence interval:
9  0.06114303 0.25887357
10 sample estimates:
11 mean of the differences
12      0.160083
```



# Descriptive statistics III

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

# Graphs from previous slides



# Population statistics by poppr()

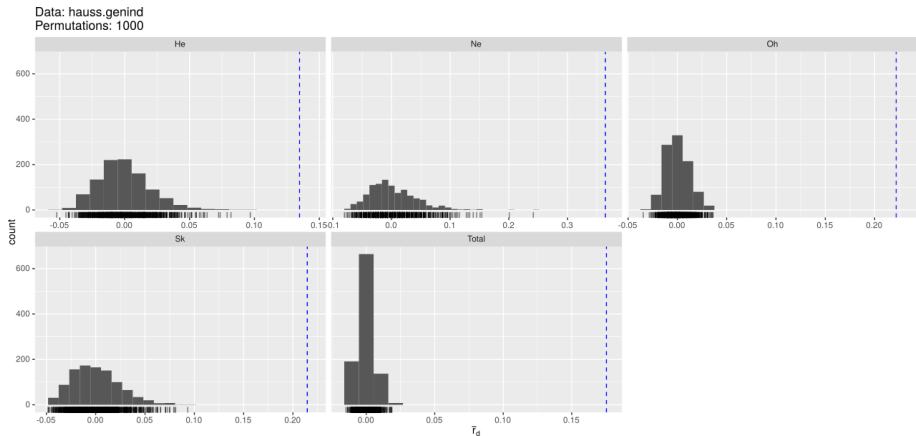
- poppr() is central function of poppr package calculating plenty of population genetic indices

```
1 ?poppr # See details
2 poppr(dat=hauss.genind, total=TRUE, sample=1000, method=4,
3   missing="geno", cutoff=0.15, quiet=FALSE, clonecorrect=FALSE,
4   plot=TRUE, index="rbarD", minsamp=1, legend=TRUE)
5 ... # Output table with indices:
6 Pop  N  MLG  eMLG      SE      H      G lambda  E.5  Hexp  Ia # More...
7 He  13  11  1.97  1.58e-01  2.352  9.94  0.899  0.941  0.503  1.42 ...
8 Ne   6   5  1.93  2.49e-01  1.561  4.50  0.778  0.930  0.604  3.44 ...
9 .....
10 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 $r_{barD}$ 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
- **H<sub>exp</sub>** — **Nei's gene diversity** (expected heterozygosity) — unbiased gene diversity (from 0 = no diversity to 1 = highest diversity)
- **I<sub>a</sub>** — 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

```
1 # According to loci
2 hauss.hwe.test <- hw.test(x=hauss.loci, B=1000)
3 hauss.hwe.test
4 # According to populations
5 # Separate genind object into list of genind objects for individual
6 # populations
7 hauss.pops <- seppop(hauss.genind)
8 hauss.pops
9 # Convert genind back to loci (list of loci objects according to
10 # populations)
11 hauss.pops.loci <- lapply(X=hauss.pops, FUN=genind2loci)
12 # Calculate the results per populations
13 lapply(X=hauss.pops.loci, FUN=hw.test, B=1000)
```

## Departure from HWE — results per locus

```
hauss.hwe.test
```

	chi <sup>2</sup>	df	Pr(chi <sup>2</sup> >)	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-equilibrium 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

```
1 # Fit, Fst and Fis for each locus
2 Fst(x=hauss.loci, pop=1)
3           Fit           Fst           Fis
4 msta93  0.31835291 0.17867087 0.17006829
5 msta101 -0.09968472 0.04064928 -0.14628018
6     ...           ...           ...
```

# F-statistics II

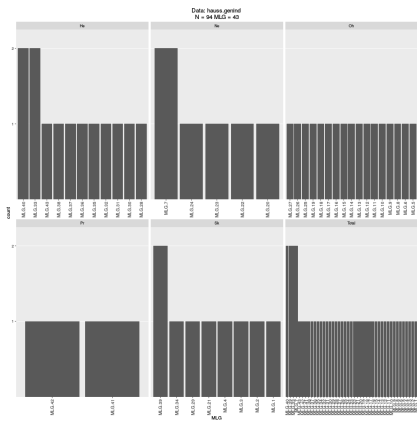
```
1 # Multilocus estimators of variance components and F-statistics,  
2 # alternative to Fst  
3 library(hierfstat)  
4 fstat(x=hauss.genind, pop=NULL, fstonly=FALSE)  
5           pop      Ind  
6 Total 0.1589501 0.2582641  
7 pop   0.0000000 0.1180834  
8 # Nei's pairwise Fst between all pairs of populations.  
9 # 0 = no structure; 1 = maximal difference  
10 pairwise.fst(x=hauss.genind, pop=NULL, res.type="matrix")  
11           He      Ne      Oh      Pr      Sk  
12 He 0.00000000 0.19960826 0.11391904 0.09404571 0.11184561  
13 Ne 0.19960826 0.00000000 0.07265306 0.19220430 0.10112859  
14 Oh 0.11391904 0.07265306 0.00000000 0.05302854 0.06287497  
15 Pr 0.09404571 0.19220430 0.05302854 0.00000000 0.10436469  
16 Sk 0.11184561 0.10112859 0.06287497 0.10436469 0.00000000
```

# F-statistics for mixed ploidy data

```
1 # stampFst requires population factor in genlight (here, population
2 # code consists of first three letters of individual's name)
3 indNames(arabidopsis.genlight)
4 pop(arabidopsis.genlight) <- substr(x=indNames(arabidopsis.genlight),
5   start=1, stop=3)
6 # Check it
7 pop(arabidopsis.genlight)
8 popNames(arabidopsis.genlight)
9 arabidopsis.fst <- StAMPP::stampFst(geno=arabidopsis.genlight,
10  nboots=100, percent=95, nclusters=1)
11 # For large data use higher nclusters to parallelize calculations
12 ?StAMPP::stampFst # See method details
13 arabidopsis.fst[["Fsts"]] # Matrix of Fst among populations
14 arabidopsis.fst[["Pvalues"]] # Matrix of P values
```

- Methods from StAMPP package (the same is the case for any method working somehow with distances) are sensitive to missing data...
- Populations must be already defined in the genlight object

# Multi locus genotypes

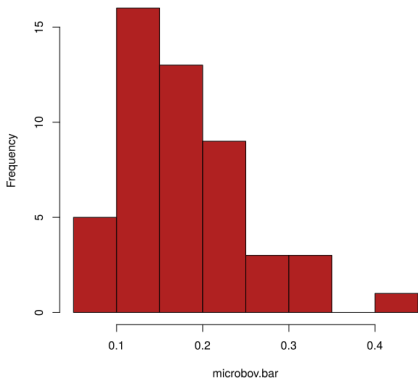


```
1 # Total number of MLGs
2 # (simple value)
3 mlg(gid=hauss.genind, quiet=FALSE)
4 # MLGs shared among populations
5 mlg.crosspop(gid=hauss.genind,
6   df=TRUE, quiet=FALSE)
7 # Detailed view on distribution
8 # of MLGs into populations
9 # (table and/or plot)
10 mlg.table(gid=hauss.genind,
11   bar=TRUE, total=TRUE,
12   quiet=FALSE)
13 mlg.vector(hauss.genind)
14 mlg.id(hauss.genind)
```

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

# Inbreeding

Average inbreeding in Salers cattles



```
1 # Load training data (cattle)
2 data(microbov)
3 # Separate populations of Salers
4 microbov.pops <- seppop(microbov)
5   [["Salers"]]
6 microbov.pops # See it
7 # Calculate the inbreeding
8 microbov.inbr <- inbreeding(x=
9   microbov.pops, N=100)
10 # Check for more settings
11 ?inbreeding
12 # population means for plotting
13 microbov.bar <- sapply(X=
14   microbov.inbr, FUN=mean)
15 # Plot it
16 hist(x=microbov.bar, col=
17   "firebrick", main="Average
18   inbreeding in Salers cattle")
```



# Basic distances

```
1 # See ?dist.gene for details about methods of this distance
2 hauss.dist.g <- dist.gene(x=hauss.genind@tab, method="pairwise")
3 # Euclidean distance for individuals (plain ordinary distance matrix)
4 hauss.dist <- dist(x=hauss.genind, method="euclidean", diag=T, upper=T)
5 hauss.dist
6 # Nei's distance (not Euclidean) for populations
7 # (other methods are available, see ?dist.genpop)
8 hauss.dist.pop <- dist.genpop(x=hauss.genpop, method=1, diag=T, upper=T)
9 # Test if it is Euclidean
10 is.euclid(hauss.dist.pop, plot=TRUE, print=TRUE, tol=1e-10) # FALSE = No
11 # Turn to be Euclidean
12 hauss.dist.pop <- cailliez(distmat=hauss.dist.pop, print=FALSE,
13   tol=1e-07, cor.zero=TRUE)
14 # Test if it is Euclidean
15 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, Pythagorean theorem is valid, etc.). If the distance matrix contains non-Euclidean distances, the result can be weird...

# Distances reflecting microsatellite repeats

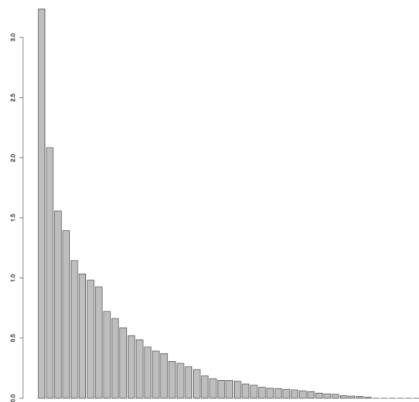
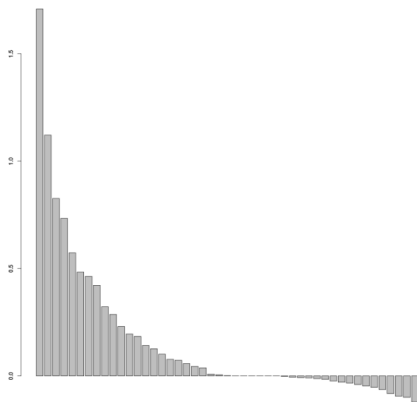
```
1 # Bruvo's distances weighting SSRs repeats - take care about replen
2 # parameter - requires repetition length for every SSRs locus
3 hauss.dist.bruvo <- bruvo.dist(pop=hauss.genind, replen=rep(2, 12),
4   loss=TRUE)
5 # Test if it is Euclidean
6 is.euclid(hauss.dist.bruvo, plot=TRUE, print=TRUE, tol=1e-10)
7 # Turn to be Euclidean
8 hauss.dist.bruvo <- cailliez(distmat=hauss.dist.bruvo, print=FALSE,
9   tol=1e-07, cor.zero=TRUE)
10 # Test if it is Euclidean
11 is.euclid(hauss.dist.bruvo, plot=TRUE, print=TRUE, tol=1e-10)
12 # Show it
13 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...

```
1 # Nei's distance (not Euclidean) for individuals
2 # (other methods are available, see ?nei.dist from poppr package)
3 hauss.dist.nei <- nei.dist(x=hauss.genind, warning=TRUE)
4 is.euclid(distmat=hauss.dist.nei, plot=TRUE, print=TRUE, tol=1e-10)
5 hauss.dist.nei
6 # Dissimilarity matrix returns a distance reflecting the number of
7 # allelic differences between two individuals
8 hauss.dist.diss <- diss.dist(x=hauss.genind, percent=FALSE, mat=TRUE)
9 is.euclid(as.dist(hauss.dist.diss), plot=TRUE, print=TRUE, tol=1e-10)
10 hauss.dist.diss
```

## Import own distance matrix from another software:

```
1   Fe      He      Oh      ...
2 Fe 0.00000 132.019 109.159 ...
3 He 132.0191 0.00000 9.89111 ...
4 Oh 109.1590 9.89111 0.00000 ...
5 Pr 139.5669 8.55312 4.40562 ...
6 Ne 156.7619 9.96143 16.6927 ...
7 ... ..      ...      ...      ...
```

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

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

Different distances available in package poppr

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 <sup>1</sup>	rogers.dist	—	Yes
Bruvo 2004	bruvo.dist	Step-wise Mutation	No

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

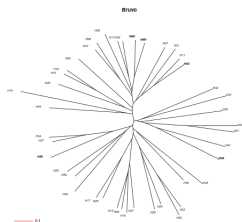
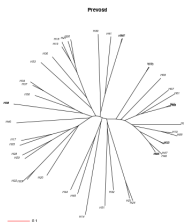
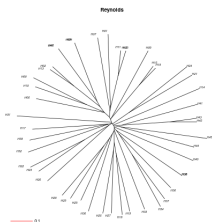
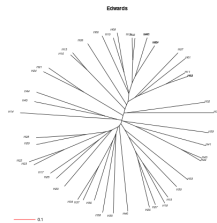
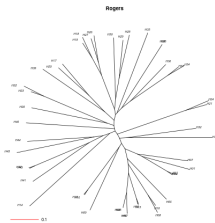
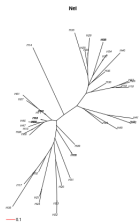
<sup>1</sup>Rogers (1972): Measures of genetic similarity and genetic distances. Pp. 145-153 of Studies in Genetics. University of Texas Publishers

# Comparison of different matrices

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

# Neighbor-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...

```
1 # There are various models available
2 ?dist.dna
3 # Create the distance matrix
4 usflu.dist <- dist.dna(x=usflu.dna, model="TN93")
5 # Check the resulting distance matrix
6 usflu.dist
7 class(usflu.dist)
8 dim(as.matrix(usflu.dist))
9 # Create another distance matrix
10 meles.dist <- dist.dna(x=meles.dna, model="F81")
11 # Check it
12 meles.dist
13 class(meles.dist)
14 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):

```
1 usflu.dists.l <- seploc(usflu.genlight, n.block=10, parallel=FALSE)
2 class(usflu.dists.l)
3 usflu.dists <- lapply(X=usflu.dists.l, FUN=function(DDD)
4   dist(as.matrix(DDD)))
5 class(usflu.dists)
6 names(usflu.dists)
7 class(usflu.dists[[1]])
8 usflu.distr <- Reduce(f="+", x=usflu.dists)
9 class(usflu.distr)
10 usflu.distr
11 # It is possible to use just basic dist function on whole
12 # genlight object (might require a lot of RAM)
13 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

# Distances in mixed-ploidy data sets I

```
1 library(StAMPP) # Load required library
2 # stampNeisD requires population factor in genlight
3 # Nei's 1972 distance between individuals
4 # (use pop=TRUE to calculate among populations)
5 arabidopsis.dist <- stampNeisD(geno=arabidopsis.genlight, pop=FALSE)
6 # Check it
7 head(arabidopsis.dist)
8 dim(arabidopsis.dist)
9 class(arabidopsis.dist)
10 # The same on population level
11 arabidopsis.dist.pop <- stampNeisD(geno=arabidopsis.genlight, pop=TRUE)
12 # Check it
13 head(arabidopsis.dist.pop)
14 dim(arabidopsis.dist.pop)
15 class(arabidopsis.dist.pop)
```

# Distances in mixed-ploidy data sets II

```
1 # Export the distance matrix as Phylip format for usage in
2 # external software (e.g. SplitsTree)
3 stampPhylip(distance.mat=arabidopsis.dist, file="arabidopsis_dist.txt")
4 # Genomic relationship matrix
5 ?stampGmatrix # Method details
6 arabidopsis.genomat <- stampGmatrix(geno=arabidopsis.genlight)
7 # Check it
8 head(arabidopsis.genomat)
9 dim(arabidopsis.genomat)
10 class(arabidopsis.genomat)
```

- If there are plenty of missing data and/or the distance is far from being Euclidean, it will not work very well... — sanitize missing data prior calculating distance (e.g. using `poppr::missingno`)
- Always check created distances

# Over 40 distances from philentropy package

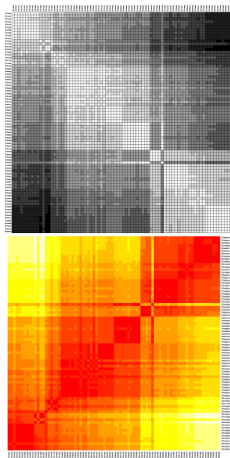
- There is enormous number of various distance measures...
- For example **Jaccard index** is used to compare binary (presence/absence) data like **AFLP**

```
1 library(philentropy) # Load the library
2 getDistMethods() # See available distances
3 ?distance # See details of distances
4 # Calculate e.g. Jaccard index for AFLP data
5 # amara.aflp has 30 columns, see dim(amara.aflp)
6 # column 1 contains names, see head(amara.aflp)
7 amara.jac <- distance(x=amara.aflp[,2:30], method="jaccard")
8 # See result
9 class(amara.jac)
10 amara.jac
11 # Make it distance matrix
12 amara.jac <- as.dist(m=amara.jac, diag=TRUE, upper=TRUE)
13 amara.jac
```

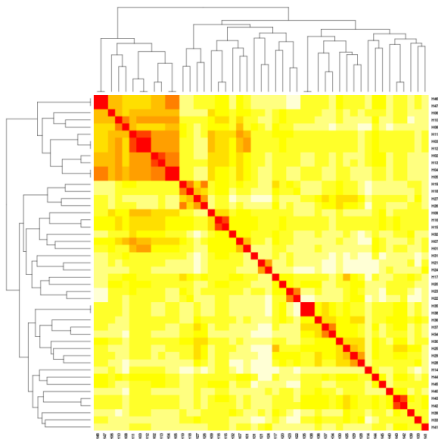
# Visualize pairwise genetic similarities

```
1 # table.paint() requires data
2 # frame, dist can't be directly
3 # converted to DF
4 table.paint(df=as.data.frame(
5   as.matrix(usflu.dist)), cleg=0,
6   clabel.row=0.5, clabel.col=0.5)
7 # Same visualization, colored
8 # heatmap() reorders values
9 # because by default it plots
10 # also dendrograms on the edges
11 heatmap(x=as.matrix(usflu.dist),
12   Rowv=NA, Colv=NA, symm=TRUE)
```

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



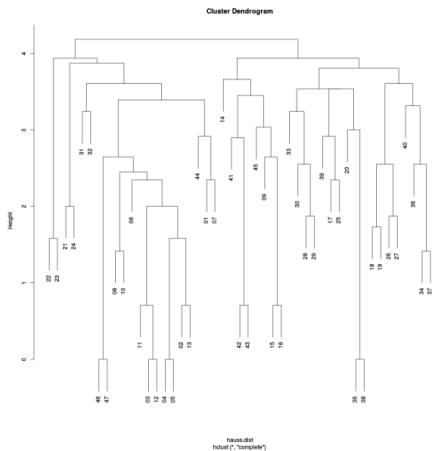
# Heat maps



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

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

# Hierarchical clustering — UPGMA and others



```
1 # According to distance used
2 # see ?hclust for methods
3 plot(hclust(d=hauss.dist,
4 method="complete"))
5 plot(hclust(d=hauss.dist.pop,
6 method="complete"))
7 plot(hclust(d=hauss.dist.
8 bruvo, method="complete"))
```

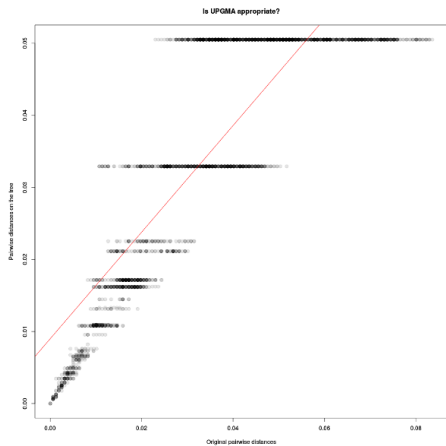
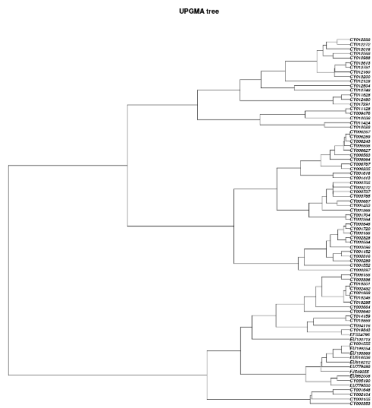
- This is very basic function to make dendrogram
- There are better possibilities (NJ etc — see slide 149 and onward)

# UPGMA and its test

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

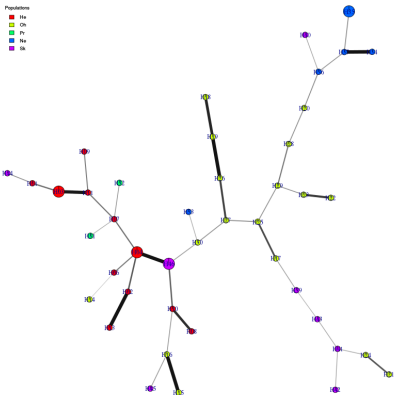
- Analysis of molecular variance tests if there are significant differences among populations (and/or another levels)
- Some implementations can partition variance into various levels
- `pegas::amova` returns a table of sums of square deviations (SSD), mean square deviations (MSD), and the number of degrees of freedom (df), and a vector of variance components (`sigma2`)
- See `sigma2` column for how much of the variance is on which level — percentage can be calculated as percentage of each level from total
- For more complicated hierarchy see `?poppr::poppr.amova`
- For mixed-ploidy dat sets see `?StAMPP::stampAmova`

## AMOVA II

```
1 hauss.pop <- pop(hauss.genind)
2 hauss.amova <- pegas::amova(hauss.dist~hauss.pop, data=NULL,
3   nperm=1000, is.squared=TRUE)
4 # See results
5 hauss.amova
6 ...
7           SSD      MSD df
8 hauss.pop 30.71923 7.679809 4
9 Error     119.58100 2.847167 42
10 Total     150.30023 3.267396 46
11 ...
12 Variance components:
13       sigma2 P.value
14 hauss.pop 0.55738      0 # From here we can calculate the percentage
15 Error     2.84717
16 ...
```

# Minimum Spanning Network

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

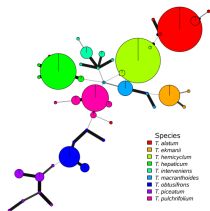


```

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

```

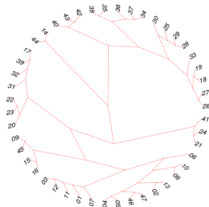
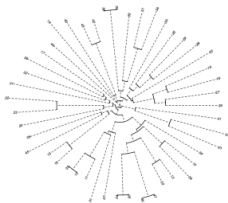
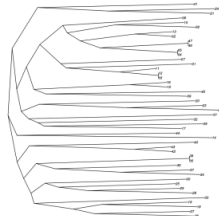
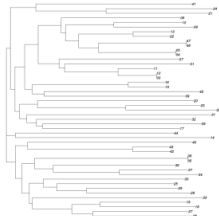
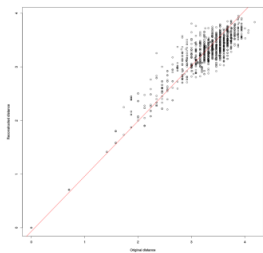
1 ?bruvo.msn # See details...



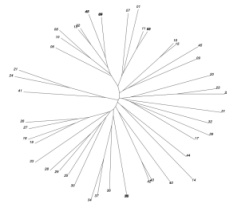
# Calculate and test NJ tree

```
1 # Calculates the tree (try with various distances)
2 hauss.nj <- nj(hauss.dist)
3 # Test tree quality - plot original vs. reconstructed distance
4 plot(as.vector(hauss.dist), as.vector(as.dist(cophenetic(hauss.nj))),
5      xlab="Original distance", ylab="Reconstructed distance")
6 abline(lm(as.vector(hauss.dist) ~
7          as.vector(as.dist(cophenetic(hauss.nj)))), col="red")
8 # Linear model for above graph
9 summary(lm(as.vector(hauss.dist) ~
10          as.vector(as.dist(cophenetic(hauss.nj)))) # Prints summary text
11 # Plot a basic tree - see ?plot.phylo for details
12 plot.phylo(x=hauss.nj, type="phylogram")
13 plot.phylo(x=hauss.nj, type="cladogram", edge.width=2)
14 plot.phylo(x=hauss.nj, type="fan", edge.width=2, edge.lty=2)
15 plot.phylo(x=hauss.nj, type="radial", edge.color="red",
16           edge.width=2, edge.lty=3, cex=2)
17 # There are enormous graphical possibilities...
```

# Choose your tree...



Neighbour-joining tree



# Bootstrap

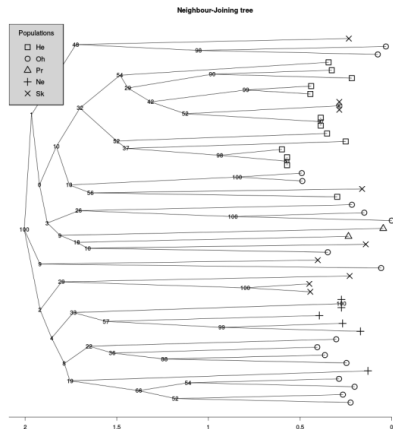
```
1 # boot.phylo() re-samples all columns - remove population column first
2 hauss.loci.nopop <- hauss.loci
3 hauss.loci.nopop[["population"]] <- NULL
4 # Calculate the bootstrap
5 hauss.boot <- boot.phylo(phy=hauss.nj, x=hauss.loci.nopop,
6   FUN=function(XXX) nj(dist(loci2genind(XXX))), B=1000)
7 # boot.phylo returns NUMBER of replicates - NO PERCENTAGE
8 # Plot the tree
9 plot.phylo(x=hauss.nj, type="unrooted", main="Neighbor-Joining tree")
10 # Labels for nodes - bootstrap - see ?nodelabels for graphical settings
11 nodelabels(text=round(hauss.boot/10))
12 ?boot.phylo # See details
13 # Another possibility
14 hauss.aboot <- aboot(x=hauss.genind, tree="nj", distance=nei.dist,
15   sample=100) # Bootstrap values are in slot node.label
16 # Plot the tree, explicitly display node labels
17 plot.phylo(x=hauss.aboot, show.node.label=TRUE)
18 ?aboot # Package poppr
```

# Nicer trees

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



## Choose your tree...



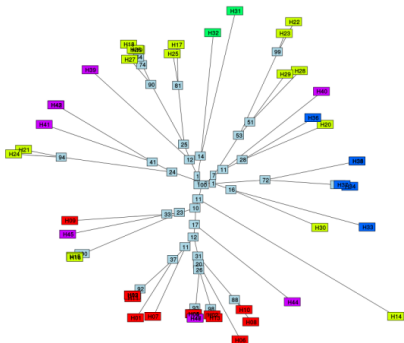
# Trees based on Bruvo's distance

Package poppr (bootstrap is incorporated within the function)

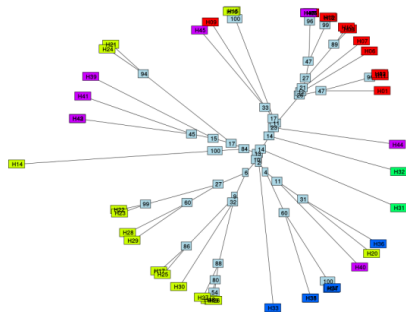
```
1 # NJ
2 hauss.nj.bruvo <- bruvo.boot(pop=hauss.genind, replen=rep(2, 12),
3   sample=1000, tree="nj", showtree=TRUE, cutoff=1, quiet=FALSE)
4 plot.phylo(x=hauss.nj.bruvo, type="unrooted", show.tip=FALSE,
5   edge.width=3, main="Neighbor-Joining tree")
6 # Call node labels as phylo$node.labels or phylo[["node.labels"]]
7 nodelabels(hauss.nj.bruvo[["node.labels"]])
8 tiplabels(hauss.nj.bruvo[["tip.label"]], bg=fac2col(x=hauss.genind$pop,
9   col.pal=nj.rainbow))
10 # UPGMA
11 hauss.upgma <- bruvo.boot(pop=hauss.genind, replen=rep(2, 12),
12   sample=1000, tree="upgma", showtree=TRUE, cutoff=1, quiet=FALSE)
13 plot.phylo(hauss.upgma, type="unrooted", show.tip=FALSE, edge.width=3,
14   main="UPGMA tree")
15 nodelabels(hauss.upgma[["node.labels"]])
16 tiplabels(hauss.upgma[["tip.label"]], bg=fac2col(x=hauss.genind@pop,
17   col.pal=nj.rainbow))
```

# Choose your tree...

Neighbor-Joining tree.



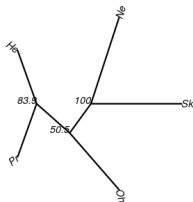
UPGMA tree



# NJ tree of populations

```
1 # about() can use distances implemented in poppr:  
2 ?poppr::about  
3 ?poppr::nei.dist  
4 # Calculations  
5 hauss.nj.pop <- about(x=hauss.genpop, tree="nj", distance="nei.dist",  
6   sample=1000, showtree=FALSE)  
7 print.phylo(hauss.nj.pop) # Information about result  
8 # Plot a tree  
9 plot.phylo(x=hauss.nj.pop, type="radial", show.node.label=TRUE,  
10  cex=1.2, edge.width=3, main="Neighbor-Joining tree of populations")
```

Neighbor-Joining tree of populations



# NJ tree based on DNA sequences

```
1 # Calculate the tree
2 usflu.tree <- nj(X=usflu.dist)
3 # Plot it
4 plot.phylo(x=usflu.tree, type="unrooted", show.tip=FALSE)
5 title("Unrooted NJ tree")
6 # Colored tips
7 usflu.pal <- colorRampPalette(topo.colors(length(levels(as.factor(
8   usflu.annot[["year"]])))))
9 # Tip labels
10 tiplabels(text=usflu.annot$year, bg=num2col(usflu.annot$year,
11   col.pal=usflu.pal), cex=0.75)
12 # Legend - describing years - pretty() automatically shows best
13 # values from given range, num2col() selects colors from color scale
14 legend(x="bottomright", fill=num2col(x=pretty(x=1993:2008, n=8),
15   col.pal=usflu.pal), leg=pretty(x=1993:2008, n=8), ncol=1)
```

# Root the tree

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

# Bootstrap rooted tree

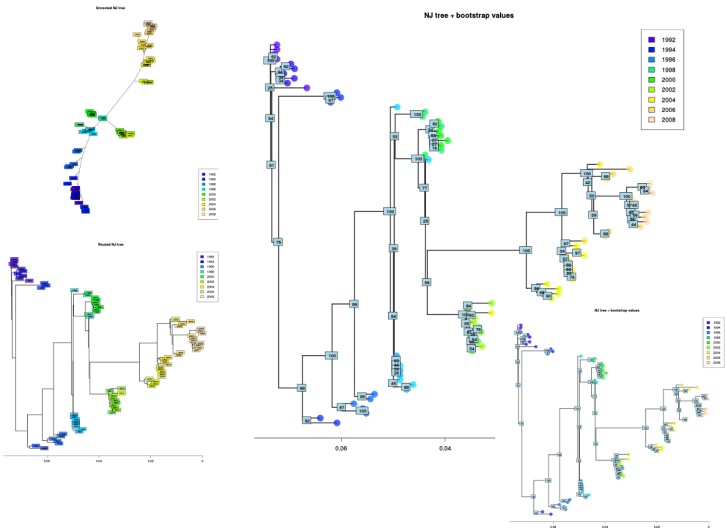
```
1 # Calculate it
2 usflu.boot <- boot.phylo(phy=usflu.tree.rooted, x=usflu.dna,
3   FUN=function(EEE) root(nj(dist.dna(EEE, model="TN93")),
4     outgroup=1), B=1000)
5 # Plot the tree
6 plot.phylo(x=usflu.tree.rooted, show.tip=FALSE, edge.width=2)
7 title("NJ tree + bootstrap values")
8 tiplabels(frame="none",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

# Collapse branches with low bootstrap support

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



# The trees



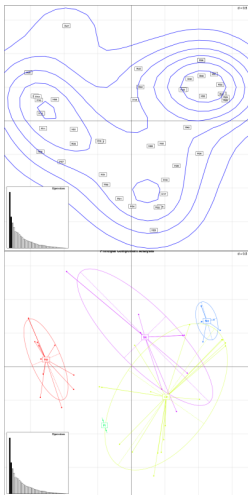
# NJ is death. Long live NJ!

- “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

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

## PCoA II

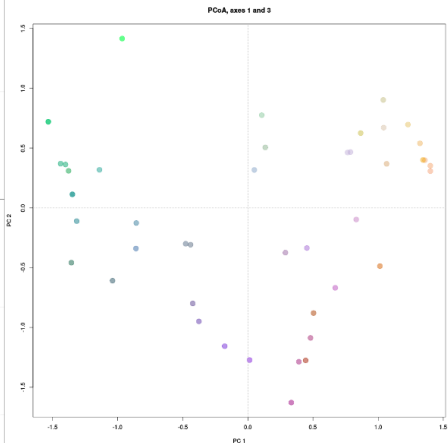
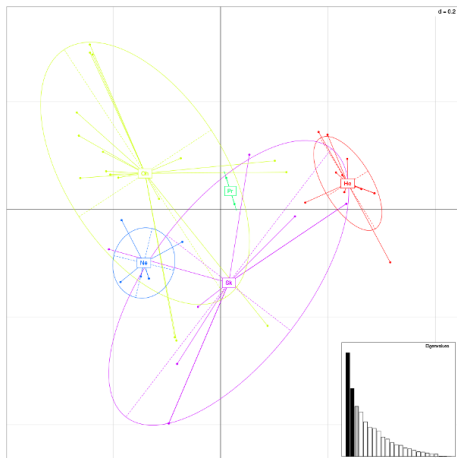


```

1 # Based on Bruvo's distance
2 hauss.pcoa.bruvo <- dudi.pco(d=
3   bruvo.dist(pop=hauss.genind,
4     replen=rep(2, 12)),
5     scannf=FALSE, nf=3)
6 s.class(dfxy=hauss.pcoa.bruvo$li,
7     fac=pop(hauss.genind),
8     col=hauss.pcoa.col)
9 add.scatter.eig(hauss.pcoa.bruvo$
10   eig, posi="bottomright", 3,1,2)
11 # Another possibility for colored
12 # plot (see ?colorplot for details)
13 colorplot(xy=hauss.pcoa$li[c(1,2)],
14   X=hauss.pcoa$li, transp=TRUE,
15   cex=3, xlab="PC 1", ylab="PC 2")
16 title(main="PCoA, axes 1 and 3")
17 abline(v=0, h=0, col="gray", lty=2)

```

# PCoA — Bruvo and colorplot



# Process more data

Not all combinations and possibilities were shown...

## Tasks

- 1 Most of examples were shown with the *Taraxacum haussknechtii* microsatellite dataset — try to do some analyses with another imported data
  - 2 Try some of the introduced analyses with your own custom imported data
- Of course, following chapters will show more possible analyses...
  - Previous examples are not covering all possibilities...
  - It is crucial to be able to edit the introduced commands to be able to handle your data
  - Check help pages of the functions for more options what to do with your data

# Discriminant Analysis of Principal components

## 5 DAPC

Bayesian clustering

Discriminant analysis and visualization

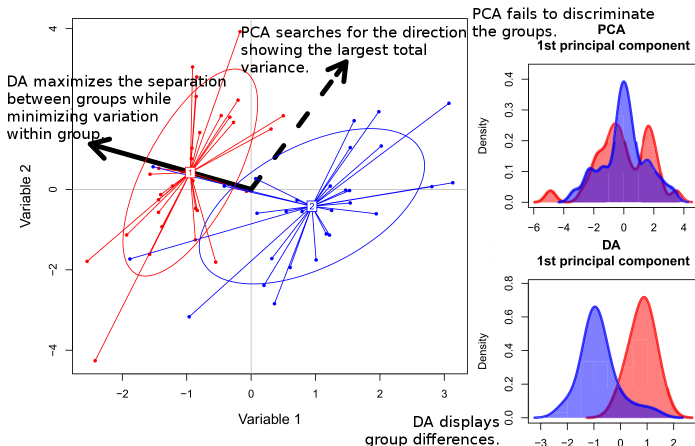
# DAPC

- 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

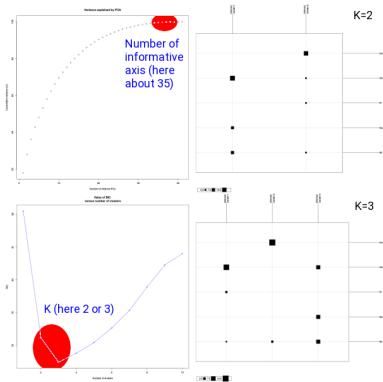
```

1 # Retain all informative PC (here about 35)
2 # According to second graph select best K (here 2 or 3)
3 # Now we select K=2 and later rerun the analysis for K=3 (lines 14-18)
4 hauss.kfind <- find.clusters(x=hauss.genind, stat="BIC",
5   choose.n.clust=TRUE, max.n.clust=10, n.iter=100000, n.start=100,
6   scale=FALSE, truenames=TRUE)
7 # See results as text
8 table(pop(hauss.genind), hauss.kfind$grp)
9 hauss.kfind
10 # Graph showing table of original and inferred populations and
11 # assignment of individuals
12 table.value(df=table(pop(hauss.genind), hauss.kfind$grp), col.lab=
13   paste("Inferred\ncluster", 1:length(hauss.kfind$size)), grid=TRUE)
14 # For K=3 - note parameters n.pca and n.clust - we just rerun the
15 # analysis and when results are stable, no problem here
16 hauss.kfind3 <- find.clusters(x=hauss.genind, n.pca=35, n.clust=3,
17   stat="BIC", choose.n.clust=FALSE, max.n.clust=10, n.iter=100000,
18   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



```

1 # See results as text
2 table(pop(hauss.genind),
3       hauss.kfind3$grp)
4 hauss.kfind3
5 # Graph showing table of original
6 # and inferred populations and
7 # assignment of individuals
8 table.value(
9   df=table(pop(hauss.
10  genind), hauss.kfind3$grp),
11  col.lab=paste("Inferred\n
12  cluster",
13  1:length(hauss.kfind3$size)),
14  grid=TRUE)
15 # If needed, use custom text for
16 # parameter col.lab=c("...", "...")
17 # as many labels as inferred groups

```

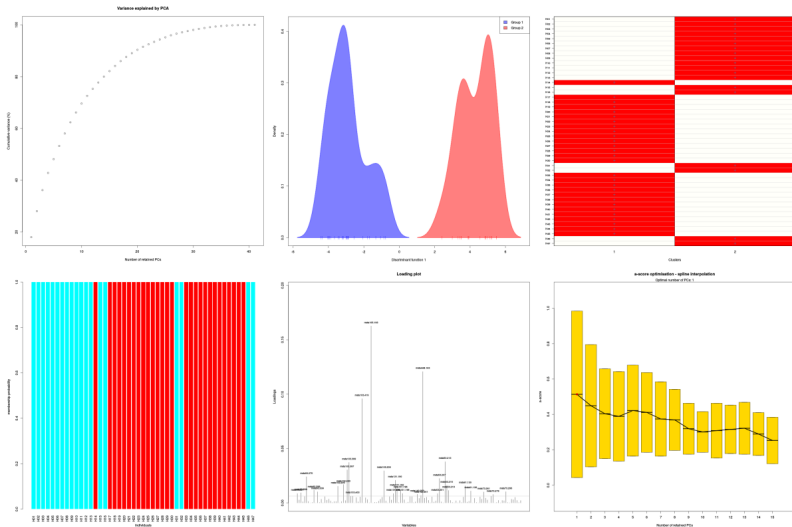
## DAPC code I

```

1 ## K=2
2 # Create DAPC
3 # Number of informative PC (Here 15, adegenet recommends < N/3). Select
4 # number of informative DA (here only one is available - no PCA graph)
5 hauss.dapc <- dapc(x=hauss.genind, pop=hauss.kfind$grp, center=TRUE,
6   scale=FALSE, var.contrib=TRUE, pca.info=TRUE, truenames=TRUE)
7 # Information
8 hauss.dapc
9 # Density function - only for first axis here!
10 scatter(x=hauss.dapc, xax=1, yax=1, main="DAPC", bg="white", solid=0.5,
11   leg=TRUE, txt.leg=c("Group 1", "Group 2"), posi.leg="topright")
12 # Assignment of individuals to clusters
13 assignplot(x=hauss.dapc)
14 # Structure-like plot
15 compoplot(x=hauss.dapc, xlab="Individuals", leg=FALSE)
16 # Loadingplot - alleles the most adding to separation of individuals
17 loadingplot(x=hauss.dapc$var.contr)

```

## DAPC for K=2



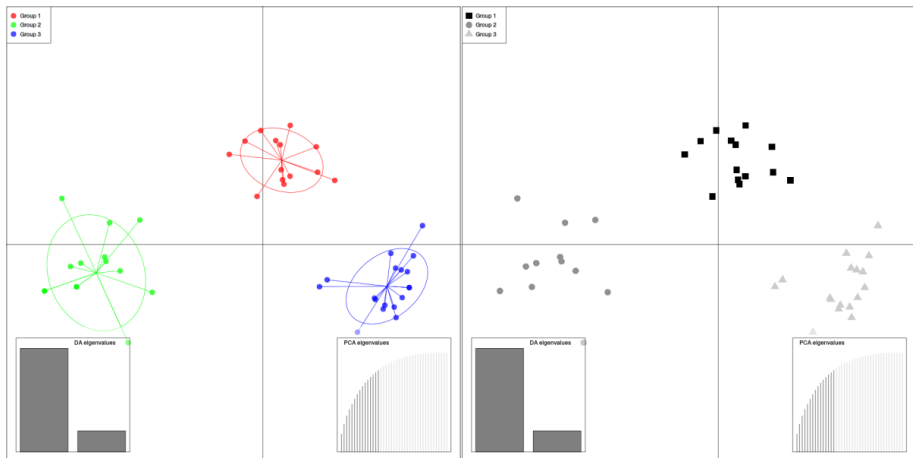
## DAPC code II

```

1 # alfa-score - according to number of PC axis
2 optim.a.score(x=hauss.dapc)
3 ## K=3
4 # Create DAPC
5 # Number of informative PC (Here 15, adegenet recommends < N/3)
6 # Select number of informative DA (here 2 - usually keep all of them)
7 hauss.dapc3 <- dapc(x=hauss.genind, pop=hauss.kfind3$grp, center=TRUE,
8   scale=FALSE, var.contrib=TRUE, pca.info=TRUE, truenames=TRUE)
9 # Information
10 hauss.dapc
11 # A la PCA graph
12 scatter(x=hauss.dapc3, main="DAPC, Taraxacum haussknechtii",
13   bg="white", cex=3, clab=0, col=rainbow(3), posi.da="bottomleft",
14   scree.pca=TRUE, posi.pca="bottomright", leg=TRUE,
15   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  $K=3$ 

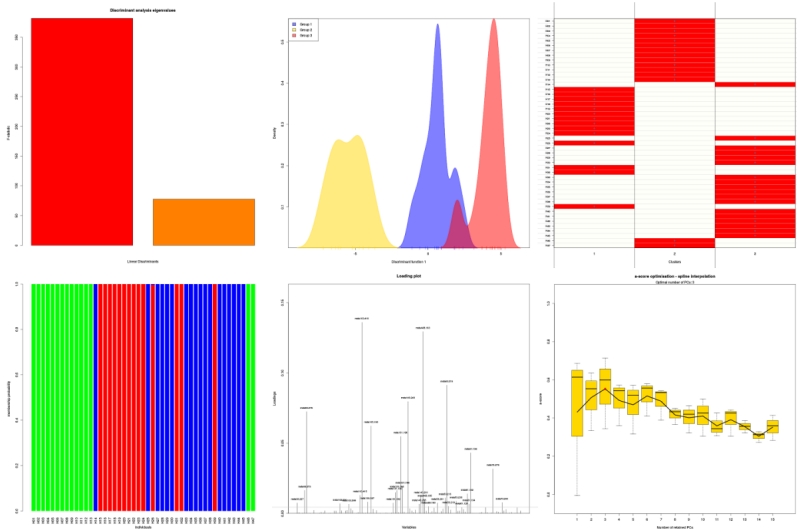
## DAPC code III

```

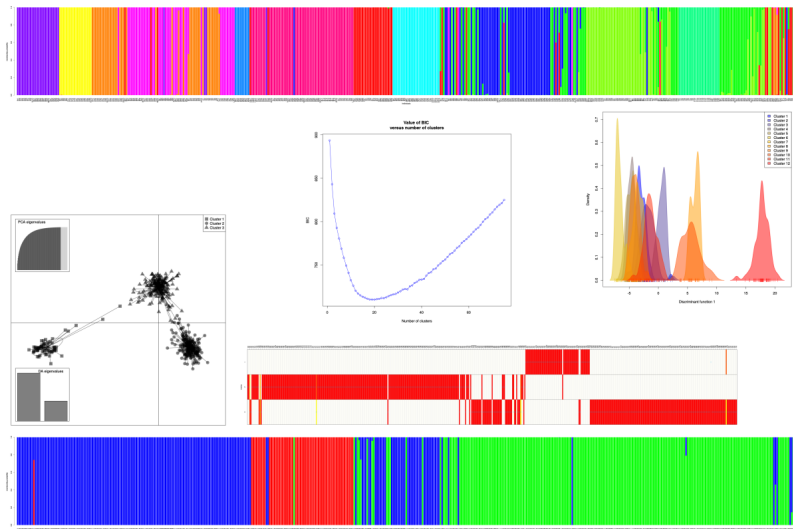
1 # Same in BW
2 scatter(x=hauss.dapc3, main="DAPC, Taraxacum haussknechtii",
3         bg="white", pch=c(15:17), cell=0, cstar=0, solid=1, cex=2.5, clab=0,
4         col=grey.colors(3, start=0, end=0.8, gamma=2, alpha=0), posi.da=
5         "bottomleft", scree.pca=TRUE, posi.pca="bottomright", leg=TRUE,
6         txt.leg=c("Group 1", "Group 2", "Group 3"), posi.leg="topleft")
7 # Density function - only for first axis here!
8 scatter(x=hauss.dapc3, xax=1, yax=1, main="DAPC", bg="white", solid=0.5,
9         leg=T, txt.leg=c("Group 1", "Group 2", "Group 3"), posi.leg="topleft")
10 # Assignment of individuals to clusters
11 assignplot(hauss.dapc3)
12 # Structure-like plot
13 compoplot(hauss.dapc3, xlab="Individuals", leg=FALSE)
14 # Loadingplot - alleles the most adding to separation of individuals
15 loadingplot(hauss.dapc3$var.contr)
16 # alfa-score - according to number of PC axis
17 optim.a.score(hauss.dapc3)

```



DAPC for  $K=3$ , extra information

# Another DAPC example



# Single Nucleotide Polymorphism

Special methods for large next-generation sequencing data

## ⑥ SNP PCA and NJ

# Special functions to work with huge SNP data sets

```

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

```

## Number of missing values in each locus

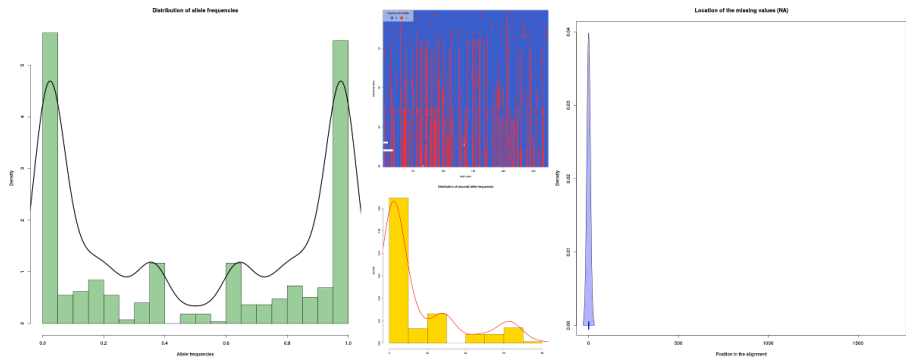
```

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

```

- Those tools are designed mainly for situation when having multiple (nearly) complete genomes — not needed for smaller data sets
- 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<sup>nd</sup> allele frequencies, missing data and number of 2<sup>nd</sup> allele, distribution of allele frequencies, and number of missing values in each locus



# PCA, NJ and genlight objects

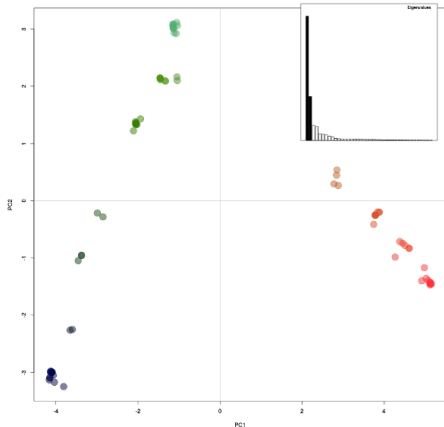
```

1 usflu.pca <- glPca(x=usflu.genlight, center=TRUE, scale=FALSE,
2   loadings=TRUE) # Select number of retained PC axes, about 10 here
3 # Plot PCA
4 scatter.glPca(x=usflu.pca, posi="bottomright")
5 title("PCA of the US influenza data")
6 # Colored plot
7 colorplot(usflu.pca$scores, usflu.pca$scores, transp=TRUE, cex=4)
8 title("PCA of the US influenza data")
9 abline(h=0, v=0, col="gray")
10 add.scatter.eig(usflu.pca[["eig"]][1:40], 2, 1, 2, posi="topright",
11   inset=0.05, ratio=0.3)
12 # Calculate phylogenetic tree
13 usflu.tree.genlight <- nj(dist(as.matrix(usflu.genlight)))
14 # Plot colored phylogenetic tree
15 plot.phylo(x=usflu.tree.genlight, type="fan", show.tip=FALSE)
16 tiplabels(pch=20, col=num2col(usflu.annot[["year"]],
17   col.pal=usflu.pal), cex=4)
18 title("NJ tree of the US influenza data")

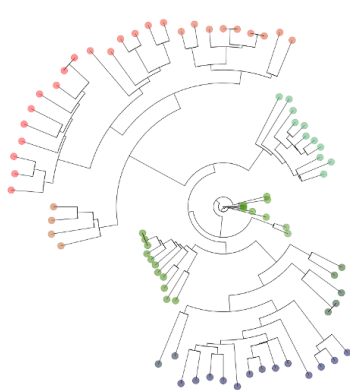
```

# PCA, NJ and genlight objects

PCA of the US influenza data



NJ tree of the US influenza data





# Spatial analysis and genetic data

Correlation of genes and space, spatial structure of genotypes

## 7 Spatial analysis

Moran's I

sPCA

Monmonier

Mantel test

Geneland

Plotting maps

# Short overview of spatial genetics (in R)

## Basic approaches

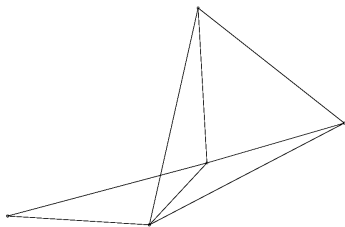
- Moran's  $I$  — several implementations (here as basic autocorrelation index, in sPCA and in Monmonier's algorithm), generally it is autocorrelation coefficient with broader use
- Mantel test — several implementations, popular, although recently criticized as biologically irrelevant, generally correlation 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

```

1 # Load required library
2 library(spdep)
3 # Creates connection network
4 hauss.connectivity <- chooseCN(xy=hauss.genind$other$xy, type=5,
5   d1=0, d2=1, plot.nb=TRUE, result.type="listw", edit.nb=FALSE)
6 hauss.connectivity
7 # Test of Moran's I for 1st PCoA axis
8 # Results can be checked against permuted values of moran.mc()
9 moran.test(x=hauss.pcoa[["li"]][,1], listw=hauss.connectivity,
10  alternative="greater", randomisation=TRUE)
11 Moran's I test under randomisation
12 data:  hauss.pcoa$li[, 1]
13 weights: hauss.connectivity
14 Moran I statistic standard deviate = -18.514, p-value = 1
15 alternative hypothesis: greater
16 sample estimates:
17 Moran I statistic      Expectation      Variance
18   -0.5232003724      -0.0217391304      0.0007336276

```

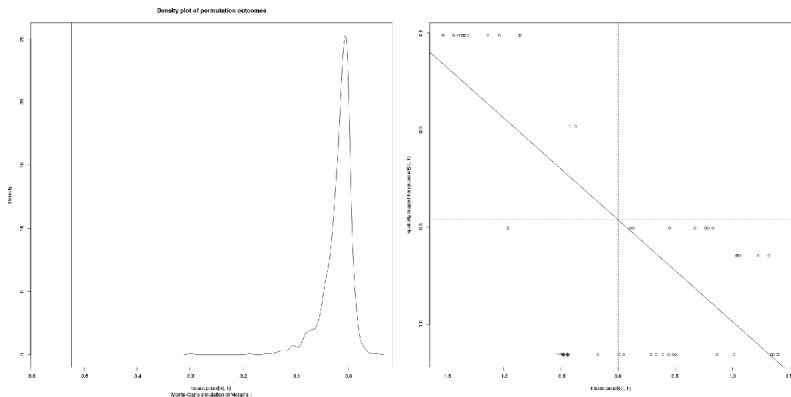
# Calculation of Moran's I

```

1 # Test of Moran's I for 1st PCoA axis
2 hauss.pcoa1.mctest <- moran.mc(x=hauss.pcoa$li[,1],
3   listw=hauss.connectivity, alternative="greater", nsim=1000)
4 hauss.pcoa1.mctest
5 # Output:
6 Monte-Carlo simulation of Moran I
7 data: hauss.pcoa$li[, 1]
8 weights: hauss.connectivity
9 number of simulations + 1: 1001
10 statistic = -0.5163, observed rank = 1, p-value = 0.999
11 alternative hypothesis: greater
12 # Plot the results
13 plot(hauss.pcoa1.mctest) # Plot of density of permutations
14 moran.plot(x=hauss.pcoa$li[,1], listw=hauss.connectivity) # PC plot

```

# Moran's I for our 1<sup>st</sup> 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<sup>nd</sup> axis)

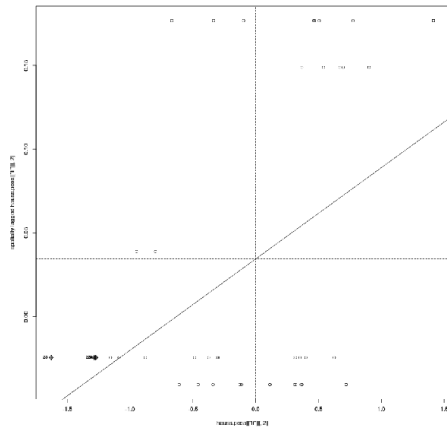
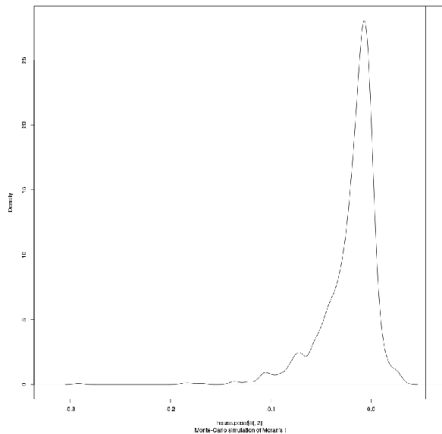
```

1 # Test of Moran's I for 2nd PCoA axis
2 moran.test(x=hauss.pcoa$li[,2], listw=hauss.connectivity,
3   alternative="greater", randomisation=TRUE)
4 hauss.pcoa2.mctest <- moran.mc(x=hauss.pcoa$li[,2],
5   listw=hauss.connectivity, alternative="greater", nsim=1000)
6 hauss.pcoa2.mctest
7 # Output
8 Monte-Carlo simulation of Moran's I
9 data: hauss.pcoa$li[, 2]
10 weights: hauss.connectivity
11 number of simulations + 1: 1001
12 statistic = 0.0545, observed rank = 1001, p-value = 0.000999
13 alternative hypothesis: greater
14 # Plot the results
15 plot(hauss.pcoa2.mctest) # Plot of density of permutations
16 moran.plot(x=hauss.pcoa$[,2], listw=hauss.connectivity) # PC plot

```

# Second axis is surprisingly significant

Density plot of permutation outcomes



- 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

```

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

```

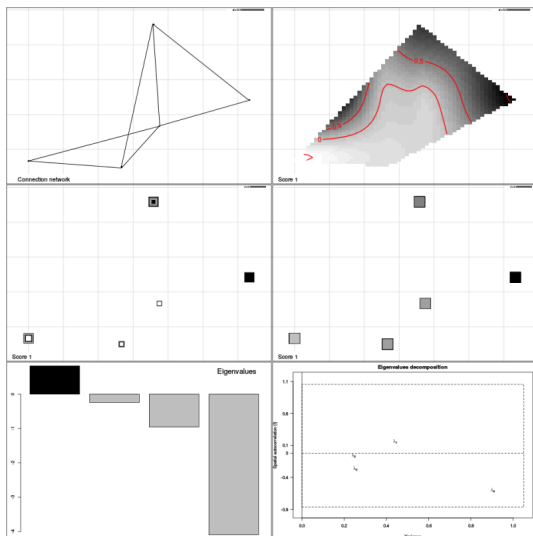
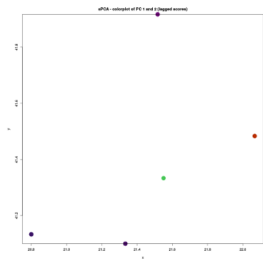
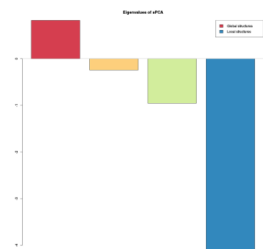
# Calculations of sPCA

```

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

```

# sPCA outputs I



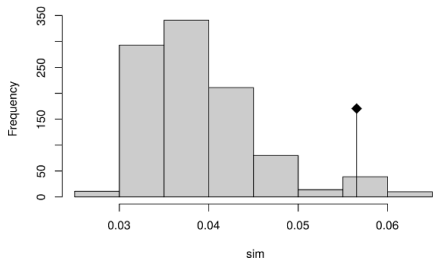
# Test if global/local structure is significant

```

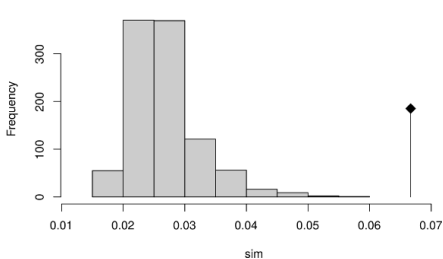
1 hauss.spca.glo <- global.rtest(X=hauss.genind$tab, listw=hauss.spca$lw,
2   nperm=999)
3 hauss.spca.glo
4 plot(hauss.spca.glo)
5 hauss.spca.loc <- local.rtest(X=hauss.genind$tab, listw=hauss.spca$lw,
6   nperm=999)
7 hauss.spca.loc
8 plot(hauss.spca.loc)

```

Histogram of sim



Histogram of sim



# Map of genetic clines

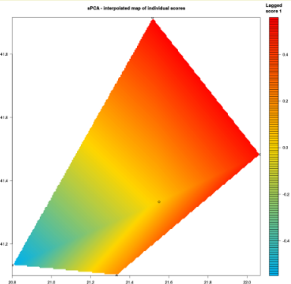
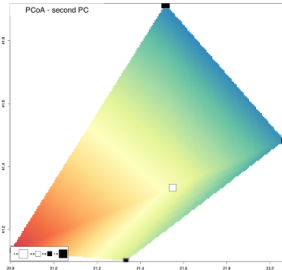
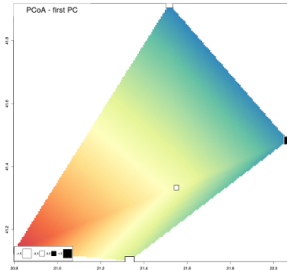
```
1 library(akima) # It is needed for manipulation with coordinates
2 # Transform the coordinates
3 hauss.spca.temp <- interp(other(hauss.genind)$xy[,1],
4   other(hauss.genind)$xy[,2], hauss.spca$ls[,1],
5   xo=seq(min(other(hauss.genind)$xy[,1]),
6   max(other(hauss.genind)$xy[,1]), le=200),
7   yo=seq(min(other(hauss.genind)$xy[,2]),
8   max(other(hauss.genind)$xy[,2]), le=200), duplicate="median")
9 # For 1st axis
10 image(x=hauss.spca.temp, col=spectral(100))
11 s.value(dfxy=hauss.genind$other$xy, z=hauss.pcoa$li[,1],
12   add.p=TRUE, csize=0.5, sub="PCoA - first PC", csub=2,
13   possub="topleft")
14 # For 2nd axis
15 image(x=hauss.spca.temp, col=spectral(100))
16 s.value(dfxy=hauss.genind$other$xy, z=hauss.pcoa[["li"]][,2],
17   add.p=TRUE, csize=0.5, sub="PCoA - second PC", csub=2,
18   possub="topleft")
```

# sPCA outputs II

```

1 # Interpolated lagged score on a map
2 hauss.spca.annot <- function() {
3   title("sPCA - interpolated map of individual scores")
4   points(other(hauss.genind)$xy[,1], other(hauss.genind)$xy[,2])
5 }
6 filled.contour(hauss.spca.temp, color.pal=colorRampPalette(
7   lightseason(100)), pch=20, nlevels=100, key.title=title("Lagged\n
8   score 1"), plot.title=hauss.spca.annot())

```

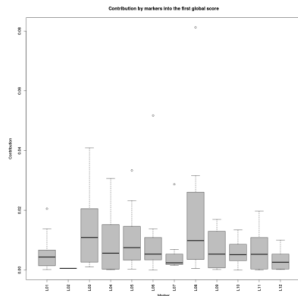
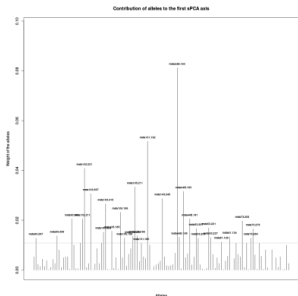


# Loading plots — which alleles contribute the most?

```

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

```



# 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**

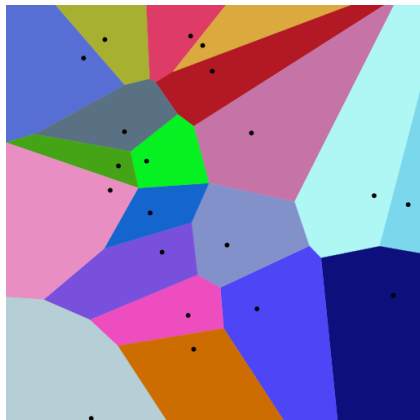
```

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

```



# Voronoi tessellation

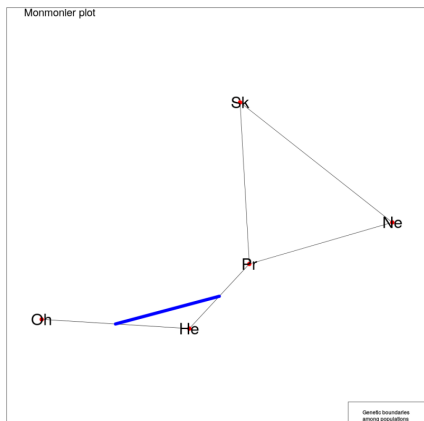


- In simplest case, all points have certain area and all points

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 and there are large distances among populations

# Plot genetic boundaries



```

1 plot.monmonier(hauss.monmonier,
2   add.arrows=FALSE, bwd=10,
3   sub="Monmonier plot", csub=2)
4 points(hauss.genpop$other$xy,
5   cex=2.5, pch=20, col="red")
6 text(x=hauss.genpop$other$xy$lon,
7   y=hauss.genpop$other$xy$lat,
8   labels=popNames(hauss.genpop),
9   cex=3)
10 legend("bottomright",
11   leg="Genetic boundaries\n
12   among populations")
13 # For plotting see
14 ?points
15 ?text
16 ?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`):

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

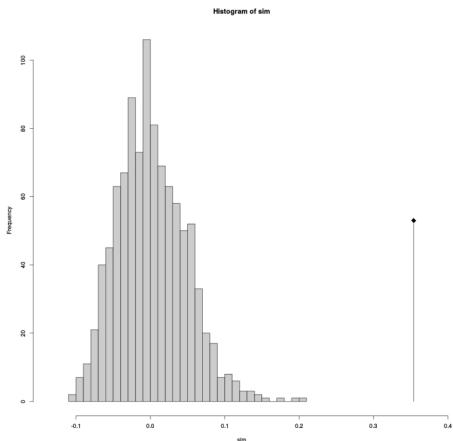
# Mantel test

- 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

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

# Mantel test outputs — strongly significant



```

1 hauss.mantel # See output
2 Monte-Carlo test
3 Call: mantel.randtest(m1 =
4   hauss.dist, m2 =
5   hauss.gdist, nrepet = 1000)
6 Observation: 0.35409
7 Based on 1000 replicates
8 Simulated p-value: 0.000999001
9 Alternative hypothesis: greater
10 Std.Obs Expectation Variance
11 7.61967545 0.001687140 0.0021389

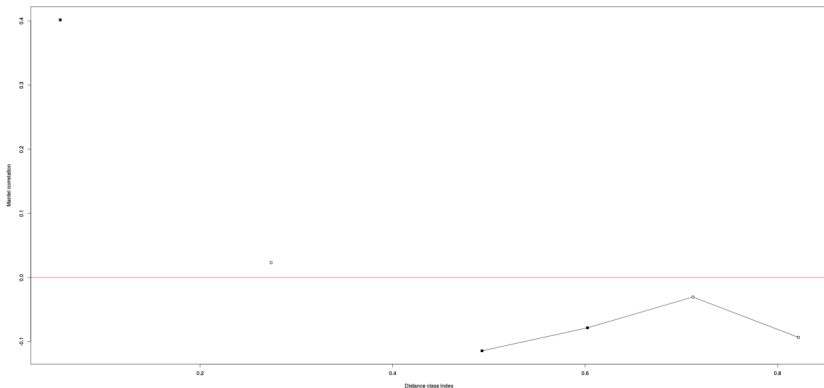
```

```

1 # Plot correlogram (next slide)
2 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

```

1 hauss.mantel.cor # See the text output:
2 Mantel Correlogram Analysis
3 Call:
4 mantel.correlog(D.eco = hauss.dist, D.geo = hauss.gdist, XY = NULL,
5   n.class = 0, break.pts = NULL, cutoff = FALSE, r.type = "pearson",
6   nperm = 1000, mult = "holm", progressive = TRUE)
7       class.index      n.dist Mantel.cor Pr(Mantel) Pr(corrected)
8 D.cl.1      0.054757 532.000000  0.409545   0.0010   0.000999 ***
9 D.cl.2      0.164271  0.000000      NA        NA        NA
10 D.cl.3      0.273784  52.000000  0.028055   0.2797   0.279720
11 D.cl.4      0.383298  0.000000      NA        NA        NA
12 D.cl.5      0.492812 466.000000 -0.097214   0.0160   0.031968 *
13 D.cl.6      0.602325  36.000000 -0.086288   0.0140   0.041958 *
14 D.cl.7      0.711839 108.000000 -0.044109   0.1568   0.313686
15 D.cl.8      0.821353 458.000000 -0.095780   0.0100   0.049950 *
16 ...
17 ---
18 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
    
```



# About Geneland

- For installation see slide 47
- Works with haploid and diploid co-dominant markers (microsatellites or SNPs)
- Spatially explicit Bayesian clustering
- Produces maps of distribution of inferred genetic clusters
- Relative complicated tool with various modeling options
- For more information see

<https://www2.imm.dtu.dk/~gigu/Geneland/>

```
1 # Load needed libraries
2 library(PBSmapping) # Required to transform coordinates
3 library(Geneland)
4 # Graphical interface is available, we will use only command line...
5 Geneland.GUI()
```



# Loading and conversions of coordinates

```
1 # Geneland requires specific coordinate space
2 # hauss.coord is DF, we need just plain matrix
3 hauss.geneland.coord <- as.matrix(hauss.coord)
4 colnames(hauss.geneland.coord) <- c("X", "Y")
5 attr(hauss.geneland.coord, "projection") <- "LL"
6 attr(hauss.geneland.coord, "zone") <- NA
7 hauss.geneland.coord.utm <- convUL(hauss.geneland.coord)
8 dim(hauss.geneland.coord)
9 hauss.geneland.coord
10 dim(hauss.geneland.coord.utm)
11 hauss.geneland.coord.utm # Final coordinates
12 # Load data (only haploid or diploid data are supported)
13 # only plain table with alleles
14 hauss.geneland.data <- read.table(file=
15   "https://soubory.trapa.cz/rcourse/haussknechtii_geneland.txt",
16   na.string="-999", header=FALSE, sep="\t")
17 dim(hauss.geneland.data)
18 hauss.geneland.data
```

# Before running MCMC

- Monte Carlo Markov Chains (MCMC) require usually millions of generations (iterations, `nit`) to find optimal solution
- Beginning ( $\sim 10\text{--}20\%$ ) 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<sup>th</sup> generation
- Denser sampling produces smoother data, but can consume too much disk space...

Directory structure for Geneland:

```
geneland/
├── 1
│   └── admixture
├── 2
│   └── admixture
├── 3
│   └── admixture
```

# Settings and running MCMC

```

1 hauss.geneland.nrun <- 5 # Set number of independent runs
2 hauss.geneland.burnin <- 100 # Set length of burnin chain
3 hauss.geneland.maxpop <- 10 # Set maximal K (number of populations)
4 # FOR loop will run several independent runs and produce output maps
5 # of genetic clusters - outputs are written into subdirectory within
6 # geneland directory (this has to exist prior launching analysis)
7 for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
8   hauss.geneland.path.mcmc <- paste("geneland/", hauss.geneland.irun,
9     "/", sep="") # paste is good especially for joining several texts
10  # On Windows, remove following line and create subdirectories from
11  # 1 to max K manually (creating subdirs in Windows in R is complicated)
12  system(paste("mkdir ", hauss.geneland.path.mcmc)) # Creates subdirs
13  # Inference - MCMC chain - see ?MCMC for details
14  MCMC(coordinates=hauss.geneland.coord.utm, geno.dip.codom=
15    hauss.geneland.data, path.mcmc=hauss.geneland.path.mcmc,
16    delta.coord=0.001, varnpop=TRUE, npopmin=1, npopmax=
17    hauss.geneland.maxpop, nit=10000, thinning=10,
18    freq.model="Uncorrelated", spatial=TRUE)
19  # For loop continues on next slide

```

# Running MCMC

```

1 # Start of FOR loop is on previous page
2 # In practice set much higher number of iterations (nit, millions),
3 # appropriate sampling (thinning, thousands) and longer burnin
4 # Post-process chains
5 PostProcessChain(coordinates=hauss.geneland.coord.utm,
6   path.mcmc=hauss.geneland.path.mcmc, nxdom=500, nydom=500,
7   burnin=hauss.geneland.burnin)
8 # Output
9 # Simulated number of populations
10 Plotnpop(path.mcmc=hauss.geneland.path.mcmc, printit=TRUE,
11   file=paste(hauss.geneland.path.mcmc, "/geneland-number_of_clusters
12   .pdf", sep=""), format="pdf", burnin=hauss.geneland.burnin)
13 dev.off() # We must close graphical device manually
14 # Map of estimated population membership
15 PosteriorMode(coordinates=hauss.geneland.coord.utm,
16   path.mcmc=hauss.geneland.path.mcmc, printit=TRUE, format="pdf",
17   file=paste(hauss.geneland.path.mcmc, "/geneland-map.pdf", sep=""))
18 dev.off() # We must close graphical device manually
19 } # End of FOR loop from previous slide

```

# Estimate $F_{ST}$

```
1 # Prepare list to record values of Fst for all runs
2 hauss.geneland.fstat <- list()
3 # Estimate Fst
4 for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
5   hauss.geneland.path.mcmc <- paste("geneland/",
6   hauss.geneland.irun, "/", sep="")
7   # F-statistics - Fis and Fst
8   hauss.geneland.fstat[[hauss.geneland.irun]] <- Fstat.output(
9     coordinates=hauss.geneland.coord.utm,
10    genotypes=hauss.geneland.data,
11    burnin=hauss.geneland.burnin, ploidy=2,
12    path.mcmc=hauss.geneland.path.mcmc)
13 }
14 # Print Fst output
15 hauss.geneland.fstat
```

# MCMC inference under the admixture model

```

1 for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
2   hauss.geneland.path.mcmc <- paste("geneland/",
3     hauss.geneland.irun, "/", sep="")
4   hauss.geneland.path.mcmc.adm <- paste(hauss.geneland.path.mcmc,
5     "admixture", "/", sep="")
6   # On Windows, remove following line of code and create in each
7   # result directory (from 1 to max K) new subdirectory "admixture"
8   # (creating subdirs in Windows in R is complicated)
9   system(paste("mkdir ", hauss.geneland.path.mcmc.adm))
10  HZ(coordinates=hauss.geneland.coord.utm, geno.dip.codom=
11    hauss.geneland.data, path.mcmc.noadm=hauss.geneland.path.mcmc,
12    nit=10000, thinning=10,
13    path.mcmc.adm=hauss.geneland.path.mcmc.adm)
14 }

```

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



## Produce maps of respective inferred clusters

```

1 for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
2   hauss.geneland.path.mcmc <- paste("geneland/",
3     hauss.geneland.irun, "/", sep="")
4   # Maps - tessellations
5   PlotTessellation(coordinates=hauss.geneland.coord.utm,
6     path.mcmc=hauss.geneland.path.mcmc, printit=TRUE,
7     path=hauss.geneland.path.mcmc)
8   for (hauss.geneland.irun.img in 1:hauss.geneland.maxpop) {
9     dev.off() } # We must close graphical device manually
10 }

```

- 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)

# Estimate frequencies of null alleles

```
1 hauss.geneland.fna <- list()
2 for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
3   hauss.geneland.path.mcmc <- paste("geneland/", hauss.geneland.irun,
4     "/", sep="")
5   # Estimation
6   hauss.geneland.fna[[hauss.geneland.irun]] <-
7     EstimateFreqNA(path.mcmc=hauss.geneland.path.mcmc)
8 }
9 # See output
10 hauss.geneland.fna
```

- Each item of the `list` object `hauss.geneland.fna` (from 1 to number of runs) contains vector of estimated frequencies of null alleles for every locus

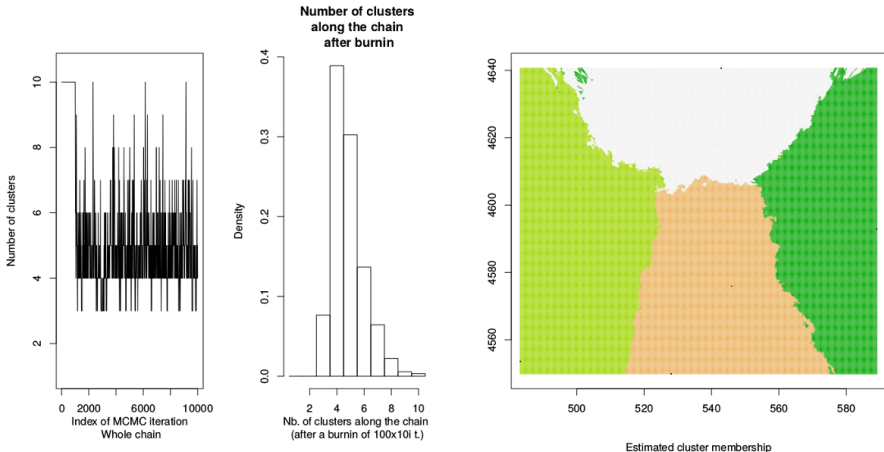
# Determine which run is the best

```
1 # Calculate average posterior probability
2 hauss.geneland.lpd <- rep(NA, hauss.geneland.nrun)
3 for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
4   hauss.geneland.path.mcmc <- paste("geneland/",
5     hauss.geneland.irun, "/", sep="")
6   hauss.geneland.path.lpd <- paste(hauss.geneland.path.mcmc,
7     "log.posterior.density.txt", sep="")
8   hauss.geneland.lpd[hauss.geneland.irun] <-
9     mean(scan(hauss.geneland.path.lpd)[-(1:hauss.geneland.burnin)]) }
10 # Sorts runs according to decreasing posterior probability
11 # the first one is the best
12 order(hauss.geneland.lpd, decreasing=TRUE)
13 [1] 5 1 4 3 2 # Run 5 is the best here
14 hauss.geneland.lpd # Here the runs are unsorted
15 [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

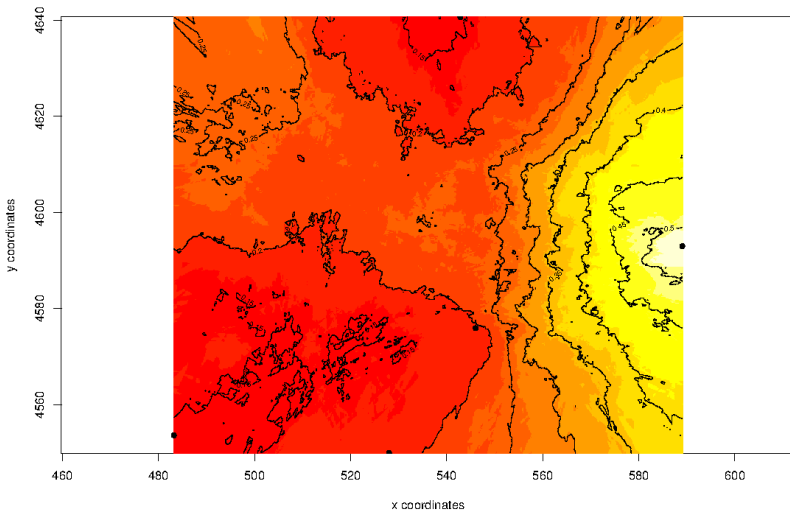
# MCMC chain, number of clusters and their map

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

Map of posterior probability to belong to cluster 1



## When using Geneland, remember...

- Within `MCMC()`, there must be at least hundreds thousands or millions of generations (`nit`) and appropriate sampling (thousands or higher, `thinning` — not to fill whole disk)
- To analyze geo-referenced data with a non-spatial prior set in `MCMC()` `spatial=FALSE`
- To analyze non-spatial data remove parameter `coordinates` from `MCMC()` function
- To obtain structure-like plots, file `proba.pop.membership.indiv.txt` in Geneland output directory can be used as input file for `distruct`
- To use SNPs, ATCG bases must be recoded as 1, 2, 3, 4, fixed alleles must be removed
- Geneland can handle only haploids and diploids (no ploidy mixing)
- When unsure, consult [manual](#)

# Very basic mapping in R

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

# Basic map and Google map

```

1 # Insert legend
2 legend(x="topright", inset=1/50,
3       legend=c("He", "Oh", "Pr", "Ne",
4               "Sk"), col="red", border="black",
5       pch=15:19, pt.cex=2, bty="o",
6       bg="lightgrey", box.lwd=1.5,
7       cex=1.5, title="Populations")
8 # Google map is produced into a
9 # file. Parameter markers contain
10 # data frame with coordinates and
11 # possibly with more information
12 hauss.gmap <- GetMap(center=
13   c(lat=41, lon=21), size=c(640,
14   640), destfile="gmap.png",
15   zoom=8, markers=hauss.coord,
16   maptype="terrain") # Plot map:
17 PlotOnStaticMap(MyMap=hauss.gmap)

```





# More options for Google map

```

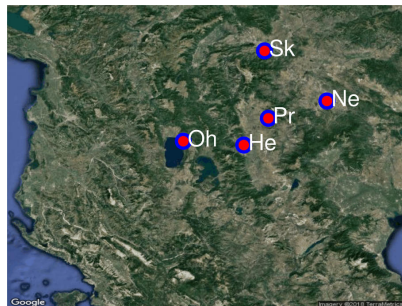
1 hauss.gmap2 <- GetMap(center=c(
2   lat=41, lon=21), size=c(640,
3   640), destfile="gmap2.png",
4   zoom=8, maptype="satellite")
5 PlotOnStaticMap(MyMap=hauss.gmap2,
6   lat=hauss.genpop@other$xy[["lat"
7   ]], lon=hauss.genpop@other$xy
8   [["lon"]], FUN=points, pch=19,
9   col="blue", cex=5)
10 PlotOnStaticMap(MyMap=hauss.gmap2,
11   lat=hauss.genpop@other$xy[["lat"
12   ]], lon=hauss.genpop@other$xy
13   [["lon"]], add=TRUE, FUN=points,
14   pch=19, col="red", cex=3)
15 PlotOnStaticMap(MyMap=hauss.gmap2,
16   lat=hauss.genpop@other$xy[["lat"
17   ]], lon=hauss.genpop@other$xy
18   [["lon"]], add=TRUE, FUN=text,

```

```

1 labels=as.vector(popNames(
2   hauss.genind)), pos=4, cex=3,
3   col="white")
4 # Google maps have their own
5 # internal scaling, adding of
6 # points by standard functions
7 # will not work correctly

```



# Adding pie charts to map I

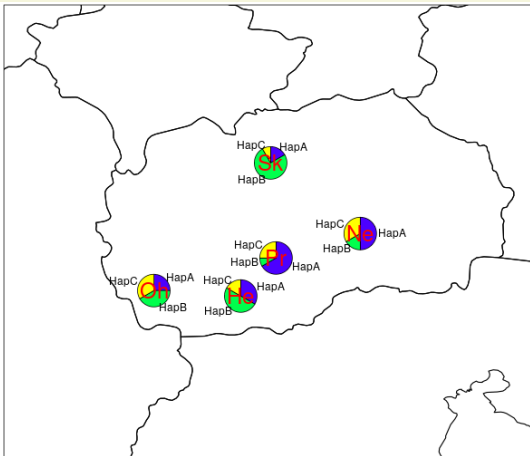
```
1 # Prepare matrix with some data (exemplary distribution of haplotypes)
2 hauss.pie <- cbind(c(20, 30, 15, 40, 10), c(30, 10, 25, 5, 45),
3   c(10, 20, 20, 15, 5))
4 # Add row names according to populations
5 rownames(hauss.pie) <- popNames(hauss.genpop)
6 # Add column names according to data displayed
7 colnames(hauss.pie) <- c("HapA", "HapB", "HapC")
8 class(hauss.pie) # Check it is matrix
9 hauss.pie # See resulting matrix
10 # Plot basic map with state boundaries within selected range
11 plot(x=getMap(resolution="high"), xlim=c(20, 23), ylim=c(41, 42),
12   asp=1, lwd=1.5)
13 box() # Add frame around the map
14 # Plot the pie charts
15 for (L in 1:5) { add.pie(z=hauss.pie[L,], x=as.vector(hauss.genpop@other
16   $xy[["lon"]])[L], y=as.vector(hauss.genpop@other$xy[["lat"]])[L],
17   labels=names(hauss.pie[L,]), radius=0.1, col=topo.colors(3)) }
18 ?add.pie # See more options
```

# Adding pie charts to map II

```

1 # Add population labels
2 text(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$xy[["lat"]],
3      labels=as.vector(popNames(hauss.genind)), col="red", cex=2)

```



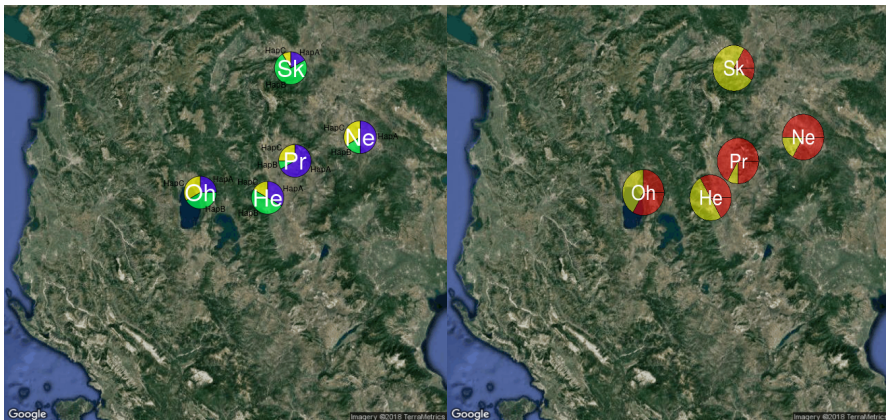
# Pie charts on Google map I

```
1 # Prepare list to store recalculated coordinates
2 hauss.gmap2.coord <- list()
3 # Calculation of coordinates to form required by Google Maps
4 for (LC in 1:5) { hauss.gmap2.coord[[LC]] <- LatLon2XY.centered(MyMap=
5   hauss.gmap2, lat=as.vector(hauss.genpop@other$xy[["lat"]])[LC],
6   lon=as.vector(hauss.genpop@other$xy[["lon"]])[LC], zoom=8) }
7 hauss.gmap2.coord # See result
8 # Plot plain map
9 PlotOnStaticMap(MyMap=hauss.gmap2)
10 # Plot pie charts
11 for (LP in 1:5) { add.pie(z=hauss.pie[LP,], x=hauss.gmap2.coord[[LP]]
12   $newX, y=hauss.gmap2.coord[[LP]]$newY, labels=names(hauss.pie[LP,]),
13   radius=25, col=topo.colors(n=3, alpha=0.7)) }
14 # Alternative option to plot pie charts
15 for (LF in 1:5) { plotrix::floating.pie(xpos=hauss.gmap2.coord[[LF]]
16   $newX, ypos=hauss.gmap2.coord[[LF]]$newY, x=hauss.pie[LF,], radius=30,
17   col=heat.colors(n=3, alpha=0.5) ) }
```

# Pie charts on Google map II

```
1 # Add population text labels
```

```
2 PlotOnStaticMap(MyMap=hauss.gmap2, lat=hauss.genpop@other$xy[["lat"]],
3 lon=hauss.genpop@other$xy[["lon"]], add=TRUE, FUN=text,
4 labels=as.vector(popNames(hauss.genind)), cex=3, col="white")
```



# OpenStreetMaps and data sets from mapproj

```
1 # Plot on OpenStreetMaps - server is commonly overloaded and does not
2 # respond correctly - in fact, it rarely works...
3 GetOsmMap(lonR=c(18, 24), latR=c(39, 44), scale=200000, destfile=
4   "osmmap.png", format="png", RETURNIMAGE=TRUE, GRAYSCALE=FALSE,
5   NEWMAP=TRUE, verbose=1)
6 # Plot on data sets from mapproj package
7 library(maps) # Various mapping tools (plotting, ...)
8 library(mapdata) # More detailed maps, but political boundaries often
9   # outdated, see https://CRAN.R-project.org/package=mapdata
10 library(mapproj) # Convert latitude/longitude into projected coordinates
11 # Plot a map, check parameters
12 # Check among others "projection" and ?mapproject for its details
13 map(database="worldHires", boundary=TRUE, interior=TRUE, fill=TRUE,
14   col="lightgrey", plot=TRUE, xlim=c(16, 27), ylim=c(37, 46))
15 # If you'd use projection, use mapproject to convert also coordinates!
16 # See ?mapproject for details
17 points(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$xy
18   [["lat"]], pch=15:19, col="red", cex=3)
```

# Plotting on SHP files I

- Get SHP files from <https://soubory.trapa.cz/rcourse/macedonia.zip> and unpack them into R working directory
- Data were originally downloaded from <https://download.geofabrik.de/europe/macedonia.html>
- R working directory has to contain also respective DBF and SHX files (same name, only different suffix)

```

1 library(maptools)
2 # Load SHP file
3 dir() # Verify required files are unpacked in the working directory
4 # There are several functions readShape* - select appropriate
5 # according to data stored in respective SHP file
6 # Check correct import by plotting all layers
7 macedonia_building <- readShapeLines(fn="macedonia_buildings.shp")
8 plot(macedonia_building) # Continues on next slide...

```

# Plotting on SHP files II

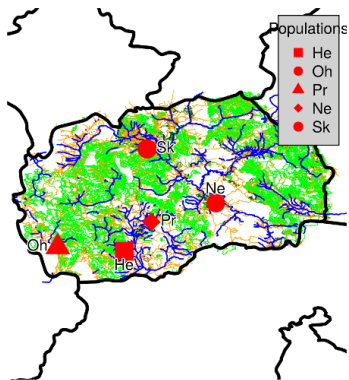
```
1 macedonia_landuse <- readShapeLines(fn="macedonia_landuse.shp")
2 plot(macedonia_landuse)
3 macedonia_natural <- readShapeLines(fn="macedonia_natural.shp")
4 plot(macedonia_natural)
5 macedonia_railways <- readShapeLines(fn="macedonia_railways.shp")
6 plot(macedonia_railways)
7 macedonia_roads <- readShapeLines(fn="macedonia_roads.shp")
8 plot(macedonia_roads)
9 macedonia_waterways <- readShapeLines(fn="macedonia_waterways.shp")
10 plot(macedonia_waterways)
11 # Plot all layers into single image, add more information
12 plot(macedonia_building)
13 plot(macedonia_landuse, add=TRUE, col="darkgreen", fill=TRUE)
14 plot(macedonia_natural, add=TRUE, col="green", fill=TRUE)
15 plot(macedonia_railways, add=TRUE, col="brown", lty="dotted")
16 plot(macedonia_roads, add=TRUE, col="orange")
17 plot(macedonia_waterways, add=TRUE, col="blue", lwd=2)
18 # Ends on next slide...
```



# Plotting on SHP files III

```
1 # Add state boundaries
2 plot(x=getMap(resolution="high"), xlim=c(19, 24), ylim=c(39, 44), asp=1,
3     lwd=5, add=TRUE) # Or e.g.
4 map(database="worldHires", boundary=TRUE, interior=TRUE, fill=FALSE,
5     col="red", add=TRUE, plot=TRUE, xlim=c(16, 27), ylim=c(37, 46), lwd=5)
6 # Add sampling points
7 points(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$
8     xy[["lat"]], pch=15:19, col="red", cex=4)
9 # Add description of sampling points
10 shadowtext(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$
11     xy[["lat"]], labels=as.vector(popNames(hauss.genind)), col="black",
12     bg="white", theta=seq(pi/4, 2*pi, length.out=8), r=0.15,
13     pos=c(1, 3, 2, 4, 4), offset=0.75, cex=1.5)
14 # Add legend
15 legend(x="topright", inset=1/50, legend=c("He", "Oh", "Pr", "Ne",
16     "Sk"), col="red", border="black", pch=15:19, pt.cex=2, bty="o",
17     bg="lightgrey", box.lwd=1.5, cex=1.5, title="Populations")
18 # Results on next slide...
```

# Plotting on SHP files IV and maps from mapproj



# Bayesian clustering with Structure

## 8 Structure

Running Structure from R  
ParallelStructure on Windows  
Post processing

# Structure I

- Population genetic software for Bayesian clustering, <https://web.stanford.edu/group/pritchardlab/structure.html>
- De facto standard for population genetics, one of the most used software tools
- 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, e.g. from K=1 to K=10 (grouping into 1, 2, 3, ..., 10 clusters)
  - User must run it repeatedly for each K to see if the result is stable (because of stochasticity of the computational algorithm), e.g. 10 times each
  - Finally there use to be hundreds of runs... (e.g. 100 like in this example)

# Structure II

- The best grouping into  $K$  groups is selected according to posterior probability of each run (different groupings have different probability of being correct, we need to select the best one) and stability of runs (independent runs for a given same  $K$  return same results)
- 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, depending on user's preference and knowledge)
- 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>
- Relatively complicated tool requiring plenty of work...

# 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 **distrupt**

# 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

```

1 # Prepare special new empty directory and set working directory
2 setwd("~/dokumenty/fakulta/vyuka/r_mol_data/examples/structure/")
3 install.packages("ParallelStructure",
4   repos="https://r-forge.r-project.org") # Install the package
5 library(ParallelStructure) # Load the library
6 # It takes more or less same parameter as normal Structure
7 ?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

```

1 parallel_structure(joblist="joblist.txt", n_cpu=3, structure_path=
2   "~/bin/", infile="hauss_stru.in", outpath="results/", numinds=47,
3   numloci=12, plot_output=1, label=1, popdata=1, popflag=1,
4   phenotypes=0, markernames=1, mapdist=0, onerowperind=0, phaseinfo=0,
5   extracol=0, missing=-9, ploidy=2, usepopinfo=0, revert_convert=1,
6   printqhat=1, locdata=0, recessivealleles=0, phased=0, noadmix=0,
7   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](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

```
1 # Install Rmpi library required by ParallelStructure for
2 # parallelisation on Windows (installation can be very problematic)
3 install.packages("Rmpi")
4 library(Rmpi)
5 # Instead of parallel_structure() use MPI_structure()
6 # with same arguments
7 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](#)

```
1 # Load the script
2 source("https://soubory.trapa.cz/rcourse/structure-sum-2011.r")
3 # Create new directory with result files results_job*_f and set
4 # working directory accordingly
5 setwd("/home/vojta/dokumenty/fakulta/vyuka/r_mol_data/examples/
6 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](https://soubory.trapa.cz/rcourse/list_k.txt)

```
2 results_job_S010_f
```

```
3 results_job_S011_f
```

```
... ..
```

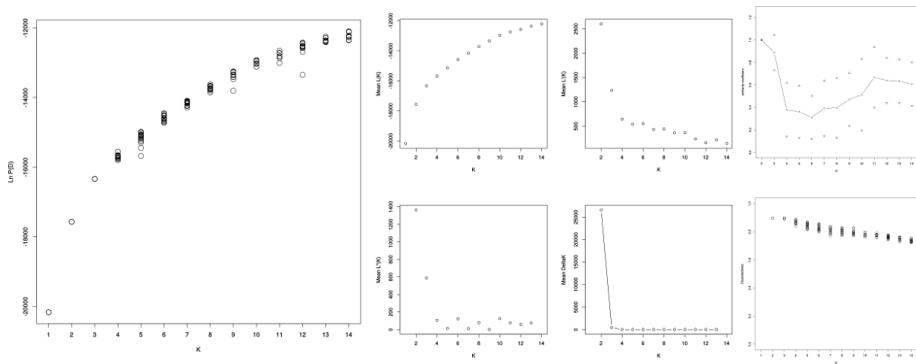
# Run Structure-sum

```
1 # See documentation for details. Functions take as an argument
2 # list_k file and number of populations
3 Structure.table("list_k.txt", 5)
4 Structure.simil("list_k.txt", 5)
5 Structure.deltaK("list_k.txt", 5)
6 graphics.off() # Close graphics
7 Structure.cluster("list_k.txt", 5)
8 # Reordering ("alignment") of runs to get same clusters in same
9 # columns (prepare respective list_k files - one for each K)
10 # Preparing data for CLUMPP
11 Structure.order("list_k_02.txt", 5)
12 Structure.order("list_k_03.txt", 5)
13 Structure.order("list_k_04.txt", 5)
14 Structure.order("list_k_05.txt", 5)
15 Structure.order("list_k_06.txt", 5)
16 Structure.order("list_k_07.txt", 5)
17 # 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

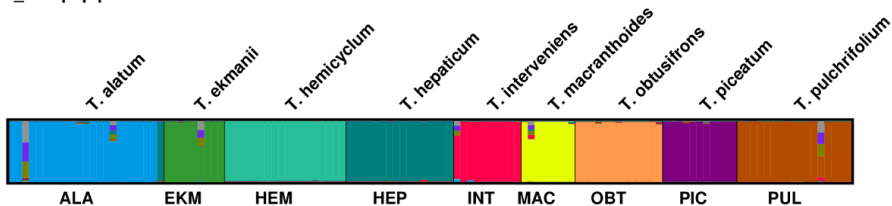


# Example of final Structure plot

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>

t\_12.1.popq



# Multiple sequence alignment

## 9 Alignment

Overview and MAFFT

MAFFT, Clustal, MUSCLE and T-Coffee

Display and cleaning



# Multiple sequence alignment

- Good alignment is basic condition for any analysis of DNA sequences
- DNA/RNA and protein sequences must be aligned prior any subsequent analysis (tree building, ...)
- 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

```

1 meles.mafft
2 class(meles.mafft)
3 # read ?clustal and documentation of Clustal, Muscle and T-Coffee
4 # when using them to set correct parameters
5 meles.clustal <- ape::clustal(x=meles.dna, pw.gapopen=10, pw.gapext=0.1,
6   gapopen=10, gapext=0.2, exec="/usr/bin/clustalw2", quiet=FALSE,
7   original.ordering=TRUE) # Change "exec" to fit your path to clustal!
8 meles.clustal
9 class(meles.clustal)
10 meles.muscle <- muscle(x=meles.dna, exec="muscle", quiet=FALSE,
11   original.ordering=TRUE) # Change "exec" to fit your path to muscle!
12 meles.muscle
13 class(meles.muscle)
14 # Remove gaps from alignment - destroy it
15 meles.nogaps <- del.gaps(meles.muscle)
16 ?del.gaps # See for details!

```

# Multiple sequence alignment with MUSCLE

```
1 # Plot the alignment select bases to plot and/or modify colors
2 image(x=meles.muscle, c("a", "t", "c", "g", "n"), col=rainbow(5))
3 # Add gray dotted grid
4 grid(nx=ncol(meles.muscle), ny=nrow(meles.muscle), col="lightgrey")
```



# Align multiple genes

- NGS/HTS introduced work with hundreds and thousands of genes, it makes sense to process them in batch and not manually one-by-one

```

1 # Create a list of DNABin objects to process
2 multialign <- list(meles.dna, usflu.dna, usflu.dna2, usflu.dna3)
3 # See it
4 multialign
5 class(multialign)
6 lapply(X=multialign, FUN=class)
7 # Do the alignment
8 multialign.aln <- lapply(X=multialign, FUN=phyloch::mafft,
9   method="localpair", maxiterate=100, path="/usr/bin/mafft")
10 # Change "path" to fit your path to mafft!
11 # See result
12 multialign.aln
13 multialign.aln[[1]]
14 lapply(X=multialign.aln, FUN=class)

```

# Align multiple genes in parallel

- There are plenty of implementations of parallelisation and using of computer clusters, see <https://cran.r-project.org/web/views/HighPerformanceComputing.html>

```
1 library(parallel)
2 # Do the same in parallel (mclapply do the tasks in parallel, not
3 # one-by-one like lapply)
4 multialign.aln2 <- mclapply(X=multialign, FUN=ape::muscle,
5   exec="muscle", quiet=FALSE, original.ordering=TRUE)
6 # Change "path" to fit your path to muscle!
7 # mclapply() relies on forking and hence is not available on Windows
8 # unless "mc.cores=1"
9 # See result
10 multialign.aln2
11 lapply(X=multialign.aln2, FUN=class)
12 ?mclapply # See more options
13 ?clusterApply # See more options (parLapply should work on Windows)
```

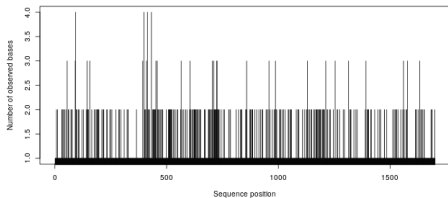
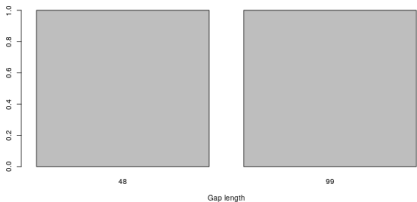
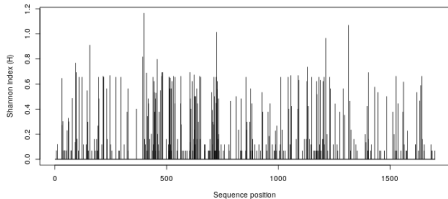
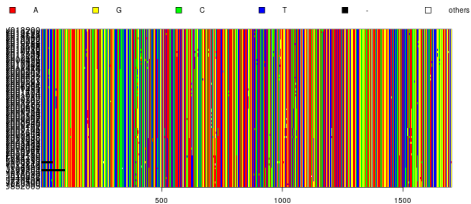
# Checking the alignment

```

1 # Shortcut for plotting alignment
2 image.DNABin(x=meles.mafft)
3 # Display aligned sequences with gaps
4 image.DNABin(x=usflu.dna)
5 # Check the alignment
6 checkAlignment(x=usflu.dna, check.gaps=TRUE, plot=TRUE, what=1:4)
7 checkAlignment(x=as.matrix.DNABin(x=meles.dna), check.gaps=TRUE,
8   plot=TRUE, what=1:4)
9 ?checkAlignment # See details
10 # DNABin can be technically list or matrix - some functions require
11 # list, some matrix, some can handle both - check manual and if needed,
12 # use:
13 as.matrix.DNABin()
14 as.list.DNABin()
15 # Matrix makes sense only for alignments, list for any import
16 # (sequences do not have to have same lengths)

```

# Checking the alignment





# Cleaning the alignment

```
1 # Delete all columns containing any gap
2 library(ips)
3 usflu.dna.ng <- deleteGaps(x=usflu.dna, nmax=0)
4 usflu.dna.ng
5 # See of settings of "nmax" value - threshold for gap deletion
6 ?deleteGaps # "nmax=0" deletes all columns with any gap
7 multialign.aln.ng <- lapply(X=multialign.aln, FUN=deleteGaps, nmax=0)
8 multialign.aln.ng
9 # Do not confuse with function delete.gaps() from phyloch package
10 # Display the result
11 image.DNABin(x=usflu.dna.ng)
12 lapply(X=multialign.aln.ng, FUN=image.DNABin)
13 # Delete positions in alignment containing only missing data/N
14 # (rows as well as columns)
15 ?ips::deleteEmptyCells # See help page for details
16 ?phyloch::delete.empty.cells # See help page for details
17 # Delete only empty columns/rows (usage as above)
18 ?ape::del.colgaponly # See help page for details
19 ?ape::del.rowgaponly # See help page for details
```

# Manipulation, display and analysis of sets of trees

Work with individual trees and sets of trees, finding species trees from multiple gene trees

## 10 Trees

Manipulations

Seeing trees in the forest

MP

Comparisons

Notes about plotting the trees

# Read and write tree and drop tips

```
1 # Read trees in NEWICK format - single or multiple tree(s)
2 oxalis.trees <- read.tree
3   (file="https://soubory.trapa.cz/rcourse/oxalis.nwk")
4 summary(oxalis.trees)
5 length(oxalis.trees)
6 names(oxalis.trees)
7 # Export trees in NEWICK format
8 write.tree(phy=oxalis.trees, file="trees.nwk")
9 oxalis.trees[[1]][["tip.label"]] # See tip labels
10 # Drop a tip from multiPhylo
11 plot.multiPhylo(x=oxalis.trees)
12 oxalis.trees.drop <- lapply(X=oxalis.trees, FUN=drop.tip,
13   tip="O._callosa_S15")
14 class(oxalis.trees.drop) <- "multiPhylo"
15 plot.multiPhylo(x=oxalis.trees.drop)
16 plot.phylo(hauss.nj) # Drop a tip from single tree
17 hauss.nj[["tip.label"]] # See tip labels (and numbers)
18 hauss.nj.drop <- drop.tip(phy=hauss.nj, tip=47)
19 plot.phylo(hauss.nj.drop)
```

# Extract clades from trees and drop extinct tips

```
1 # Interactively extract tree
2 # Plot source tree
3 plot.phylo(hauss.nj)
4 nodelabels() # See node labels (numbers) - needed for some tasks
5 # Select clade to extract by clicking on it
6 hauss.nj.extracted <- extract.clade(phy=hauss.nj, interactive=TRUE)
7 # See new extracted tree
8 plot.phylo(hauss.nj.extracted)
9 # Non-interactively extract tree
10 hauss.nj.extracted <- extract.clade(phy=hauss.nj, node=60,
11   interactive=FALSE)
12 # See new extracted tree
13 plot.phylo(hauss.nj.extracted)
14 # Drop "extinct" tips - those who don't reach end the tree
15 # tolerance is respective to the used metrics
16 plot.phylo(hauss.nj)
17 axisPhylo()
18 hauss.nj.fossil <- drop.fossil(phy=hauss.nj, tol=0.4)
19 plot.phylo(hauss.nj.fossil)
```



# Join two trees, rotate tree

```

1 # Bind two trees into one
2 hauss.nj.bind <- bind.tree(x=hauss.nj.fossil, y=hauss.nj.extracted,
3   where="root", position=0, interactive=FALSE)
4 plot.phylo(hauss.nj.bind)
5 # Bind two trees interactively
6 # Plot tree receiving the new one
7 plot.phylo(hauss.nj.fossil)
8 # Select where to bind new tree to
9 hauss.nj.bind <- bind.tree(x=hauss.nj.fossil, y=hauss.nj.extracted,
10  interactive=TRUE)
11 plot.phylo(hauss.nj.bind)
12 # Rotate tree
13 # plot.phylo plots tree in exact order as it is in the phylo object
14 plot.phylo(hauss.nj)
15 nodelabels()
16 hauss.nj.rotated <- rotate(phy=hauss.nj, node="70")
17 plot.phylo(hauss.nj.rotated)

```

# Ladderize and (un)root the tree

```
1 # Ladderize the tree
2 plot.phylo(hauss.nj)
3 hauss.nj.ladderized <- ladderize(hauss.nj)
4 plot.phylo(hauss.nj.ladderized)
5 # Root the tree
6 plot.phylo(hauss.nj)
7 print.phylo(hauss.nj)
8 # resolve.root=TRUE ensures root will be bifurcating (needed here)
9 # (without this parameter it sometimes doesn't work)
10 hauss.nj.rooted <- root(phy=hauss.nj, resolve.root=TRUE, outgroup=10)
11 print.phylo(hauss.nj.rooted)
12 plot.phylo(hauss.nj.rooted)
13 # Root the tree interactive
14 plot.phylo(hauss.nj)
15 hauss.nj.rooted <- root(phy=hauss.nj, interactive=TRUE)
16 plot.phylo(hauss.nj.rooted)
17 unroot() # Unroot the tree
18 is.rooted() # Check if it is rooted
```

# Check tree and compute branch lengths and times

```
1 # Check if the tree is ultrametric - is variance of distances
2 # of all tips to node 0? It is required for some analysis
3 is.ultrametric()
4 # Make tree ultrametric
5 chronos()
6 ?chronos # Check it for mode how to calculate the lengths
7 # chronos has more uses - it is mainly used for dating
8 # Compute branch lengths for trees without branch lengths
9 ?compute.brklen # Check it for mode how to calculate the lengths
10 compute.brklen()
11 # Computes the branch lengths of a tree giving its branching
12 # times (aka node ages or heights)
13 compute.brtime()
14 ?compute.brtime # 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` (afterward use `class(x) <- "multiPhylo"`)

# 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 become Euclidean) by functions like `quasi.euclid` 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

```

1 colnames(oxalis.trees.d) <- names(oxalis.trees) # Add names of
2 rownames(oxalis.trees.d) <- names(oxalis.trees) # columns and rows
3 # Make matrix symmetric
4 oxalis.trees.d[lower.tri(oxalis.trees.d)] <-
5   t(oxalis.trees.d)[lower.tri(oxalis.trees.d)]
6 # Create heat maps using heatmap.2 function from gplots package
7 heatmap.2(x=oxalis.trees.d, Rowv=FALSE, Colv="Rowv", dendrogram="none",
8   symm=TRUE, scale="none", na.rm=TRUE, revC=FALSE, col=rainbow(15),
9   cellnote=oxalis.trees.d, notecex=1, notecol="white", trace="row",
10  linecol="black", labRow=names(oxalis.trees),
11  labCol=names(oxalis.trees), key=TRUE, keysize=2,
12  density.info="density", symkey=FALSE, main="Correlation matrix of
13  topographical distances", xlab=names(oxalis.trees),
14  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)

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

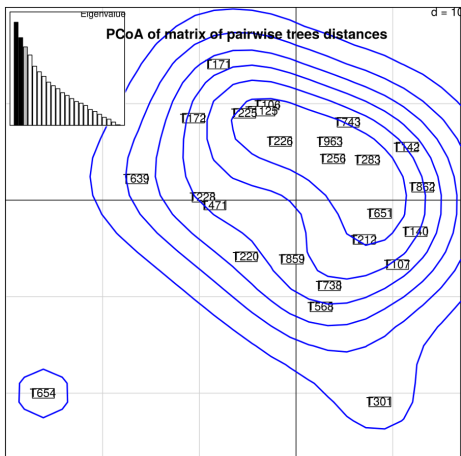


# PCoA from distance matrices of topographical differences among trees — the code

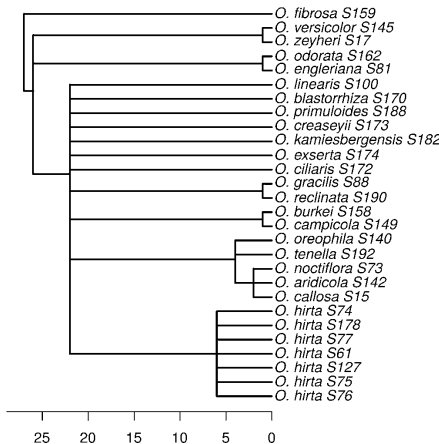
PC plots help to identify outliers — trees with noticeably different topology

```
1 # Test if the distance matrix is Euclidean or not
2 is.euclid(distmat=as.dist(oxalis.trees.d), plot=TRUE)
3 [1] FALSE # We can use e.g. quasieuclid() to make it Euclidean
4 # Calculate the PCoA
5 oxalis.trees.pcoa <- dudi.pco(d=quasieuclid(as.dist(oxalis.trees.d)),
6   scannf=TRUE, full=TRUE)
7 # Plot PCoA
8 s.label(dfx=oxalis.trees.pcoa$li)
9 # Add kernel densities
10 s.kde2d(dfx=oxalis.trees.pcoa$li, cpoint=0, add.plot=TRUE)
11 # Add histogram of eigenvalues
12 add.scatter.eig(oxalis.trees.pcoa[["eig"]], 3,1,2, posi="topleft")
13 # Add title to the plot
14 title("PCoA of matrix of pairwise trees distances")
15 # Alternative function to plot PCA plot
16 scatter(x=oxalis.trees.pcoa, posieig="topleft")
```

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



# Consensus tree

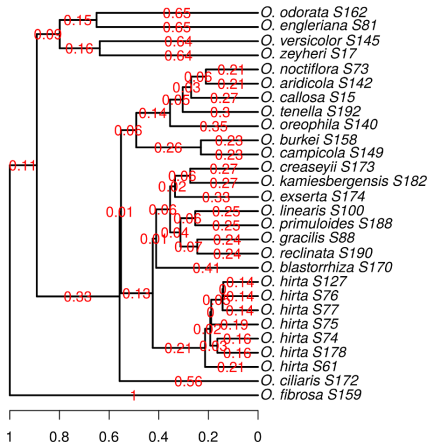


```

1 # Root all trees
2 lapply(X=oxalis.trees, FUN=root,
3       "O._fibrosa_S159",
4       resolve.root=TRUE)
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



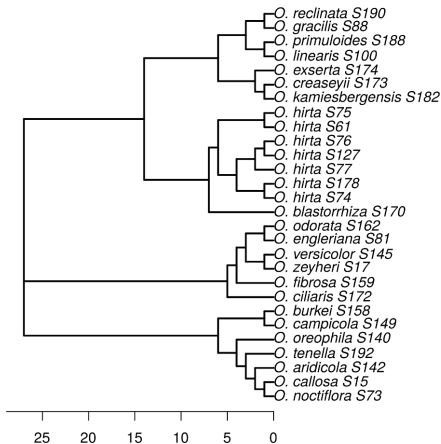
```

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



```

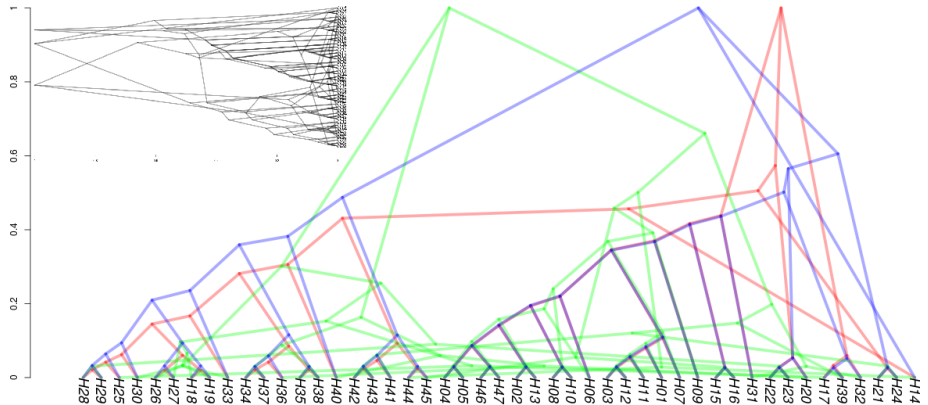
1 library(phytools)
2 oxalis.tree.sp <- mrp.supertree
3   (tree=oxalis.trees.rooted,
4     method="optim.parsimony",
5     rooted=TRUE)
6 print.phylo(oxalis.tree.sp)
7 plot.phylo(oxalis.tree.sp,
8   edge.width=2, label.offset=0.01)
9 axisPhylo(side=1)
10 # Similar function
11 ?phangorn::superTree

```

# Density tree I

```
1 # Prepare list of trees to show
2 hauss.nj.trees <- list(hauss.nj, hauss.nj.bruvo, hauss.nj.rooted)
3 hauss.nj.trees <- lapply(X=hauss.nj.trees, FUN=compute.brlen)
4 hauss.nj.trees <- lapply(X=hauss.nj.trees, FUN=chronos)
5 class(hauss.nj.trees) <- "multiPhylo"
6 # The trees should be (otherwise plotting works, but may be ugly)...
7 is.rooted.multiPhylo(hauss.nj.trees) # rooted,
8 is.ultrametric.multiPhylo(hauss.nj.trees) # ultrametric and
9 is.binary.multiPhylo(hauss.nj.trees) # binary bifurcating.
10 # Plotting has various options, play with it
11 phangorn::densiTree(x=hauss.nj.trees, direction="downwards",
12   scaleX=TRUE, col=rainbow(3), width=5, cex=1.5) # See next slide
13 densiTree(x=hauss.nj.trees, direction="upwards", scaleX=TRUE, width=5)
14 densiTree(x=hauss.nj.trees, scaleX=TRUE, width=5, cex=1.5)
15 # Compare this option with similar on following slide
16 ?phangorn::densiTree
17 ?phytools::densityTree
```

# Density tree II

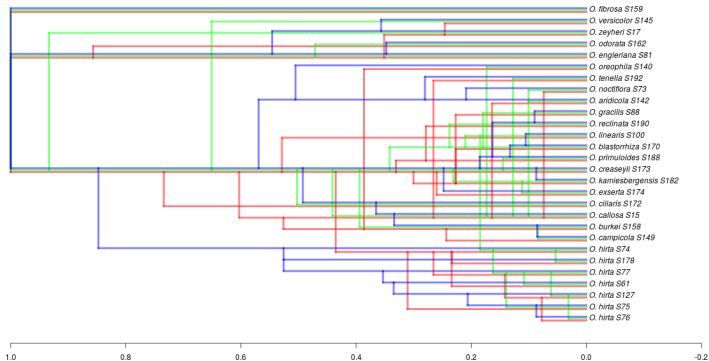


# Density tree III

```

1 phytools::densityTree(trees=oxalis.trees.ultra, fix.depth=TRUE,
2   use.gradient=TRUE, alpha=0.5, lwd=4) # Probably to much noise... :-?
3 phytools::densityTree(trees=oxalis.trees.ultra[1:3], fix.depth=TRUE,
4   use.gradient=TRUE, alpha=0.5, lwd=4) # Nice selection
5 phytools::densityTree(trees=oxalis.trees.ultra[c(2,4,6,7)],
6   fix.depth=TRUE, use.gradient=TRUE, alpha=0.5, lwd=4) # Nice selection

```

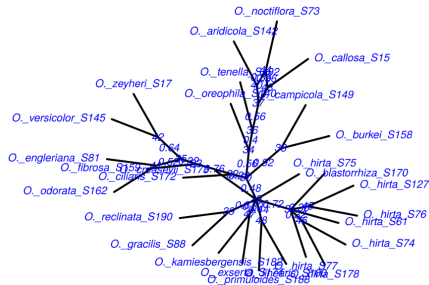


# Networks

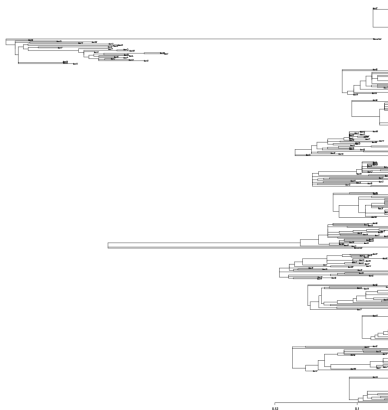
```

1 library(phangorn)
2 oxalis.tree.net <- consensusNet(oxalis.trees.rooted, prob=0.25)
3 plot(x=oxalis.tree.net, planar=FALSE, type="2D", use.edge.length=TRUE,
4     show.tip.label=TRUE, show.edge.label=TRUE, show.node.label=TRUE,
5     show.nodes=TRUE, edge.color="black", tip.color="blue") # 2D (down)
6 plot(x=oxalis.tree.net, planar=FALSE, type="3D", use.edge.length=TRUE,
7     show.tip.label=TRUE, show.edge.label=TRUE, show.node.label=TRUE,
8     show.nodes=TRUE, edge.color="black", tip.color="blue") # 3D

```



# Kronoviz — see all trees on same scale



```

1 kronoviz(x=oxalis.trees.rooted,
2         layout=length(oxalis.trees.
3         rooted), horiz=TRUE)
4 # Close graphical device to
5 # cancel division of plotting
6 # device
7 dev.off()

```

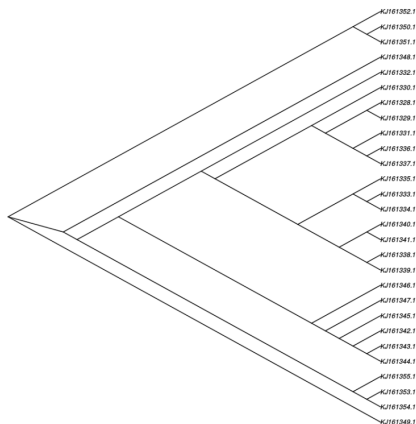
- The plot can be very long and it can be hard to see details
- But one can get impression if all trees are more or less in same scale (have comparable length) or not

# 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  $\sim 9$  taxa, branch-and-bound swapping (guaranteeing finding the best tree) until  $\sim 20$  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 149) is used to start the search
- With rising performance of computers, it uses to be replaced maximum likelihood or Bayesian methods

# Maximum parsimony — code and result

Maximum-parsimony tree of Meles



```

1 ?parsimony # Parsimony details
2 # Conversion to phyDat
3 meles.phydat <-
4   as.phyDat(meles.dna)
5 # Prepare starting tree
6 meles.tre.ini <- nj(dist.
7   dna(x=meles.dna, model="raw"))
8 # Maximum parsimony score
9 parsimony(tree=meles.tre.ini,
10  data=meles.phydat)
11 # Optimization
12 # Maximum parsimony tree
13 meles.tre.pars <- optim.
14   parsimony(tree=meles.tre.
15   ini, data=meles.phydat)
16 # Draw a tree
17 plot.phylo(x=meles.tre.pars,
18  type="clad", edge.width=2)

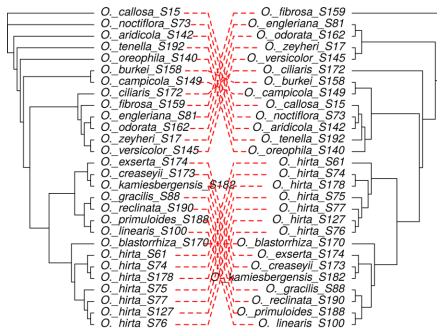
```



# Compare two trees

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

# Cophyloplot comparing two trees



- ladderize() pre-sorts tips in the tree — it can help to cophyloplot() to reduce crossings

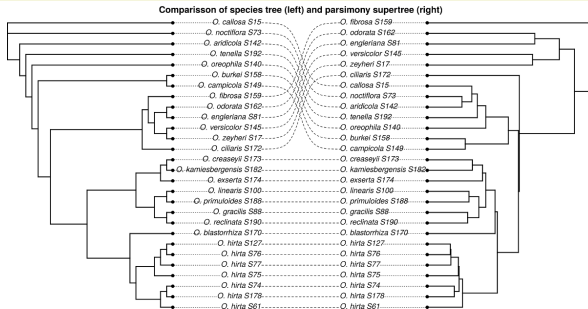
- cophyloplot() has not any optimization to plot the lines
- Automatic plot is usually not perfect — there use to be unneeded crossing lines — rotate=TRUE is recommended to can fix this manually by clicking to the nodes
- cophyloplot() has similar parameters like plot.phylo() — play with it and/or adjust in graphical editor

# Alternative implementation — phytools::cophylo

```

1 ?cophylo # See options
2 # Prepare the object for plotting
3 oxalis.cophylo <- cophylo(tr1=oxalis.tree.sp, tr2=oxalis.tree.sp.mean,
4   assoc=(cbind(sort(oxalis.tree.sp$tip.label),
5     sort(oxalis.tree.sp$tip.label))), rotate=TRUE)
6 plot.cophylo(x=oxalis.cophylo, lwd=2, link.type="curved") # Plot it
7 title("\nComparisson of species tree (left) and parsimony supertree
8   (right)")

```



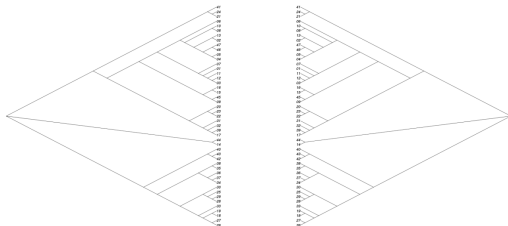
# Change orientation of plots

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

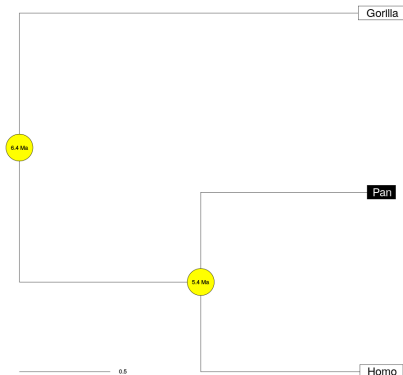
```

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

```



# Highlighted labels



```

1 # Load tree in text format
2 trape <- read.tree(text=
3   "((Homo, Pan), Gorilla);")
4 # Plot the tree
5 plot.phylo(x=trape,
6   show.tip.label=FALSE)
7 # Add colored tip labels
8 tiplabels(trape[["tip.label"]],
9   bg=c("white", "black",
10    "white"), col=c("black",
11    "white", "black"), cex=2)
12 # Add colored node labels
13 nodelabels(text=c("6.4 Ma",
14   "5.4 Ma"), frame="circle",
15   bg="yellow")
16 add.scale.bar() # Add scale bar
17 # Note vectors for tip/nodelabels

```

# Reconstruction of evolution of traits

## 11 Evolution

PIC

Autocorrelation

Decomposition

PGLS

GEE

Phylosignal

pPCA

Ancestral state

Phenogram

# Overview of methods of reconstruction of evolution of traits I

- Testing if there is correlation between evolution of two or more characters (if they evolve together)
- Testing if there is correlation between one character and phylogenetic history (if trait changes follow evolution)
- Reconstruction of ancestral states of character
- For some methods, taxonomic level can be taken into account (if there is significant evolutionary signal on the trait evolution on e.g. level of genus or family)
- Generally available for continuous as well as discrete characters (not in all methods)
- Some methods can handle more observations per accession
- There are various methods how to display everything

# Overview of methods of reconstruction of evolution of traits II

- Methods and models are highly debated in the literature
  - Different experts commonly disagree what is the best method
  - General methods are not usable everywhere (e.g. evolution of genome size must take into account polyploidization — [chromEvol](#))
  - Usage is better to be consulted with some relevant expert
  - This is very difficult chapter by meaning of how to find the best method to analyze particular data...
- Always read manual and original papers explaining the methods



# 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

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

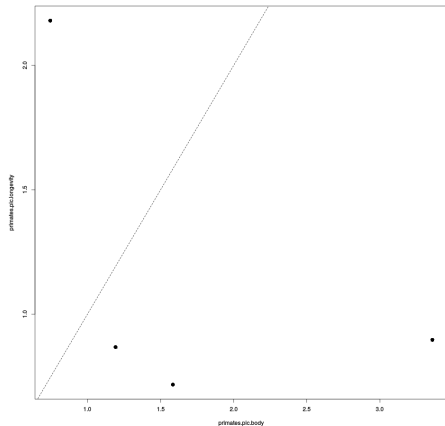
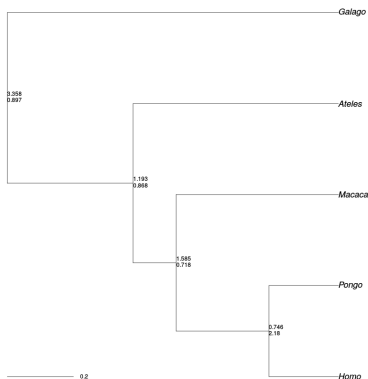
# PIC and its plotting

```

1 primates.pic.body <- pic(x=primates.body, phy=primates.tree,
2   scaled=TRUE, var.contrasts=FALSE, rescaled.tree=FALSE)
3 primates.pic.longevity <- pic(x=primates.longevity, phy=primates.tree,
4   scaled=TRUE, var.contrasts=FALSE, rescaled.tree=FALSE)
5 # Plot a tree with PIC values
6 plot.phylo(x=primates.tree, edge.width=2, cex=1.5)
7 nodelabels(round(primates.pic.body, digits=3), adj=c(0, -0.5),
8   frame="none")
9 nodelabels(round(primates.pic.longevity, digits=3), adj=c(0, 1),
10  frame="none")
11 add.scale.bar()
12 # Plot PIC
13 plot(x=primates.pic.body, y=primates.pic.longevity, pch=16, cex=1.5)
14 abline(a=0, b=1, lty=2) # x=y line
15 # Correlation coefficient of both PICs
16 cor(x=primates.pic.body, y=primates.pic.longevity, method="pearson")
17 [1] -0.5179156

```

# Plot of PIC (on the tree)



# Test it

```
1 lm(formula=primates.pic.longevity~primates.pic.body)
2 Coefficients:
3 (Intercept) primates.pic.body
4 1.6957 -0.3081
5 # Because PICs have expected mean zero - such linear regressions
6 # should be done through the origin (the intercept is set to zero)
7 lm(formula=primates.pic.longevity~primates.pic.body-1)
8 Coefficients:
9 primates.pic.body
10 0.4319
11 # Permutation procedure to test PIC
12 lmorigin(formula=primates.pic.longevity~primates.pic.body, nperm=1000)
13 Regression through the origin
14 Permutation method = raw data
15 Coefficients and parametric test results
16 Coefficient Std_error t-value Pr(>|t|)
17 primates.pic.body 0.43193 0.28649 1.5077 0.2288
18 F-statistic: 2.273067 on 1 and 3 DF:
19 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...

```

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

```

# Explanation of the cbind trick

```

1 cbind(primates.body, jitter(primates.body), jitter(primates.body))
2 Homo          4.09434 4.113074 4.038092
3 Pongo         3.61092 3.671217 3.558953
4 Macaca        2.37024 2.426757 2.430310
5 Ateles        2.02815 1.986006 2.091402
6 Galago        1.46968 1.494281 1.496831
7 cbind(primates.body, jitter(primates.body), jitter(primates.body))[1,]
8 4.094340      4.072501      4.035324
9 cbind(primates.body, jitter(primates.body), jitter(primates.body))[2,]
10 3.610920     3.572728     3.664654
11 class(cbind(primates.body, jitter(primates.body),
12           jitter(primates.body)))
13 [1] "matrix"
14 class(cbind(primates.body, jitter(primates.body),
15           jitter(primates.body))[1,])
16 [1] "numeric"
17 # 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

```
1 # Let's choose weights as  $w_{ij} = 1/d_{ij}$ , where the d's is the distances
2 # measured on the tree - cophenetic() calculates cophenetic distances
3 # can be just cophenetic(primates.tree) or some other transformation
4 primates.weights <- 1/cophenetic(primates.tree)
5 primates.weights # See it
6 class(primates.weights)
7 diag(primates.weights) <- 0 # Set diagonal to 0
```

# Testing of Moran's I

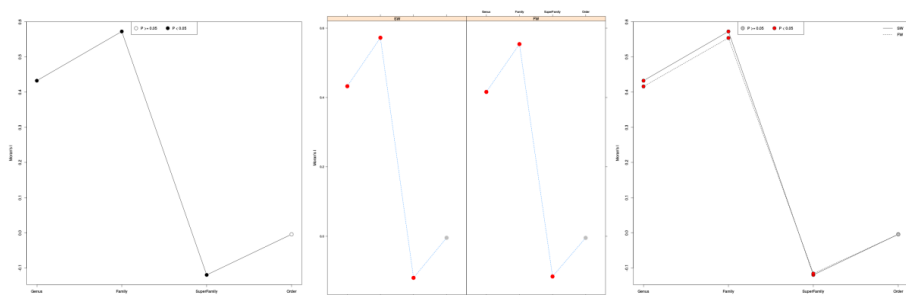
```
1 # Calculate Moran's I
2 # Slightly significant positive phylogenetic correlation among body mass
3 Moran.I(x=primates.body, weight=primates.weights,
4         alternative="greater")
5 # Positive, but non-significant
6 Moran.I(x=primates.longevity, weight=primates.weights,
7         alternative="greater")
8 # Test of Moran's with randomization procedure
9 # Body is significant - nonrandom, longevity not (random)
10 gearymoran(bilis=primates.weights, X=data.frame(primates.body,
11          primates.longevity), nrepet=1000)
12 # Test of Abouheif designed to detect phylogenetic autocorrelation in
13 # a quantitative trait - in fact Moran's I test using a particular
14 # phylogenetic proximity between tips
15 library(adephylo)
16 abouheif.moran(x=cbind(primates.body, primates.longevity),
17               W=primates.weights, method="oriAbouheif", nrepet=1000,
18               alter="greater")
```



# Correlogram to visualize results of phylogenetic autocorrelation analysis

```
1 data(carnivora) # Loads training data set
2 head(carnivora) # Look at the data
3 # Calculate the correlogram
4 carnivora.correlogram <- correlogram.formula
5   (formula=SW~Order/SuperFamily/Family/Genus, data=carnivora)
6 carnivora.correlogram # See results
7 # Calculate the correlogram - test for both body masses
8 carnivora.correlogram2 <- correlogram.formula
9   (formula=SW+FW~Order/SuperFamily/Family/Genus, data=carnivora)
10 carnivora.correlogram2 # See results
11 plot.correlogram(x=carnivora.correlogram, legend=TRUE,
12   test.level=0.05, col=c("white", "black")) # Plot it
13 # Plot it - test for both body masses - two or one graph(s)
14 plot.correlogramList(x=carnivora.correlogram2, lattice=TRUE,
15   legend=TRUE, test.level=0.05)
16 plot.correlogramList(x=carnivora.correlogram2, lattice=FALSE,
17   legend=TRUE, test.level=0.05)
```

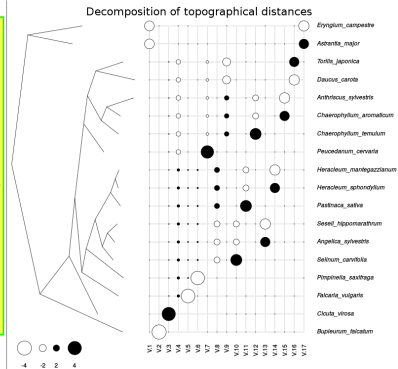
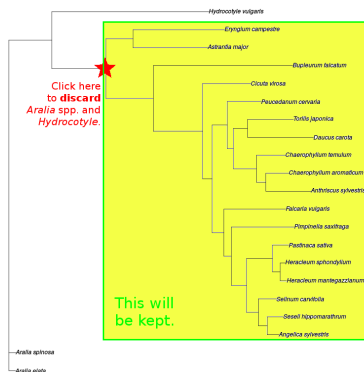
# Correlograms of SW and SW+FW (in one or two graphs) depending on taxonomic level with marked significance



# Prepare toy data set (tree)

```
1 # Load MrBayes tree in NEXUS format
2 apiaceae.tree <- read.nexus
3   (file="https://soubory.trapa.cz/rcourse/apiaceae_mrbayes.nexus")
4 print.phylo(apiaceae.tree) # See it
5 plot.phylo(apiaceae.tree) # See it
6 # Root the tree
7 apiaceae.tree <- root(apiaceae.tree, "Aralia_elata")
8 # Remove "_" from taxa names
9 # plot.phylo() by default omits "_" from tip names
10 apiaceae.tree$tip.label <- gsub(pattern="_", replacement=" ",
11   x=apiaceae.tree$tip.label)
12 # Drop outgroup (Aralia and Hydrocotyle)
13 # Click on last common ancestor of ingroup desired to be kept
14 plot.phylo(apiaceae.tree)
15 apiaceae.tree <- extract.clade(apiaceae.tree, interactive=TRUE)
16 plot.phylo(apiaceae.tree)
17 library(adephylo)
18 library(phylobase)
```

# Modified tree



```

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

# Prepare toy data set (the variable)

```

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

```

- `sim.char()` creates an array (we keep only numeric vector of 1<sup>st</sup> 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

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

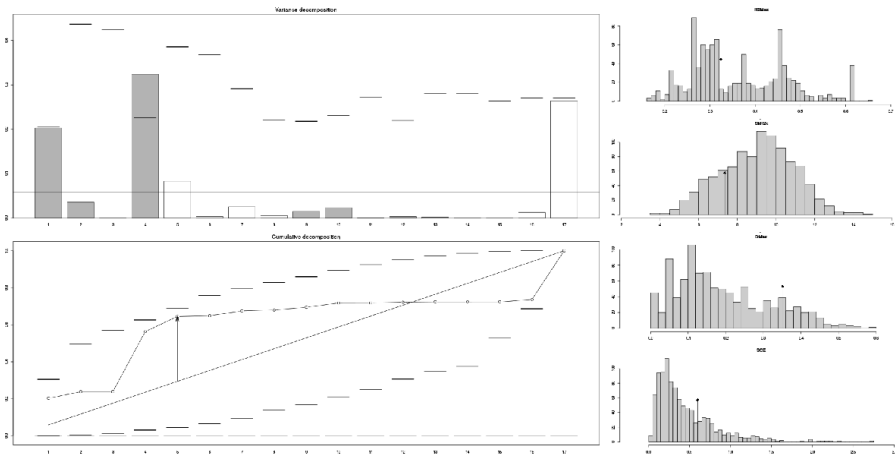
```
1           Df   Sum Sq  Mean Sq  F value  Pr(>F)
2 as.matrix...  2  0.063689  0.031845   2.2275  0.1422
3 Residuals    15  0.214443  0.014296
```

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

```
1 orthogram(x=apiaceae.eco, tre=apiaceae.tree, nrepet=1000,  
2   alter="two-sided")  
3 ?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 inertia of the trait to phylogeny...



# Phylogenetic Generalized Least Squares

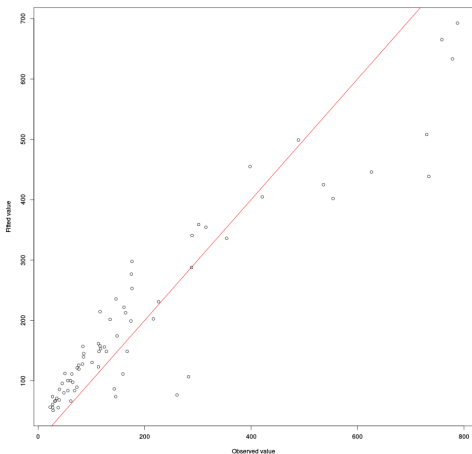
- Model-based testing if there is significant correlation between two traits (after removing the phylogenetic component)
- `nlme::gls` fits a linear model using generalized least squares
- Functions `corBlomberg`, `corBrownian`, `corMartins` and `corPagel` from `ape` package create correlation matrix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

```
1 library(caper) # Load needed library
2 data(shorebird) # Load training data, see ?shorebird.data
3 # Calculate the model
4 shorebird.pgls <- pgls(formula=shorebird.data[["F.Mass"]] ~
5   shorebird.data[["Egg.Mass"]], data=comparative.data(phy=
6   shorebird.tree, data=as.data.frame(cbind(shorebird.data[["F.Mass"]],
7   shorebird.data[["Egg.Mass"]], shorebird.data[["Species"]])),
8   names.col=V3, vcv=TRUE))
9 # See the result
10 summary(shorebird.pgls)
11 # See the plot of observer and fitted values
12 plot(shorebird.pgls)
13 abline(a=0, b=1, col="red")
14 # ANOVA view of the model
15 anova(shorebird.pgls)
16 # Akaike's information criterion (smaller = better)
17 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)

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

# Not significant in this case...

```
1 Call: compar.gee(formula = primates.longevity ~
2   primates.body, phy = primates.tree)
3 Number of observations: 5
4 Model:
5           Link: identity
6 Variance to Mean Relation: gaussian
7 QIC: 7.310142
8 Summary of Residuals:
9           Min      1Q      Median      3Q      Max
10 -0.8031302 -0.0132754  0.0999588  0.1988258  0.2862064
11 Coefficients:
12           Estimate      S.E.      t Pr(T > |t|)
13 (Intercept)  1.0670417  0.5838429  1.827618  0.2695894
14 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

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

- 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

```
1 # sapply performs analysis on list of variables (numeric vectors)
2 sapply(X=list(body=primates.body, longevity=primates.longevity),
3       FUN=Kcalc, phy=primates.tree, checkdata=FALSE)
4 sapply(X=list(body=primates.body, longevity=primates.longevity),
5       FUN=phylosignal, phy=primates.tree, reps=1000)
6 # Alternative to use multiPhylosignal instead of sapply
7 multiPhylosignal(x=as.data.frame(cbind(primates.body,
8   primates.longevity)), phy=primates.tree, reps=1000)
9 # Note sapply() and multiPhylosignal() return same data, but the
10 # matrices are transposed - use t() to transpose one to look like
11 # the other:
12 t(multiPhylosignal(x=as.data.frame(cbind(primates.body,
13   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; from e.g. `plotrix::std.error`)
- `phylosig()` can be used as an alternative to `phylosignal()` — the functions are similar in basic usage

```
1 library(phytools)
2 ?phylosig # See for details
3 # Test for phylogenetic signal (here without SE)
4 phylosig(tree=apiaceae.tree, x=apiaceae.eco, method="K", test=TRUE,
5          nsim=1000)
6 phylosig(tree=primates.tree, x=primates.body, method="lambda",
7          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 (`AIC(pglS(...))`)

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

# Phylogenetic principal component analysis

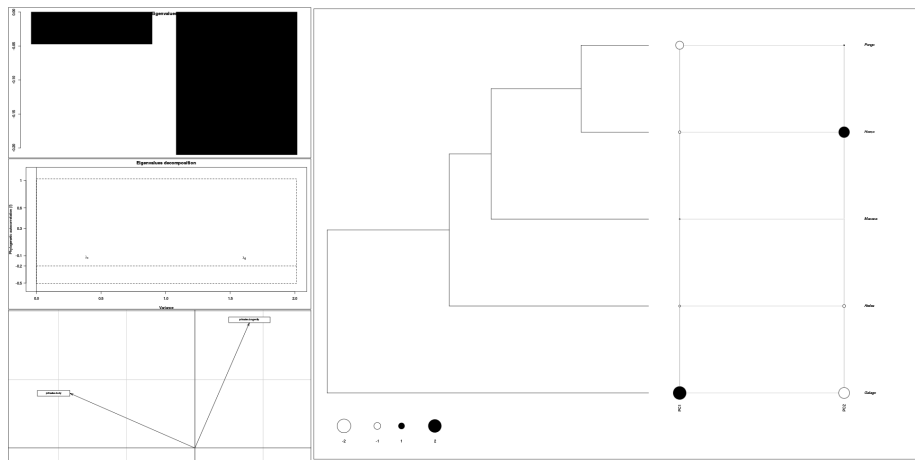
## PCA corrected for phylogeny

- 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

```
1 # Library needed to create phylo4d object required by ppca
2 library(adephylo)
3 # Calculate pPCA
4 primates.ppca <- ppca(x=phylo4d(x=primates.tree, cbind(
5   primates.body, primates.longevity)), method="patristic",
6   center=TRUE, scale=TRUE, scannf=TRUE, nfposi=1, nfnega=0)
7 # Print results
8 print(primates.ppca)
9 # See summary information
10 summary(primates.ppca)
11 # See PCA scores for variables on phylogenetic tree
12 scatter(primates.ppca)
13 # See decomposition of pPCA eigenvalues
14 screeplot(primates.ppca)
15 # Plot pPCA results - global vs. local structure, decomposition
16 # of pPCA eigenvalues, PCA plot of variables and PCA scores
17 # for variables on phylogenetic tree
18 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



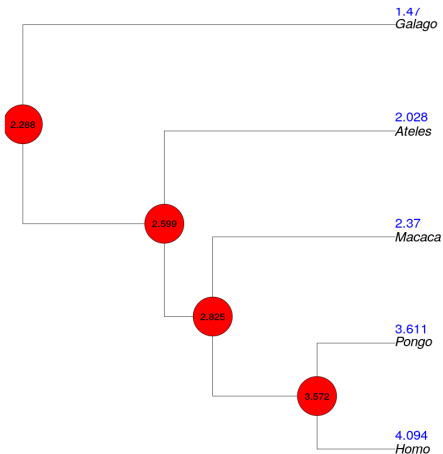
# Ancestral state reconstruction

- 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

```
1 library(ape)
2 # See ?ace for possible settings and estimations
3 primates.body.ace <- ace(x=primates.body, phy=primates.tree,
4   type="continuous", method="REML",
5   corStruct=corBrownian(value=1, phy=primates.tree))
6 # See result - reconstructions are in $ace slot
7 # To be plotted on nodes - 1st column are node numbers
8 primates.body.ace
9 # Plot it
10 plot.phylo(x=primates.tree, edge.width=2, cex=2)
11 tiplabels(round(primates.body, digits=3), adj=c(0, -1),
12   frame="none", col="blue", cex=2)
13 nodelabels(round(primates.body.ace$ace, digits=3),
14   frame="circle", bg="red", cex=1.5)
```

# Ancestral state reconstructions of primates body weights

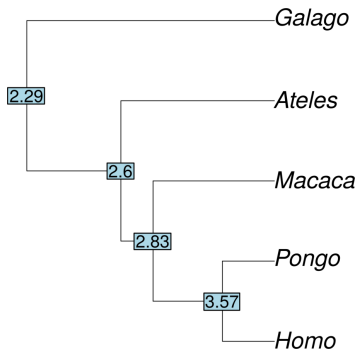
- ACE returns long numbers — truncate them by e.g. `round(x=..., digits=3)` (`x` is vector with ACE values)



## Another possibilities (package phytools)

```
1 # More possibilities
2 plot.phylo(x=primates.tree,
3   edge.width=2, cex=2)
4 # ML estimation of a continuous
5 # trait, can compute confidence
6 # interval (see further)
7 nodelabels(fastAnc(tree=
8   primates.tree, x=primates.body))
9 # ACE for Brownian evolution with
10 # directional trend
11 plot.phylo(x=primates.tree,
12   edge.width=2, cex=2)
13 nodelabels(anc.trend(tree=
14   primates.tree, x=primates.body,
15   maxit=100000)$ace)
16 # ACE for Brownian evolution using
17 # likelihood
18 plot.phylo(x=primates.tree,
```

```
1   edge.width=2, cex=2)
2 nodelabels(round(anc.ML(tree=
3   primates.tree, x=primates.body,
4   maxit=100000, model="BM")$ace,
5   digits=2), cex=1.5)
```



# Bayesian ancestral character estimation I

```
1 primates.body.ace.bayes <- anc.Bayes(tree=primates.tree,  
2   x=primates.body, ngen=100000) # Use more MCMC generations  
3 primates.body.ace.bayes  
4 # Get end of ancestral states from Bayesian posterior distribution  
5 # (it should converge to certain values)  
6 tail(primates.body.ace.bayes[["mcmc"]])  
7 primates.body.ace.bayes[["mcmc"]][1001,3:6]  
8 # Get means of ancestral states from Bayesian posterior distribution  
9 colMeans(primates.body.ace.bayes[["mcmc"]][201:nrow  
10 (primates.body.ace.bayes[["mcmc"]]),as.character(1:primates.tree  
11 $Nnode+length(primates.tree$tip.label))])  
12 # Plot the likelihood from posterior distribution (it should converge  
13 # to certain values)  
14 plot(primates.body.ace.bayes)  
15 # Plot the tree and reconstructed ancestral states  
16 plot.phylo(x=primates.tree, edge.width=2, cex=2)  
17 nodelabels(round(x=primates.body.ace.bayes[["mcmc"]][1001,3:6],  
18   digits=3), cex=1.5)  
19 ?phangorn::ancestral.pml # Another possibility
```

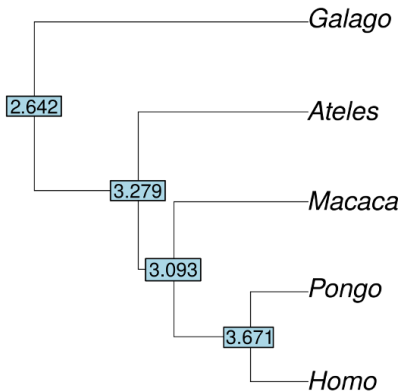
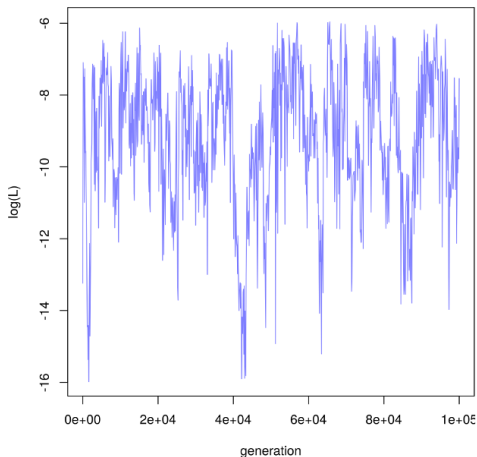


# Bayesian ancestral character estimation II

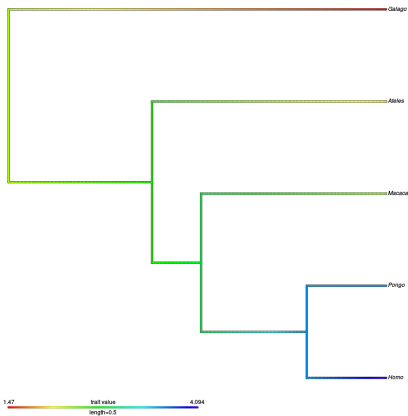
```
1 primates.body.ace.bayes # See the values
2         6         7         8         9
3 2.418557 2.569484 2.844268 3.584153
4 tail(primates.body.ace.bayes[["mcmc"]]) # See end of the table
5     gen     sig2         6         7         8         9     logLik
6 996   99500 2.328336 2.999055 2.875331 3.492034 4.220528 -9.178014
7 997   99600 2.998972 3.796833 2.938607 3.622099 3.690610 -10.166447
8 998   99700 2.677339 3.888228 3.200297 2.935587 3.585092 -9.467951
9 999   99800 1.960194 4.129754 3.196746 3.310278 3.425686 -9.780220
10 1000  99900 1.636984 3.556224 2.701105 2.720756 2.965226 -9.770242
11 1001 100000 1.588710'2.641937 3.279439 3.093420 3.670807' -7.538937
12 primates.body.ace.bayes[["mcmc"]][1001,3:6] # See end of the table
13         6         7         8         9
14 1001 2.641937 3.279439 3.09342 3.670807
15 colMeans(primates.body.ace.bayes[["mcmc"]][201:nrow(primates.body.ace.
16 bayes[["mcmc"]]),as.character(1:primates.tree$Nnode+length
17 (primates.tree$tip.label))]) # Get means of ancestral states
18         6         7         8         9
19 2.418557 2.569484 2.844268 3.584153
```

# Bayesian ancestral character estimation III

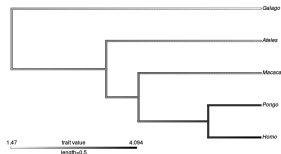
Likelihood of Bayesian posterior probability and the tree with reconstructions



# Continuous map

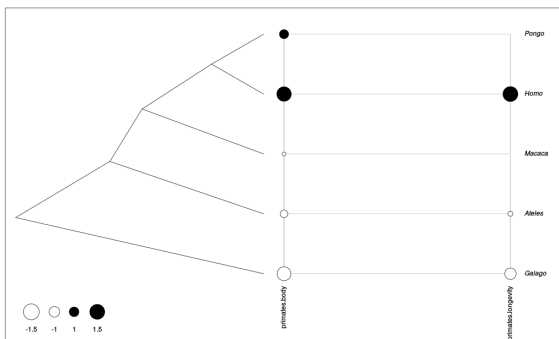


```
1 library(phytools)
2 contMap(tree=primates.tree,
3         x=primates.body)
4 # Change colors with setMap()
5 primates.contmap <- setMap(x=
6   contMap(primates.tree,
7           primates.body),
8   colors=c("white", "black"))
9 plot(primates.contmap)
10 # See ?par for more settings
```



# Display more characters on a tree in a table

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



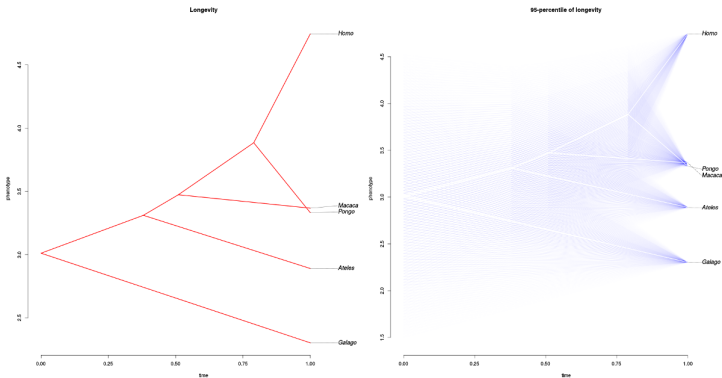
# Phenogram

Vertical axis shows character values

```

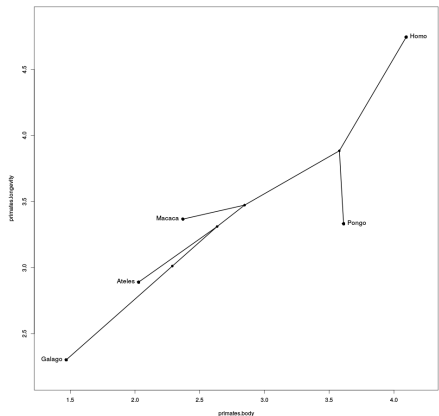
1 phenogram(tree=primates.tree, x=primates.longevity, fsize=1.2,
2   ftype="i", colors="red", main="Longevity")
3 fancyTree(tree=primates.tree, type="phenogram95", x=primates.longevity,
4   fsize=1.2, ftype="i", main="95-percentile of longevity")

```



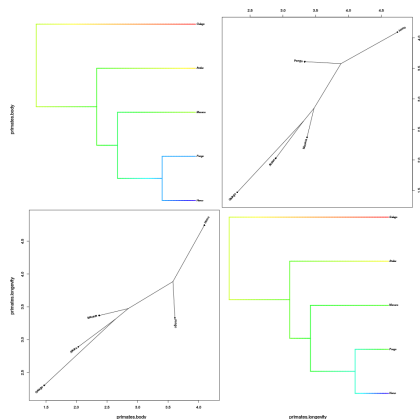
# Display 2 continuous characters in space and 3D tree connecting them

```
1 # 2 characters on 2 axis
2 phylomorphospace(tree=
3   primates.tree, X=cbind
4   (primates.body,
5   primates.longevity),
6   label="horizontal",
7   lwd=2, fsize=1.5)
8 # 3D (3rd character is fake here)
9 # 3 characters in a rotating cube
10 phylomorphospace3d(tree=
11   primates.tree, X=cbind
12   (primates.body,
13   primates.longevity,
14   abs(primates.body-
15   primates.longevity)),
16   label=TRUE)
```



# Combine phenograms and ancestral state reconstructions

```
1 # 2 characters on 2 axis
2 fancyTree(tree=primates.tree,
3           type="scattergram",
4           X=cbind(primates.body,
5                 primates.longevity),
6           res=500, ftype="i")
7 # See manuals for more settings
8 ?fancyTree
9 ?phenogram
10 ?phylomorphospace
11 ?phylomorphospace3d
12 ?contMap
13 ?setMap
14 ?par
```



# Plotting traits on trees — code

```
1 # Load training data to display
2 data(shorebird, package="caper")
3 ?caper::shorebird # See information about the data
4 library(phytools) # Load library
5 # Prepare named vector of values to plot
6 shorebird.toplot <- shorebird.data[["Egg.Mass"]]
7 names(shorebird.toplot) <- rownames(shorebird.data)
8 shorebird.toplot # See it
9 # See options for plotting functions
10 ?plotTree.wBars
11 ?dotTree
12 ?plotSimmap
13 # Plot the trees
14 plotTree.wBars(tree=shorebird.tree, x=shorebird.toplot,
15   tip.labels=TRUE)
16 dotTree(tree=shorebird.tree, x=shorebird.toplot, tip.labels=TRUE,
17   type="cladogram")
```





# Process more data

Not all combinations and possibilities were shown...

## Tasks

- 1 Try to do some analyses with another introduced toy data
  - 2 Try some of the introduced analyses with your own custom imported data
- Working code can be easily recycled to process another data in similar way
  - R is always moving forward — new and new options are arising — be opened for news and search them on the Internet
  - Previous examples are not covering all possibilities...
  - It is crucial to be able to edit the introduced commands to be able to handle your data
  - Check help pages of the functions for more options what to do with your data

# Final topics

General remarks about graphics, introduction to scripting, documentation and help resources, overview of packages

## 12 The end

Graphics

GitHub

Scripts

Functions

Loops

If-else branching

Solving problems

Resources

Summary

The end

# 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.

```

1 # Output figure will be saved to the disk as OutputFile.png
2 png(filename="OutputFile.png", width=720, height=720, bg="white")
3 # Here can go any number of functions making plots...
4 plot(...) # Whatever...
5 # When using plotting commands, nothing is shown on the screen
6 # The final plot(s) will be saved by:
7 dev.off() # Closes graphical device - needed after use of plotting
8           # functions png(), svg(), pdf(), ... followed by any
9           # function like plot() to write the file(s) to the disk
10 filename="OutFiles_%03d.png" # Returns list of files named
11                               # OutFiles_001.png, OutFiles_002.png, ...
12                               # Useful for functions returning more
13                               # graphs.
14 ?png # These functions have various possibilities to set size, whatever.
15 ?svg # Exact possibilities of all 3 functions vary from system to system
16 ?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 \(web 1, web 2\)](#)
  - They have enormous possibilities, 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`, `mfc`, `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)

```
1 # Needed library
2 install.packages("devtools")
3 library(devtools)
4 dev_mode(on=TRUE)
5 # Install selected package from GitHub (user/project)
6 install_github("thibautjombart/adegenet")
7 # when finished go back to normal version
8 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](#)

```
1 # General syntax:
2 MyFunction <- function (x, y) {
3   # Any commands can be here...
4   x + y
5 }
6 # Use as usually:
7 MyFunction(5, 8)
8 MyFunction(1, 4)
9 MyFunction(x=4, y=7)
10 MF <- MyFunction(9, 15)
11 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

```
1 # Simplest loop - print value of "i" in each step
2 # "i" is commonly used for various indexing
3 for (i in 1:5) { print(i) }
4 [1] 1 # This is the value of "i"...
5 [1] 2
6 [1] 3
7 [1] 4
8 [1] 5
```

# For and while loops

```
1 # In every step modify value of variable "X" (add 1 to previous value)
2 X <- 0 # Set initial value
3 for (i in 1:10) {
4   # Any commands can be here...
5   print("Loop turn") # Some message for user
6   print(i) # Print number of turn - note it is decreasing
7   X <- X+i # Rise value of "X" by current value of "i" (previous line)
8   print(paste("Variable value:", X)) # Print current value of "X"
9 }
10 for (i in 10:5) { print(i) } # Can be descending...
11 # Work on each item of a list object
12 # Print length of each sequence in nothofagus.sequences
13 for (L in 1:length(nothofagus.sequences)) {
14   print(length(nothofagus.sequences[[L]])) }
15 # While loop - it is done while the condition is valid
16 # While value of "Q" is < 5 (starting from 0), print it and add 1
17 Q <- 0
18 while (Q < 5) { print(Q <- Q+1) }
```

# If-else branching I

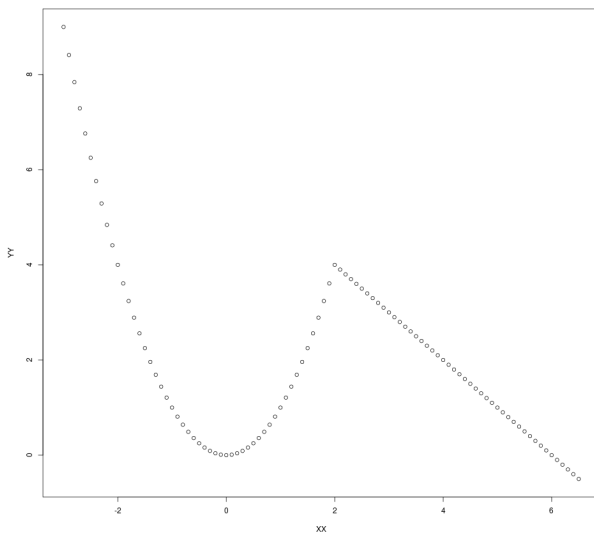
- Basic method of branching the code — **if** the condition is met, one branch is followed, **else** — in any other case — the other branch of the code is executed
- else** part can be missing — the code is executed **only if** the condition is met

```
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

```
1 # Plot from example from previous slide
2 plot(XX, YY) # See the result
3 # Or (different possibility to get very same result)
4 # Note "XX" is reused from the previous slide
5 CC <- function(AA) {
6   if(AA <= 2) { # Executed for XX <= 2
7     BB <- AA^2
8   } else { # Executed for XX > 2
9     BB <- 6-AA
10    }
11   return(BB) # The output value
12 }
13 CC # Previously, "YY" contained values to plot made by the for loop,
14    # here "CC" contains function to be used by sapply() when plotting
15 plot(sapply(XX, CC)) # See the result
16 # 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 73)
  - 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)
  - R is waiting for some user input

# Most common problems and their solutions II

- 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 168) — 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 346 and onward)
- R and packages are more or less changing from version to version
  - Old methods can become outdated and not working anymore
  - Check release notes and change logs for new versions, manual pages and on-line forums (slide 346 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 voluntarily 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?

# Where to look for the help I

## Question must have certain form!

Before asking, **ensure your question is in answerable form** — slide 344.

- Sloppily asked question can't be answered at all...
- Check documentation, manuals and search the Internet before asking

- R homepage <https://www.r-project.org/> and packages <https://cran.r-project.org/web/packages/> (with documentation and links)
- 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>

# Where to look for the help II

- Bioconductor home page <https://bioconductor.org/> and support forum <https://support.bioconductor.org/>
- Adegnet help mailing list <http://lists.r-forge.r-project.org/cgi-bin/mailman/listinfo/adegetnet-forum> and GitHub page <https://github.com/thibautjombart/adegetnet/wiki>
- Poppr forum <https://groups.google.com/forum/#!forum/poppr>
- 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>

## Where to look for the help III

- R at StackOverflow StackExchange (for programmers)  
<https://stackoverflow.com/questions/tagged/r>
- R at CrossValidated StackExchange (for stasticians, mathematicians, etc.) <https://stats.stackexchange.com/questions/tagged/r>
- Biostars — general bioinformatics forum  
<https://www.biostars.org/>
- 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... — they wish their packages to be working and useful
- **Uncle Google** is your friend here (*“how to XXX in R”*)...

# 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
- Check original papers to fully understand respective method

## 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

# Learning resources I

- R homepage <https://www.r-project.org/> and packages <https://cran.r-project.org/web/packages/> (with documentation and links)
- Books about R  
<https://www.r-project.org/doc/bib/R-books.html>
- List of R documentation  
<https://cran.r-project.org/manuals.html>
- Bioconductor help pages  
<https://master.bioconductor.org/help/>
- R phylo wiki [https://www.r-phylo.org/wiki/Main\\_Page](https://www.r-phylo.org/wiki/Main_Page)
- R phylogenetics at CRAN  
<https://cran.r-project.org/web/views/Phylogenetics.html>

## Learning resources II

- Integrated documentation search  
<https://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/>
- 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/>



## Learning resources III

- 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/> and GitHub page  
<https://github.com/thibautjombart/adegenet/wiki>
- APE home page <http://ape-package.ird.fr/>
- 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>

## Learning resources IV

- Population Genetics in R <http://popgen.nescent.org/> by Kamvar et al
- 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 <https://CRAN.R-project.org/package=phangorn>
- The R journal <https://journal.r-project.org/>
- RStudio Cheat Sheets <https://www.rstudio.com/resources/cheatsheets/> and Online learning resources <https://www.rstudio.com/online-learning/>
- R-bloggers — aggregation of R blogs <https://www.r-bloggers.com/>

## Learning resources V

- R on The Molecular Ecologist  
<https://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 <https://ropensci.org/>
- Statistics with R [http://zoonek2.free.fr/UNIX/48\\_R/all.html](http://zoonek2.free.fr/UNIX/48_R/all.html)
- The R Inferno book <https://www.burns-stat.com/documents/books/the-r-inferno/>  
(Feeling like being in hell when using R?)
- Springer R series  
<https://www.springer.com/series/6991?detailsPage=titles>

# Learning resources VI

- ggplot2 (the most powerful graphical library used by many packages) information <http://ggplot2.org/>
- plyr documentation <http://plyr.had.co.nz/> — manipulation with data (split-apply-combine)
- Leaflet for R <https://rstudio.github.io/leaflet/>
- Learning R blog <https://learnr.wordpress.com/>
- R for Community Ecologists  
<http://ecology.msu.montana.edu/labdsv/R/>
- Try R on-line course full of adventure and heroic quests  
<http://tryr.codeschool.com/> (Tired of orcs, dragons and princesses? Beat R!:) )
- Quick-R learning resource <https://www.statmethods.net/>
- R manual and help search <http://finzi.psych.upenn.edu/>

## Learning resources VII

- **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")`
- And finally: **Reading documentation is not wasting of time!** ;-)

## 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** — GUI, **adegraphics** — extra graphical functions, commonly used internally)
- **akima**: cubic spline interpolation methods for irregular and regular grids 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

## Packages we used... II

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

- **fields**: tools for spatial data
- **geiger**: fitting macroevolutionary models to phylogenetic trees
- **Geneland**: stochastic simulation and MCMC inference of structure from genetic data
- **ggplot2**: data visualizations 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

## Packages we used... III

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

- [mapplots](#): Extra map plotting, pie charts and more
- [mapproj](#): converts latitude/longitude into projected coordinates
- [maps](#): draws geographical maps
- [maptools](#): manipulating and reading geographic data
- [MASS](#): functions and data sets 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



## Packages we used... IV

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

- [pegas](#): population and evolutionary genetics analysis
- [permute](#): restricted permutation designs
- [phangorn](#): phylogenetic analysis
- [philentropy](#): over 40 optimized distance and similarity measures for comparing probability functions
- [phylobase](#): phylogenetic structures and comparative data
- [phyloch](#): interfaces and graphic tools for phylogenetic data
- [phytools](#): phylogenetic analysis, comparative biology, graphics
- [picante](#): integrates phylogeny and ecology
- [plotrix](#): various labeling, axis and color scaling functions
- [poppr](#): genetic analysis of populations with mixed reproduction
- [RandomFields](#): simulation of Gaussian fields (+ [RandomFieldsUtils](#))

## Packages we used... V

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

- [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
- [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
- [StAMPP](#): statistical analysis of mixed ploidy populations
- [stringi](#) and [stringr](#): character string processing, internally used by many packages

## Packages we used... VI

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

- **TeachingDemos**: demonstrations for teaching and learning
- **vcfR**: import/export, basic checking and manipulations of VCF
- **vegan**: community ecology
- **XVector**: representation and manipulation 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 phylogeny
- [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
- **distory**: geodesic distance between phylogenetic trees
- **diversitree**: comparative phylogenetic analyses of diversification

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

For your own explorations...

- **diveRcity**: calculation of both genetic diversity partition statistics, genetic differentiation statistics, and locus informativeness for ancestry assignment
- **ecodist**: Dissimilarity-based functions for ecological analysis, spatial and community data
- **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

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

For your own explorations...

- [genotypeR](#): design of genotyping markers from VCF files, output of markers for multiplexing on various platforms, various QA/QC and analysis
- [geomorph](#): geometric morphometric analyses of 2D/3D landmark data
- [ggtree](#): visualization and annotation of phylogenetic trees
- [HardyWeinberg](#): statistical tests and graphics for HWE
- [heatmaply](#): interactive cluster heat maps
- [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

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

For your own explorations...

- [HyPhy](#): macroevolutionary phylogenetic analysis of species trees and gene trees
- [iteRates](#): iterates through a phylogenetic tree to identify regions of rate variation
- [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 disequilibrium between SNPs
- [leaflet](#): interactive web maps with the JavaScript Leaflet library
- [Linarius](#): dominant marker analysis with mixed ploidy levels



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

For your own explorations...

- [markophylo](#): markov chain models for phylogenetic trees
- [MCMCgImm](#): MCMC generalized linear mixed models
- [MINOTAUR](#): multivariate visualization and outlier analysis
- [MonoPhy](#): visualization and exploration of monophyletic clades on a tree
- [MPSEM](#): modeling phylogenetic signals using eigenvector maps
- [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

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

For your own explorations...

- **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
- **pcadapt**: PCA and search for loci responsible for the grouping (no support for mixing ploidy levels), uses VCF
- **PCPS**: principal coordinates of phylogenetic structure
- **Phybase**: read, write, manipulate, simulate, estimate, and summarize phylogenetic trees (gene trees and species trees)
- **phyclust**: phylogenetic clustering

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

For your own explorations...

- **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
- **phyloIm**: 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
- **pmc**: phylogenetic Monte Carlo

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

For your own explorations...

- [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
- [rCharts](#): interactive JS charts
- [RColorBrewer](#): ColorBrewer palettes
- [rdryad](#): access for Dryad web services
- [Reol](#): interface to the Encyclopedia of Life
- [rMaps](#): interactive maps
- [rphast](#): interface to PHAST software for comparative genomics
- [RMesquite](#): interoperability with Mesquite

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

For your own explorations...

- [Rphylip](#): interface for PHYLIP
- [Rsamtools](#): BAM, FASTA, BCF and tabix file import and manipulations
- [SigTree](#): identify and visualize significantly responsive branches in a phylogenetic tree
- [SNPRelate](#): parallel computing toolset for relatedness and principal component analysis of SNP data
- [spatstat](#): spatial point pattern analysis
- [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

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

For your own explorations...

- [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
- [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 phylogeny
- [TreePar](#): estimating birth and death rates based on phylogeny
- [TreeSim](#): simulating phylogenetic trees
- [VariantAnnotation](#): annotation of genetic variants (useful to filter VCF)

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 X<sub>Y</sub>L<sup>A</sup>T<sub>E</sub>X on openSUSE GNU/Linux, February 23, 2018.