

# 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 8 to 11, 2021



# Outline I

## 1 Introduction

## 2 R

Installation

Let's start with R

Basic operations in R

Tasks

Packages for our work

## 3 Data

Overview of data and data types

Microsatellites

AFLP

Notes about data

DNA sequences, SNP

VCF

# Outline II

Export  
Tasks

## 4 Alignment

Overview and MAFFT

MAFFT, Clustal, MUSCLE and T-Coffee

Multiple genes

Display and cleaning

Tasks

## 5 Basic analysis

First look at the data

Statistics

MSN

Genetic distances

## Outline III

AMOVA

Hierarchical clustering

NJ (and UPGMA) tree

PCoA

Tasks

### 6 SNP

PCA and NJ

### 7 DAPC

Bayesian clustering

Discriminant analysis and visualization

Tasks

### 8 Structure

Running Structure from R

# Outline IV

ParallelStructure on Windows  
Post processing

## 9 Spatial analysis

Moran's I

sPCA

Monmonier

Mantel test

Geneland

Plotting maps

Tasks

## 10 Trees

Manipulations

MP

# Outline V

Seeing trees in the forest

Comparisons

Notes about plotting the trees

Tasks

## 11 Evolution

PIC

Autocorrelation

Decomposition

PGLS

GEE

Phylosignal

pPCA

Ancestral state

# Outline VI

Phenogram  
Tasks

## 12 The end

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-2021>
  - Český: <https://trapa.cz/cs/kurz-molekularni-data-r-2021>
- 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)
- 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 (slide 16) 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
- [Kate](#)
- [KWrite](#)
- [Vim](#)
- [GNU Emacs](#)
- [Geany](#)
- [Bluefish](#)
- Open multiple files
- 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 (slide 16)...

# 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...
  - 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 (its grammatical structure and what and how 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 — **read also 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, ...
  - Alignments
  - NJ, UPGMA, PCoA, DAPC, Bayesian clustering, ML, maximum parsimony, ...
  - Character evolution, ancestral state reconstructions, ...
  - 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, proteomics, ...)

# 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

Tasks

Packages for our work

# About R

- Project for Statistical Computing
- Open-source — freely available with source code — anyone can use and modify 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) I

- Most users use some GUI – it is more convenient than plain command line
- Provide more comfortable interface for work with scripts (source code highlight, ...), overview of loaded packages and variables, easier work with figures, ...
- RStudio <https://rstudio.com/products/rstudio/> – probably the most common, multi-platform, very powerful
- RKWard <https://rkward.kde.org/> – feature very rich, developed mainly for Linux, available also for another operating systems
  - RKWard must be compiled for the same version as you use
  - If downloading for Windows or macOS, check your version of R and download respective version of RKWard
  - On Linux, do not mix package repositories, ensure RKWard is compiled for your R version (typically install both from same resource)
- R commander (Rcmdr) <https://www.rcommander.com/> – multi-platform, not so rich as previous



## Graphical user interfaces (GUI) II

- 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>
- Emacs speaks statistics, R Tools for MS Visual Studio, Rattle, Radiant, ...
- And more...
- Pick one you like (from above list or any else) and install it...

# RKward

The screenshot displays the RKward R environment. On the left, a file browser shows the project structure with folders like 'GlobalEnv', 'Random seed', and 'afla'. The central script editor contains R code for hierarchical clustering using the 'afla' package. The code includes loading the 'afla' package, reading data from 'afla.genind', and performing clustering with 'find.clusters' and 'table.pop'. The output is a dendrogram on the right, which visualizes the hierarchical clustering of 60 samples (POP7568 to POP6045) into two main clusters: 'Inferred cluster 1' and 'Inferred cluster 2'. The dendrogram shows a clear separation between the two clusters, with 'Inferred cluster 1' containing samples POP7568 through POP6249 and 'Inferred cluster 2' containing samples POP6249 through POP6045.

```

49 srs.genind <- df.genind(0=srs, sep="||", pop
50 srs.genind$other$locid <- srs$locid[1,])
51 srs.genind
52
53 # K-Ind
54
55 # AFL
56 # Hierarchická klasifikace - K-way cluster
57 # Získání výstupní informací o K-Ind oběh
58 afl.kfind <- find.clusters(0=afl.genind, sta
59 # Poliska se na výpis
60 table.pop(afl.genind, afl.kfind$grp)
61 afl.kfind
62
63 # Ústí: Interpretace přířazení plošných populac
64 table.value(table.pop(afl.genind, afl.kfind
65 # Průběh se na výpis
66 afl.kfind$
67 table.pop(afl.genind, afl.kfind$grp)
68 afl.kfind
69
70 # Ústí: Interpretace přířazení plošných populac
71 table.value(table.pop(afl.genind, afl.kfind
72 # Průběh se na výpis
73 afl.kfind$
74 afl.kfind$ <- find.clusters(0=afl.genind, b
75 # Poliska se na výpis
76 table.pop(afl.genind, afl.kfind$grp)
77 afl.kfind$
78 # Ústí: Interpretace přířazení plošných populac
79 table.value(table.pop(afl.genind, afl.kfind
80 # Průběh se na výpis
81 afl.kfind$
82 srs.kfind <- find.clusters(srs.genind, stat
83 # Poliska se na výpis
84 table.pop(srs.genind, srs.kfind$grp)
85 srs.kfind
86
87 # Ústí: Interpretace přířazení plošných populac
88 table.value(table.pop(srs.genind, srs.kfind
89 # Průběh se na výpis
90 srs.kfind$ <- find.clusters(0=srs.genind, b
91 # Poliska se na výpis
92 table.pop(srs.genind, srs.kfind$grp)
93 srs.kfind$
94 # Ústí: Interpretace přířazení plošných populac
95 table.value(table.pop(srs.genind, srs.kfind
96 # Průběh se na výpis
97 srs.kfind$
  
```

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

The screenshot illustrates the Rkward interface with several key elements:

- Main Window:** Displays R code for SNP analysis, including commands like `nothofagus$seq`, `nothofagus$sequences`, `getSequence`, `get_annot`, `getAnnot`, `closeBank`, `convert_sequences`, and `nothofagus.dna`.
- Settings Menu:** Opened, with 'Configure Editor...' selected.
- Configure - Rkward Dialog:**
  - Appearance Tab:** Shows 'Dynamic Word Wrap' set to 'Follow Line Numbers' and 'Align dynamically wrapped lines to indentation depth' set to '80% of View Width'.
  - Font & Color Schemas Tab:** Shows the 'Vim (dark)' schema selected. It displays color swatches for 'Text Area', 'Selected Text', 'Current Line', 'Search Highlight', 'Replace Highlight', 'Icon Border', 'Background Area', 'Line Numbers', 'Current Line Number', and 'Separator'.

## RStudio

The screenshot shows the RStudio environment with a script editor on the left containing R code for population genetics analysis. The console on the bottom left displays the R version and copyright information. The environment pane on the right lists installed and required packages. An 'Update Packages' dialog box is open in the center, showing a table of packages with their current and available versions.

**Script Editor Code:**

```

27 srs3n.loci.missingno <- as.loci(srs3n.missingno)
28 srs3n.loci.missingno
29
30 # Summary statistics
31 srs3n.sum <- summary(srs3n.missingno)
32 plot(srs3n.sum$Hexp, srs3n.sum$Hobs, main = "Observed vs. expected heterozygosity")
33 abline(0, 1, col = "red")
34 t.test(srs3n.sum$Hexp, srs3n.sum$Hobs, paired = TRUE, var.equal = TRUE)
35 bartlett.test(list(srs3n.sum$Hexp, srs3n.sum$Hobs))
36 par(mfrow=c(2,2))
37 plot(srs3n.sum$pop.eff, srs3n.sum$pop.nall, xlab="Populations sample size", ylab="Number of alleles", main="Alleles numbers and sample sizes")
38 text(srs3n.sum$pop.eff, srs3n.sum$pop.nall, lab.names=srs3n.sum$pop.eff)
39 barplot(srs3n.sum$loc.nall, ylab="Number of alleles", main="Number of alleles per locus", names.arg=srs3n.missingno$loc.names, las="x")
40 barplot(srs3n.sum$Hexp~srs3n.sum$Hobs, main="Heterozygosity: expected-observed", ylab="Hexp - Hobs", names.arg=srs3n.missingno$c.names, las="x")
41 barplot(srs3n.sum$pop.eff, main="Sample sizes per species", ylab="Number of genotypes", las=3)
42 dev.off()
43 svg("poppr_303d.svg", family="sans", bg="white", antialias="subpixel")
44 poppr(srs3n.genind, total=TRUE, sample=10000, method=4, missing="geno", cutoff=0.15, quiet=FALSE, clonecorrect=FALSE, hist=TRUE, missamp=1, legend=TRUE)
45 dev.off()

```

**Environment Pane:**

```

library(ppr) # Required by pegas
library(pegas)
library(poppr)
library(sp) # Required by rworldmap
library(rworldmap)
library(ade4) # Required by pegas, adegenet and poppr
library(adegenet) # Required by pegas and poppr
library(ape) # Required by pegas
library(pegas)
library(poppr)
library(sp) # Required by rworldmap
library(rworldmap)
srs3n.sum <- summary(srs3n.missingno)
theta.msat(srs3n.loci.missingno)

```

**Update Packages Dialog:**

Package	Installed	Available	NEWS
<input checked="" type="checkbox"/> BH	1.55-0-2	1.55-0-3	<input type="checkbox"/>
<input checked="" type="checkbox"/> HSAUR	1.3-4	1.3-5	<input type="checkbox"/>
<input checked="" type="checkbox"/> openair	1.0	1.1-0	<input type="checkbox"/>
<input checked="" type="checkbox"/> rgi	0.95.1108	0.95.1201	<input type="checkbox"/>
<input checked="" type="checkbox"/> sparseM	1.05	1.6	<input type="checkbox"/>
<input checked="" type="checkbox"/> strame	2012.5-1	2015.1-1	<input type="checkbox"/>
<input checked="" type="checkbox"/> THdata	1.0-5	1.0-6	<input type="checkbox"/>

**Console Output:**

```

R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-suse-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |

```

**Environment Pane (Packages):**

Package	Version
abc	2.0
abind	1.4-1
abind	1.4-0
Ace	0.8-8
adepack	1.3-3-3
adepack	1.3-3-3
ad4	1.2
ad4	1.2
ada	2.0-3
adapMCMC	1.1
adef	1.6-2
adef	1.6-2
adefGUI	0.2-8
adegenet	1.4-2
adegenetics	0.9
adehabitat	1.8.15
adehabitatHR	0.4.12
adehabitatHS	0.3.10
adehabitatLT	0.3.17
adehabitatMA	0.3-8

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

The screenshot displays the RStudio environment with several windows and settings:

- Main Editor:** Shows R code with soft line wrap enabled. The code includes comments and function definitions like `plot <- function(x, y, ...)`.
- Console:** Displays the output of the code execution, including a list of installed packages.
- Options Dialog:**
  - General:** Shows the RStudio theme set to 'Modern'.
  - Code:** Shows 'Soft wrap' is checked.
  - Appearance:** Shows 'Use light theme' is checked.
  - Editor font:** Shows 'Editor Font size' is set to 11.
  - Editor theme:** Shows 'Solarized Dark' is selected.
  - Terminal:** Shows 'Terminal' is checked.
- Package Management Dialog:**
  - CRAN mirror:** Shows 'Choose HTTP(S) CRAN Mirror' dialog with 'Softeq (Prague) - Sofia University' selected.
  - Package development:** Shows options like 'Use devtools package functions if available' and 'Automatically rewrites editor to build errors'.
- Help Menu:** Shows the 'Global Options...' option highlighted.
- Code Editor:** Shows R code with line numbers 1-13 and comments explaining the code.

# MS Windows & Apple macOS

- 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 [macOS](#), 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 packages `patterns-openSUSE-devel_basis` and `gcc-fortran` for compilation of R packages when installing them from R (only some R packages are available in openSUSE 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/>
- 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](#)

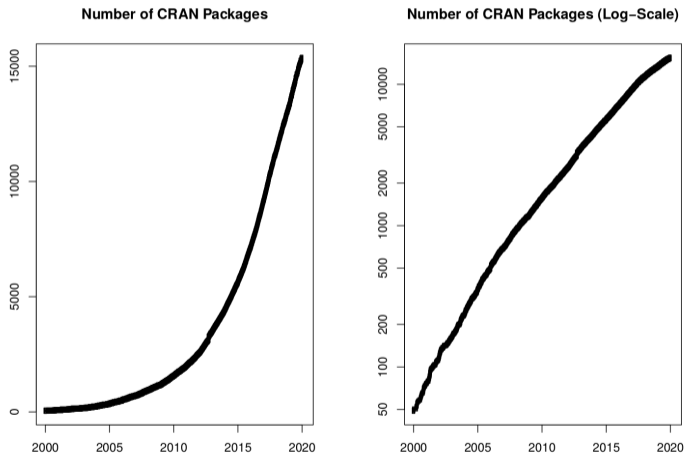
# Important note about names of directories

- 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 than creating new directory/user)
- If the user has e.g. on Windows (where the problem is the biggest) path like `C:/Documents and Settings/Šíleně úpějící kůň/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 macOS 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 16,000 packages + many orphaned and archived — abandoned by developers, might be working)
- Bioconductor <https://bioconductor.org/> — mainly bioinformatics packages, genomic data (over 1,900 packages)
- R-Forge <https://r-forge.r-project.org/> (over 2,100 packages)
- RForge <https://www.rforge.net/> (much smaller)
- And more ([GitHub](#)), custom webs, ...
- 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

# CRAN keeps growing...

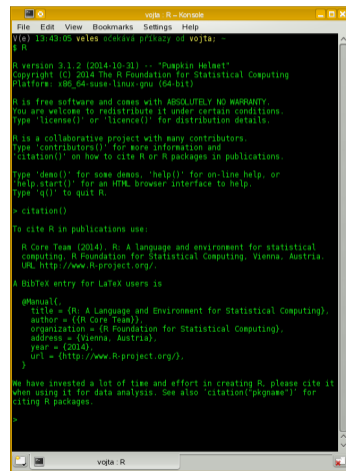


<https://journal.r-project.org/archive/2019-2/cran.pdf>

# 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



```
vojta R - Konsole
File Edit View Bookmarks Settings Help
V(e) 13:43:05 veles ocekava prikazy od vojta: -
$ R
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-suse-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 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 commas):  
`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 require(sos) # More comprehensive search from packages
9 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
- Output value(s)
- About author(s)
- References to cite
- Example code



# Importance of working directory

- Very important point to get familiar with in R
- Default place to load/save, import/export data/results
  - It changes paths — one of the most common mistakes — something (input file, ...) 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**
  - When starting RStudio/RKward, R usually starts in user's home directory or so
- RStudio and RKward also save session information (list of opened files, ...) — very convenient
- Regularly save your work to prevent loses in case of crash or any other accident

# 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 plain R (**File | working directory**), in RStudio (**Session | Set working directory**) or in RKWard (**Workspace | Set Working Directory**) set it in menu or by the above command
- R saves history of commands into file `.Rhistory` file within working directory (by default hidden in Linux/macOS)
- 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 (files can be renamed)
- RStudio and RKWard help with this very much

# Types of objects

- As any programming language, R has plenty of types of objects (variables) with different features, usage and aims
- **Vectors** — numbers, characters, boolean
- **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 (“sheet of Excel”)
- **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

- `AABin` — stores amino acid sequences (aligned or not)
- `alignment` — aligned sequences (package `seqinr`)
- `dapc` — results of DAPC
- `dist` — distance matrices
- `DNABin` — stores DNA sequences (aligned or not)
- `genind` — stores various genetic information for individuals
- `genlight` — variant of `genind` to store large multiple genomes
- `genpop` — like `genind`, but on population level
- `haplonet` — networks without reticulation
- `haplotype` — unique sequences from `DNABin`

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

- `hclust` — output of hierarchical clustering, can be converted to phylo
- `loci` — extension of data frame (DF), stores information about loci
- `matching` — binary phylogenetic trees
- `matrix` — general matrix (numeric or not)
- `pco` ; `dudi` — results of PCA, PCoA, ...
- `phyDat` — “preparation” of data for some phylogenetic analysis (usually sequences and characters)
- `phylo` — phylogenetic information, typically trees
- `phylo4` — derived from phylo (more data), S4 instead of S3
- `SNPbin` — stores large SNP data for single genome

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

- `spca` – results of sPCA
- `treeshape` – derived from `hclust`
- `vcfR` – imported (and possibly edited) VCF
- 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("data.frame")` (Or any other class name of loaded package) – there are information about slots within that classes you can access
- Common task is conversion among various formats – functions commonly require different input format

# 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

## Conversions among data types II

From	To	Command	Package
treeshape	phylo	<code>as.phylo(x)</code>	apTreeshape
splits	phylo	<code>as.phylo(x)</code>	phangorn
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



# Conversions among data types III

From	To	Command	Package
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
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

# Conversions among data types IV

From	To	Command	Package
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
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

# Conversions among data types V

From	To	Command	Package
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

R doesn't ask neither notifies when overwriting objects! Be careful!

```
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 dim(y) # Dimensions of "y"
15 y[,2] # Prints second column
```

# Basic operations with data II

```

1 y[3,] # Prints third row
2 y[4,3] # Prints element from fourth row and third column
3 c(x, y[4,]) # Concatenate "x" and 4th row of "y"
4 x2 <- c(x, 3, 2, 1) # Concatenate "x" and values "3", "2" and "1"
5 x <- y[2,] # Replaces "x" by second row of "y" (no warning)
6 rm(x) # Deletes x (no warning)
7 y[,1:3] # Prints first through third column of the matrix
8 y[3,] <- rep(x=20, each=4) # Replaces third line by value of 20
9 y[y==20] <- 10 # If value of y's element is 20, replace it by 10
10 summary(y) # Basic statistics - according to columns
11 colnames(y) <- c("A", "B", "C", "D") # Set column names
12 # Objects and functions are without quotation marks; files and text with
13 colnames(y) # Prints column names, use rownames() in very same way
14 y # See modified object
15 y[, "C"] # Prints column C (R is case sensitive!)
16 y[,c("C", "B")] # Extract columns "C" and "B"

```

# Basic operations with data III

```

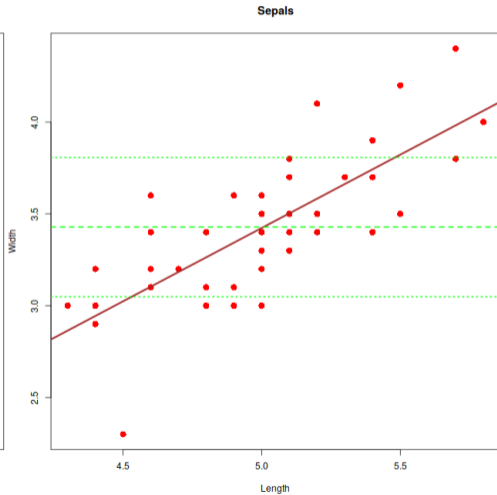
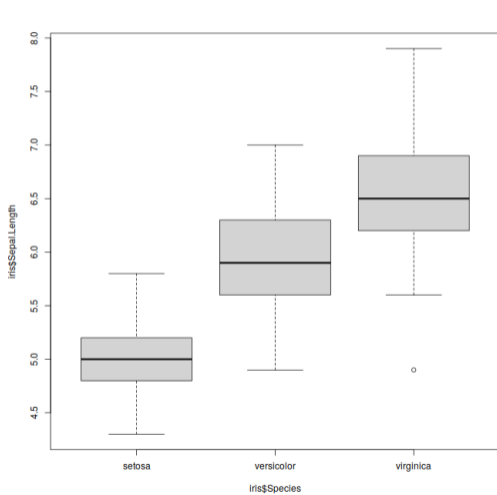
1 t(y) # Transposes the matrix
2 diag(y) # Get diagonal of the matrix
3 # Replace diagonal by repetition of values 50 and 100
4 diag(y) <- rep(x=c(50, 100), times=2)
5 y # See modified object
6 y <- as.data.frame(y) # Turns into DF (see other functions as.*)
7 class(y) # Is it data frame now?
8 y[y==17] <- "NA" # Removes values of 17 (NA = not available = missing)
9 y$B # Gets variable B of data frame y ($ works similarly in S3 objects)
10 # When loading saved project, you have to load again libraries and
11 # scripts (see further), data objects are restored
12 # This can be conveniently done in RStudio/RKward
13 save(list=ls(), file="test.RData") # Saves all objects during the work
14 load("test.RData") # Loads saved R environment with all objects
15 fix(y) # Use to edit matrices, data frames, functions, ...
16 rm(y) # Removing...

```

# Some basic statistics

```
1 summary(iris) # Basic summary statistics
2 # Boxplot comparing sepal lengths of the three species
3 boxplot(formula=iris$Sepal.Length~iris$Species)
4 setosa <- iris[which(iris$Species=="setosa"),] # Extract only setosa
5 # Testing correlation between sepal length and width of setosa
6 cor.test(x=setosa$Sepal.Length, y=setosa$Sepal.Width)
7 # Plot correlation between sepal length and width of setosa
8 plot(x=setosa$Sepal.Length, y=setosa$Sepal.Width, main="Sepals",
9      xlab="Length", ylab="Width", pch=16, cex=1.5, col="red")
10 # Add linear model line
11 abline(reg=lm(setosa$Sepal.Width~setosa$Sepal.Length), col='brown', lwd=3)
12 # Add sepal width mean and standard deviation
13 abline(h=mean(setosa$Sepal.Width), col="green", lwd=2, lty=2)
14 abline(h=mean(setosa$Sepal.Width) + sd(setosa$Sepal.Width), col="green",
15        lwd=2, lty=3)
16 abline(h=mean(setosa$Sepal.Width) - sd(setosa$Sepal.Width), col="green",
17        lwd=2, lty=3)
```

# Figures from previous basic statistics





# Analysis of Variance (ANOVA)

- ANOVA tells us if there is significant difference among means of samples

```

1 aov(formula=iris[["Sepal.Length"]]~iris[["Species"]]) # Output:
2 Call:
3   aov(formula = iris[["Sepal.Length"]] ~ iris[["Species"]])
4 Terms:
5             iris[["Species"]] Residuals
6 Sum of Squares           63.21213  38.95620
7 Deg. of Freedom           2          147
8 Residual standard error: 0.5147894
9 Estimated effects may be unbalanced
10 summary(aov(formula=iris[["Sepal.Length"]]~iris[["Species"]])) # Output:
11              Df Sum Sq Mean Sq F value Pr(>F)
12 iris[["Species"]]  2  63.21  31.606   119.3 <2e-16 ***
13 Residuals       147  38.96   0.265
14 ---
15 Signif. codes:  0 '***', 0.001 '**', 0.01 '*', 0.05 '.', 0.1 ' ', 1

```

# Practice basic operations I

## Tasks I

- 1 Load R training dataset `iris` (`data(iris)`) and read about it (`?iris`).
- 2 Explore it — print it, display only beginning of data (`head()`). How many rows and columns does it have? Which variables? Which species (`levels()`)?
- 3 Extract from `iris` dataset only sepal lengths and save it as new vector.
- 4 Extract from `iris` dataset petal length and species and save it as new data frame.
- 5 Extract from `iris` dataset lines 10–20.
- 6 Read help of `rep` function and
  - 1 Create repetition of five times value of “A”.
  - 2 Create repetition of five times vector of “1, 3”.

# Practice basic operations II

## Tasks II

- 1 Read help of `seq` function and create sequence from 5 to 20 with steps of 0.5 and save it as new vector.
- 2 Extract from that vector 2<sup>nd</sup>, 4<sup>th</sup> and 8<sup>th</sup> position and save it as new vector.
- 3 Remove from that vector 2<sup>nd</sup> element.
- 4 Concatenate previous two vectors into new vector.
- 5 What is difference between `iris["Sepal.Width"]`, `iris[2]`, `iris[["Sepal.Width"]]` and `iris$Sepal.Width`? Compare. What are the different ways good for?
- 6 Try similar tasks as on slide 46 with another species and characters. Try to improve the plots.

# Packages, repositories 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 54)
  - 2 Specify non-CRAN repository every time installing from it (e.g. slide 60) or use special tools (e.g. for **Bioconductor** — slide 59)

# Repositories

- Repositories (internet directories full of R packages — slide 28) can be set via `options(repos=c())` or as `repos` parameter for each `install.packages()` command (slide 54 and onward)
- Repositories don't have to be set as global options, e.g. Bioconductor (slide 59) has its own way to manage packages
- Similar concepts as app stores of Android, iOS, etc.

## Installation of packages in GUI

- **RStudio:** set repositories by command from slide 54 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 54. **OK**. Go to menu **Settings | Manage R packages and plugins...**, click to **Install...**, select and install desired packages...

# Set repositories

```

1 # Basic package installation
2 install.packages("PackageName") # Case sensitive!
3 ?install.packages # Shows all available parameters (options)
4 getOption("repos") # Shows actual repositories
5 options(repos=c("https://mirrors.nic.cz/R/",
6   "https://r-forge.r-project.org/", "https://rforge.net/"))
7 options() # Generic function to modify various settings
8 ?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 – check output and examine why installation failed – commonly due to missing external dependency (read installation output and look for notes about missing libraries, etc.)
- Avoid mixing of several R versions
- After upgrade of R (e.g. from 3.6 to 4.0), user **must** reinstall all packages

# Install packages

- If repositories from slide 54 are not set, it is possible to install in several steps packages from main repository (CRAN) and from another sources (following slides)
- This is the basic and default the most common usage
- After upgrade of R (e.g. from 3.6 to 4.0), all packages must be reinstalled

```
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" - they must be loaded to use them:
7 library(PackageName) # Loads installed package (we will do it on the fly)
8 # Updates installed packages (by default from CRAN)
9 update.packages(repos=getOption("repos"), ask=FALSE)
```

# Install packages needed for the course

```
1 # Install packages
2 # Installation of multiple packages may sometimes fail - install then
3 # packages in smaller groups or one by one
4 install.packages(pkgs=c("ade4", "adegenet", "adegraphics", "adephylo",
5 "akima", "ape", "BiocManager", "caper", "corrplot", "devtools",
6 "gee", "geiger", "ggplot2", "gplots", "hierfstat", "ips", "kdetrees",
7 "lattice", "mapdata", "mapplots", "mapproj", "maps", "maptools",
8 "nlme", "PBSmapping", "pegas", "phangorn", "philentropy", "phylobase",
9 "rentrez", "phytools", "picante", "plotrix", "poppr", "raster",
10 "rgdal", "RgoogleMaps", "Rmpi", "rworldmap", "rworldxtra", "seqinr",
11 "shapefiles", "snow", "sos", "sp", "spdep", "splancs", "StAMPP",
12 "TeachingDemos", "tripack", "vcfR", "vegan"),
13 repos="https://mirrors.nic.cz/R/", dependencies="Imports")
14 # Regularly update installed packages
15 update.packages(ask=FALSE)
16 # Upgrade all packages e.g. from R 3.6 to 4.0
17 install.packages(pkgs=installed.packages())
```



# Install Geneland package

- Since version 4, not in CRAN anymore, check its [manual](#), [GitHub](#) and homepage <https://i-pri.org/special/Biostatistics/Software/Geneland/>
- On Windows install [Rtools](#) first to be able to compile source package on Windows

```
1 # Other packages used when using Geneland
2 # Needed is PBSmapping or mapproj for conversion of coordinates
3 # GUI uses for parallelisation snow and Rmpi
4 # RgoogleMaps (requires rgdal) can be used to plot output on top of Google
5 # map, maptools, shapefiles (requires foreign) and tripack on GIS layer
6 install.packages(pkgs=c("PBSmapping", "mapproj", "rgdal", "RgoogleMaps",
7   "Rmpi", "sp", "maptools", "shapefiles", "snow", "tripack"),
8   repos="https://mirrors.nic.cz/R/", dependencies="Imports")
9 # Install Geneland from GitHub
10 # Devtools package is required to install from GitHub
11 if(! "devtools" %in% installed.packages()) {install.packages("devtools")}
12 # Install Geneland itself
13 devtools::install_github("gilles-guillot/Geneland")
```

# 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, ips seems to develop more and phyloch will probably be deprecated...

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

# 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, see [web](#)) and **use Bioconductor's special helper package**
- Explore available packages: <https://bioconductor.org/packages/release/BiocViews.html>

```
1 # Install CRAN package BiocManager used to manage Bioconductor packages
2 if (!requireNamespace("BiocManager")) install.packages("BiocManager")
3 ?BiocManager::install # See options
4 # Install Bioconductor packages
5 BiocManager::install("muscle") # Simplest usage
6 BiocManager::install(pkgs=c("Biostrings", "muscle"))
7 # Update installed packages
8 BiocManager::install()
9 # Search for Bioconductor packages
10 BiocManager::available() # List everything
11 ?BiocManager::available # See options
```

# Bioconductor and others – differences from another repositories

Bioconductor has its own installation method

```
1 # Standard installation
2 install.packages(c("adegenet", "poppr", "phytools"))
3 update.packages() # 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 # Install CRAN package BiocManager used to manage Bioconductor packages
9 if (!requireNamespace("BiocManager")) install.packages("BiocManager")
10 # Install Bioconductor packages
11 BiocManager::install("muscle") # Simplest usage
12 BiocManager::install(pkgs=c("Biostrings", "muscle"))
13 ?BiocManager::install # See more options
14 BiocManager::install() # Update installed packages
```

# 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 (similarly to BASH, Python, etc.)
  - User must install this software manually
- Clustal (W/X; Omega is not used in the course) <http://clustal.org/>
  - Aligner of sequences (from slide 125)
- Gdal  
<https://trac.osgeo.org/gdal/wiki/DownloadingGdalBinaries>
  - Recommended for conversion of coordinates for Geneland (from slide 260), loading of SHP files, etc.
  - Optional for Windows users, Linux users should use `gdal` and respective development (`-dev(e1)`) packages
- GIMP <https://www.gimp.org/>

## Non-R software II

- Image manipulation, free open-source alternative to products of Adobe or Corel
- Optional, one of possibilities to view and edit output graphics from R
- 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 125)
- MPI <http://fisher.stats.uwo.ca/faculty/yu/Rmpi/>
  - Library for parallelisation used by Rmpi package (optional, recommended especially on Windows; macOS and Linux have more options for parallelisation)
  - 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 125)
- proj <https://proj4.org/download.html>

## Non-R software III

- Required for conversion of coordinates for Geneland (from slide 260)
- Optional for Windows users, Linux users should use `proj` and respective development (`-dev(e1)`) packages
- Rtools <https://cran.r-project.org/bin/windows/Rtools/>
  - **On Windows only** – to be able to compile source packages on Windows
- 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 224)
  - 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

Tasks



# 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
  - Old fashioned, but same mathematical tools can be used to analyze any presence/absence (1/0) data matrices
- **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 (especially **Bioconductor**) has plenty of packages for specialized protein analysis and more
- **Nucleic acid sequences** (in nearly any form) — DNA or RNA
  - From “classical” Sanger sequencing — long individual reads (of single/few genes)
  - From “modern” **HTS (NGS)** — **454 pyrosequencing**, **Illumina**, ... methods
    - Probably most used are **RADseq** scanning whole genome, **HybSeq** and other **target enrichment** methods 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), **Genotyping by sequencing (GBS)**; 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
- Modern methods are quickly developing and able to produce  $\sim 10^{12}$  bp per run and multiplex many individuals
- 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
  - Explore more options yourselves
- 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 already used `iris`)
- 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; for example phylogeny and species traits
5 # of shorebirds from package caper (we will use it much later)
6 data(shorebird, package="caper")
7 # Optional method (load respective library and then data)
8 library(caper) # Library containing (among others) desired data
9 data(shorebird) # Loading the data
10 ?shorebird # See content of the dataset # or ?caper::shorebird
11 ?adegenet::microbov
12 ?adegenet::rupica
13 ?ape::carnivora
    
```

# Our data... I

- We will use...
  - Modified subset of diploid microsatellite data of *Taraxacum haussknechtii* (Asteraceae) from Macedonia by [Zeisek et al 2015](#)
    - Population genetics – genetic structure, characteristics and comparisons of sampled populations
    - Spatial genetics – genetic relationships in spatial context – relationships among populations regarding their spatial position
  - 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](#)
    - Population genetics
    - Species delimitation (genotyping, identification and assignment of species)
  - Small subset of population AFLP data of *Cardamine amara* (Brassicaceae) from Europe by [Karol Marhold](#) and his team
    - Population genetics
  - 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 Bohutínská](#) and her colleagues

## Our data... II

- Population genomics
  - Associations of various traits (physiological, ...) with particular genetic loci/alleles
- Small subset of trees and taxa from phylogeny of *Oxalis* spp. (Oxalidaceae) from South Africa (the Cape region) by [Schmickl et al. 2016](#)
  - Phylogenomics, target enrichment (HybSeq) sequencing of multiple genes – construction of individual gene trees, their evaluation (identification of trees with significantly different topology – genes with different evolutionary pathway) and construction of species trees and networks
- Internal transcribed spacer sequences of *Gunnera* spp. (Gunneraceae) from [Wanntorp et al 2014](#) downloaded from [GenBank](#)
  - Phylogenetic relationships among species with “Gondwana” distribution (south of South America, South Africa, Australia, New Zealand, ...)
- Maturase K (matK) plastid sequences of *Nothofagus* (Nothofagaceae) downloaded from [EMBL-EBI](#) (various authors; mainly from phylogenetic studies)
- Training data from packages
  - Sequence of influenza from USA sampled in several years (package adegenet, part of its vignette)

## Our data... III

- SSRs genotypes of cattle breeds, `?adegenet::microbov` – population genetics (genetic structure)
- *Rupicapra rupicapra* (Bovidae) SSRs genotypes from French Bauges mountains, `?adegenet::rupica` – spatial genetics (spatial distribution of genotypes and their relationships)
- Morphological traits and phylogeny of 17 *Maples* species, `?adephylo::maples` – phylogeny and correlated evolution of characters
- Morphological traits of Carnivora, `?ape::carnivora` – life history patterns: allometric, phylogenetic and ecological associations
- Measurements of *Iris setosa*, *I. versicolor* and *I. virginica*, `?iris` – measurements of morphological characters, their correlations and differences among species
- Phylogeny and morphological traits of shorebirds, `?caper::shorebird` – comparative evolution of characters

## Our data... IV

- Work with microsatellites is in most cases (except some methods taking advance from microsatellite mutational nature) same as with presence-absence data and methods can handle both data types in nearly same fashion
  - Examples are shown mainly with microsatellites, but another (sequencing, presence/absence, ...) 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
- Many methods can process (nearly) any input data type

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)
  - Sequences (usually short from Illumina) mapped to reference
  - Each file contains data for single sample
  - Contains also information about mapping quality etc., complex structure
  - Compressed version of SAM (see below)
  - Special applications are needed to work with BAM files
  - Suffix `*.bam`
- **CSV** (Comma Separated Values)
  - “One sheet of Excel”
  - Common format to store data (traits, coordinates, ...), similar to TSV (see below)
  - 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`
  - Can have many formatting forms — carefully set parameter of import function

# Input/output data formats II

## Representative selection

- **FASTA**

- Simple and popular text format to store genetic sequences (DNA, RNA, proteins)
- Each file contains one or more sequences
- **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 `>`
- Each sequence can be on single line, on multiple lines
- Can store also alignments (practically sequences of same length, with marked gaps and missing data)
- Larger sequences are sometimes compressed
- Suffix usually `*.fasta` (generic, also `*.fas`, `*.fa`, `*.seq`, `*.fsa`), `*.fna` (nucleic acid), `*.ffn` (nucleotides, coding regions), `*.faa` (amino acids), `*.frn` (non-coding RNA), ...

# Input/output data formats III

## Representative selection

- **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
- Probably the most common format for output of modern high throughput sequencing machines (e.g. Illumina), each file contains huge number of sequences
- Commonly compressed by gzip ( \*.gz ), sometimes by other compression methods
- 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, ...)
- File can contain one or more trees

# Input/output data formats IV

## Representative selection

- E.g. `(A, B, (C, D)E)F`; or `(A:0.1, B:0.2, (C:0.3, D:0.4):0.5)`;
- Simple logic, but very hard to read by human
- 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`
- **SAM** (Sequence Alignment Map)
  - Text-based format for storing biological sequences aligned to a reference sequence
  - Each file contains data for one sample

# Input/output data formats V

## Representative selection

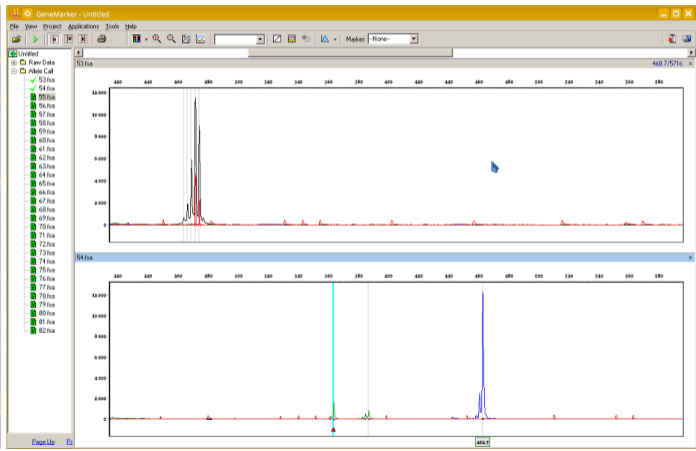
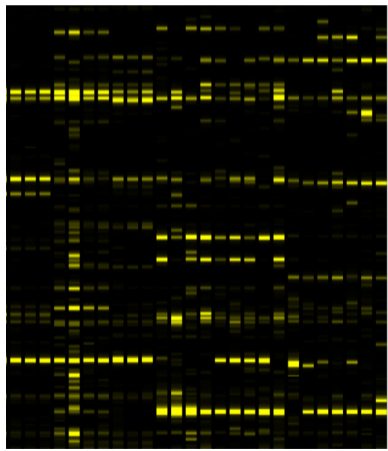
- Structure is relatively complex
- Used by applications like [bamtools](#) or [SAMtools](#)
- Suffix usually `*.sam`
- **TSV** (TAB separated values)
  - “One sheet of Excel”
  - Common format to store data (traits, coordinates, ...), similar to CSV
  - Columns (cells) are separated by tabs (`\t`) – it is important to check structure before import into R (and verify 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 VI

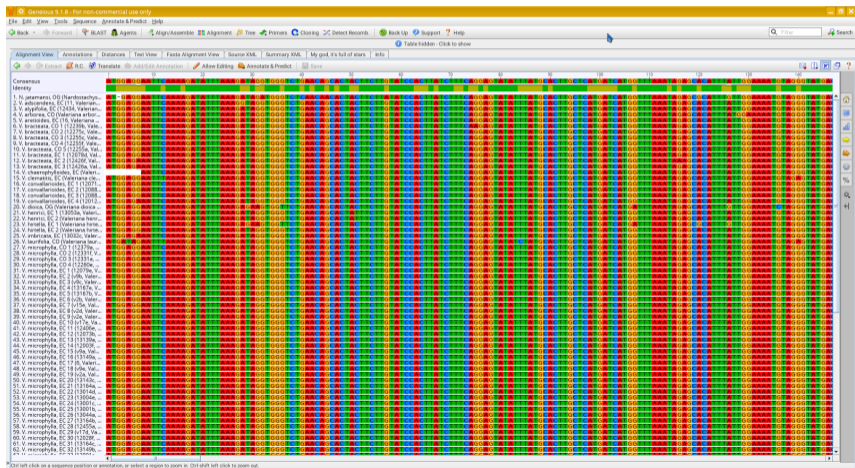
## Representative selection

- **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 methods
  - 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

```

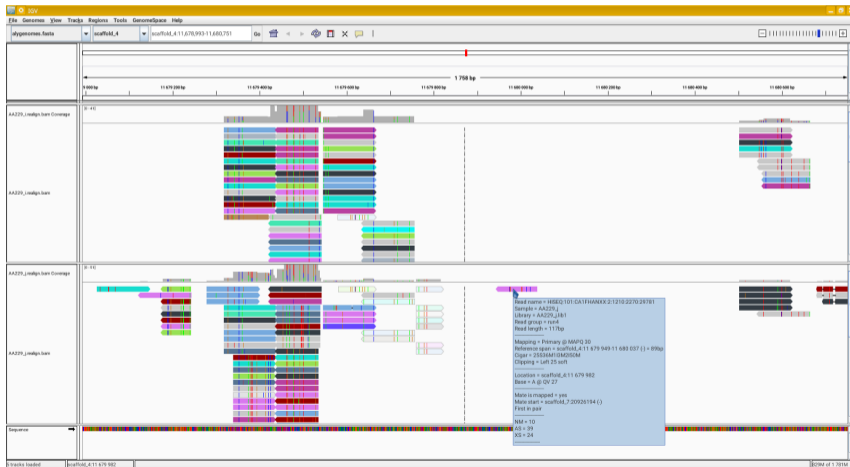
1 > CY013200
2 atgaagactatcattgctttgagctacatthttatgtctggthttcgctcaaaaacttcccctagtgaaaaca
3 ggaaatgacaacagcacagcaacgctgtgcctgggacaccatgcagtgccaacggaacgatcacgaatgat
4 > CY013781
5 atgaagactatcattgctttgagctacatthttatgtctggthttcgctcaaaaacttccccaattgaagtg
6 ... # FASTA continues...
    
```

```

1 @7001425F:141:CAV4JANXX:3:1104:1496:1976 1:N:0:GTGTCC
2 NCGATTCATTTTAGCAATTAGACGTGAAGGTCTCTTGATGAAAGACACTAACGAACTCTTTCCTTGGACACC
3 +
4 #<<BBFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
5 @7001425F:141:CAV4JANXX:3:1104:1250:1987 1:N:0:GTGTCC
6 TCGATTCACTGAACTGAATGTCCGACAACTTTAGTTTGTCGTTTCTACCTCACCAAAGTTCGGAGCTTCGA
7 +
8 ... # FASTQ continues...
    
```

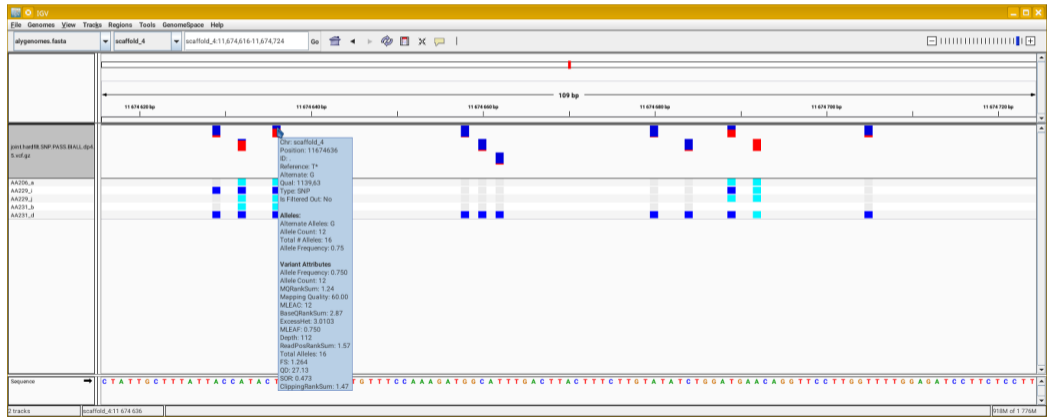
# Examples of data and formats IV – BAM displayed in IGV

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, macOS, ...) 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:`)

## Notes about paths to import the data II

- 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"`)
- 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 (macOS, 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()`)

## Notes about paths to import the data III

- 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 384

### Working in dedicated directory

R always work in some directory (see by `getwd()`) and by default load input files from there (see them by `dir()`) and save output there – relative paths starts there. This is common source of confusion and errors for beginners.

# 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

- Import, basic checking, manipulation, conversion, and export of basic common data types
- Analyses are in further chapters...

```
1 # Load needed libraries
2 library(ape)
3 library(ade4)
4 library(adegenet)
5 library(pegas)
6 library(poppr)
7 # Now let's start to work...
```



# Load microsatellite data

```

1 # Source data
2   pop  msta93 msta101 msta102 msta103 msta105 msta131 ...
3 H01  He  269/269 198/198 221/223 419/419 197/197 196/196 ...
4 H02  He  275/283 198/198 221/223 419/419 193/193 168/190 ...
5 ...  ...      ...      ...      ...      ...      ...      ...
6 # Loading the data
7 # Load training data (Taraxacum haussknechtii from Macedonia)
8 hauss.loci <- read.loci(file="https://soubory.trapa.cz/rcourse/
9   haussknechtii_ssrs.txt", header=TRUE, loci.sep="\t", allele.sep="/",
10  col.pop=2, col.loci=3:14, row.names=1) # \t means TAB key
11 hauss.loci # Data control
12 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 pop(hauss.genind) # See population names
4 hauss.genind$pop # "$" separates extra slots within object

```

```

1 # Source data
2 Ind      lon    lat
3 H01    21.3333  41.1
4 ...      ...    ...
5 # Read coordinates
6 hauss.coord <- read.csv("https://soubory.trapa.cz/rcourse/haussknechtii_
7   coordinates.csv", header=TRUE, sep="\t", quote="", dec=".", row.names=1)
8 hauss.coord

```

- Coordinates can be in any projection or scale — according to aim
- Take care about parameters of `read.csv()` ! See `?read.csv`
- `pegas::geod` calculates geodesic distances (on Earth surface)

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

**Always 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 presence/absence (e.g. 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  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, ploidy and their mixing doesn't matter
- Not all methods introduced later 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], type="PA")
15 indNames(aflp.genind) <- amara.aflp[,1] # Add individual names
16 aflp.genind
17 # You can add any other variables into genind$other$XXX

```



# Another data manipulation

```

1 ?genind2df # adegenet - export into data frame
2 ?genind2genalex # poppr - export for genalex
3 ?splitcombine # poppr - edits population hierarchy
4 ?popsb # 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 manuals (?...) of the functions before usage
    
```

- SNPs can be into genind imported in same way as AFLP (PA)
- `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, character, etc. 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 is able to store mixed ploidy data, but not all analysis are able to handle it

# 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 (Adegenet toy data)
3 usflu.dna <- read.dna(file="https://adegenet.r-forge.r-project.org/
4   files/usflu.fasta", format="fasta")
5 class(usflu.dna) # Check the object
6 usflu.dna # Check the object
7 # Another possibility (only for FASTA alignments, same result):
8 usflu.dna2 <- fasta2DNABin(file="https://adegenet.r-forge.r-project.org
9   /files/usflu.fasta") # Normally keeps only SNP - see ?fasta2DNABin
10 class(usflu.dna2) # Check the object
11 usflu.dna2 # Check the object
12 as.character(usflu.dna2)[1:5,1:10] # Check the object
13 dim(usflu.dna2) # Does it have correct size?
14 # Read annotations
15 usflu.annot <- read.csv("https://adegenet.r-forge.r-project.org/files/
16   usflu.annot.csv", header=TRUE, row.names=1)
17 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 sequence data in NEXUS (similar to reading FASTA)
6 ?read.nexus.data
    
```

- RNA or protein sequences can be handed in same way – see `?read.dna` and `?read.FASTA`
- For nucleic acid sequences there is R class `DNABin`, for protein sequences `AABin` – they are handled in same way
- Do not confuse `read.nexus.data` (reads sequences) and `read.nexus` (reads trees)

# Import sequences from GenBank

- Data from <https://www.ncbi.nlm.nih.gov/popset/22854787> (*Gunnera* spp., phylogenetic study of [Wanntorp et al. 2002](#))

```

1 # Importing sequences according to sequence ID
2 gunnera.dna <- read.GenBank(c("AF447749.1", "AF447748.1", "AF447747.1",
3 "AF447746.1", "AF447745.1", "AF447744.1", "AF447743.1", "AF447742.1",
4 "AF447741.1", "AF447740.1", "AF447739.1", "AF447738.1", "AF447737.1",
5 "AF447736.1", "AF447735.1", "AF447734.1", "AF447733.1", "AF447732.1",
6 "AF447731.1", "AF447730.1", "AF447729.1", "AF447728.1"))
7 gunnera.dna
8 class(gunnera.dna)
  
```

- To be converted into `genind` (useful for many population genetic analysis), sequences in `DNABin` must be aligned (from slide 125)
- To query on-line database as through web we use [seqinr](#) (next slide) or [rentrez](#) (slide after)

# 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",query="SP=Nothofagus AND K=rbcl",
7   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 nothofagus.annot <- getAnnot(nothofagus[["req"]]) # Get annotations
13 nothofagus.annot
14 closebank() # Close the bank when work is over
15 # Convert sequences from a list to DNABin (functions as.DNABin*)
16 nothofagus.dna <- as.DNABin.list(nothofagus.sequences)
17 nothofagus.dna # See it
```

# Query NCBI databases

```
1 library(rentrez)
2 entrez_dbs() # Genetic banks available for rentrez
3 entrez_db_summary("nucleotide") # Brief description of what the database is
4 # Set of search terms that can used with this database
5 entrez_db_searchable("nucleotide")
6 # Search the database and get IDs of matched records
7 nothofagus.search <- entrez_search(db="nucleotide",
8   term="Nothofagus[ORGN] AND rbcL[GENE]")
9 nothofagus.search
10 # Fetch desired records according to their IDs
11 nothofagus.fasta <- entrez_fetch(db="nucleotide",
12   id=nothofagus.search[["ids"]], rettype="fasta")
13 nothofagus.fasta
14 # Conversion into DNABin requires saving to disk as FASTA and then loading
15 write(x=nothofagus.fasta, file="nothofagus.fasta")
16 nothofagus.dna <- read.dna(file="nothofagus.fasta", format="fasta")
17 nothofagus.dna
```

# Importing SNP

- Import from **PLINK** requires saving of data with option “`--recodeA`”

```
1 ?read.PLINK # How to read PLINK files
```

- 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="https://adegenet.r-forge.r-project.org/files/usflu.fasta",
3  quiet=FALSE, saveNbAlleles=TRUE)
4 usflu.genlight # See genlight
5 ?fasta2genlight # Function has several options to speed up reading
6 # If it crashes (on Windows), try to 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](#), `?read.snp`

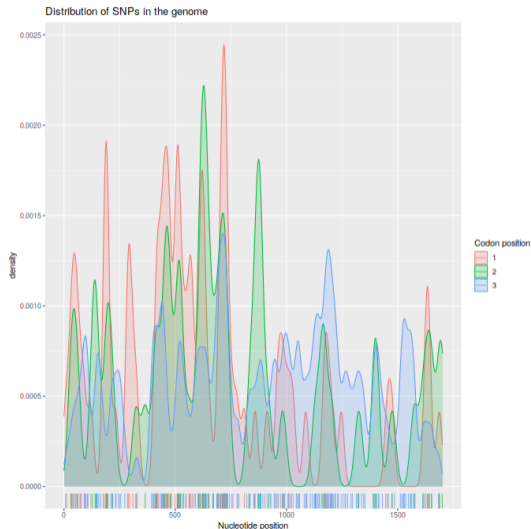


# Checking SNPs

```

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   usflu.dna), codon=FALSE)
6 # Position of polymorphism within
7 # alignment-differentiating codons
8 snpossi.plot(as.matrix(usflu.dna))
9 # When converting DNABin to genind
10 # only polymorphic loci are kept -
11 # threshold for polymorphism can
12 # be arbitrary (polyThres=...)
13 usflu.genind2 <- DNABin2genind(x=
14   usflu.dna, polyThres=0.01)
15 usflu.genind2 # See it

```



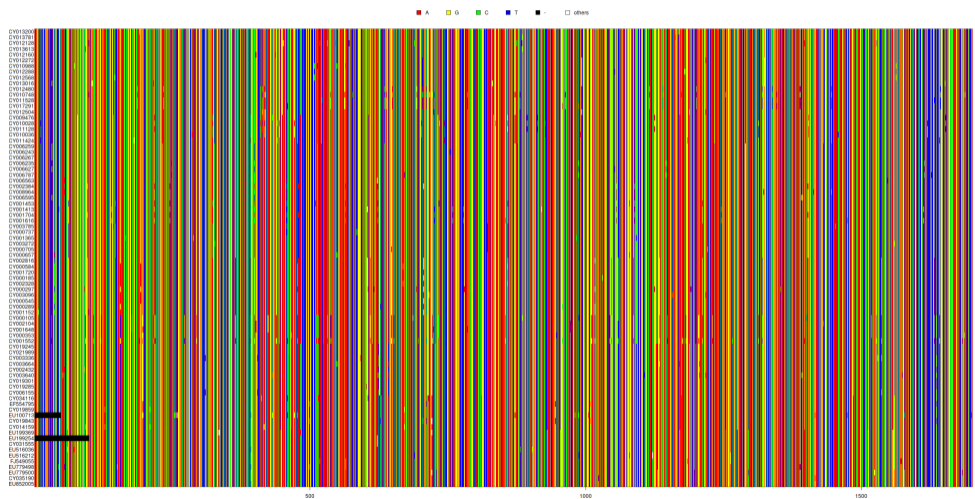
# Checking sequences

```

1 # Test is distribution of SNPs is random (1000 permutations)
2 snpposi.test(as.matrix(usflu.dna))
3 pegas::nuc.div(x=usflu.dna) # Nucleotide diversity
4 ape::base.freq(x=usflu.dna) # Base frequencies
5 ape::GC.content(x=usflu.dna) # GC content
6 # Number of times any dimer/trimer/etc oligomers occur in a sequence
7 # Note: count() requires single sequence as DNABin/character
8 seqinr::count(seq=as.character.DNABin(gunnera.dna[["AF447749.1"]]),
9   wordsize=3)
10 # View sequences - all must be of the same length
11 # Function "imag.DNABin" requires as input matrix
12 # So that sequences must be of same length
13 image.DNABin(x=usflu.dna)
14 # Sequences must be of same length - as.matrix.DNABin() can help
15 image.DNABin(x=as.matrix.DNABin(usflu.dna))

```

# U.S. flu 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 into working directory input file <https://soubory.trapa.cz/rcourse/arabidopsis.vcf.gz> and reference sequence <https://soubory.trapa.cz/rcourse/alygenomes.fasta>
- 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; other option is usage of [VariantAnnotation](#)

```

1 library(vcfR) # Required library
2 # Pick up downloaded file 'arabidopsis.vcf.gz' from the disk
3 arabidopsis.vcf <- read.vcfR(file=file.choose())
4 # File choose dialog can open in background - search for it :- )
5 # Or directly load remote file
6 arabidopsis.vcf <- read.vcfR
7   (file="https://soubory.trapa.cz/rcourse/arabidopsis.vcf.gz")

```

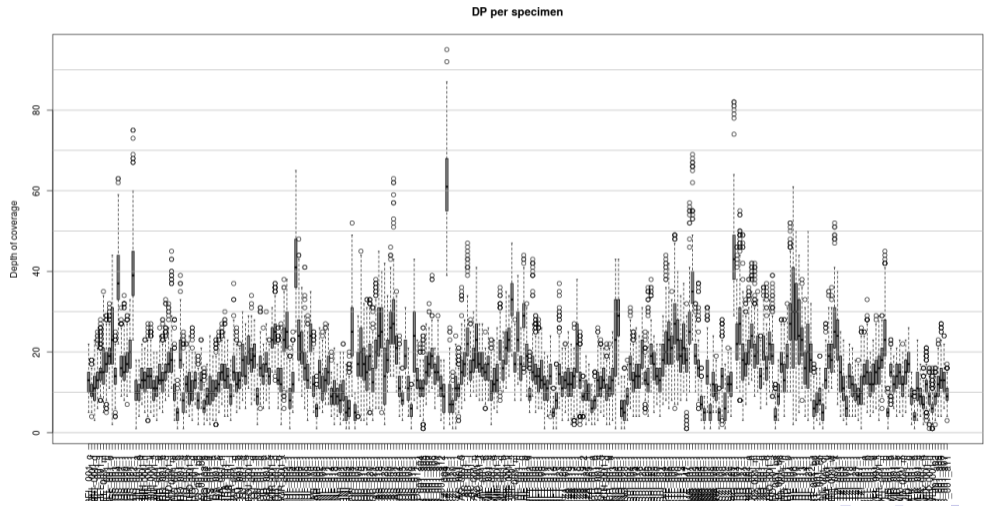
# Checking VCF

```
1 # It returns object of class vcfR-class
2 ?read.vcfR # See more import options
3 ?pegas::read.vcf # This one returns list of objects loci and data.frame
4 arabidopsis.vcf
5 head(arabidopsis.vcf)
6 arabidopsis.vcf@fix[1:10, 1:5]
7 strwrap(arabidopsis.vcf@meta[1:21])
8 queryMETA(x=arabidopsis.vcf)
9 queryMETA(x=arabidopsis.vcf, element="DP")
10 queryMETA(x=arabidopsis.vcf, element="FORMAT.+DP")
11 queryMETA(x=arabidopsis.vcf, element="FORMAT=<ID=DP")
12 head(x=getFIX(x=arabidopsis.vcf))
13 head(x=is.polymorphic(x=arabidopsis.vcf, na.omit=TRUE))
14 head(x=is.biallelic(x=arabidopsis.vcf))
15 arabidopsis.vcf@gt[1:10, 1:4]
16 # See description of depth of coverage (DP) slot
17 strwrap(x=grep(pattern="ID=DP,", x=arabidopsis.vcf@meta, value=TRUE))
```

# Checking depth of coverage (DP)

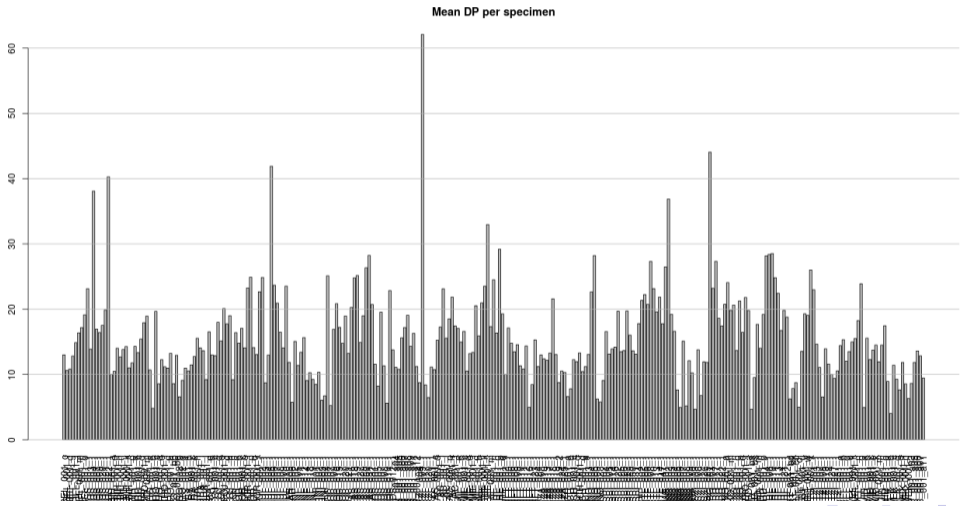
```
1 arabidopsis.vcf.dp <- extract.gt(x=arabidopsis.vcf, element="DP",
2   as.numeric=TRUE) # GT:GQ:DP:HQ
3 dim(arabidopsis.vcf.dp) # See it
4 head(arabidopsis.vcf.dp)
5 # Boxplot of DP
6 boxplot(x=arabidopsis.vcf.dp, col="#808080", ylab="Depth of coverage",
7   las=3)
8 title("DP per specimen")
9 abline(h=seq(from=0, to=90, by=10), col="#b3b3b3")
10 # Bar plot of mean DP
11 barplot(apply(X=arabidopsis.vcf.dp, MARGIN=2, FUN=mean, na.rm=TRUE), las=3)
12 title("Mean DP per specimen")
13 abline(h=seq(from=0, to=60, by=10), col="#b3b3b3")
14 # Heatmap of DP (subset)
15 heatmap.bp(x=arabidopsis.vcf.dp[1:100,1:100], col.ramp=rainbow(n=100,
16   start=0.1)) # Subset - only first 100 loci and individuals
17 title("DP per specimens and loci")
```

# DP per specimen





# Bar plot of mean DP





# Extract the genotype quality (GQ)

```
1 # Extract the GQ
2 arabidopsis.vcf.gq <- extract.gt(x=arabidopsis.vcf, element="GQ",
3   as.numeric=TRUE)
4 dim(arabidopsis.vcf.gq)
5 arabidopsis.vcf.gq[1:10,1:10]
6 # Heatmap of GQ (subset)
7 heatmap.bp(x=arabidopsis.vcf.gq[1:100,1:100])
8 # Bar plot of mean GQ
9 barplot(apply(X=arabidopsis.vcf.gq, MARGIN=2, FUN=mean, na.rm=TRUE), las=3)
10 abline(h=seq(from=0, to=90, by=5), col="grey")
11 # Boxplot of GQ
12 boxplot(arabidopsis.vcf.gq, las=2, main="Genotype Quality (GQ)")
13 abline(h=seq(from=0, to=100, by=5), col="grey")
14 # Basically same as work with DP...
```

# Missing data

```

1 # Extract information about missing data
2 arabidopsis.vcf.miss <- apply(X=arabidopsis.vcf.dp, MARGIN=2,
3   FUN=function(x){sum(is.na(x))})
4 arabidopsis.vcf.miss <- arabidopsis.vcf.miss/nrow(arabidopsis.vcf)
5 # Bar plot of missing data
6 barplot(height=arabidopsis.vcf.miss, ylab="Percentage of missing data",
7   las=2)
8 abline(h=seq(from=0, to=1, by=0.05), col="grey")
9 # Histogram of frequencies of missing data
10 arabidopsis.vcf.missg <- apply(X=arabidopsis.vcf.dp, MARGIN=1,
11   FUN=function(x){sum(is.na(x))})
12 arabidopsis.vcf.missg <-
13   arabidopsis.vcf.miss/ncol(arabidopsis.vcf@gt[, -1])
14 hist(x=arabidopsis.vcf.missg, xlab="Missingness (%)")
15 abline(h=seq(from=0, to=350, by=25), col="grey")
16 # Set abline parameters according to your data

```

# Remove non-biallelic loci and indels

- This is commonly done with SNPs as many downstream analysis are well defined only for biallelic loci (and do not work well with other loci)
- Other than biallelic loci are often suspicious of being laboratory/computational artefact

```
1 # Remove indels
2 arabidopsis.vcf <- extract.indels(x=arabidopsis.vcf)
3 # Remove non-biallelic loci
4 arabidopsis.vcf <- arabidopsis.vcf[is.biallelic(x=arabidopsis.vcf), ]
5 # See result
6 arabidopsis.vcf
```

# ChromR - filtration of VCF I

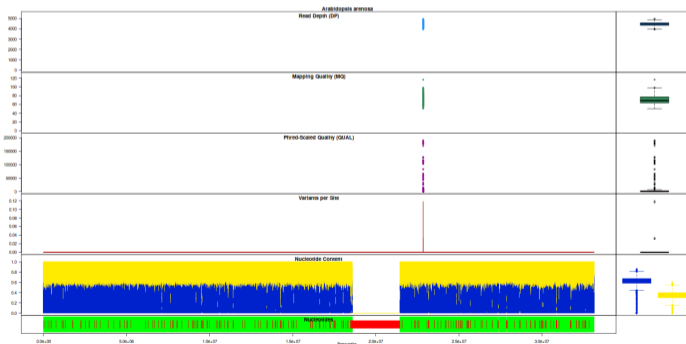
```
1 # Loading reference sequence - download
2 # https://soubory.trapa.cz/rcourse/alygenomes.fasta into working directory
3 arabidopsis.dna <- read.dna(file="alygenomes.fasta", format="fasta")
4 arabidopsis.dna
5 # Conversion into chromosome object (ChromR)
6 arabidopsis.chrom <- create.chromR(vcf=arabidopsis.vcf,
7   name="Arabidopsis arenosa", seq=arabidopsis.dna)
8 arabidopsis.chrom
9 plot(arabidopsis.chrom)
10 # Masking sites with too low/high DP and/or MQ
11 arabidopsis.chrom.mask <- masker(x=arabidopsis.chrom, min_QUAL=1, min_DP=8,
12   max_DP=5000, min_MQ=40, max_MQ=200)
13 arabidopsis.chrom.mask
14 plot(arabidopsis.chrom.mask)
15 variant.table(arabidopsis.chrom.mask)
16 # Saving mask into new object
17 arabidopsis.chrom.fin <- proc.chromR(x=arabidopsis.chrom.mask)
```

# ChromR - filtration of VCF II

```

1 # See results from the previous slide
2 arabidopsis.chrom.fin
3 chromoqc(chrom=arabidopsis.chrom.fin)
4 # The plot is bit empty as we have only single gene

```



# Convert VCF into various objects for later processing

```

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

```

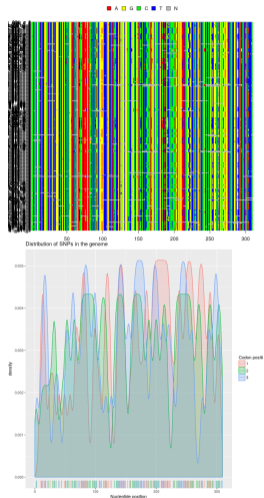


# Convert vcfR into DNABin

```

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

```



# 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=as.character.DNABin(gunnera.dna),
11   names=names(gunnera.dna), file.out="gunnera.fasta", open="w")
12 # Export DNA sequences as NEXUS
13 write.nexus.data(x=gunnera.dna, file="gunnera.nexus", format="dna")
14 write.vcf(x=arabidopsis.vcf, file="arabidopsis.vcf.gz") # Export VCF
15 # Export tree(s) (objects of class phylo) - will be introduced later
16 write.tree(phy=hauss.nj.bruvo, file="haussknechtii.nwk") # In NEWICK
17 write.nexus(hauss.nj.bruvo, file="haussknechtii.nexus") # In NEXUS
```

# Import your own 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...
    - Import preferably your data — you'll later use them to perform selected analysis
  - 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...

# Multiple sequence alignment

## 4 Alignment

Overview and MAFFT

MAFFT, Clustal, MUSCLE and T-Coffee

Multiple genes

Display and cleaning

Tasks

### Importance of alignment

- **All sequences must be aligned prior to any analysis!**
- Be sure to either import already aligned sequences of same length — or align them
- Aligned sequences commonly require post-processing — trimming, ... as especially e.g. distance-based analysis are sensitive to missing data

# 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 [Unipro UGENE](#), [Geneious](#) or [CLC Genomics Workbench](#))
- 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...
  - See e.g. [https://github.com/V-Z/hybseq-scripts/blob/master/bin/hybseq\\_4\\_alignment\\_3\\_run.r](https://github.com/V-Z/hybseq-scripts/blob/master/bin/hybseq_4_alignment_3_run.r) from [HybSeq](#) scripts
- 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

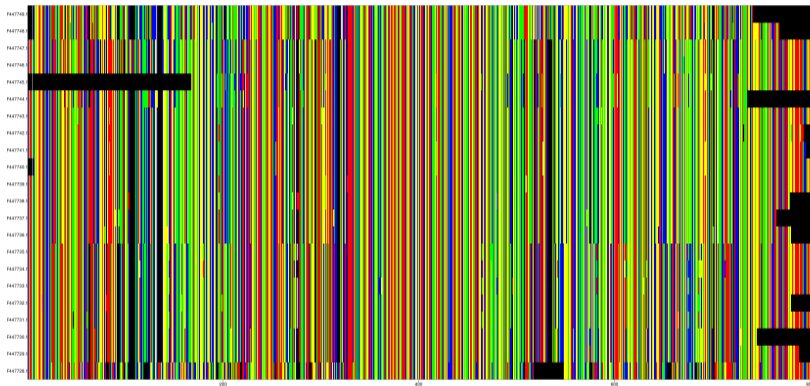
- MUSCLE is available in packages `muscle` and `ape` – first one reads “`*StringSet`” classes R objects and writes “`*MultipleAlignment`” R classes objects; the latter reads and writes object of class “`DNABin`”
- `ape` also contains functions to use Clustal and T-Coffee – both read and write `DNABin`
- MAFFT is available from (same author) in packages `ips` and `phyloch` – both read and write `DNABin`

```
1 library(ape)
2 library(ips)
3 # Requires path to MAFFT binary - set it according to your installation
4 # read ?mafft and mafft's documentation
5 # Change "exec" to fit your path to mafft!
6 gunnera.mafft <- mafft(x=gunnera.dna, method="localpair", maxiterate=100,
7   options="--adjustdirection", exec="/usr/bin/mafft")
8 gunnera.mafft
```

# Clustal, MUSCLE and T-Coffee from ape

```
1 class(gunnera.mafft)
2 image.DNABin(gunnera.mafft)
3 # read ?clustal and documentation of Clustal, Muscle and T-Coffee
4 # when using them to set correct parameters
5 gunnera.clustal <- ape::clustal(x=gunnera.dna, pw.gapopen=10, pw.gapext=
6   0.1, gapopen=10, gapext=0.2, exec="/usr/bin/clustalw2", quiet=FALSE,
7   original.ordering=TRUE) # Change "exec" to fit your path to clustal!
8 gunnera.clustal
9 class(gunnera.clustal)
10 image.DNABin(gunnera.clustal)
11 gunnera.muscle <- muscle(x=gunnera.dna, exec="muscle", quiet=FALSE,
12   original.ordering=TRUE) # Change "exec" to fit your path to muscle!
13 gunnera.muscle
14 class(gunnera.muscle)
15 image.DNABin(gunnera.muscle)
16 ?muscle::muscle # See option in muscle package
```

# Multiple sequence alignment with MUSCLE



```
1 # Remove gaps from alignment - destroy it
2 gunnera.nogaps <- del.gaps(gunnera.muscle)
3 ?del.gaps # See for details
```



# 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(gunnera.dna, usflu.dna, usflu.dna2)
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=ips::mafft, method="localpair",
9 maxiterate=100, exec="/usr/bin/mafft")
10 # Change "exec" 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

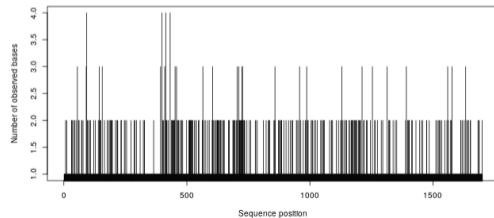
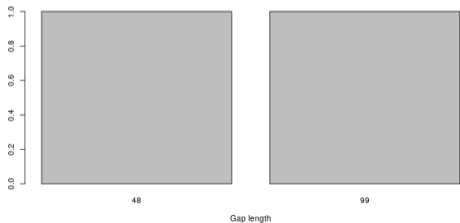
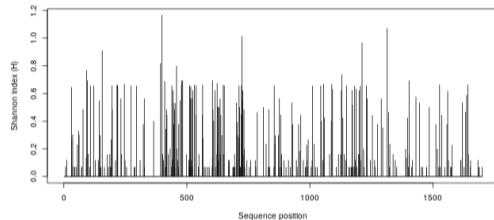
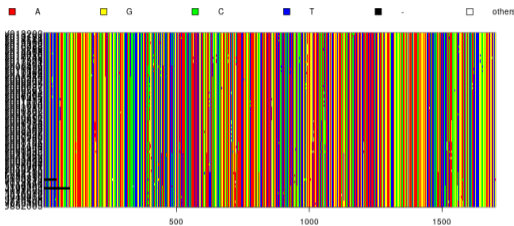
- There are plenty of implementations of parallelisation and using of computer clusters, see <https://CRAN.R-project.org/web/views/HighPerformanceComputing.html>

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

# Checking the alignment



# Cleaning the alignment

```
1 # Delete all columns/rows containing only gaps or missing data (N, ?, -)
2 gunnera.mafft <- deleteEmptyCells(DNAbin=gunnera.mafft)
3 ?ips::deleteEmptyCells # See help page for details
4 ?phyloch::delete.empty.cells # See help page for details
5 # Delete all columns containing at least 25% of gaps
6 gunnera.mafft.ng <- deleteGaps(x=gunnera.mafft, gap.max=nrow(gunnera.mafft)/4)
7 gunnera.mafft.ng
8 # Do not confuse with function delete.gaps() from phyloch package
9 # See of settings of "nmax" value - threshold for gap deletion
10 ?deleteGaps # "nmax=0" deletes all columns with any gap
11 multialign.aln.ng <- lapply(X=multialign.aln, FUN=deleteGaps, gap.max=5)
12 multialign.aln.ng
13 # Delete every line (sample) containing at least 20% of missing data
14 gunnera.mafft.ng <- del.rowgapsonly(x=gunnera.mafft.ng, threshold=0.2,
15   freq.only=FALSE)
16 gunnera.mafft.ng
17 ?ape::del.rowgapsonly # See help page for details
```

# Cleaning the alignment

```
1 # Delete every alignment position having at least 20% of missing data
2 gunnera.mafft.ng <- del.colgaponly(x=gunnera.mafft.ng, threshold=0.2,
3   freq.only=FALSE)
4 gunnera.mafft.ng
5 ?ape::del.colgaponly # See help page for details
6 # Display the result
7 image.DNABin(x=gunnera.mafft.ng)
8 lapply(X=multialign.aln.ng, FUN=image.DNABin)
```

- “Strictness” of alignment cleaning depends on following steps — NJ (and another distance-based methods) doesn’t like more than  $\sim 10\text{--}15\%$  of missing data, but some tree builders are able to work with gaps — check their documentation...
- Automated cleanup is useful especially if batch processing plenty of genes

# Practice alignment

## Tasks

- 1 Download (import from on-line database or your file from disk) sequences of ITS (or other variable gene if you prefer) of your favourite organism.
- 2 Check the imported data.
- 3 Align sequences with your preferred aligner.
- 4 Check the resulting alignment.
- 5 Trim the alignment — delete columns/rows with too much missing data. Think about various thresholds and their implications.
- 6 Compare outputs of several aligners and/or different parameters (gap penalty etc.).

# Basic analysis

## 5 Basic analysis

First look at the data

Statistics

MSN

Genetic distances

AMOVA

Hierarchical clustering

NJ (and UPGMA) tree

PCoA

Tasks



# Introductory overview of statistics and methods I

- **Selected method depends on data type, question to answer, ...** — see further slides and chapters
  - Check assumptions and requirements of the methods before usage
  - Think if the method answers your question
  - Always be opened for new possibilities coming with new methods and packages developed...
- **Population-genetic indices** — from slide 145
  - 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 166
  - **It is crucial to select appropriate distance method for given data type**
  - Usually require the distance matrix to be Euclidean

# Introductory overview of statistics and methods II

- Distance matrix has one single number (index) for each pair of comparisons (individuals, populations) – rough
- 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 181
  - 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 185
  - A tree starting from the two most similar individuals and connecting in the next steps next and next the most similar individual

# Introductory overview of statistics and methods III

- In some cases artificially chains individuals
- Several methods try to improve it – slide 199
- **Principal Coordinates Analysis (PCoA)** – from slide 200
  - 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 165
  - 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 300
  - Generally, the methods are looking for the most simple solution under given model, e.g. to construct phylogenetic tree requiring the lowest number of evolutionary changes (DNA mutations)

# Introductory overview of statistics and methods V

- It is easy to score how good the solution is (comparing to another solution), 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 (comparing to another solution), 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, scoring them and approaching the best runs in steps (iterative generations)
    - After some time it converges to find optimal solution (usually described by logarithms of likelihood of given model)

# Introductory overview of statistics and methods VI

- Usually,  $\sim$ millions (or even more) of generations are required
- 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  $\sim$ 10–25% 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  $\sim$ 100–1000 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

## Questions and data

- Methods in this section answer questions about genetic characteristics of individuals/populations
  - Overall genetic similarity of individuals/populations (without spatial or another context) — various population-genetic indices, PCoA, ...
  - Distribution of genotypes within/among populations
  - Description of genetic characteristics of populations — heterozygosity, Hardy-Weinberg equilibrium, F-statistics, ...
  - Hierarchical relationships among individuals/populations (UPGMA, NJ, ...)
- Any data can be used
  - Population genetic studies use to use as variable genetic markers as possible (depending on scale, e.g. SSRs, AFLP, RAD-Seq, highly variable introns like ITS, ...)
- Nearly all data types are processed in similar/same way
  - It's important to select correct genetic index, distance method, etc. for particular data types and/or question

# Load needed libraries

```
1 library(ape) # Analysis of phylogenetics and evolution
2 library(ade4) # Analysis of ecological data, multivariate methods
3 library(adegenet) # Exploratory analysis of genetic and genomic data
4 library(pegas) # Population and evolutionary genetics
5 # Population genetic analysis, including populations with mixed
6 # reproduction
7 library(poppr)
8 library(hierfstat) # Hierarchical F-statistics
9 library(corrplot) # Visualization of correlation matrix
10 library(StAMPP) # Statistical analysis of mixed ploidy populations
11 library(philentropy) # Various genetic distances
```

- Of course, there are plenty of another options...
  - See e.g. [CRAN Views](#) for [genetics](#) or [multivariate statistics](#)
  - Representative, but not exhaustive examples are shown



# Descriptive statistics I

- We will now work mainly with diploid SSRs of *Taraxacum haussknechtii*, you can **try with 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 it looks like big difference
5 plot(x=hauss.summ$Hexp, y=hauss.summ$Hobs,
6      main="Observed vs expected heterozygosity",
7      xlab="Expected heterozygosity", ylab="Observed heterozygosity")
8 abline(0, 1, col="red")
9 # Bartlett's K-squared test of difference
10 # between observed and expected heterozygosity - not significant
11 bartlett.test(list(hauss.summ$Hexp, hauss.summ$Hobs))
12           Bartlett test of homogeneity of variances
13 data:  list(hauss.summ$Hexp, hauss.summ$Hobs)
14 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

```

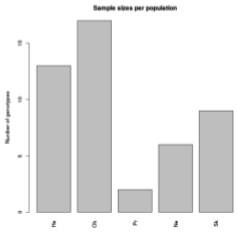
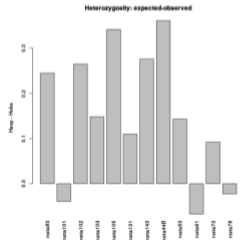
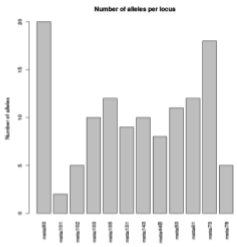
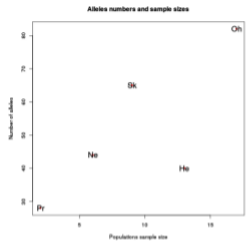
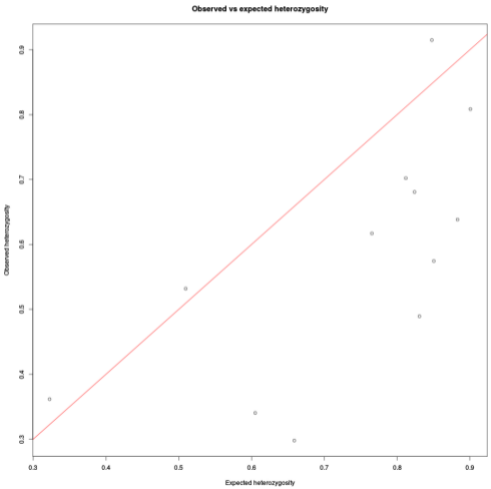
1 # T-test of difference between observed and expected heterozygosity
2 t.test(x=hauss.summ$Hexp, y=hauss.summ$Hobs, paired=TRUE, var.equal=T)
3     Paired t-test
4 data:  hauss.summ$Hexp and hauss.summ$Hobs
5 t = 3.5622, df = 11, p-value = 0.004456 # strongly significant
6 alternative hypothesis: true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

# Graphs from previous slides



# Population statistics returned by `poppr()` I

## Too much to choose from?

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

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

# Population statistics returned by poppr() II

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

## Population statistics returned by poppr() III

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

## Population statistics returned by poppr() IV

- The index of association is a calculation based on the ratio of the variance of the raw number of differences between individuals and the sum of those variances over each locus
- It is 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 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



# Population statistics by poppr()

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

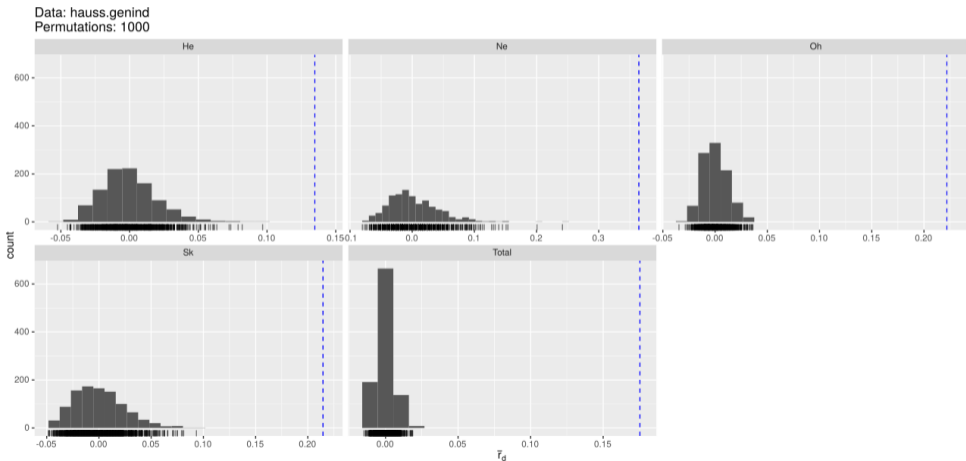
```

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



# 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

```

1 hauss.hwe.test # See results per locus
2               chi^2   df   Pr(chi^2 >)   Pr.exact
3 msta93      383.5519728 190 3.552714e-15   0.000
4 msta101     0.6927242   1  4.052393e-01   0.657
5 msta102     83.0741964  10 1.250111e-13   0.000
6 msta103     77.1819098  45 1.998865e-03   0.000
7     ...           ...   ...           ... # Another loci...
    
```

- 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
- If there is significant departure from HWE, think about biological process (with respect to life traits of species studied) which could cause such structuring

# 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`, `theta.msat` and another similar functions the data object **must** contain population column (see manual of respective function)

```

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 # Convert Adegnet's genind to format of hierfstat package
2 ?genind2hierfstat
3 # Nei's pairwise Fst between all pairs of populations.
4 # 0 = no structure; 1 = maximal difference
5 pairwise.neifst(dat=genind2hierfstat(dat=hauss.genind))
6
7      He      Ne      Oh      Pr      Sk
8 He      NA 0.3204 0.1824 0.1942 0.1599
9 Ne 0.3204      NA 0.1210 0.2548 0.1229
10 Oh 0.1824 0.1210      NA 0.1038 0.0833
11 Pr 0.1942 0.2548 0.1038      NA 0.1407
12 Sk 0.1599 0.1229 0.0833 0.1407      NA
    
```

- Check also another functions of **hierfstat** package, there are more options
- Hierfstat package implements  $F_{ST}$  only for haploid and diploid populations, **StAMPP** (next slide) also for another ploidies and mixed ploidy data

# F-statistics for mixed ploidy data I

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

```
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 # Population code consists of first three letters of individual's name -
5 # extract the population name part
6 substr(x=indNames(arabidopsis.genlight), start=1, stop=3)
7 pop(arabidopsis.genlight) <- substr(x=indNames(arabidopsis.genlight),
8   start=1, stop=3)
9 pop(arabidopsis.genlight) # Check it
10 popNames(arabidopsis.genlight)
11 ?StAMPP::stampFst # See method details
```

# F-statistics for mixed ploidy data II

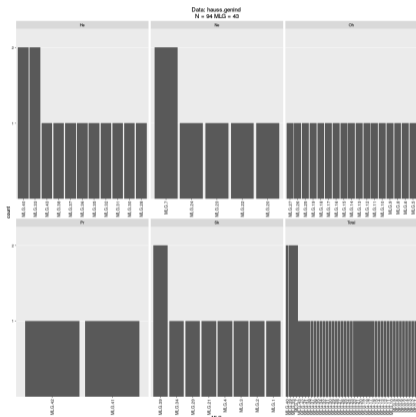
```
1 # Calculating the Fst
2 arabidopsis.fst <- StAMPP::stampFst(geno=arabidopsis.genlight,
3   nboots=100, percent=95, nclusters=1)
4 # For large data use higher nclusters to parallelize calculations
5 arabidopsis.fst[["Fsts"]] # Matrix of Fst among populations
6 arabidopsis.fst[["Pvalues"]] # Matrix of P values
7 # Save results - open in spreadsheet (e.g. LibreOffice Calc)
8 write.table(x=arabidopsis.fst[["Fsts"]], file="arabidopsis_fst.tsv",
9   quote=FALSE, sep="\t")
10 # Correlation plot of pairwise Fst
11 corrplot(corr=arabidopsis.fst[["Fsts"]], method="circle", type="lower",
12   col=funky(15), title="Correlation matrix of Fst among populations",
13   is.corr=FALSE, diag=FALSE, outline=TRUE, order="alphabet", tl.pos="lt",
14   tl.col="black")
15 ?corrplot # See for more options
16 # Display in similar way also another Fst tables
```



# Multi locus genotypes and inbreeding coefficient

- Especially when working with clonal species or species with low genetic structure it is beneficial to know how many unique genotypes there are and how they are distributed across populations
- Inbreeding coefficient estimates level of mating among individuals with (nearly) identical genotypes — important e.g. for conservation studies or work with agricultural species
- Such studies require sufficiently variable marker (e.g. SSRs) so that results show real genetic structuring and not “just” low variability of selected marker

# Multi locus genotypes



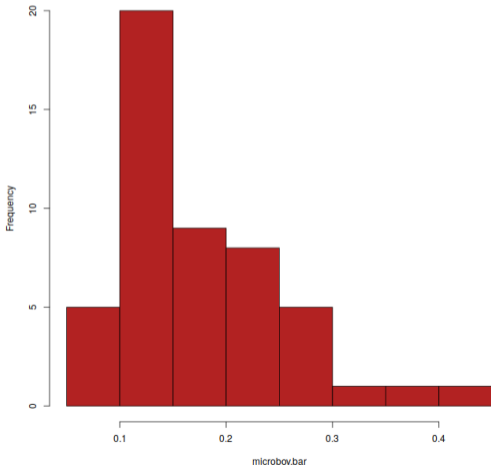
```

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 cattle



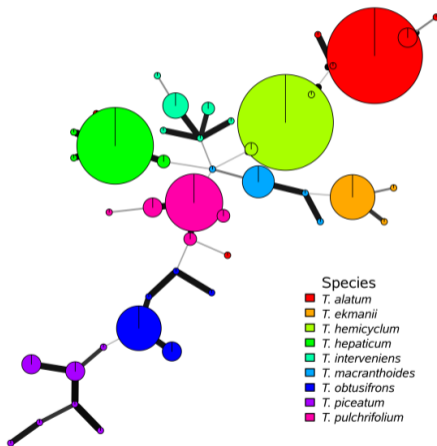
```

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 ?inbreeding # Check for settings
11 # population means for plotting
12 microbov.bar <- sapply(X=
13   microbov.inbr, FUN=mean)
14 # Plot it
15 hist(x=microbov.bar, col=
16   "firebrick", main="Average
17   inbreeding in Salers cattle")

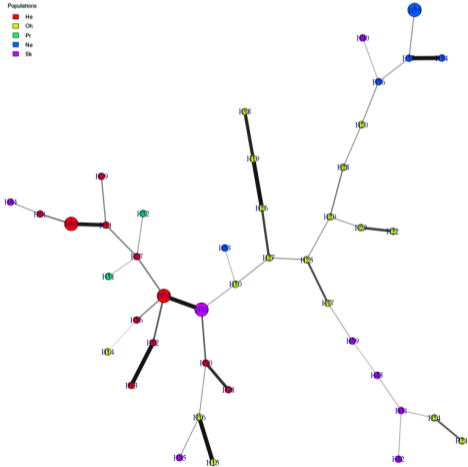
```

# Minimum Spanning Network (MSN)

- Package poppr, for SSRs based on Bruvo's distance, can handle any data type (requires `genind` objects)
- Shows relationships among haplotypes (unique genotypes), can be labeled by population, haplotype, ...
  - Size of pie is proportional to number of individuals assigned
  - Lines connect haplotypes according to their similarity
- Suitable for less variable datasets, e.g. some mitochondrial or plastide genes (or SSRs for less variable species) – otherwise the figure is messy



# Minimum Spanning Network



```

1 # See details and options...
2 ?bruvo.msn
3 # Get the MSN
4 # Note SSRs repeats 'rep(2, 12)' -
5 # change according to your data
6 bruvo.msn(gid=hauss.genind,
7   replen=rep(2, 12), loss=TRUE,
8   palette=rainbow, vertex.label
9   ="inds", gscale=TRUE,
10  wscale=TRUE, showplot=TRUE)
11 # For another data types
12 ?msn.poppr
13 # Interactive creation of MSN
14 # Opens browser window
15 ?imsn
    
```

# Basic distances

```
1 # Simple dissimilarity distance matrix
2 ?dist.gene # Details about methods of this distance constructions
3 hauss.dist <- dist.gene(x=hauss.genind@tab, method="pairwise")
4 # Nei's distance (not Euclidean) for populations (other methods are
5 # available, see ?dist.genpop)
6 hauss.dist.pop <- dist.genpop(x=hauss.genpop, method=1, diag=T, upper=T)
7 # Test if it is Euclidean
8 is.euclid(hauss.dist.pop, plot=TRUE, print=TRUE, tol=1e-10) # FALSE = No
9 # Turn to be Euclidean
10 hauss.dist.pop <- cailliez(distmat=hauss.dist.pop, print=FALSE, tol=1e-07,
11   cor.zero=TRUE)
12 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

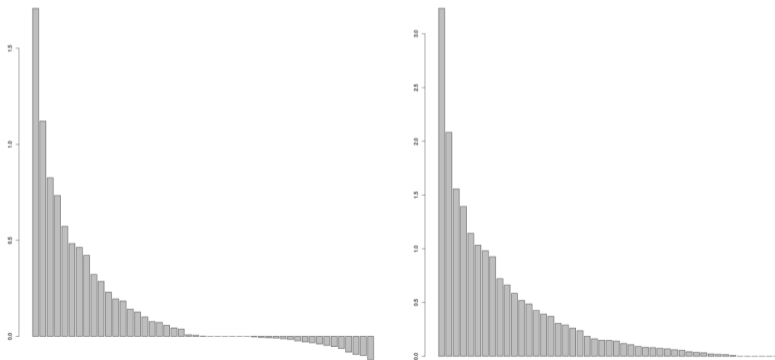
```
1 # Bruvo's distances weighting SSRs repeats - take care about replen
2 # parameter - requires repetition length for every SSRs locus
3 hauss.dist.bruvo <- bruvo.dist(pop=hauss.genind, replen=rep(2, 12),
4   loss=TRUE)
5 # Test if it is Euclidean
6 is.euclid(hauss.dist.bruvo, plot=TRUE, print=TRUE,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

- 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 # Dissimilarity matrix returns a distance reflecting the number of
6 # allelic differences between two individuals
7 hauss.dist.diss <- diss.dist(x=hauss.genind, percent=FALSE, mat=TRUE)
8 is.euclid(as.dist(hauss.dist.diss), plot=TRUE, print=TRUE, tol=1e-10)
    
```

## 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	<code>prevosti.dist</code> , <code>diss.dist</code>	—	No
Nei 1972, 1978	<code>nei.dist</code>	Infinite Alleles, Genetic Drift	No
Edwards 1971	<code>edwards.dist</code>	Genetic Drift	Yes
Reynolds 1983	<code>reynolds.dist</code>	Genetic Drift	Yes
Rogers 1972 <sup>1</sup>	<code>rogers.dist</code>	—	Yes
Bruvo 2004	<code>bruvo.dist</code>	Step-wise Mutation	No

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

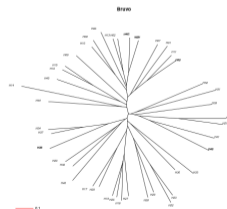
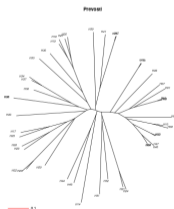
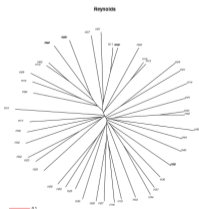
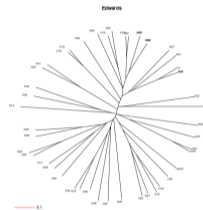
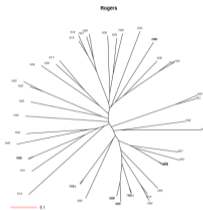
<sup>1</sup>Rogers (1972): Measures of genetic similarity and genetic distances. Pp. 145-153 of Studies in Genetics. University of Texas Publishers

# 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 dists[["Bruvo"]] <- hauss.dist.bruvo # Add Bruvo distance
11 dists # Check list of distances
12 par(mfrow=c(2, 3)) # Split graphical device into 2 lines, 3 panes each
13 # Calculate NJ and plot all trees
14 x <- lapply(names(dists), function(x) { plot(njs(dists[[x]]), main=x,
15   type="unrooted")
16   add.scale.bar(lcol="red", length=0.1) })
17 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 gunnera.dist <- dist.dna(x=gunnera.mafft.ng, model="F81")
11 # Check it
12 gunnera.dist
13 class(gunnera.dist)
14 dim(as.matrix(gunnera.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(D) dist(as.matrix(D)))
4 class(usflu.dists)
5 names(usflu.dists)
6 class(usflu.dists[[1]])
7 usflu.distr <- Reduce(f="+", x=usflu.dists)
8 class(usflu.distr)
9 usflu.distr
10 # It is possible to use just basic dist function on whole genlight object
11 # (might require a lot of RAM)
12 usflu.distg <- dist.gene(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 # stampNeisD requires population factor in genlight Nei's 1972 distance
2 # between individuals (use pop=TRUE to calculate among populations)
3 arabidopsis.dist <- stampNeisD(geno=arabidopsis.genlight, pop=FALSE)
4 # Check it
5 head(arabidopsis.dist)
6 dim(arabidopsis.dist)
7 class(arabidopsis.dist)
8 # The same on population level
9 arabidopsis.dist.pop <- stampNeisD(geno=arabidopsis.genlight, pop=TRUE)
10 # Check it
11 head(arabidopsis.dist.pop)
12 dim(arabidopsis.dist.pop)
13 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

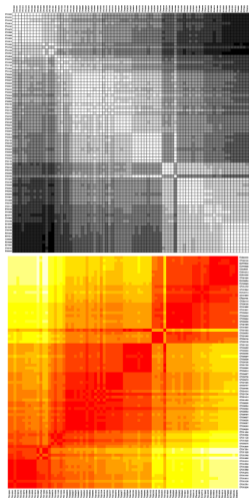
- There is enormous number of various distance measures...
- For example [Jaccard index](#) is used to compare binary (presence/absence) data like [AFLP](#)

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



# AMOVA I

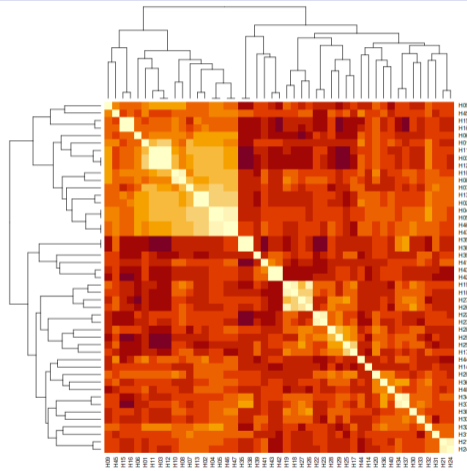
- Analysis of molecular variance tests if there are significant differences among populations (and/or another levels)
- Some implementations can partition variance into various levels
- `pegas::amova` returns a table of sums of square deviations ( `SSD` ), mean square deviations ( `MSD` ), and the number 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=FALSE)
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 ...
    
```

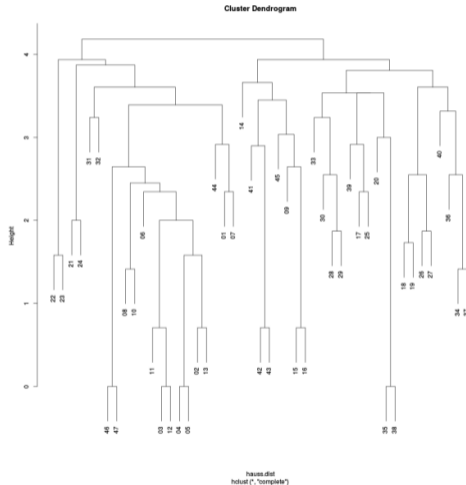
# Heat maps



```

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)
15 # See settings like colors,
16 # dendrogram, etc.
17 ?heatmap
    
```

# Hierarchical clustering – UPGMA and others



```

1 # According to distance used
2 # How to use hierarchical
3 # clustering
4 ?hclust
5 plot(hclust(d=hauss.dist,
6   method="complete"))
7 plot(hclust(d=hauss.dist.pop,
8   method="complete"))
9 plot(hclust(d=hauss.dist.bruvo,
10  method="complete"))

```

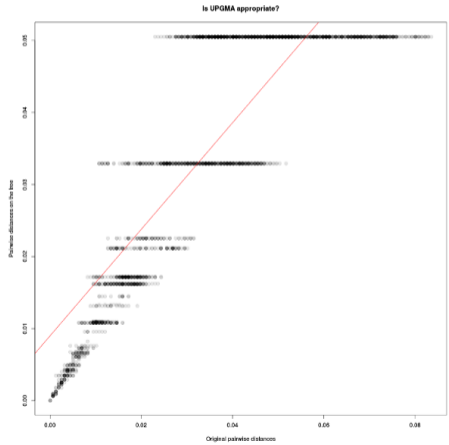
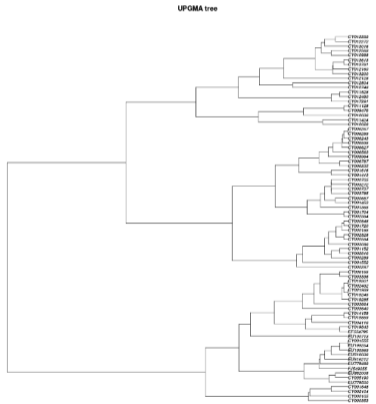
- This is very basic function to make dendrogram
- There are better possibilities (NJ etc – see slide 185 and onward)

# 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...

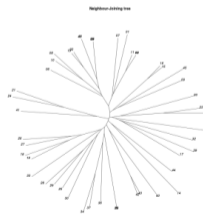
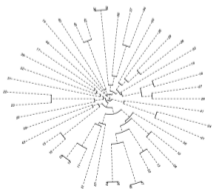
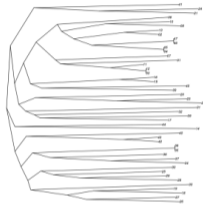
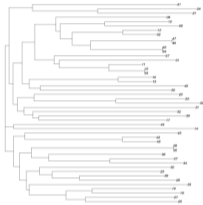
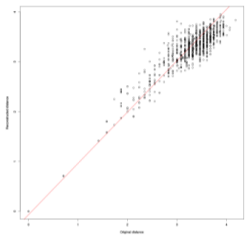




# Calculate and test NJ tree

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

# Choose your 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, FUN=function
6   (XXX) nj(dist.gene(loci2genind(XXX)@tab, method="pairwise")), 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 ?aboot # Package poppr
```

# Plotting bootstrap and nicer trees

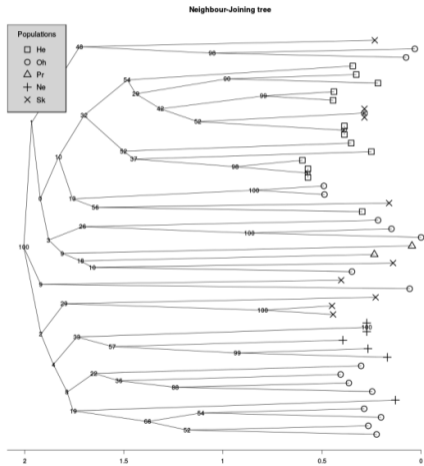
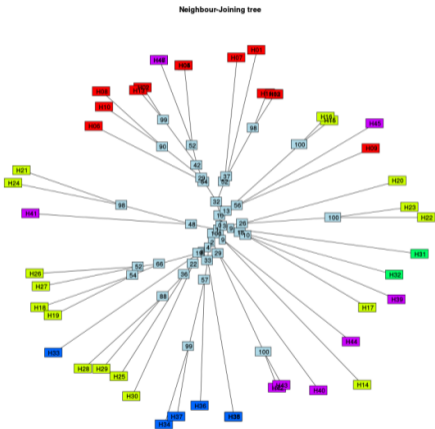
```
1 # Plot the tree, explicitly display node labels
2 plot.phylo(x=hauss.aboot, show.node.label=TRUE)
3 ?plot.phylo # See details...
4 ## Plot a nice tree with colored tips
5 plot.phylo(x=hauss.nj, type="unr", show.tip=F, edge.width=3, main="NJ")
6 # Labels for nodes - bootstrap - see ?nodelabels for graphical settings
7 nodelabels(text=round(hauss.boot/10))
8 # Colored labels - creates vector of colors according to populations
9 nj.rainbow<-colorRampPalette(rainbow(length(levels(pop(hauss.genind)))))
10 tiplabels(text=indNames(hauss.genind), bg=fac2col(x=pop(hauss.genind),
11   col.pal=nj.rainbow)) # Colored tips
12 ## Plot BW tree with tip symbols and legend
13 plot.phylo(x=hauss.nj, type="clad", show.tip=F, edge.width=3, main="NJ")
14 axisPhylo() # Add axis with distances
15 # From node labels let's remove unneeded frame
16 nodelabels(text=round(hauss.boot/10), frame="none", bg="white")
```

# Nicer trees

```
1 # As tip label we use only symbols - see ?points for graphical details
2 tiplabels(frame="none", pch=rep(0:4, times=c(13, 17, 2, 6, 9)), lwd=2,
3   cex=2)
4 # Plot a legend explaining symbols
5 legend(x="topleft", legend=c("He", "Oh", "Pr", "Ne", "Sk"),
6   border="black", pch=0:4, pt.lwd=2, pt.cex=2, bty="o", bg="lightgrey",
7   box.lwd=2, cex=1.2, title="Populations")
8 # See more options...
9 ?plot.phylo
10 ?nodelabels
11 ?legend
12 ?axis.phylo
```

- Functions in `ape` (`plot.phylo` and others), `adegenet` and another packages provide plenty of options to manipulate and display trees, for users of `ggplot2`, `ggtree` is an interesting alternative

# Choose your tree...

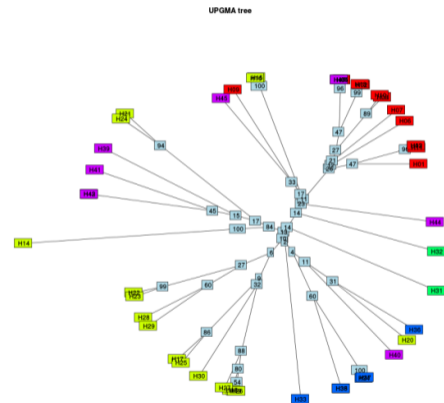
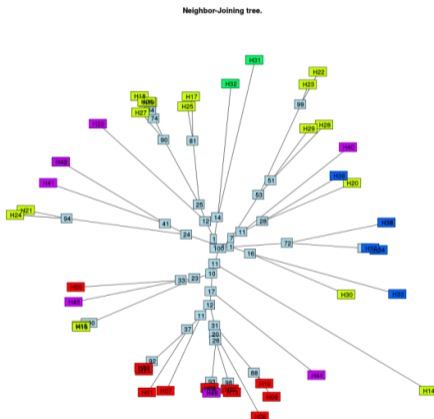


# Trees based on Bruvo's distance

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

# Choose your tree...





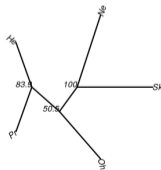
# NJ tree of populations

```

1 ?poppr::aboot # aboot() can use distances implemented in poppr:
2 ?poppr::nei.dist
3 # Calculations
4 hauss.nj.pop <- aboot(x=hauss.genpop, tree="nj", distance="nei.dist",
5   sample=1000, showtree=FALSE)
6 print.phylo(hauss.nj.pop) # Information about result
7 # Plot a tree
8 plot.phylo(x=hauss.nj.pop, type="radial", show.node.label=TRUE,
9   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.phylo(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, FUN=
3   function(EEE) root.phylo(nj(dist.dna(EEE, model="TN93")), outgroup=1),
4   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, col=transp(num2col(x=usflu.annot[["year"]],
9   col.pal=usflu.pal), alpha=0.7), cex=3.5, fg="transparent")
10 axisPhylo()
11 # Legend - describing years - pretty() automatically shows best
12 # values from given range, num2col() selects colors from color scale
13 legend(x="topright", fill=num2col(x=pretty(x=1993:2008, n=8),
14   col.pal=usflu.pal), leg=pretty(x=1993:2008, n=8), ncol=1)
15 # Plots bootstrap support - note usflu.boot contains raw numbers
16 # transform it into percent
17 nodelabels(text=round(usflu.boot/10), cex=0.75)
```

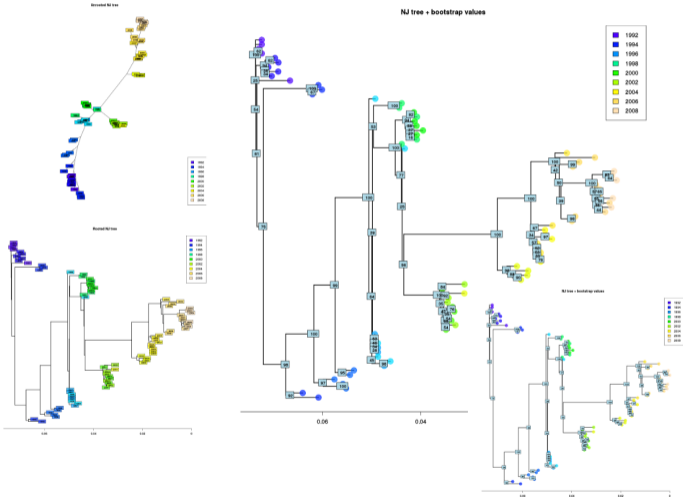
# Collapse branches with low bootstrap support

```

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$edge.length[usflu.tocollapse] <- 0
8 # Create new tree
9 usflu.tree.collapsed <- di2multi(usflu.tree.usflu.na.density, tol=0.00001)
10 # Plot the consensus tree
11 plot.phylo(x=usflu.tree.collapsed, show.tip=FALSE, edge.width=2)
12 title("NJ tree after collapsing weak nodes")
13 tiplabels(text=usflu.annot$year, bg=transp(num2col(x=usflu.annot
14   [["year"]], col.pal=usflu.pal), alpha=0.7), cex=0.5, fg="transparent")
15 axisPhylo()
16 legend(x="topright", fill=num2col(x=pretty(x=1993:2008, n=8),
17   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`

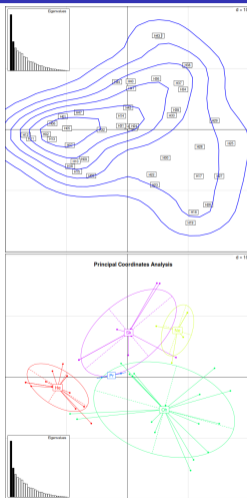
# PCoA I

```

1 hauss.pcoa <- dudi.pco(d=dist.gene(x=scaleGen(x=hauss.genind, center=TRUE,
2   scale=FALSE, truenames=TRUE), method="pairwise"), scannf=FALSE, nf=3)
3 s.label(dfxy=hauss.pcoa$li, clabel=0.75) # Basic display
4 # To plot different axes use for example dfxy=hauss.pcoa$li[c(2, 3)]
5 s.kde2d(dfxy=hauss.pcoa$li, cpoint=0, add.plot=TRUE) # Add kernel density
6 # Add histogram of Eigenvalues
7 add.scatter.eig(w=hauss.pcoa$eig, nf=3, xax=1, yax=2, posi="topleft",
8   sub="Eigenvalues")
9 # Percentage of variance explained by each PC axis
10 100*hauss.pcoa$eig/sum(hauss.pcoa$eig)
11 # Colored display according to populations
12 # Creates vector of colors according to populations
13 hauss.pcoa.col <- rainbow(length(levels(pop(hauss.genind))))
14 s.class(dfxy=hauss.pcoa$li, fac=pop(hauss.genind), col=hauss.pcoa.col)
15 add.scatter.eig(w=hauss.pcoa$eig, nf=3, xax=1, yax=2, posi="bottomleft",
16   sub="Eigenvalues")
17 title("Principal Coordinates Analysis") # Adds title to the graph
    
```



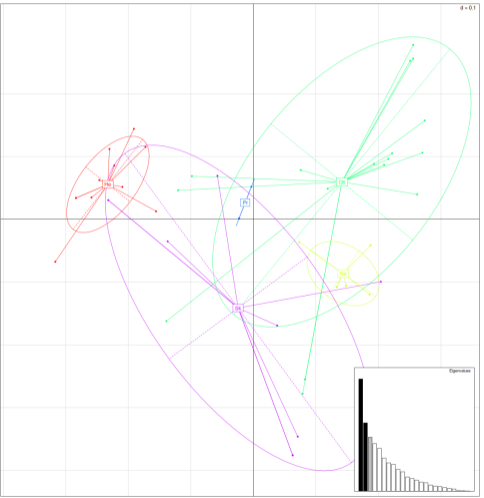
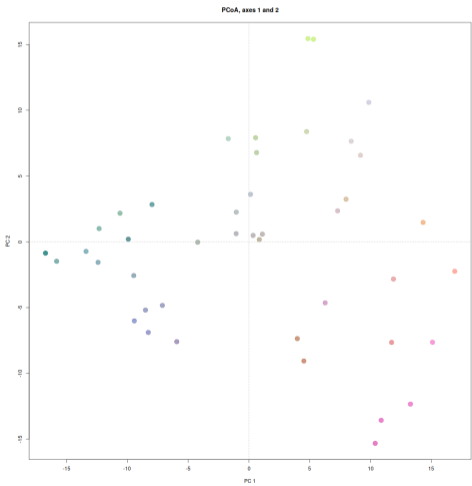
# PCoA II



```

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 2")
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 of basic analysis were shown with the *Taraxacum haussknechtii* microsatellite dataset — try to do some analysis with another imported data
  - 2 Try some of the introduced analysis with your own custom imported data
  - 3 Try at least 2–3 analysis according to your interests
- Of course, following chapters will show more possible analysis...
  - 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

# Single Nucleotide Polymorphism

Special methods for large next-generation sequencing data

## 6 SNP

### PCA and NJ

- Large whole-genome sequencing data (usually from Illumina sequencing) use to be imported as VCF (and converted to `genlight`) or read from FASTA directly into `genlight` – it's designed to efficiently store and process large datasets
- Several special functions are available to directly work with `genlight`, taking advantage of its design, and using multiple CPU cores (on Linux and macOS)

# 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 frequencies",
7     main="Distribution of (second) allele frequencies")
8 lines(x=density(usflu.freq)$x, y=density(usflu.freq)$y^1.5, col="red",
9     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", xlab="Allele
15     frequencies", main="Distribution of allele frequencies", nclass=20)
16 lines(x=density(usflu.mean, bw=0.05)$x, y=density(usflu.mean, bw=0.05)$y^2,
17     lwd=3)
```

# Number of missing values in each locus

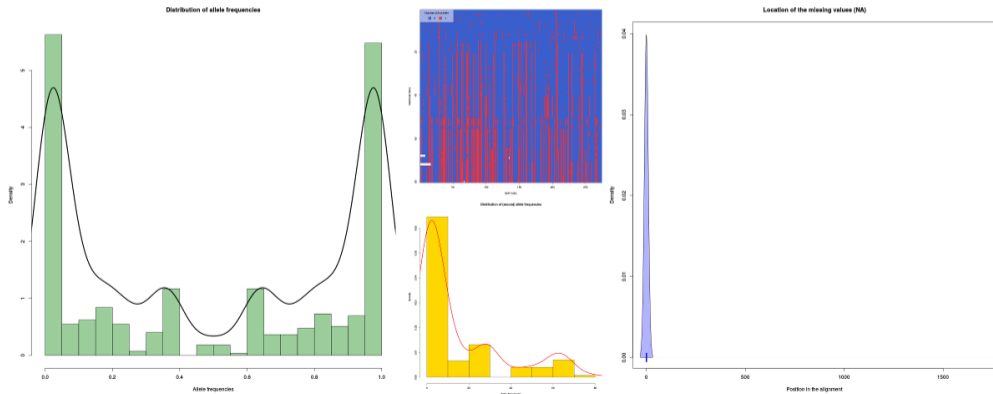
```

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 of original alignment
4 plot(x=usflu.na.density, type="n", xlab="Position in the alignment",
5      main="Location of the missing values (NA)", xlim=c(0, 1701))
6 polygon(c(usflu.na.density$x, rev(usflu.na.density$x)),
7         c(usflu.na.density$y, rep(0, length(usflu.na.density$x))),
8         col=transp("blue", alpha=0.3))
9 points(glNA(usflu.genlight), rep(0, nLoc(usflu.genlight)), pch="|", cex=2,
10        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...
- **Windows users:** To speed up the processing, `gl*` functions use parallelisation library unavailable on Windows – add parameter `parallel=FALSE` to be able to use them

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



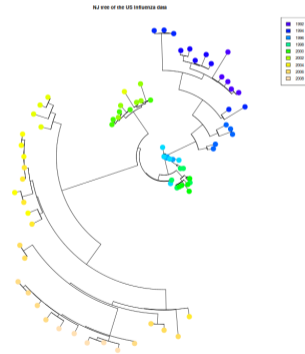
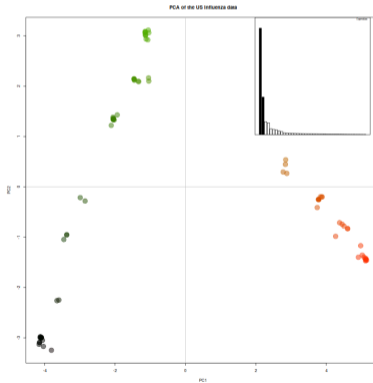
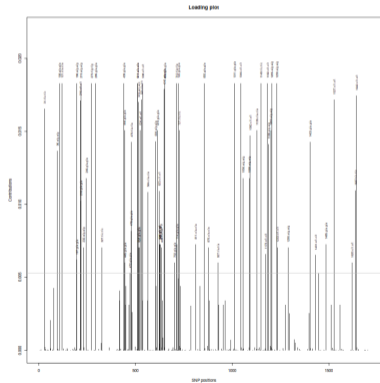
# PCA, NJ and genlight objects

```
1 usflu.pca <- glPca(x=usflu.genlight, center=TRUE, scale=FALSE,
2   loadings=TRUE) # Select number of retained PC axes, about 10 here
3 scatter.glPca(x=usflu.pca, posi="bottomright") # Plot PCA
4 title("PCA of the US influenza data")
5 # Loading plot - contribution of variables to the pattern observed
6 loadingplot.glPca(x=usflu.pca)
7 colorplot(usflu.pca$scores, usflu.pca$scores, transp=TRUE, cex=4) # Cols
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 usflu.tree.genlight <- nj(dist.gene(as.matrix(usflu.genlight))) # Get tree
13 # Plot colored phylogenetic tree
14 plot.phylo(x=usflu.tree.genlight, type="fan", show.tip=FALSE)
15 tiplabels(pch=20, col=num2col(usflu.annot[["year"]], col.pal=usflu.pal),
16   cex=4)
17 title("NJ tree of the US influenza data")
```



# PCA, NJ and genlight objects

```
1 legend(x="topright", fill=num2col(x=pretty(x=1993:2008, n=8),
2   col.pal=usflu.pal), leg=pretty(x=1993:2008, n=8), ncol=1)
```



# Discriminant Analysis of Principal components

## 7 DAPC

Bayesian clustering

Discriminant analysis and visualization

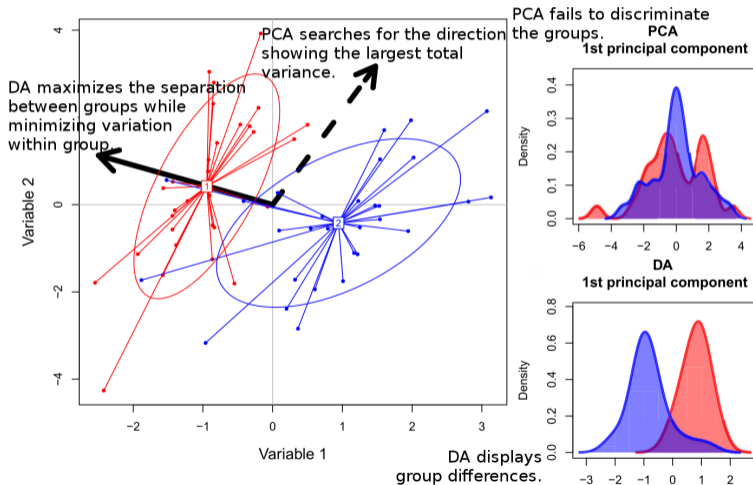
Tasks

# 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)
- User selects best **K** – number of clusters; according to scores shown
- 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 <https://adegenet.r-forge.r-project.org/>
- If following commands would seem too complicated, try web interface

```
1 library(adegenet)
2 adegenetServer("DAPC") # Recommended to open in Google Chrome/Chromium
3 adegenetTutorial("dapc") # Tutorial, more information about DAPC
```

# Principal difference between PCA and DA

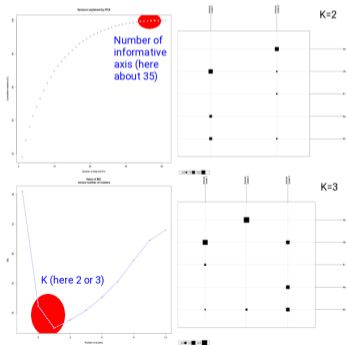


# K-find — Bayesian K-means clustering

```
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 table(pop(hauss.genind), hauss.kfind$grp) # See results as text
8 hauss.kfind
9 # Graph showing table of original and inferred populations and
10 # assignment of individuals
11 table.value(df=table(pop(hauss.genind), hauss.kfind$grp), col.lab=
12   paste("Inferred\ncluster", 1:length(hauss.kfind$size)), grid=TRUE)
13 # For K=3 - note parameters n.pca and n.clust - we just rerun the
14 # analysis and when results are stable, no problem here
15 hauss.kfind3 <- find.clusters(x=hauss.genind, n.pca=35, n.clust=3,
16   stat="BIC", choose.n.clust=FALSE, n.iter=100000, n.start=100,
17   scale=FALSE, truenames=TRUE)
```

# K-find outputs

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

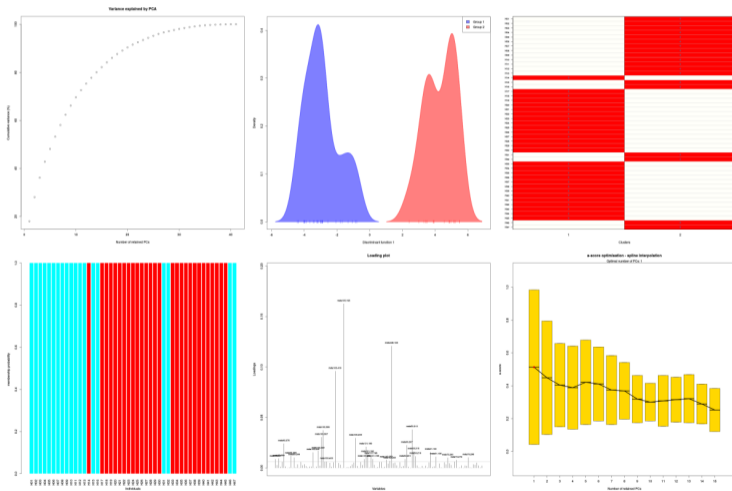


```
1 # See results as text
2 table(pop(hauss.genind),
3       hauss.kfind3$grp)
4 hauss.kfind3
5 # Graph showing table of original
6 # and inferred populations and
7 # assignment of individuals
8 table.value(
9   df=table(pop(hauss.
10  genind), hauss.kfind3$grp),
11  col.lab=paste("Inferred\n
12  cluster",
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



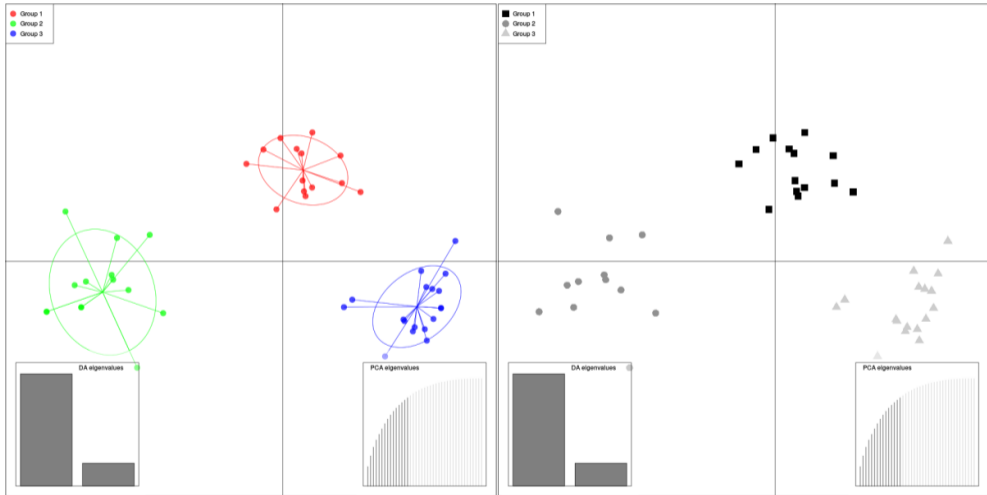


# DAPC code II

```
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 hauss.dapc # Information
10 # A la PCA graph
11 scatter(x=hauss.dapc3, main="DAPC, Taraxacum haussknechtii",
12   bg="white", cex=3, clab=0, col=rainbow(3), posi.da="bottomleft",
13   scree.pca=TRUE, posi.pca="bottomright", leg=TRUE,
14   txt.leg=c("Group 1", "Group 2", "Group 3"), posi.leg="topleft")
```

- Especially graphical parameters have huge possibilities...
- See `?scatter` and play with it...

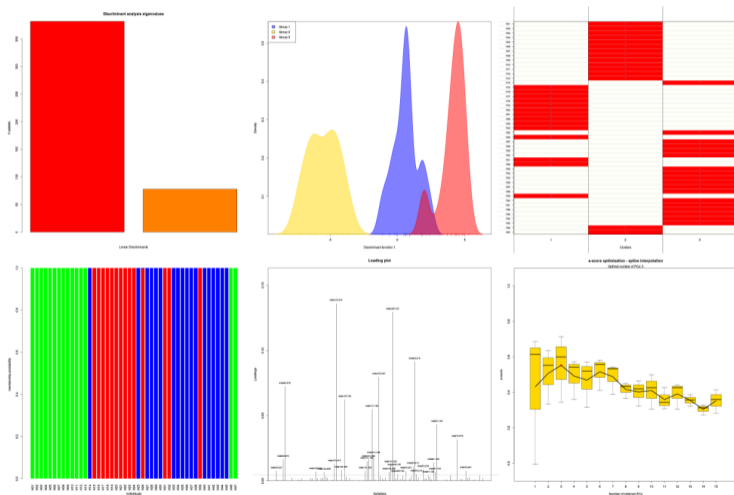
# DAPC for K=3



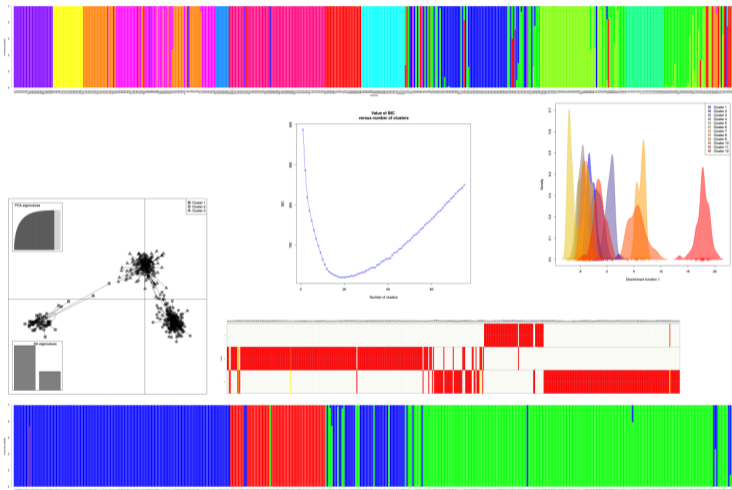
# DAPC code III

```
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=gray.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



# Try DAPC

## Tasks

- 1 Try DAPC with `microbov` dataset ( `data(microbov)` , see `?microbov` ), the U.S. flu dataset (the `usflu.genind` object), or some other data according to your choice.
  - 1 Try K-means clustering as well as DAPC itself.
  - 2 Try various Ks (if appropriate).
  - 3 Try various displays.
- 2 Interpret the results.
- 3 When is DAPC good tool and when should you avoid it?
- 4 Which data can you cluster using K-means clustering?

# 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)
  - 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 II

- 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...
- Here is shown processing option using [ParallelStructure](#), another option is package [strataG](#) (see `?strataG::structure`)
- **Does not work (at all) on Windows** (although see further)

# 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
  - E.g. [Structure Harvester](#) or Structure-sum R script (see later) can do this
- 3 Post process Structure outputs to prepare them for plotting
  - Sort and align Structure outputs
  - Probably most commonly done in [CLUMPP](#)
- 4 Plot the final graphs
  - Can be done in nearly any enough advanced graphical tool (including some R functions), probably the most commonly used is [distruct](#)

# Running Structure in parallel with R

- Let's use modern multi-core CPUs and plenty of RAM in current computers — parallelisation saves time
- **ParallelStructure** R library can optimally distribute computations of independent Structure runs among CPU cores
- When using it, cite [Besnier & Glover 2013](#)
- I show slightly modified way from [The Molecular Ecologist](#)
- Authors recommend to run it without GUI and not on Windows...
- For this chapter start new R project in new working directory

```
1 # Prepare special new empty directory and set working directory
2 setwd("~/dokumenty/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 alphanumeric 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/course/hauss\\_stru.in](https://soubory.trapa.cz/course/hauss_stru.in) and joblist <https://soubory.trapa.cz/course/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 require(Rmpi)
4 # Instead of parallel_structure() use MPI_structure()
5 # with same arguments
6 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, macOS, 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("~/dokumenty/vyuka/r_mol_data/examples/structure/structure_sum/")
```

- When using it, cite [Ehrich 2006](#). If you don't have the script, ask (it is not my so I don't want to post it on the net), see [manual](#)
- Prepare **list\_k.txt** containing on each line K and name of output file
- Get list K (example of the joblist below) from [https://soubory.trapa.cz/rcourse/list\\_k.txt](https://soubory.trapa.cz/rcourse/list_k.txt)

```
2 results_job_S010_f
3 results_job_S011_f
```

```
... ..
```



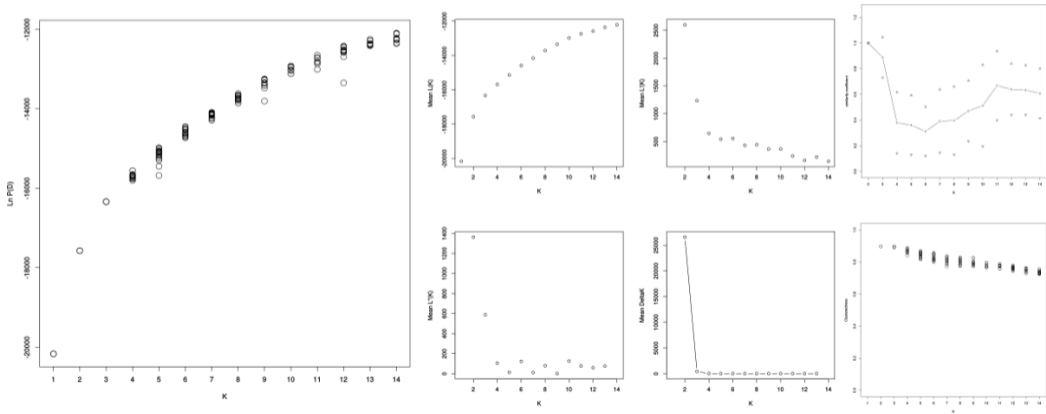
# Run Structure-sum

```
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 columns
9 # (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) # 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

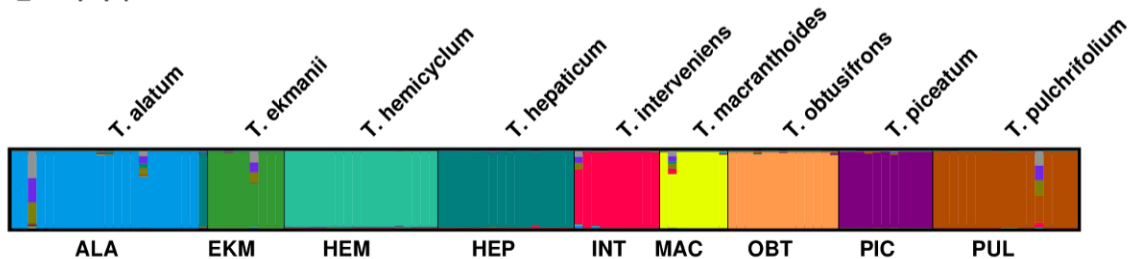


# Example of final Structure plot

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



# Spatial analysis and genetic data

Correlation of genes and space, spatial structure of genotypes

```
1 # After working on STRUCTURE go back to the original working directory
2 # Go to YOUR OWN directory, same as on beginning
3 setwd("~/dokumenty/vyuka/r_mol_data/examples/")
```

## 9 Spatial analysis

Moran's I

sPCA

Monmonier

Mantel test

Geneland

Plotting maps

Tasks

# Short overview of spatial genetics (in R)

## Basic approaches

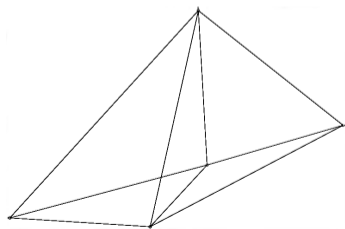
- **Moran's I** — several implementations (here as basic autocorrelation index, in sPCA and in Monmonier's algorithm), generally it is autocorrelation coefficient with broader use
- **Mantel test** — several implementations, popular, although recently criticized as biologically irrelevant, generally 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)
- Plenty of options with plotting maps, including interactive maps (e.g. with **R Leaflet**)
- 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
- Used in many variants for plenty of applications
- This (or similar) approach can be used to test correlation between another characteristics (typically used in ecology)
- Calculations are done according to matrix of geographic distances, or 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 library(adegenet) # Load required libraries
2 library(spdep)
3 # Creates connection network
4 hauss.connectivity <- chooseCN(xy=hauss.genind$other$xy, type=5, d1=0,
5   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 Moran I statistic standard deviate = -18.514, p-value = 1
14 alternative hypothesis: greater
15 sample estimates:
16 Moran I statistic      Expectation      Variance
17   -0.5232003724      -0.0217391304      0.0007336276

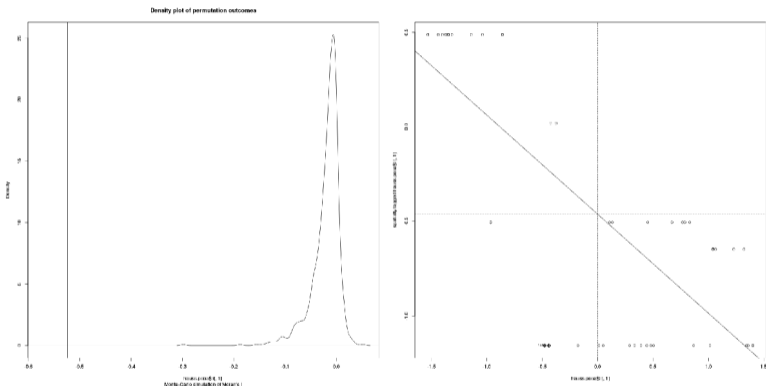
```

# 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

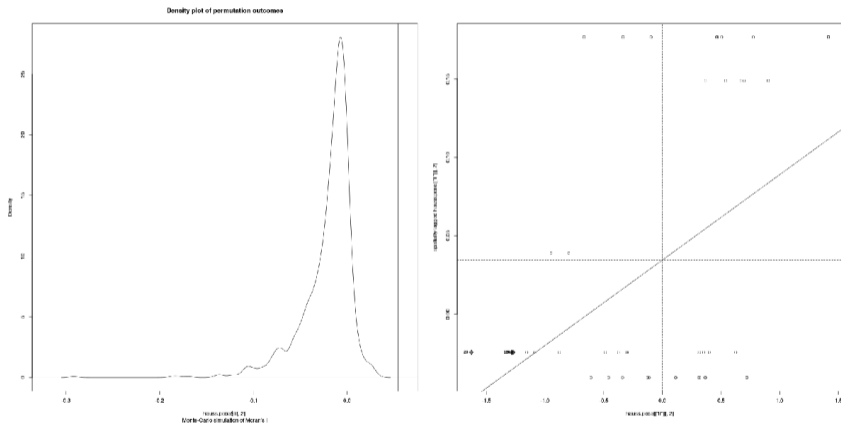


- Tested hypothesis “greater” — **no** significant **positive** autocorrelation
- If testing for hypothesis “less” — significant **negative** autocorrelation — individuals are significantly different

# 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



- 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 # See for more details - select the best "type" for your data
6 chooseCN(xy=rupica$other$xy, ask=TRUE, type=5/6/7, plot.nb=TRUE,
7          edit.nb=FALSE, ...) # Play with settings little bit...

```

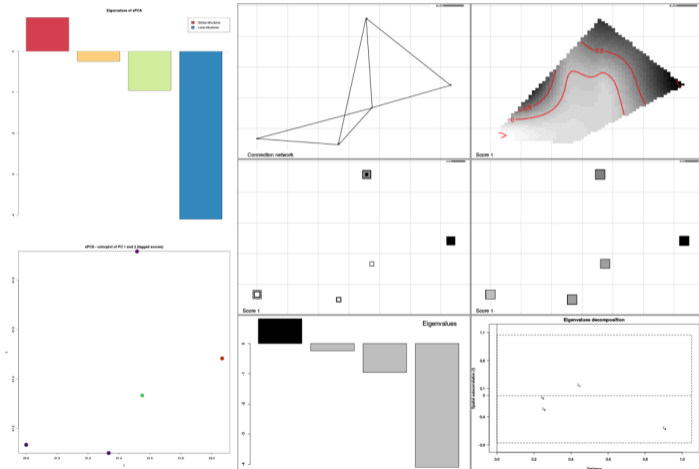
# Calculations of sPCA

```

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

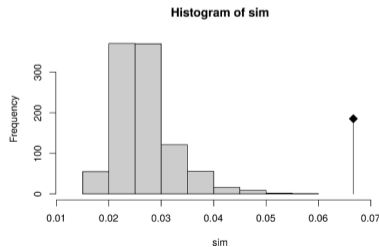
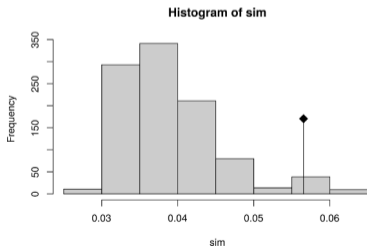


# Test if global/local structure is significant

```

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)

```



# 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], add.p=TRUE,
12   csize=0.5, sub="PCoA - first PC", csub=2, possub="topleft")
13 # For 2nd axis
14 image(x=hauss.spca.temp, col=spectral(100))
15 s.value(dfxy=hauss.genind$other$xy, z=hauss.pcoa[["li"]][,2],
16   add.p=TRUE, csize=0.5, sub="PCoA - second PC", csub=2, possub="topleft")

```

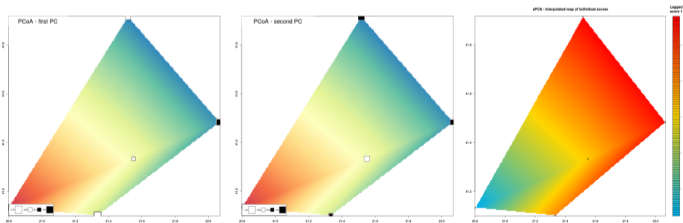


# sPCA outputs II

```

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())

```

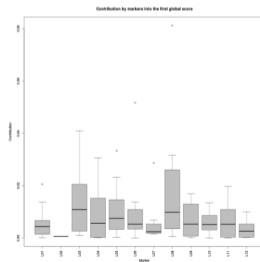
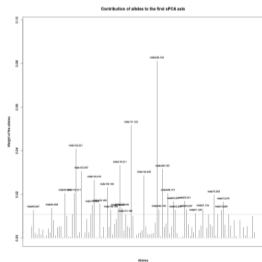


# Loading plots — which alleles contribute the most?

```

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 into
7   the first global score", col="gray")

```

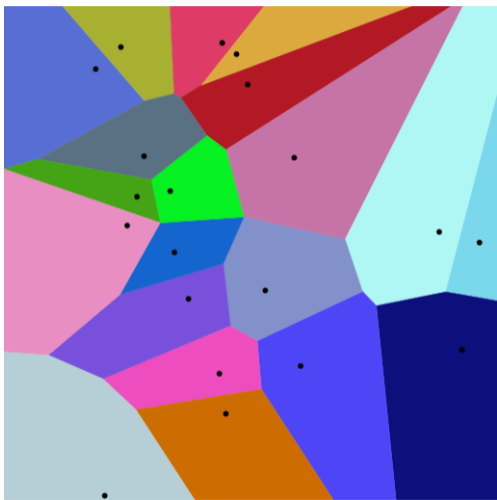


# Monmonier's algorithm — genetic boundaries

- 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](#) — it is used by a lot of spatial analysis, especially by tools relying on Bayesian statistics

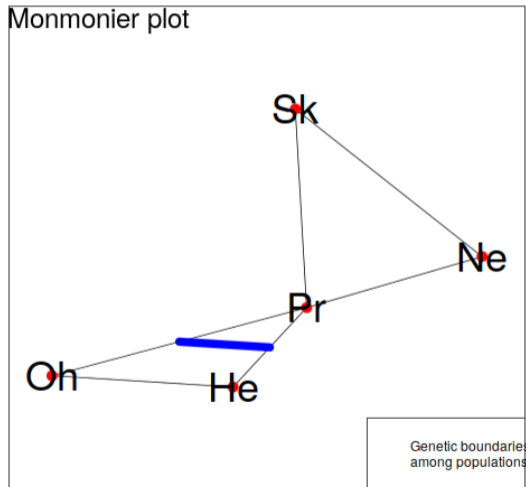
```
1 # Calculates Monmonier's function (for threshold use 'd')
2 hauss.monmonier <- monmonier(xy=hauss.genpop$other$xy, dist=dist(
3   hauss.genpop@tab), cn=chooseCN(hauss.genpop$other$xy, ask=FALSE,
4   type=6, k=2, plot.nb=FALSE, edit.nb=FALSE), nrun=1)
5 coords.monmonier(hauss.monmonier) # See result 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
- The more similar sizes of polygons and the more even distribution of samples, the more stable and reliable results we get

# 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`) by something like:

```
1 monmonier(..., dist=dudi.pco(d=dist.gene(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
- When running `monmonier(...)` when it asks for threshold of sorted local distances, try several values and see differences in output
- 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
- Distance matrix can be calculated as simple Euclidean geometric distance by `dist()`, or for larger areas by geodesic distance along Earth surface (e.g. `pegas::geod()`), or using special package like [geosphere](#)

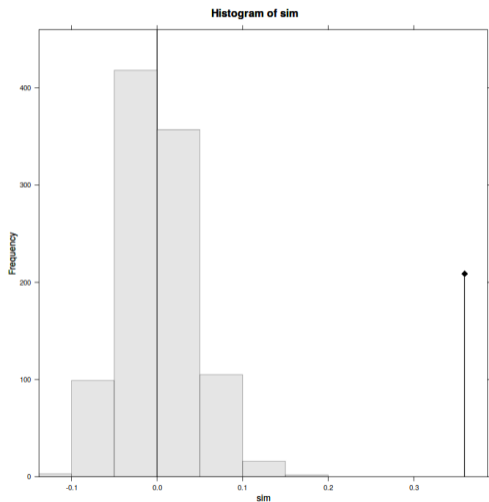
```
library(pegas) # Geodesic distance
```

# Mantel test — isolation by distance

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

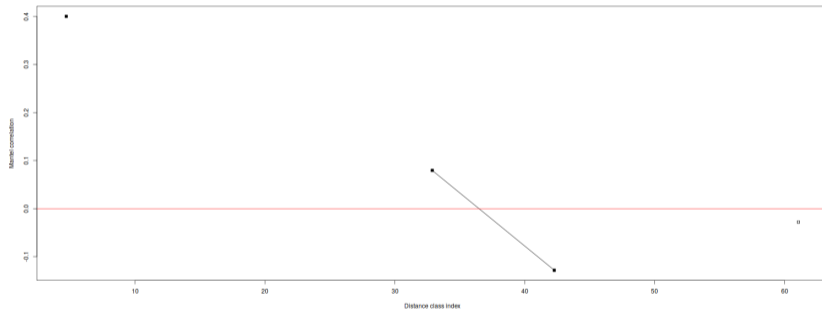
```

```

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      4.697165 532.000000  0.400101  0.0010  0.000999 ***
9 D.cl.2     14.091494  0.000000      NA      NA      NA
10 D.cl.3     23.485823  0.000000      NA      NA      NA
11 D.cl.4     32.880153  52.000000  0.079433  0.0490  0.048951 *
12 D.cl.5     42.274482 466.000000 -0.128236  0.0010  0.002997 **
13 D.cl.6     51.668811  0.000000      NA      NA      NA
14 D.cl.7     61.063140  36.000000 -0.027890  0.2547  0.254745
15 ...
16 ---
17 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

# About Geneland

- For installation see slide [57](#)
- 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://i-pri.org/special/Biostatistics/Software/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

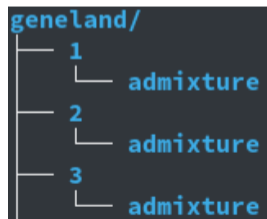
```
1 # Geneland requires specific coordinate space
2 # hauss.coord is DF, we need just plain matrix
3 hauss.geneland.coord <- as.matrix(hauss.coord)
4 colnames(hauss.geneland.coord) <- c("X", "Y")
5 attr(hauss.geneland.coord, "projection") <- "LL"
6 attr(hauss.geneland.coord, "zone") <- NA
7 hauss.geneland.coord.utm <- 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= "https://soubory.trapa.cz/rcourse/
15   haussknechti_geneland.txt", na.string="-999", header=FALSE, sep="\t")
16 dim(hauss.geneland.data)
17 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 analysis 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  $1000\text{--}10000^{\text{th}}$  generation
- Denser sampling produces smoother data, but can consume too much disk space...

Directory structure for Geneland:



# 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=hauss.geneland
15  .data, path.mcmc=hauss.geneland.path.mcmc, delta.coord=0.001, varnpop=
16  TRUE, npopmin=1, npopmax=hauss.geneland.maxpop, nit=10000, thinning=10,
17  freq.model="Uncorrelated", spatial=TRUE) # Loop continues on next slide
```



# Running MCMC

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

```

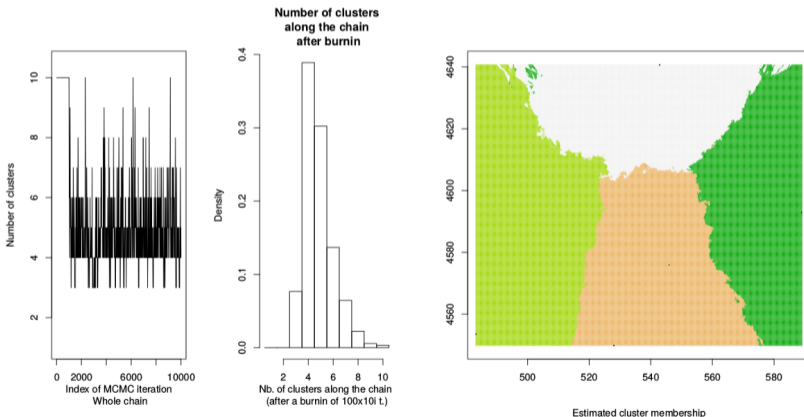
1 # Calculate average posterior probability
2 hauss.geneland.lpd <- rep(NA, hauss.geneland.nrun)
3 for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
4   hauss.geneland.path.mcmc <- paste("geneland/", hauss.geneland.irun, "/",
5     sep="")
6   hauss.geneland.path.lpd <- paste(hauss.geneland.path.mcmc,
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 order(hauss.geneland.lpd, decreasing=TRUE) # Sorts runs according to decre-
11 [1] 5 1 4 3 2 # Run 5 is the best here      # using posterior probability
12 hauss.geneland.lpd # Here the runs are unsorted
13 [1] -645.0238 -782.7912 -676.9559 -664.9947 -601.7902 # Run 5 wins

```

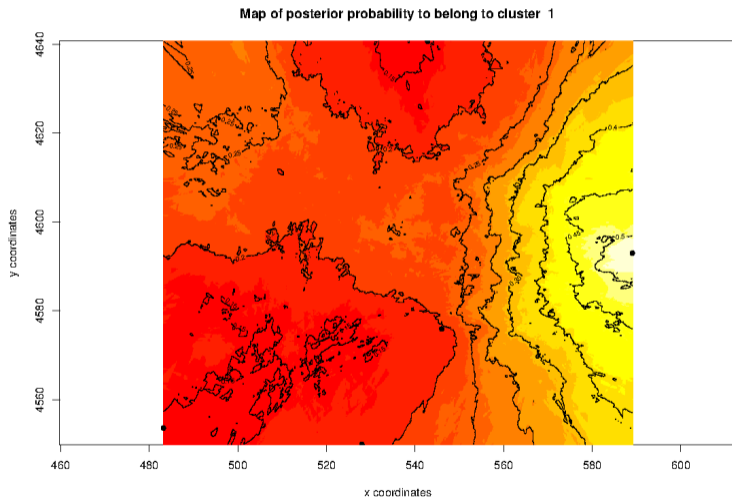
- We will use figures and  $F_{ST}$  outputs only from the best run
- It is useful to keep all runs especially for comparison if there are different solutions with similar posterior probability

# 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





## 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](#)

# Mapping overview

- There are plenty of options, only some basic options are shown...
- Some packages offer basic (state) maps
- It is possible to use various on-line maps
- [Shapefiles](#) (SHP) used in specialized GIS software can be used as base layer to plot points etc. to that map
- Interactive maps can be created with libraries like [leaflet](#)
- It is possible to plot to maps not only point and/or text labels, but also pie charts, bar charts, or e.g. [plot phylogeny to the map](#)
- Much more options are available in specialized mapping packages, especially with connections with GIS (e.g. [QGIS](#) or [GRASS GIS](#))

# Very basic mapping in R

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

# Basic map

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



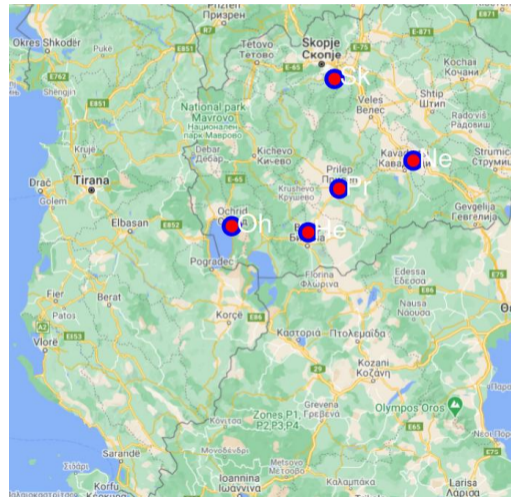
# Google map basics

1 # Basic help for Google maps

2 ?GetMap

3 ?PlotOnStaticMap

- For larger/complex maps Google requires **API key**
- See **documentation** and more options, it can use plenty of map resources (Bing maps, ...)
- Map is first downloaded with `GetMap()` into R object, and then plotted with `PlotOnStaticMap()` or with another function
- Many on-line map services do require (paid) API key...



# Plotting Google map

```
1 # Download map
2 hauss.gmap <- GetMap(center=c(lat=41, lon=21), size=c(640, 640),
3   destfile="gmap.png", zoom=8, maptype="satellite")
4 # Plot saved map, with extra data
5 PlotOnStaticMap(MyMap=hauss.gmap, lat=hauss.genpop@other$xy[["lat"]],
6   lon=hauss.genpop@other$xy[["lon"]], FUN=points, pch=19, col="blue",
7   cex=5)
8 PlotOnStaticMap(MyMap=hauss.gmap, lat=hauss.genpop@other$xy[["lat"]],
9   lon=hauss.genpop@other$xy[["lon"]], add=TRUE, FUN=points, pch=19,
10  col="red", cex=3)
11 PlotOnStaticMap(MyMap=hauss.gmap, lat=hauss.genpop@other$xy[["lat"]],
12  lon=hauss.genpop@other$xy[["lon"]], add=TRUE, FUN=text,
13  labels=as.vector(popNames(hauss.genind)), pos=4, cex=3, col="white")
14 # Google maps have their own internal scaling, adding of points by
15 # standard functions 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),asp=1,lwd=1.5)
12 box() # Add frame around the map
13 # Plot the pie charts
14 for (L in 1:5) { add.pie(z=hauss.pie[L,], x=as.vector(
15   hauss.genpop@other$xy[["lon"]])[L], y=as.vector(
16   hauss.genpop@other$xy[["lat"]])[L], labels=names(hauss.pie[L,]),
17   radius=0.1, col=topo.colors(3)) }
```

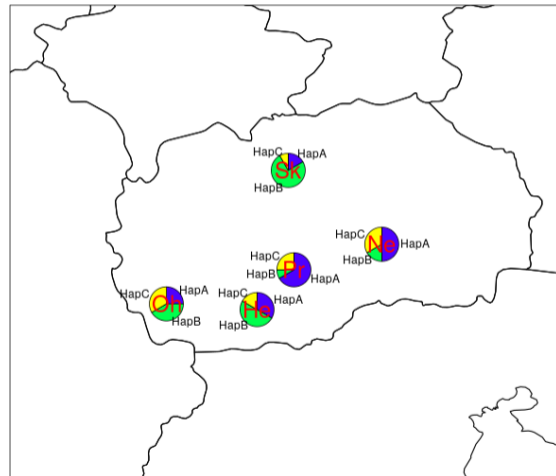
## Adding pie charts to map II

```

1 ?add.pie # See more options
2 # Add population labels
3 text(
4   x=hauss.genpop@other$xy[["lon"]],
5   y=hauss.genpop@other$xy[["lat"]],
6   labels=
7   as.vector(popNames(hauss.genind)),
8   col="red", cex=2)

```

- As usually, play with colours and display options...





# Pie charts on Google map I

```

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

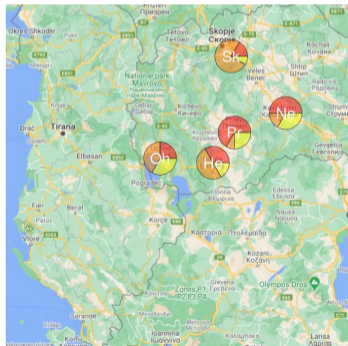
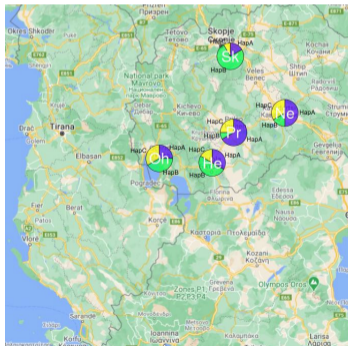
```

# Pie charts on Google map II

```

1 # Add population text labels
2 PlotOnStaticMap(MyMap=hauss.gmap, 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=2.5, col="white")

```



# Datasets from mapproj

```
1 # Plot on data sets from mapproj package
2 library(maps) # Various mapping tools (plotting, ...)
3 # More detailed maps, but political boundaries often outdated, see
4 # https://CRAN.R-project.org/package=mapdata
5 library(mapdata)
6 library(mapproj)
7 # Convert latitude/longitude into projected coordinates
8 # Plot a map, check parameters
9 # Check among others "projection" and ?mapproject for its details
10 map(database="world", boundary=TRUE, interior=TRUE, fill=TRUE,
11      col="lightgrey", plot=TRUE, xlim=c(16, 27), ylim=c(37, 46))
12 # If you'd use projection, use mapproject to convert also coordinates!
13 ?mapproject # See for details
14 points(x=hauss.genpop@other$xy[["lon"]], y=hauss.genpop@other$xy[["lat"]],
15        pch=15:19, col="red", cex=3)
```

# Plotting on SHP files I

- **Shapefile** (SHP) is common format for geographical data
- Get SHP files from <https://soubory.trapa.cz/rcourse/macedonia.zip> (from <https://download.geofabrik.de/europe/macedonia.html>) and unpack them into R working directory
- R working directory has to contain also respective DBF and SHX files (same name, only different suffix)

```
1 library(maptools)
2 library(rgdal)
3 dir() # Verify required files are unpacked in the working directory
4 # Check correct import by plotting all layers
5 macedonia_building <- readOGR(dsn="macedonia_buildings.shp")
6 plot(macedonia_building)
7 macedonia_landuse <- readOGR(dsn="macedonia_landuse.shp")
8 plot(macedonia_landuse)
```

# Plotting on SHP files II

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

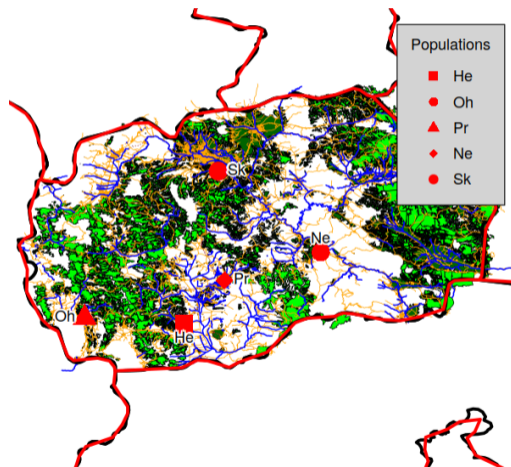
# Plotting on SHP files III

```

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="world", 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$xy[["lon"]],
8     y=hauss.genpop@other$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", "Sk"),
16     col="red", border="black", pch=15:19, pt.cex=2, bty="o",
17     bg="lightgrey", box.lwd=1.5, cex=1.5, title="Populations")

```

# Plotting on SHP files IV and maps from mapproj



# Practicing work with spatial data

## Tasks

- 1 Try some with the spatial analysis with adegenet's alpine *Rupicapra* dataset (see `?adegenet::rupica`), or cats from Nancy (see `?adegenet::nancycats`), or with any other (your) data.
- 2 Plot some of the above dataset, or some your data to map. Try several mapping options.
- 3 Which problems can you encounter when doing spatial analysis or plotting maps over large spatial scale (level of continents and above), close to equator or in polar regions?
- 4 What are possible problems when computing distances among sampling sites?
- 5 Search the Internet for another options how to plot maps in R which could be suitable for your data.



# 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

MP

Seeing trees in the forest

Comparisons

Notes about plotting the trees

Tasks

```
1 # Library  
2 library(ape)
```

# Working with phylogenetic trees in R I

- To import one or more tree(s) in **NEWICK** (`*.tre`, `*.nwk`, ...) use `read.tree()`, for trees in **NEXUS** (`*.nex`, `*.nexus`, ...) use `read.nexus()`
- To export tree in NEWICK use `write.tree()`, in NEXUS use `write.nexus()`
- Another tree (ape's class `phylo`) formats can be imported/exported/manipulated with e.g. functions from `treeio` package
- Some functions manipulating trees can work only in single tree (class `phylo`), some can work with multiple trees (class `multiPhylo`)
- `ape` has plenty of `*.multiPhylo` functions to handle tree sets (e.g. `root.phylo` vs. `root.multiPhylo`)
- If the function needed does not work with multiple trees, use `lapply()` (see further examples)

# Working with phylogenetic trees in R II

- Typical operations with trees in R
  - Calculation of individual phylogenetic trees — NJ, MP, ML, ...
  - Plotting, including various highlights and information on tips, nodes, labels, ...
  - Plotting two or more trees together to compare topologies (trees from various methods/genes, tree of plants vs. polynators, ...)
  - Preparing trees for subsequent analysis (e.g. character evolution) — removal of outgroup(s), binding of trees (e.g. crown group from one gene and and stem lineages from another), removal of certain taxa, ...
  - Comparison and evaluation of multiple gene trees, identification of trees with outlying topology
  - Construction of species tree or network from set of multiple gene trees
  - And more...
- Plenty of packages available...
- Users of `ggplot2` can e.g. use `ggtree`

# Rooting and unrooting trees I

```
1 plot.phylo(hauss.nj)
2 print.phylo(hauss.nj)
3 tiplabels() # Shows tip numbers
4 # root.phylo accepts either tip number or tip label
5 # resolve.root=TRUE ensures root will be bifurcating
6 # (without this parameter it sometimes doesn't work)
7 # outgroup can be single value or vector of multiple tips
8 hauss.nj.rooted <- root.phylo(phy=hauss.nj, resolve.root=TRUE,
9   outgroup=16) # Or
10 hauss.nj.rooted <- root.phylo(phy=hauss.nj, resolve.root=TRUE,
11   outgroup="H16") # Or
12 hauss.nj.rooted <- root.phylo(phy=hauss.nj, resolve.root=TRUE,
13   outgroup=c("H42", "H43"))
14 # root.multiPhylo() is an alias for root.phylo() for class multiPhylo
15 print.phylo(hauss.nj.rooted)
16 plot.phylo(hauss.nj.rooted)
```

# Rooting and unrooting trees II & swap clade

```
1 # Check if it is rooted - returns TRUE/FALSE
2 is.rooted(hauss.nj.rooted)
3 is.rooted(hauss.nj)
4 # Root the tree interactively - click to selected tip
5 plot.phylo(hauss.nj)
6 hauss.nj.rooted <- root.phylo(phy=hauss.nj, interactive=TRUE)
7 plot.phylo(hauss.nj.rooted)
8 ?unroot.phylo # Unroot the tree
9 # root(), unroot() and is.rooted() works with single or multiple trees
10 # (class phylo or multiPhylo)
11 # Rotate (swap) clade
12 # plot.phylo plots tree in exact order as it is in the phylo object
13 plot.phylo(hauss.nj)
14 nodelabels()
15 hauss.nj.rotated <- ape::rotate(phy=hauss.nj, node=74)
16 plot.phylo(hauss.nj.rotated)
17 nodelabels()
```

# Ladderize tree, drop fossil (extinct) tips

```
1 plot.phylo(hauss.nj)
2 # Ladderize the tree - step-wise rotation of nodes
3 hauss.nj.ladderized <- ladderize(hauss.nj)
4 # Topology is unchanged, nodes are only rotated
5 plot.phylo(hauss.nj.ladderized)
6 # Drop "extinct" tips - those who don't reach end the tree
7 # tolerance is respective to the used metrics
8 plot.phylo(hauss.nj)
9 axisPhylo()
10 hauss.nj.fossil <- drop.fossil(phy=hauss.nj, tol=0.4)
11 plot.phylo(hauss.nj.fossil)
12 # See details
13 ?drop.fossil
```

# Extract clade, part of tree

```
1 # Plot source tree
2 plot.phylo(hauss.nj)
3 # See node labels (numbers) - needed for some tasks
4 nodelabels()
5 # Non-interactively extract clade
6 hauss.nj.extracted <- extract.clade(phy=hauss.nj, node=60)
7 # See new extracted tree
8 plot.phylo(hauss.nj.extracted)
9 # Interactive extraction
10 plot.phylo(hauss.nj)
11 # Select clade to extract by clicking on it
12 hauss.nj.extracted <- extract.clade(phy=hauss.nj, interactive=TRUE)
13 # See new extracted tree
14 plot.phylo(hauss.nj.extracted)
15 # See more options
16 ?extract.clade
17 ?keep.tip # Does the opposite - keeps only selected tip(s)
```

# Bind two trees into one

```
1 # Bind donor "y" tree to a given position of the "x" tree
2 ?bind.tree # See options
3 plot.phylo(hauss.nj.fossil)
4 nodelabels()
5 plot.phylo(hauss.nj.extracted)
6 nodelabels()
7 hauss.nj.bind <- bind.tree(x=hauss.nj.fossil, y=hauss.nj.extracted,
8   where="root", position=0, interactive=FALSE)
9 plot.phylo(hauss.nj.bind)
10 # Bind two trees interactively
11 # Plot tree receiving the new one
12 plot.phylo(hauss.nj.fossil)
13 # Select where to bind new tree to
14 hauss.nj.bind <- bind.tree(x=hauss.nj.fossil, y=hauss.nj.extracted,
15   interactive=TRUE)
16 plot.phylo(hauss.nj.bind)
```



# Work with multiple trees

```
1 # Read tree(s) in NEWICK format - single or multiple tree(s)
2 oxalis.trees <- read.tree("https://soubory.trapa.cz/course/oxalis.nwk")
3 oxalis.trees
4 lapply(X=oxalis.trees, FUN=print.phylo)
5 plot.multiPhylo(x=oxalis.trees) # See all trees
6 summary(oxalis.trees)
7 length(oxalis.trees)
8 names(oxalis.trees)
9 # Export trees in NEWICK format
10 write.tree(phy=oxalis.trees, file="trees.nwk")
11 # Export trees in NEXUS format
12 write.nexus(oxalis.trees, file="trees.nexus")
13 # Root all trees
14 oxalis.trees.rooted <- root.multiPhylo(phy=oxalis.trees,
15   outgroup="O._fibrosa_S159", resolve.root=TRUE)
16 lapply(X=oxalis.trees.rooted, FUN=print.phylo)
17 plot.multiPhylo(x=oxalis.trees) # See all trees
```

# Drop a tip

```
1 plot.phylo(hauss.nj)
2 hauss.nj[["tip.label"]]
3 tiplabels()
4 hauss.nj.drop <- drop.tip(phy=hauss.nj, tip=47) # Or
5 hauss.nj.drop <- drop.tip(phy=hauss.nj, tip="H31") # Or
6 hauss.nj.drop <- drop.tip(phy=hauss.nj, tip=c("H18", "H29", "H31"))
7 plot.phylo(hauss.nj.drop)
8 # Drop a tip from multiPhylo
9 plot.multiPhylo(x=oxalis.trees)
10 # See tip labels
11 oxalis.trees[[1]][["tip.label"]]
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 lapply(X=oxalis.trees.drop, FUN=print.phylo)
```

# Branch lengths and ultrametricity of the tree

```
1 # Check if the tree is ultrametric - is variance of distances of all
2 # tips to node 0? It is required for some analysis
3 is.ultrametric(oxalis.trees)
4 # Fitting a chronogram to a phylogenetic tree whose branch lengths are
5 # in number of substitution per sites - force tree to be ultrametric
6 ?chronos
7 # Compute branch lengths for trees without branch lengths
8 ?compute.brLen
9 # Computes the branch lengths of a tree giving its branching times
10 # (aka node ages or heights)
11 ?compute.brtime
```

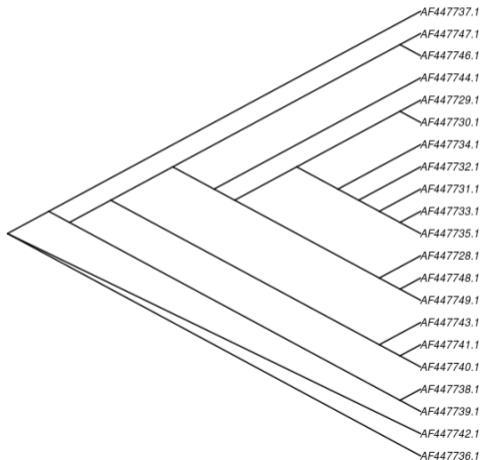
- Class `multiPhylo` is just a `list` of `phylo` objects to store multiple trees — you can perform most of analysis on it as on `phylo`, commonly using `lapply` (afterward use `class(x) <- "multiPhylo"`)

# 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 185) is used to start the search
- With rising performance of computers, it uses to be replaced maximum likelihood or Bayesian methods
- Still underlying plenty of analysis, not only reconstruction of phylogeny

# Maximum parsimony – code and result

Maximum-parsimony tree of *Gunnera* spp.



```
1 library(phangorn)
2 # Conversion to phyDat for phangorn
3 gunnera.phydat <-
4   as.phyDat(gunnera.mafft.ng)
5 # Prepare starting tree
6 gunnera.tre.ini <- nj(dist.dna(
7   x=gunnera.mafft.ng, model="raw"))
8 ?parsimony # Parsimony details
9 # Returns maximum parsimony score
10 parsimony(tree=gunnera.tre.ini,
11   data=gunnera.phydat)
12 # Optimisation - returns MP tree
13 gunnera.tre.pars <- optim.parsimony
14   (tree=gunnera.tre.ini,
15   data=gunnera.phydat)
16 plot.phylo(x=gunnera.tre.pars,
17   type="cladogram", edge.width=2)
```

# Topographical distances among trees I – implementations I

- 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
  - If the difference is very close to root, RF value can be large, even there are not much differences in the tree at all – `dist.multiPhylo` from package `distory` can be an alternative, although interpretation of that geodesic distance is sometimes not so straightforward as simple logic of RF
- 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

# Topographical distances among trees I – implementations II

- Test it with `ade4::is.euclid`, can be scaled (forced to become Euclidean) by functions like `quasieuclid` or `cailliez` in `ade4` – carefully, it can damage meaning of the data
- We get matrix of pairwise differences among trees (from multiple genes), we need display and analyze it
- Set of tools for identifying discordant phylogenetic trees are e.g. in package `kdetrees`
- Other approach are phylogenetic networks searching consensus among phylogenetic trees
- In any case, there are plenty of options how to display the differences among the trees

# 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 # Compute matrix of topological distances among phylogenetic trees
5 ?dist.topo # See details of available computing methods
6 oxalis.trees.d <- dist.topo(x=oxalis.trees, method="score")
7 # Basic information about the distance matrix
8 dim(as.matrix(oxalis.trees.d))
9 head.matrix(as.matrix(oxalis.trees.d))
```

- There are more options how to display the differences and identify (and possibly exclude) outlying trees — heatmap, PCoA, hierarchical clustering (e.g. package [dbscan](#)), ...
- Sources of incongruencies among trees: low-quality DNA/laboratory mistake, problem with alignment/gene tree reconstruction, gene duplication and [paralogy](#) (e.g. in polyploids), [ILS](#), [HGT](#), ... — such problems must be inspected



# 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
- There are plenty of heatmap functions, like `heatmap`, `heatmap.plus::heatmap.plus`, and more...

```

1 # Create heat maps using heatmap.2 function from gplots package
2 heatmap.2(x=as.matrix(x=oxalis.trees.d), Rowv=FALSE, Colv="Rowv",
3   dendrogram="none", symm=TRUE, scale="none", na.rm=TRUE, revC=FALSE,
4   col=rainbow(15), cellnote=round(x=as.matrix(x=oxalis.trees.d), digits=2),
5   notecex=1, notecol="white", trace="row", linecol="black",
6   labRow=names(oxalis.trees), labCol=names(oxalis.trees), key=TRUE,
7   keysize=2, density.info="density", symkey=FALSE, main="Correlation
8   matrix of topographical distances", xlab="Trees", ylab="Trees")

```

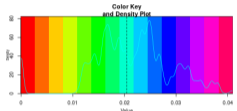
# Topographical distances among trees IV

Calculate Robinsons-Foulds distance matrix among trees and plot it

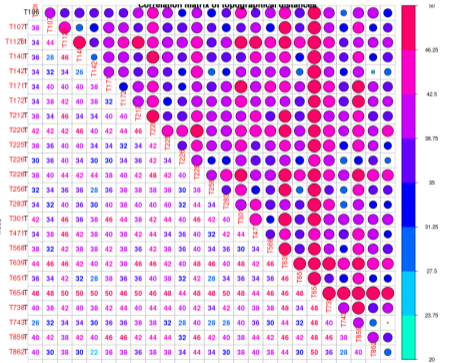
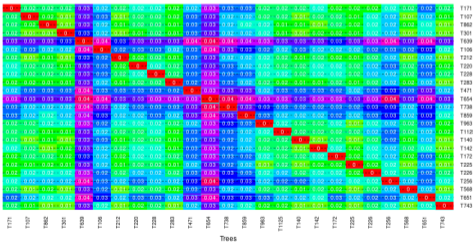
- `phytools::multiRF` can handle `multiPhylo` objects

```
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")
```

# Topographical distances among trees V – the matrices



Correlation matrix of topographical distances

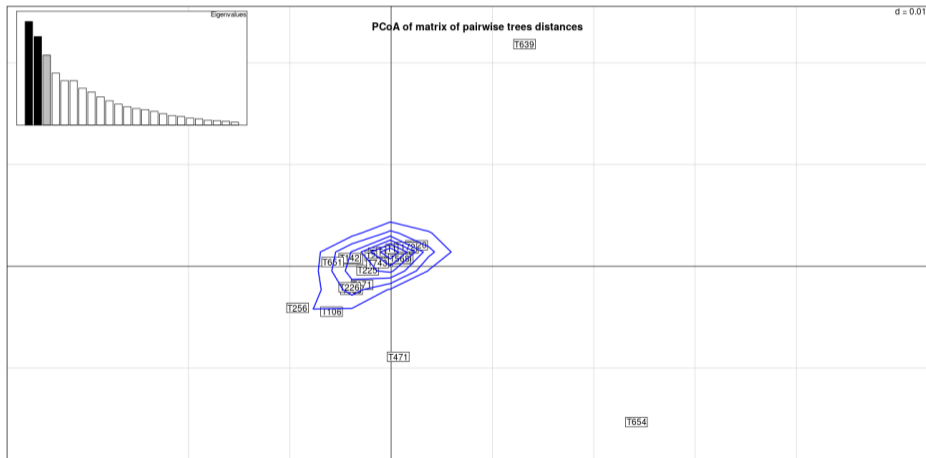


# PCoA from distance matrices of topographical differences among trees — the code

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

```
1 # Test if the distance matrix is Euclidean or not
2 library(ade4)
3 is.euclid(distmat=oxalis.trees.d, plot=TRUE)
4 [1] TRUE # If FALSE, we can use e.g. quasieuclid() to make it Euclidean
5 # Calculate the PCoA
6 oxalis.trees.pcoa <- dudi.pco(d=oxalis.trees.d, scannf=TRUE, full=TRUE)
7 # Plot PCoA and add kernel densities
8 s.label(dfxy=oxalis.trees.pcoa$li)
9 s.kde2d(dfxy=oxalis.trees.pcoa$li, cpoint=0, add.plot=TRUE)
10 # Add histogram of eigenvalues
11 add.scatter.eig(oxalis.trees.pcoa[["eig"]], 3,1,2, posi="topleft")
12 # Add title to the plot
13 title("PCoA of matrix of pairwise trees distances")
14 scatter(x=oxalis.trees.pcoa, posieig="topleft") # Alternative plotting PCA
```

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



# kdetrees — identification of outlying trees I

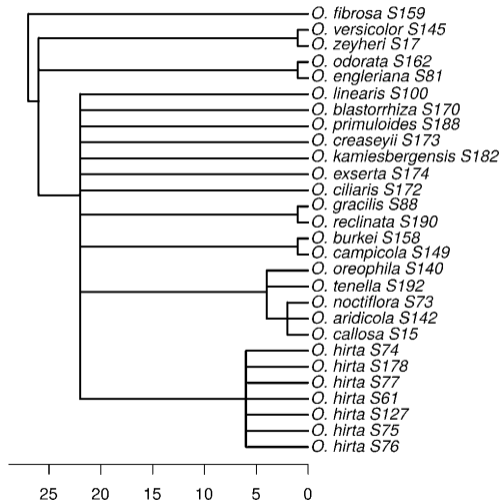
- Distance-based method of identification of trees with significantly different topology
- Function `kdetrees` has plenty of options...
- Parameter `k` sets threshold for trees to be removed — it requires repeated running with different `k` (and plotting the figures) to decide which trees to remove and which to keep

```
1 # Load library
2 library(kdetrees)
3 ?kdetrees # See options
4 # Run main function - play with parameter k
5 oxalis.kde <- kdetrees(trees=oxalis.trees, k=0.4,
6   distance="dissimilarity", topo.only=FALSE, greedy=TRUE)
7 # See results
8 oxalis.kde
9 plot(oxalis.kde)
10 hist(oxalis.kde)
```

## kdetrees — identification of outlying trees II

```
1 # See removed trees
2 plot.multiPhylo(oxalis.kde[["outliers"]])
3 # Save removed trees
4 write.tree(phy=oxalis.kde[["outliers"]], file="oxalis_trees_outliers.nwk")
5 # Save kdetrees report
6 write.table(x=as.data.frame(x=oxalis.kde), file="oxalis_trees_scores.tsv",
7   quote=FALSE, sep="\t")
8 # Extract passing trees
9 oxalis.trees.good <- oxalis.trees[names(oxalis.trees) %in%
10   names(oxalis.kde[["outliers"]]) == FALSE]
11 oxalis.trees.good
12 # Save passing trees
13 write.tree(phy=oxalis.trees.good, file="trees_good.nwk")
```

# Consensus tree

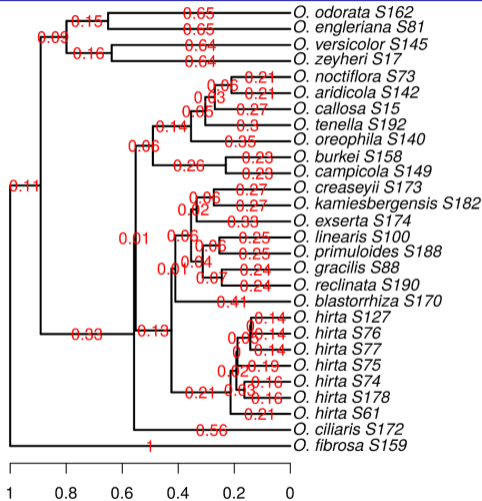


- Very basic method simply collapsing all nodes with lower than required (typically 50–75%) identity
- There is no underlying model, only comparison of topologies

```
1 # Consensus tree (50% rule)
2 oxalis.tree.con <- consensus
3   (oxalis.trees.rooted, p=0.5,
4   check.labels=TRUE)
5 print.phylo(oxalis.tree.con)
6 # Plot the tree
7 plot.phylo(oxalis.tree.con,
8   edge.width=2, label.offset=0.3)
9 axisPhylo(side=1)
10 # What a nice tree... :-P
```



# Species tree – all trees must be ultrametric

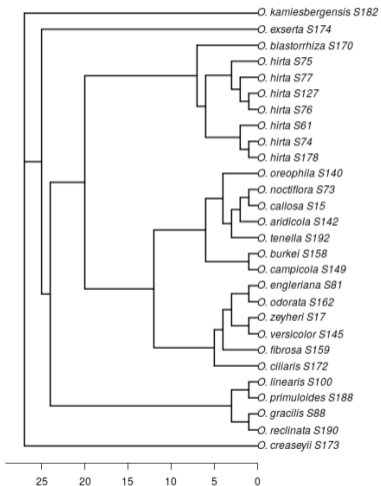


```

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 ?speciesTree # Mean distances
8 oxalis.tree.sp.mean <- speciesTree
9   (x=oxalis.trees.ultra, FUN=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



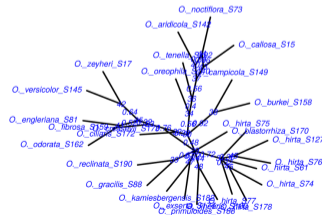
```

1 library(phytools)
2 ?mrp.supertree # See details
3 oxalis.tree.sp <- mrp.supertree
4   (tree=oxalis.trees.rooted,
5     method="optim.parsimony",
6     rooted=TRUE)
7 print.phylo(oxalis.tree.sp)
8 plot.phylo(oxalis.tree.sp,
9   edge.width=2, label.offset=0.01)
10 axisPhylo(side=1)
11 # Similar function
12 ?phangorn::superTree
13 # Coalescence model to handle
14 # multiple individuals per species
15 ?phangorn::coalSpeciesTree

```

# Networks

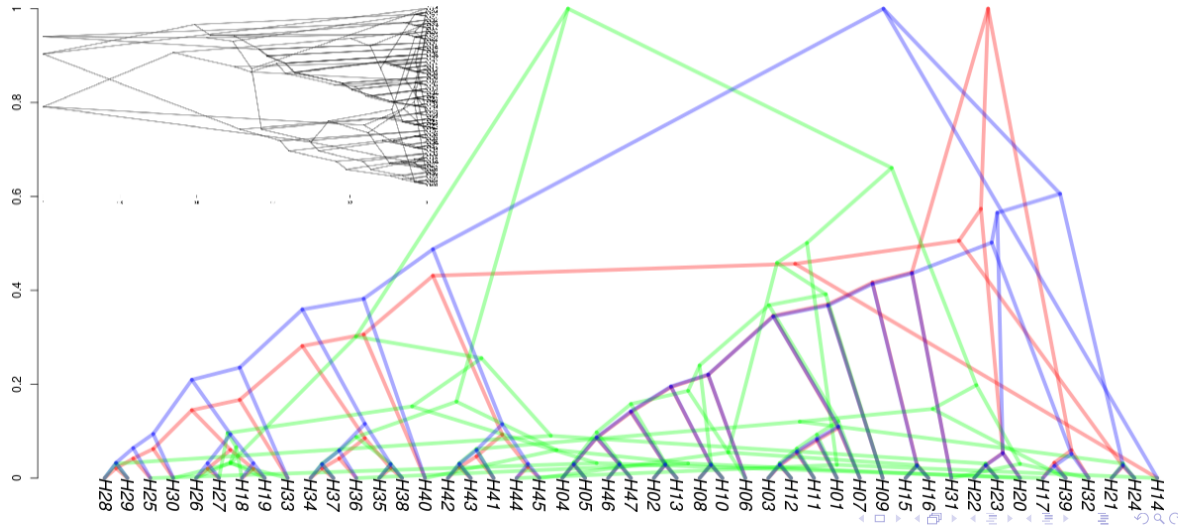
```
1 library(phangorn)
2 oxalis.tree.net <- consensusNet(oxalis.trees.rooted, prob=0.25)
3 plot(x=oxalis.tree.net, planar=FALSE, type="2D", use.edge.length=TRUE,
4     show.tip.label=TRUE, show.edge.label=TRUE, show.node.label=TRUE,
5     show.nodes=TRUE, edge.color="black", tip.color="blue") # 2D
6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



# Density tree I

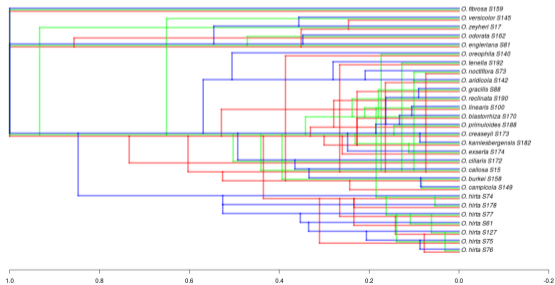
```
1 # Prepare list of trees to show
2 hauss.nj.trees <- list(hauss.nj, hauss.nj.bruvo, hauss.nj.rooted)
3 hauss.nj.trees <- lapply(X=hauss.nj.trees, FUN=compute.brLen)
4 hauss.nj.trees <- lapply(X=hauss.nj.trees, FUN=chronos)
5 class(hauss.nj.trees) <- "multiPhylo"
6 # The trees should be (otherwise 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

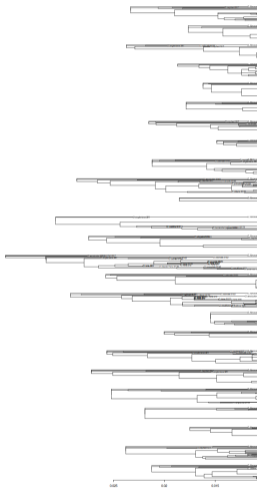


# Density tree III

```
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
```



# Kronoviz — see all trees on same scale



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

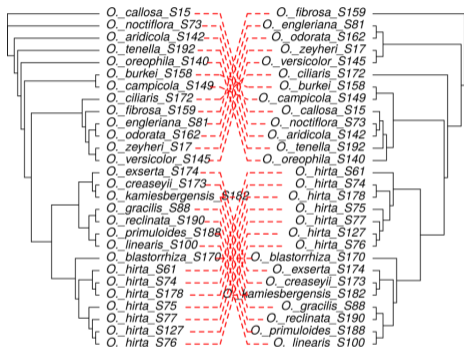
- The plot can be very long and it can be hard to see details
- But one can get impression if all trees are more or less in same scale (have comparable length) or not

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



# Cophyloplot comparing two trees

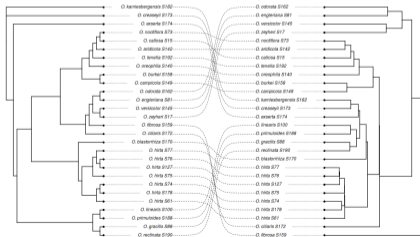


- `ladderize()` pre-sorts tips in the tree — it can help to `cophyloplot()` to reduce crossings

- `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
- Other options are in package `dendextend`

# Alternative implementation — phytools::cophylo

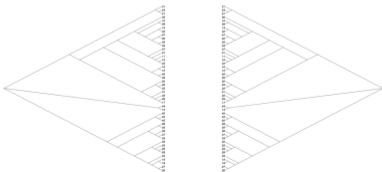
```
1 ?cophylo # See options
2 # Prepare the object for plotting
3 oxalis.cophylo <- cophylo(tr1=oxalis.tree.sp, tr2=oxalis.tree.sp.mean,
4   assoc=(cbind(sort(oxalis.tree.sp$tip.label),
5     sort(oxalis.tree.sp$tip.label))), rotate=TRUE)
6 plot.cophylo(x=oxalis.cophylo, lwd=2, link.type="curved") #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



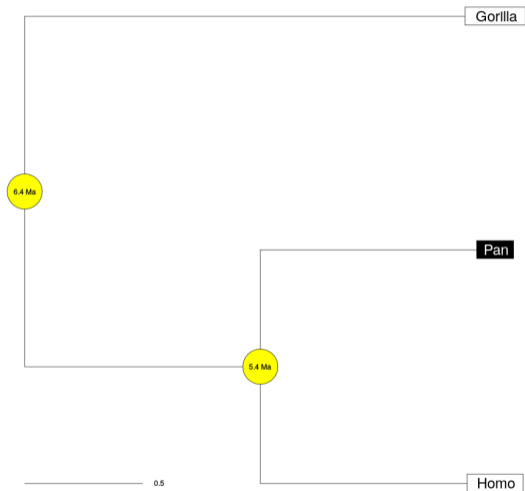
# Change orientation of plots

- `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/node labels
```

# Trees tasks

## Tasks

- 1 Import/export some tree(s), preferably your data you are working with. Try various plotting options.
- 2 Try various displays of some tree(s) from phytools dataset (see `data(package="phytools")` ).
- 3 Use some method to analyze Apicomplexa trees (see `?kdetrees::apicomplexa` ). Find discordant trees and try to construct species tree from the set of gene trees.

# Reconstruction of evolution of traits

## 11 Evolution

PIC

Autocorrelation

Decomposition

PGLS

GEE

Phylosignal

pPCA

Ancestral state

Phenogram

Tasks

# 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
- Methods and models are highly debated in the literature
  - Different experts commonly disagree what is the best method...

# Overview of methods of reconstruction of evolution of traits II

- 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 data(shorebird, package="caper") # Load training data
2 ?caper::shorebird # See it
3 head(shorebird.data) # See the data part
4 shorebird.tree # See the phylogeny
5 plot.phylo(shorebird.tree)
6 # The tree must be fully bifurcating for most of methods
7 shorebird.tree <- multi2di(phy=shorebird.tree)
8 plot.phylo(shorebird.tree) # See result

```

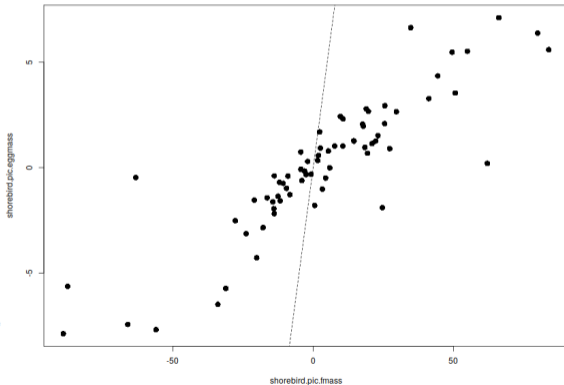
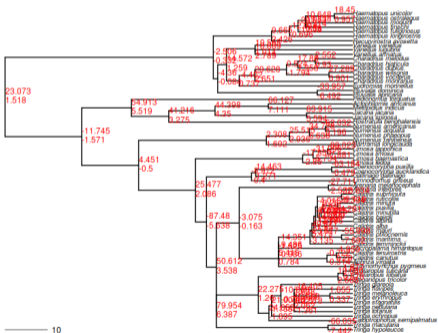
# PIC and its plotting

```
1 shorebird.pic.fmass <- pic(x=shorebird.data[["F.Mass"]], phy=
2   shorebird.tree, scaled=TRUE, var.contrasts=FALSE, rescaled.tree=FALSE)
3 shorebird.pic.eggmass <- pic(x=shorebird.data[["Egg.Mass"]], phy=
4   shorebird.tree, scaled=TRUE, var.contrasts=FALSE, rescaled.tree=FALSE)
5 # Plot a tree with PIC values
6 plot.phylo(x=shorebird.tree, edge.width=2)
7 nodelabels(round(shorebird.pic.fmass, digits=3), adj=c(0, -0.5),
8   frame="none", col="red")
9 nodelabels(round(shorebird.pic.eggmass, digits=3), adj=c(0, 1),
10  frame="none", col="red")
11 add.scale.bar()
12 # Plot PIC
13 plot(x=shorebird.pic.fmass, y=shorebird.pic.eggmass, pch=16, cex=1.5)
14 abline(a=0, b=1, lty=2) # x=y line
15 # Correlation of PIC of body mass and longevity
16 cor(x=shorebird.pic.fmass, y=shorebird.pic.eggmass, method="pearson")
17 [1] 0.879019
```

# Plot of PIC (on the tree) and test the correlation

```

1 # Correlation test
2 cor.test(x=shorebird.pic.fmass, y=shorebird.pic.eggmass,
3 alternative="greater", method="pearson")
    
```



# Test it

```

1 # Linear model of both PICs
2 lm(formula=shorebird.pic.fmass~shorebird.pic.eggmass)
3 Coefficients:
4     (Intercept)  shorebird.pic.eggmass
5         2.636             9.110
6 # Because PICs have expected mean zero - such linear regressions should
7 # be done through the origin (i.e. the intercept is set to zero)
8 lm(formula=shorebird.pic.fmass~shorebird.pic.eggmass-1)
9 Coefficients:
10     (Intercept)  shorebird.pic.eggmass
11         2.636             9.110
12 # Permutation procedure to test PIC
13 lmorigin(formula=shorebird.pic.fmass~shorebird.pic.eggmass, nperm=1000)
14 Coefficients and parametric test results
15             Coefficient Std_error t-value Pr(>|t|)
16 shorebird.pic.eggmass      9.14518   0.60274  15.173 < 2.2e-16 ***
    
```

# Intraspecific variation

- `pic.ortho()` requires list of measurements (numeric vectors) for all taxa – their lengths can differ
- We must for each `list` item provide numeric vector (construct by something like `values <- list(taxon1, taxon2, taxon3, ...)`), otherwise usage is same as with `pic()` – plotting, tests, etc.
- Transforming data into required form can require considerable work...

```

1 # See help
2 ?pic.ortho

```

# 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(shorebird.tree) or some other transformation
4 shorebird.weights <- 1/cophenetic(shorebird.tree)
5 # See it
6 class(shorebird.weights)
7 head(shorebird.weights)
8 # Set diagonal to 0
9 diag(shorebird.weights) <- 0
```

# Testing of Moran's I

```

1 # Calculate Moran's I
2 # Significant, but super small, positive phylogenetic correlation
3 Moran.I(x=shorebird.data[["M.Mass"]], weight=shorebird.weights,
4         alternative="greater")
5 Moran.I(x=shorebird.data[["F.Mass"]], weight=shorebird.weights,
6         alternative="greater")
7 # Test of Moran's with randomisation procedure
8 library(ade4)
9 ?gearymoran
10 # For all characters significant, but very small
11 gearymoran(bilis=shorebird.weights, X=shorebird.data[,2:5], nrepet=1000)
12 # Test of Abouheif designed to detect phylogenetic autocorrelation
13 # in a quantitative trait - in fact Moran's I test using a particular
14 # phylogenetic proximity between tips
15 library(ade4)
16 abouheif.moran(x=shorebird.data[,2:5], W=shorebird.weights,
17               method="oriAbouheif", nrepet=1000, alter="greater")
    
```

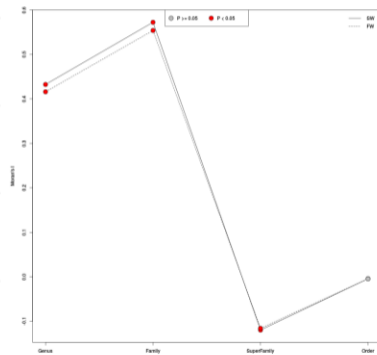
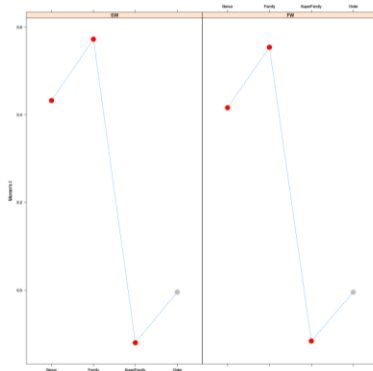
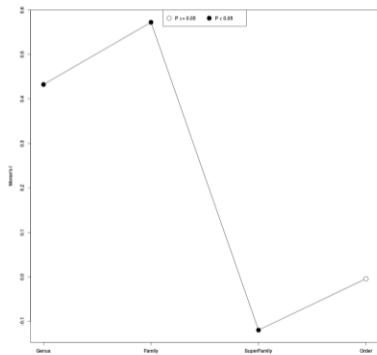
# Correlogram to visualize results of phylogenetic autocorrelation analysis

```

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

```
1 # Generate some random variable
2 library(geiger)
3 shorebird.eco <- sim.char(phy=shorebird.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(shorebird.eco) <- shorebird.tree[["tip.label"]]
8 shorebird.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(shorebird.eco ~ as.matrix(orthobasis.phylo(x=shorebird.tree,
2   method="patrinsic")[,1:2])))
3 Analysis of Variance Table
4 Response: shorebird.eco
5           Df Sum Sq Mean Sq F value    Pr(>F)
6 as.matrix...  2  226.61  113.306   54.045 8.918e-15 ***
7 Residuals    68  142.56    2.097
8 anova(lm(shorebird.data[["M.Mass"]] ~ as.matrix(orthobasis.phylo(
9   x=shorebird.tree, method="patrinsic")[,1:2])))
    
```

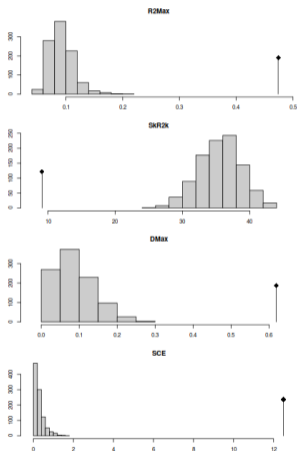
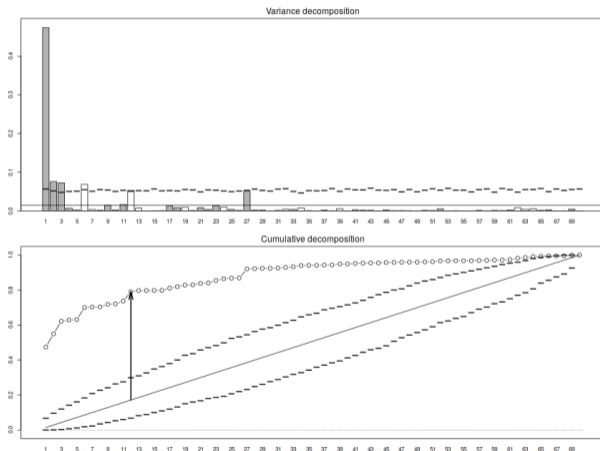
- Significant result — significant phylogenetic inertia (phylogenetic effect) — the tendency for traits to resist evolutionary change despite environmental perturbations
- `orthobasis.phylo()` return matrix, which is linear transformation of cophenetic distances — columns 1 and 2 can be used to calculate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

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

```
1 orthogram(x=shorebird.eco, tre=shorebird.tree, nrepet=1000,  
2   alter="two-sided")  
3 ?orthogram # See another calculation possibilities  
4 orthogram(x=shorebird.data[["M.Mass"]], tre=shorebird.tree, nrepet=1000,  
5   alter="two-sided")
```

- 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 out of permutations — 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

```

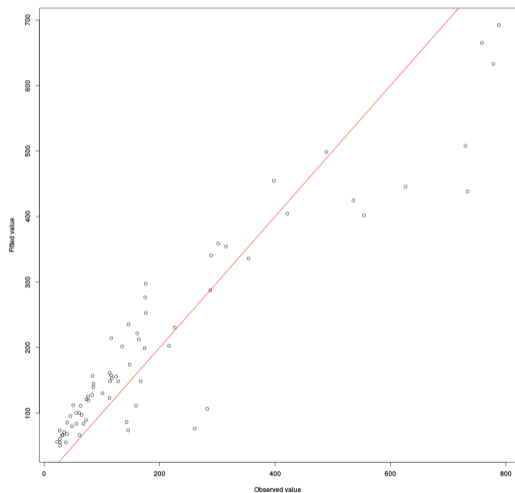
1 library(nlme)
2 summary(gls(model=F.Mass ~ Egg.Mass, data=shorebird.data,
3   correlation=corBrownian(value=1, phy=shorebird.tree)))
4   Generalized least squares fit by REML
5           AIC      BIC    logLik
6   798.813 805.5154 -396.4065
7           Value Std.Error   t-value p-value
8 (Intercept) -2.384233   73.54508  -0.032419  0.9742
9 Egg.Mass     9.145185    0.60274  15.172606  0.0000
  
```

# 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=shorebird.data[["F.Mass"]] ~
3   shorebird.data[["Egg.Mass"]], phy=shorebird.tree)
4 # or with correlation matrix
5 compar.gee(formula=shorebird.data[["F.Mass"]] ~
6   shorebird.data[["Egg.Mass"]], corStruct=corMartins(value=1,
7   phy=shorebird.tree, fixed=TRUE))
8 # for corStruct there are similar functions corBlomberg, corMartins,
9 # corPagel, corBrownian - see manuals for differences
```

# Not significant in this case...

```

1 Call: compar.gee(formula = shorebird.data[["F.Mass"]] ~
2   shorebird.data[["Egg.Mass"]], phy = shorebird.tree)
3 Number of observations: 71
4 Model:
5             Link: identity
6 Variance to Mean Relation: gaussian
7 QIC: 449593.7
8 Summary of Residuals:
9      Min      1Q   Median      3Q      Max
10 -121.577831 -48.195170 -32.598282  -2.168055  295.586322
11 Coefficients:
12             Estimate      S.E.      t  Pr(T > |t|)
13 (Intercept) -2.384233  41.1202553 -0.05798195  9.545638e-01
14 shorebird.data[["Egg.Mass"]]  9.145185  0.3370035  27.13676219  1.003642e-13
15 Estimated Scale Parameter: 6515.203
16 "Phylogenetic" df (dfP): 16.3298
    
```

# 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 # It requires named vector of trait values
3 shorebird.mmass <- shorebird.data[["M.Mass"]]
4 names(shorebird.mmass) <- rownames(shorebird.data)
5 # Bloomberg's K statistics
6 Kcalc(x=shorebird.mmass, phy=shorebird.tree, checkdata=TRUE)
7 # Test with permutations
8 phylosignal(x=shorebird.mmass, phy=shorebird.tree, reps=1000, checkdata=T)
```

- 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 # multiPhylosignal requires data frame of numerical values
2 multiPhylosignal(x=shorebird.data[,2:5], phy=shorebird.tree, reps=1000)
3           K PIC.variance.obs PIC.variance.rnd.mean PIC.variance.P...
4 M.Mass    1.089419      5.139489e+02      3.278559e+03      0.000999001...
5 P.Mass    1.092474      6.316701e+02      3.993696e+03      0.000999001...
6 Egg.Mass  1.139078      4.160653e+00      2.783029e+01      0.000999001...
7 Cl.size   0.673523      8.397776e-03      3.323744e-02      0.000999001...
8 # See help
9 ?phylosignal
10 ?multiPhylosignal
    
```

# 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=shorebird.tree, x=shorebird.mmass, method="K", test=TRUE,
5         nsim=1000)
6 plot(phylosig(tree=shorebird.tree, x=shorebird.mmass, method="K",
7             test=TRUE, nsim=1000))
8 phylosig(tree=shorebird.tree, x=shorebird.mmass, method="lambda",
9         test=TRUE)
    
```

# Alternative testing for phylogenetic signal with GLM

- It is possible to use intercept-only ( `model/formula` will be something like `variable ~ 1` instead of `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=shorebird.mmass ~ 1,
3   correlation=corBrownian(value=1, phy=shorebird.tree)))
4 summary(pgls(formula=shorebird.mmass ~ 1,
5   data=comparative.data(phy=shorebird.tree,
6   data=as.data.frame(cbind(shorebird.data[["M.Mass"]],
7   shorebird.data[["Species"]]))), names.col=V2, vcv=TRUE)))
```

# Phylogenetic principal component analysis

## PCA corrected for phylogeny

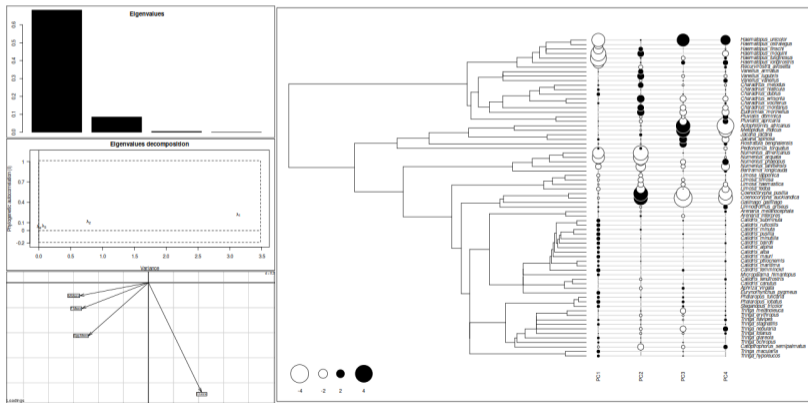
- It requires as input phylogenetic tree and respective comparative data
- Phylogenetic component is removed from the data, then classical PCA is calculated
- Together with nodes (taxa), PCA scores for PC axes are plotted — not the taxa — it shows trends of character evolution on the tree, not positions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



# Phylogenetic principal component analysis – the code

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

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



# Training data I

```
1 # Load library
2 library(ape)
3 # Loading data
4 # Ackerly & Donoghue (1998) https://doi.org/10.1086/286208
5 data(maples, package="adephylo")
6 ?adephylo::maples
7 # Process the phylogenetic tree
8 # maples data provide tree as plain text in NEWICK, must be imported
9 # into the phylo object
10 maples.tree <- read.tree(text=maples[["tre"]])
11 maples.tree
12 plot.phylo(maples.tree)
13 # For plenty of analysis it must be fully resolved (bifurcating),
14 # rooting and ultrametricity often helps
15 is.binary.phylo(maples.tree)
16 is.rooted.phylo(maples.tree)
17 is.ultrametric.phylo(maples.tree)
```

# Training data II

```

1 # See the character matrix
2 head(maples[["tab"]])
3 maples.data <- maples[["tab"]][, 1:30]
4 head(maples.data)
5 summary(maples.data)
6 # Maples mature height (m)
7 maples.height <- maples[["tab"]][["MatHt"]]
8 names(maples.height) <- rownames(maples[["tab"]])
9 maples.height
10 # Maples seed size (mg)
11 maples.sdsz <- maples[["tab"]][["SdSz"]]
12 names(maples.sdsz) <- rownames(maples[["tab"]])
13 maples.sdsz
14 # Maples leaf + petiole length (mm)
15 maples.lfpt <- maples[["tab"]][["LfPt"]]
16 names(maples.lfpt) <- rownames(maples[["tab"]])
17 maples.lfpt

```

# Ancestral state reconstruction

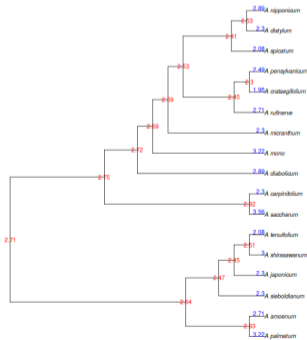
- By default `ape::ace()` performs estimation for continuous characters assuming a Brownian motion model fit by maximum likelihood
- `ace()` can handle continuous as well as discrete data

```
1 library(ape)
2 ?ace # See for possible settings
3 maples.height.ace <- ace(x=maples.height, phy=maples.tree,
4   type="continuous", method="REML", corStruct=corBrownian(value=1,
5   phy=maples.tree))
6 # See result - reconstructions are in ace slot - to be plotted on
7 # nodes - 1st column are node numbers
8 maples.height.ace
9 # Plot it
10 plot.phylo(x=maples.tree, lwd=2, cex=0.75)
11 tiplabels(maples.height, adj=c(1, 0), frame="none", col="blue", cex=0.75)
12 nodelabels(round(maples.height.ace[["ace"]], digits=1), frame="none",
13   col="red", cex=0.75)
```

# Ancestral state reconstructions of primates body weights

- ACE returns long numbers – truncate them by e.g.

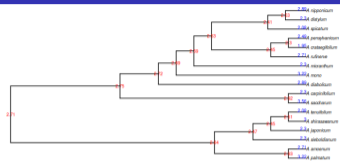
`round(x=..., digits=3)` (`x` is vector with ACE values)



```

1 # Other implementations are
2 # available in packages geiger,
3 # phangorn, ape, phytools, ...
4 # Parsimony based method
5 ?ape::MPR
6 # For continuous characters using
7 # Maximum Likelihood
8 ?geiger::fitContinuous
9 # For continuous characters using
10 # Markov Chain Monte Carlo
11 ?geiger::fitContinuousMCMC
12 # For discrete characters, various
13 # models available
14 ?geiger::fitDiscrete
15 # Marginal reconstruction of the
16 # ancestral character states
17 ?phangorn::ancestral.pml
    
```

# Implementation in phytools



```

1 # ML estimation of continuous trait
2 ?fastAnc
3 maples.height.fa <- fastAnc(tree=
4   maples.tree, x=maples.height,
5   vars=FALSE, CI=TRUE)
6 maples.height.fa
7 plot.phylo(x=maples.tree,
8   edge.width=2, cex=0.75)
9 tiplabels(maples.height,
10  adj=c(1, 0), frame="none",
11  col="blue", cex=0.75)
    
```

```

1 nodelabels(round(
2   maples.height.fa[["ace"]],
3   digits=2), frame="none",
4   col="red", cex=0.75)
5 # ACE for Brownian evolution
6 # with directional trend
7 ?anc.trend
8 maples.height.ac <- anc.trend(
9   tree=maples.tree,
10  x=maples.height, maxit=100000)
11 maples.height.ac
12 # ACE for Brownian evolution
13 # using likelihood
14 ?anc.ML
15 maples.height.ml <- anc.ML(tree=
16   maples.tree, x=maples.height,
17   maxit=100000, model="BM")
    
```

# Bayesian ancestral character estimation I

```
1 ?anc.Bayes
2 maples.height.bayes <- anc.Bayes(tree=maples.tree, x=maples.height,
3   ngen=1000000) # Use more MCMC generations
4 maples.height.bayes
5 # Get end of ancestral states from Bayesian posterior distribution
6 # (it should converge to certain values)
7 tail(maples.height.bayes[["mcmc"]])
8 maples.height.bayes[["mcmc"]][10001,3:18]
9 # Get means of ancestral states from Bayesian posterior distribution
10 colMeans(maples.height.bayes[["mcmc"]][2001:nrow(
11   maples.height.bayes[["mcmc"]]),as.character(1:
12   maples.tree$Nnode+length(maples.tree$tip.label))])
13 # Plot the ancestral states from posterior distribution
14 # (it should converge to certain values)
15 plot(maples.height.bayes)
16 # Plot the tree and reconstructed ancestral states
17 plot.phylo(x=maples.tree, edge.width=2, cex=2)
```



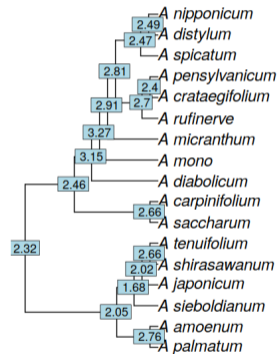
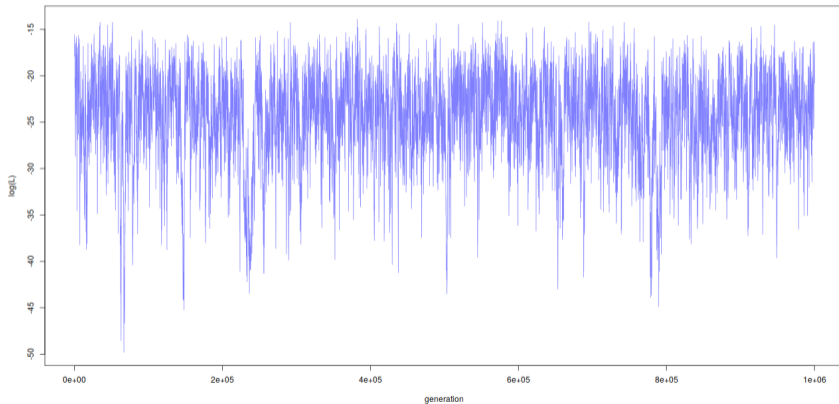
# Bayesian ancestral character estimation II

```

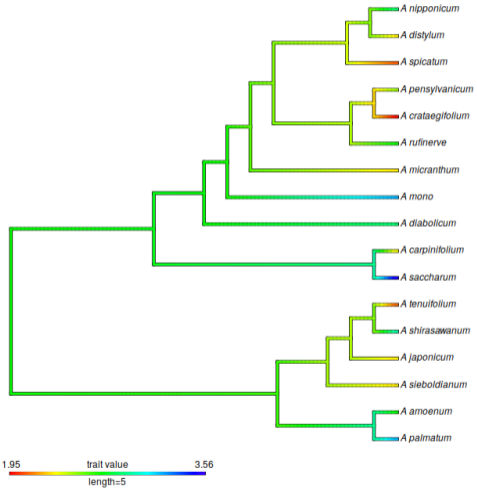
1 maples.height.bayes # Mean ancestral states from posterior distribution
2     18     19     20     21     22     23     24     25...
3 2.659527 2.662223 2.932293 2.482104 2.456929 2.510414 2.692636 2.919908...
4 tail(maples.height.bayes[["mcmc"]]) # See end of the table
5     gen     sig2     18     19     20 ...     33     logLik
6 ^I^^I...     ...     ...     ...     ...     ...     ...
7 10000  999900 0.1860167 2.284352 2.680593 2.962142 ... 2.494260 -19.00596
8 10001 1000000 0.1834573 2.315086 2.048144 2.755104 ... 2.485949 -20.80713
9 maples.height.bayes[["mcmc"]][10001,3:18] # See end of the table - values
10     18     19     20     21     22     23     24 ...
11 10001 2.315086 2.048144 2.755104 1.680306 2.019788 2.660322 2.456076 ...
12 # Get means of ancestral states from Bayesian posterior distribution
13 colMeans(maples.height.bayes[["mcmc"]][2001:nrow(
14   maples.height.bayes[["mcmc"]]),as.character(1:
15   maples.tree$Nnode+length(maples.tree$tip.label))])
16     18     19     20     21     22     23     24     25 ...
17 2.659527 2.662223 2.932293 2.482104 2.456929 2.510414 2.692636 2.919908 ...
    
```

# Bayesian ancestral character estimation III

Likelihood of Bayesian posterior probability and the tree with reconstructions

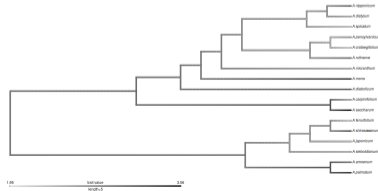


# Continuous map



```

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

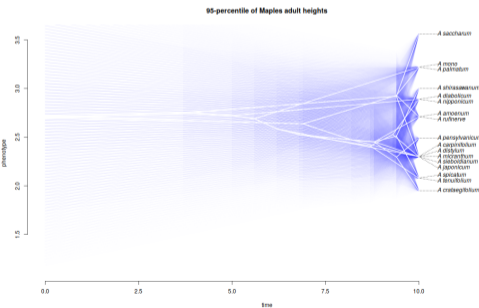
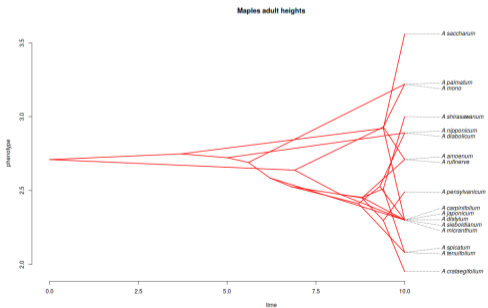




# Phenogram

Vertical axis shows character values

```
1 phenogram(tree=maples.tree, x=maples.height, ftype="i", colors="red",
2   main="Maples adult heights")
3 fancyTree(tree=maples.tree, type="phenogram95", x=maples.height,
4   ftype="i", main="95-percentile of Maples adult heights")
```

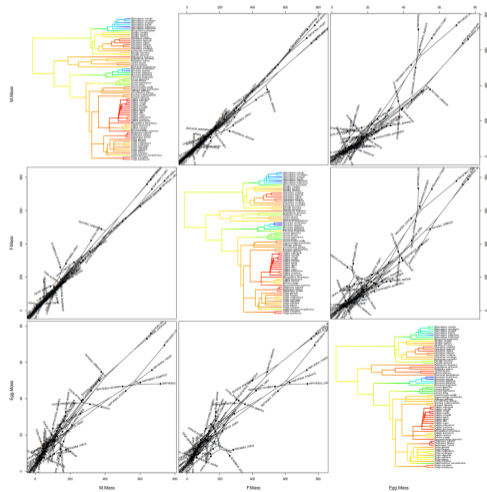




# Combine phenograms and ancestral state reconstructions

```

1 # 3 characters on 2 axis and
2 # ancestral state reconstruction
3 # for all of them
4 fancyTree(tree=shorebird.tree,
5           type="scattergram",
6           X=shorebird.data[,2:4],
7           res=500, ftype="i")
8 # See manuals for more settings
9 ?fancyTree
10 ?phenogram
11 ?phylomorphospace
12 ?phylomorphospace3d
13 ?contMap
14 ?setMap
15 ?par
16 ?plot
    
```

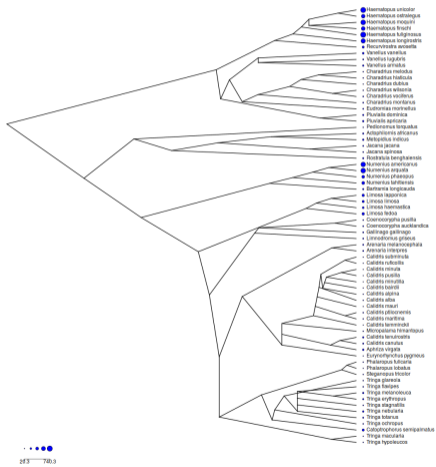
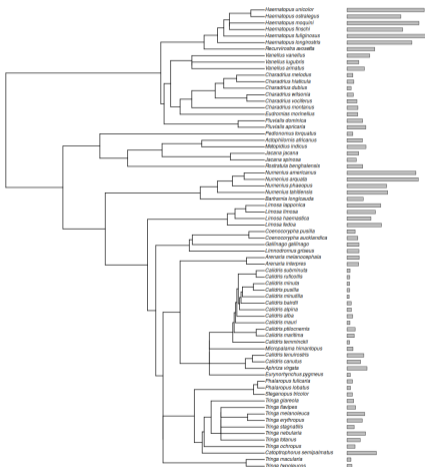


# Plotting traits on trees — code

```
1 # See options for plotting functions
2 ?plotTree.wBars # There are more variants available
3 ?dotTree
4 ?plotSimmap
5 # Plot the trees
6 # Tip labels with bars with length proportional to character values
7 plotTree.wBars(tree=shorebird.tree, x=shorebird.mmass, tip.labels=TRUE)
8 # Tip labels with dots with size proportional to character values
9 dotTree(tree=shorebird.tree, x=shorebird.mmass, tip.labels=TRUE,
10         type="cladogram")
```



# Plotting traits on trees — plots



253 740.0  
 ● ● ● ●

# Evolutionary tasks

## Tasks

- 1 Browse <http://blog.phytools.org/> and see if you find some interesting method to display your data. If so, try it with any suitable data.
- 2 See relevant training datasets  

```
data(package=c("adephylo", "caper", "geiger", "phytools"))
```

and select some training data set (or use your some own data) to try at least 1–2 of the above methods.

# Process more data

Not all combinations and possibilities were shown...

## Tasks

- 1 Try to do some analysis with another introduced toy data
  - 2 Try some of the introduced analysis with your own custom imported data
  - 3 Try at least 2–3 analysis preferably with your own data according to your interest
- 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 \(web\)](#)
  - 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 require(devtools)
3 dev_mode(on=TRUE)
4 # Install selected package from GitHub (user/project)
5 install_github("thibautjombart/adegenet")
6 # when finished go back to normal version
7 dev_mode(on=FALSE)

```

# R script and its running from command line I

- R script is just plain `TXT` file with `.r` (e.g. `myscript.r`) extension containing list of R commands
- Mark all user comments with `#` on the beginning
- In command line (Linux/macOS/Windows/...) use
  - `Rscript myscript.r` to work **interactively** – all output is written to the terminal (screen; 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 file `myscript.Rout`, terminal (screen) is clean and user can not influence the script anyhow – e.g. on [MetaCentrum](#) – be sure the script doesn't require user input and works correctly
- Script ends when there is any error or on the end of the file
- When working on both Windows and macOS/Linux, take care about end of lines, and in case of usage of accented characters (e.g. for labels) also about encoding



## R script and its running from command line II

- Windows and UNIX (Linux, macOS, ...) 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
- Linux and macOS use to use UTF-8, Windows use regional encoding, e.g. Czech CP-1250 — use advanced text editor (slide 11) to convert the encoding, or use some command line tool, like `iconv`

# 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 how it is increasing
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 for (i in 10:5) { print(i) } # Can be descending...
10 # Work on each item of a list object
11 # Print length of each sequence in gunnera.dna
12 for (L in 1:length(gunnera.dna)) {
13   print(length(gunnera.dna[[L]])) }
14 # While loop - it is done while the condition is valid
15 # While value of "Q" is < 5 (starting from 0), print it and add 1
16 Q <- 0
17 while (Q < 5) { print(Q <- Q+1) }
```

# If-else branching I

- Basic method of branching the code — **if** the condition is met, **then** 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
```

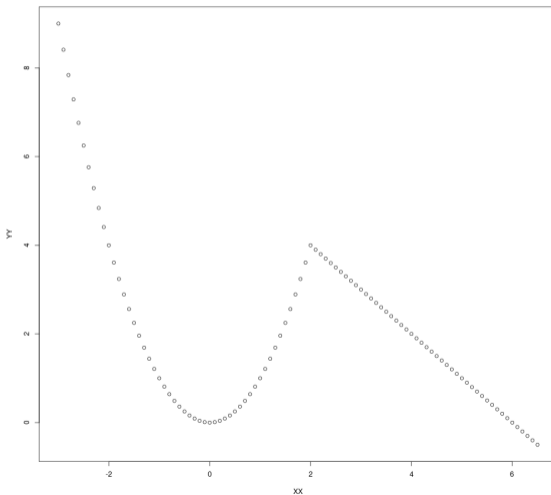
# If-else branching II

```

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 85)
  - 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 211) – 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 388 and onward)
- R and packages are more or less changing from version to version
  - Old methods can became outdated and not working anymore
  - Check release notes and change logs for new versions, manual pages and on-line forums (slide 388 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")`
- **R is free as freedom of speech — not as free beer!**

## How to ask for help II

- 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 386.

- 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/>
- Adegenet help mailing list <https://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/g/poppr/about>
- R help mailing list <https://stat.ethz.ch/mailman/listinfo/r-help> (web interface <https://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 statisticians, 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](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

 Anthony R. Ives  
Mixed and Phylogenetic Models: A  
Conceptual Introduction to Correlated  
Data  
Leanpub, 2018  
[https://leanpub.com/  
correlateddata](https://leanpub.com/correlateddata) (free to read on-line)



## 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 phylogenetics 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>
- Integrated documentation search <https://www.rdocumentation.org/>
- Better interface to R and packages documentation and integrated search <https://rdr.io/>

## Learning resources II

- 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 <https://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 <https://grunwaldlab.github.io/poppr/reference/poppr-package.html>

## Learning resources IV

- Population Genetics in R <https://popgen.nescent.org/> by Kamvar et al
- ade4 home page <https://pbil.univ-lyon1.fr/ADE-4/ade4-html/00Index.html?lang=eng>  
and documentation  
<https://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/>
- R Programming [https://en.wikibooks.org/wiki/R\\_Programming](https://en.wikibooks.org/wiki/R_Programming)
- RStudio Cheat Sheets <https://rstudio.com/resources/cheatsheets/>  
and Online learning resources <https://education.rstudio.com/learn/>
- R-bloggers — aggregation of R blogs <https://www.r-bloggers.com/>

## Learning resources V

- R on The Molecular Ecologist  
<https://www.molularecologist.com/category/software/r/>
- R tutorial <https://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 <https://ggplot2.tidyverse.org/>
- plyr documentation <https://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/>
- Quick-R learning resource <https://www.statmethods.net/>
- Visualizing and annotating phylogenetic trees with ggtree <https://4va.github.io/biodatasci/r-ggtree.html>
- R manual and help search <http://finzi.psych.upenn.edu/>
- **Uncle Google** is your friend (*“how to XXX in R”*)...

## Learning resources VII

- 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

- [ade4](#): multivariate data analysis and graphical display (enhancements: [ade4TkGUI](#) — GUI, [adegraphics](#) — extra graphical functions, commonly used internally)
- [adegenet](#): exploration of genetic and genomic data
- [adephylo](#): multivariate tools to analyze comparative data
- [akima](#): cubic spline interpolation methods for irregular and regular grids data
- [ape](#): analysis of phylogenetics and evolution
- [BiocManager](#): access the Bioconductor project package repository
- [caper](#): phylogenetic comparative analysis
- [corrplot](#): graphical display of a correlation or general matrix
- [devtools](#): package development tools, access to GitHub
- [gee](#): generalized estimation equation solver



## Packages we used... II

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

- [geiger](#): fitting macroevolutionary models to phylogenetic trees
- [Geneland](#): stochastic simulation and MCMC inference of structure from genetic data
- [ggplot2](#): data visualizations using the Grammar of Graphics
- [gplots](#): plotting data
- [hierfstat](#): estimation and tests of hierarchical F-statistics
- [ips](#): interfaces to phylogenetic software
- [kdetrees](#): non-parametric method for identifying potential outlying observations in a collection of phylogenetic trees
- [lattice](#): Trellis graphics, with an emphasis on multivariate data
- [mapdata](#): supplement to maps, larger and/or higher-resolution databases
- [mapplots](#): extra map plotting, pie charts and more

## Packages we used... III

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

- **mapproj**: converts latitude/longitude into projected coordinates
- **maps**: draws geographical maps
- **maptools**: manipulating and reading geographic data
- **muscle**: multiple sequence alignment with MUSCLE
- **nlme**: fits and compares Gaussian linear and nonlinear mixed-effects models
- **ParallelStructure**: running analysis in the population genetics software STRUCTURE
- **PBSmapping**: spatial analysis tools
- **pegas**: population and evolutionary genetics analysis
- **phangorn**: phylogenetic analysis
- **philentropy**: over 40 optimized distance and similarity measures for comparing probability functions

## Packages we used... IV

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

- [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
- [raster](#): reading, writing, manipulating, analyzing and modeling of gridded spatial data
- [rentrez](#): interface to the NCBI, allowing to search databases like GenBank
- [rgdal](#): bindings to the Geospatial Data Abstraction Library and access to projection/transformation operations library

## 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
- [Rmpi](#): interface (wrapper) to MPI (used for parallel processing)
- [rworldmap](#): mapping global data (and extra data in [rworldextra](#))
- [seqinr](#): exploratory data analysis and data visualization for biological sequence
- [shapefiles](#): read and write ESRI shapefiles
- [snow](#): simple parallel computing
- [sos](#): searches contributed R packages
- [sp](#): classes and methods for spatial data
- [spdep](#): spatial dependence: weighting schemes, statistics and models
- [splancs](#): display and analysis of spatial point pattern data

## Packages we used... VI

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

- **StAMPP**: statistical analysis of mixed ploidy populations
- **TeachingDemos**: demonstrations for teaching and learning, enhanced plotting of text
- **tripack**: constrained two-dimensional Delaunay triangulation
- **vcfR**: import/export, basic checking and manipulations of VCF
- **vegan**: community ecology

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

For your own explorations...

- **adespatial**: multiscale spatial analysis of multivariate data
- **adhoc**: ad hoc distance thresholds for DNA barcoding identification
- **addTaxa**: adding missing taxa to phylogenies
- **admixr**: interface for running **ADMIXTOOLS**
- **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

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

For your own explorations...

- **Biostrings**: string matching algorithms, and other utilities, for fast manipulation of large biological sequences or sets of sequences
- **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
- **dbscan**: implementation of several density-based algorithms (**DBSCAN**, **OPTICS**, etc.)
- **DDD**: diversity-dependent diversification
- **dendextend**: extending dendrogram objects, comparing trees
- **distory**: geodesic distance between phylogenetic trees
- **diversitree**: comparative phylogenetic analysis 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
- [dplyr](#): various manipulations with data frames
- [ecodist](#): dissimilarity-based functions for ecological analysis, spatial and community data
- [evobiR](#): comparative and population genetic analysis
- [expands](#): expanding ploidy and allele-frequency on nested subpopulations
- [fields](#): tools for spatial data
- [genetics](#): population genetics
- [genotypeR](#): design of genotyping markers from VCF files, output of markers for multiplexing on various platforms, various QA/QC and analysis
- [geomorph](#): geometric morphometric analysis of 2D/3D landmark data



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

For your own explorations...

- [geosphere](#): spherical trigonometry for geographic applications — distances and related measures for angular (longitude/latitude) locations
- [ggtree](#): visualization and annotation of phylogenetic trees ([documentation](#))
- [HardyWeinberg](#): statistical tests and graphics for HWE
- [heatmap.plus](#): advanced options to plot heatmaps
- [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
- [HyPhy](#): macroevolutionary phylogenetic analysis of species trees and gene trees
- [IRanges](#): infrastructure for manipulating intervals on sequences
- [iteRates](#): iterates through a phylogenetic tree to identify regions of rate variation

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

For your own explorations...

- [jaatha](#): simulation-based maximum likelihood parameter estimation
- [knitr](#): general-purpose tool for dynamic report generation
- [LDheatmap](#): graphical display, as a heat map, of measures of pairwise linkage disequilibrium between SNPs
- [LEA](#): landscape and ecological association studies
- [leaflet](#): interactive web maps with the JavaScript Leaflet library
- [Linarius](#): dominant marker analysis with mixed ploidy levels
- [markophylo](#): markov chain models for phylogenetic trees
- [MASS](#): functions and data sets for venables and ripley's MASS
- [MCMCglmm](#): MCMC generalized linear mixed models
- [MINOTAUR](#): multivariate visualization and outlier analysis

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

For your own explorations...

- [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
- [mvtnorm](#): multivariate normal and t probabilities
- [onemap](#): molecular marker data from model (backcrosses, F2 and recombinant inbred lines) and non-model systems (outcrossing species), constructions of genetic maps
- [OpenStreetMap](#): plotting OpenStreetMap maps (various layers)
- [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 analysis 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
- **permute**: restricted permutation designs

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

For your own explorations...

- **Phybase**: read, write, manipulate, simulate, estimate, and summarize phylogenetic trees (gene trees and species trees)
- **phyclust**: phylogenetic clustering
- **phyloclim**: integrating phylogenetics and climatic niche modeling
- **PHYLOGR**: manipulation and analysis of phylogenetically simulated data sets and phylogenetically based analysis 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
- **plotly**: creates interactive web graphics

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

For your own explorations...

- [plyr](#): splitting, applying and combining data
- [pmc](#): phylogenetic Monte Carlo
- [polyfreqs](#): Gibbs sampling algorithm to perform Bayesian inference on biallelic SNP frequencies, genotypes and heterozygosity in a population of autopolyploids
- [polysat](#): polyploid microsatellite analysis
- [RandomFields](#): simulation of Gaussian fields (+ [RandomFieldsUtils](#))
- [radiator](#): RADseq data exploration, manipulation and visualization
- [rCharts](#): interactive JS charts
- [RColorBrewer](#): ColorBrewer palettes
- [rdryad](#): access for Dryad web services
- [reshape2](#): restructure and aggregate data

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

For your own explorations...

- **rMaps**: interactive maps
- **rphast**: interface to PHAST software for comparative genomics
- **RMesquite**: interoperability with Mesquite
- **Rphylip**: interface for PHYLIP
- **Rsamtools**: BAM, FASTA, BCF and tabix file import and manipulations
- **rwty**: tests, visualizations, and metrics for diagnosing convergence of MCMC chains in phylogenetics
- **sensiPhy**: sensitivity analysis for phylogenetic comparative methods, statistical and graphical methods that estimate and report different types of uncertainty
- **seqLogo**: sequence logos for DNA sequence alignments
- **SigTree**: identify and visualize significantly responsive branches in a phylogenetic tree

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

For your own explorations...

- **SimRAD**: simulate restriction enzyme digestion, library construction and fragments size selection to predict the number of loci expected from most of the RAD and GPS approaches
- **SNPRelate**: parallel computing toolset for relatedness and principal component analysis of SNP data
- **snpStats**: classes and statistical methods for large SNP association studies
- **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 by vast range of methods, very powerful
- **stringi** and **stringr**: character string processing, internally used by many packages



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

For your own explorations...

- [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
- [tmap](#): various thematic maps
- [treebase](#): discovery, access and manipulation of TreeBASE phylogeny
- [treeio](#): read, parse and write various tree formats
- [TreePar](#): estimating birth and death rates based on phylogeny
- [TreeSearch](#): search for phylogenetic trees that are optimal using a user-defined criterion
- [TreeSim](#): simulating phylogenetic trees

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

For your own explorations...

- [treespace](#): exploration of distributions of phylogenetic trees
- [UpSetR](#): visualizations of intersecting sets using a novel matrix design, along with visualizations of several common set, element and attribute related tasks
- [VariantAnnotation](#): annotation of genetic variants (useful to filter VCF)
- [XVector](#): representation and manipulation of external sequences

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>3</sub>L<sup>A</sup>T<sub>E</sub>X on openSUSE GNU/Linux, February 5, 2021.