Molecular data in R

Phylogeny, evolution & R

Vojtěch Zeisek

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

January 27 to 30, 2020

Outline I

Outline II

Outline III

Outline IV

Outline V

Outline VI

The course information

- The course page:
  https://trapa.cz/en/course-molecular-data-r-2020
  - Česky: https://trapa.cz/cs/kurz-molekularni-data-r-2020
- Subject in SIS: https://is.cuni.cz/studium/eng/predmety/index.php?do=predmet&kod=MB120C16
  - Česky: https://is.cuni.cz/studium/predmety/index.php?do=predmet&kod=MB120C16
  - For students having subscribed the subject, 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`
- Download the script from `https://soubory.trapa.cz/rcourse/course_commands.r`, use it and write your comments and notes to it during the course
  - Note: Open the R script in some good text editor (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
- Open multiple files
- Advanced search and replace
- Use regular expressions
- Make projects, add notes
- Use command line
- Check spelling
- Debug source code

- Kate
- KWrite
- Vim
- GNU Emacs
- Geany
- Bluefish
- 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 so...
  1. **Read** the command and all comments around, do not blindly launch it
  2. Ensure you understand, what is **aim** of the command (what it is supposed to do)
  3. Ensure you understand all **limitations** of the method (when you can use it and when not)
  4. Ensure you understand **syntax** of the command (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
- Learning R is effective only if you learn R syntax (language grammar), otherwise you only memorize commands without understanding them or blindly repeat someone's code — in such case, you wouldn't be able to solve any issue with your workflow

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

We will go through...

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

We will not dig deep into...

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

The R

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

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/

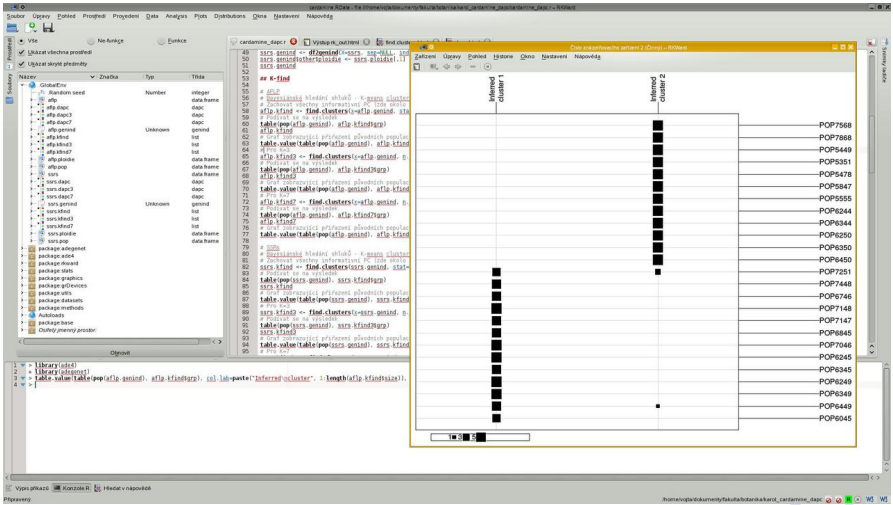
Graphical user interfaces (GUI) I

- 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

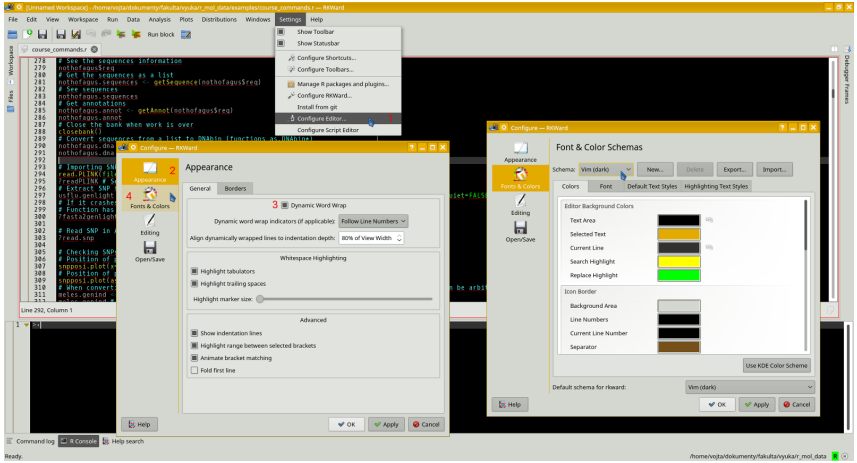
Graphical user interfaces (GUI) II

- 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
- Pick one you like (from above list or any else) and install it...

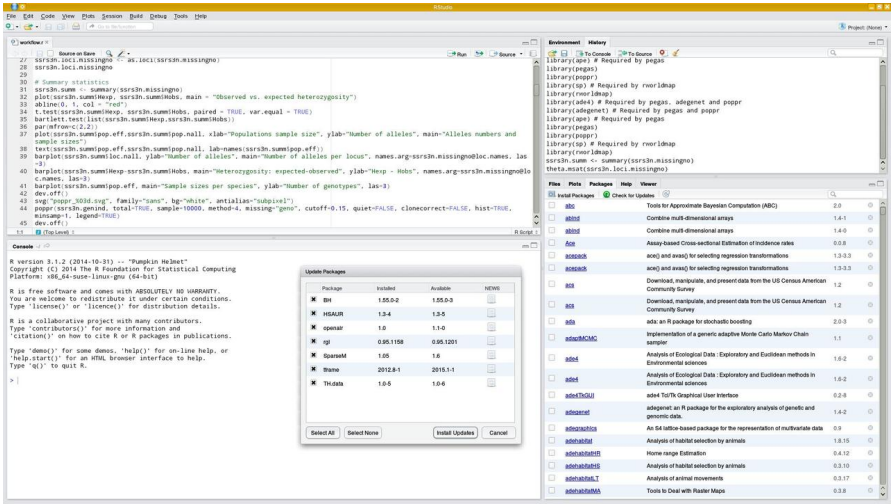
RKWard


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

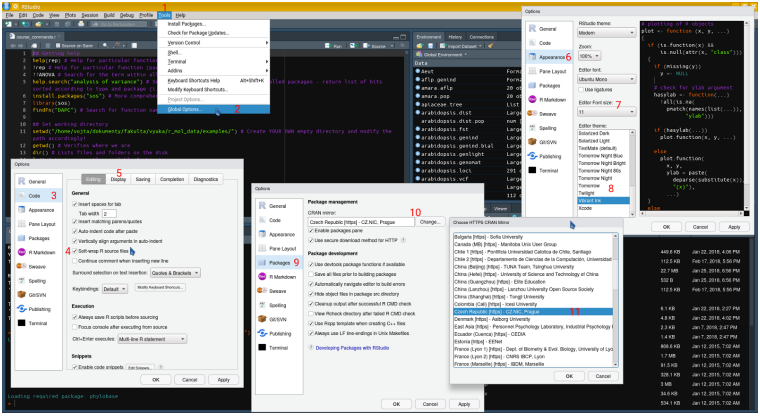
When using RKWard, consider change of settings of text editor for more comfortable work



RStudio


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Console
Update Packages
Package Installed Available NEWS
Select All
Select None
Install Updates
Cancel

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



MS Windows & Apple 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/README`
- Install packages `R-core` (the R), `R-core-devel` (required to compile additional R packages — only some are available in repositories) and optionally `rkward`
- RStudio is available from its download page

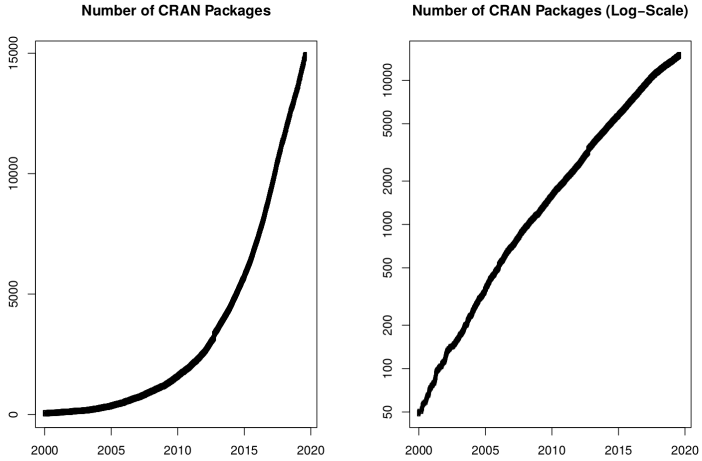
Important note about names of directories

- There must not be any spaces or accented characters in the path to R working directory or local R library, otherwise some R functions can fail (and there is no other solution)
- If the user has e.g. on Windows path like
  `C:/Documents and Settings/Šíleně úpějící kůoň/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 15,000 packages + many orphaned and archived — abandoned by developers, might be working)
- Bioconductor `https://bioconductor.org/` — mainly bioinformatics packages, genomic data (over 1,800 packages)
- R-Forge `https://r-forge.r-project.org/` (over 2,000 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...

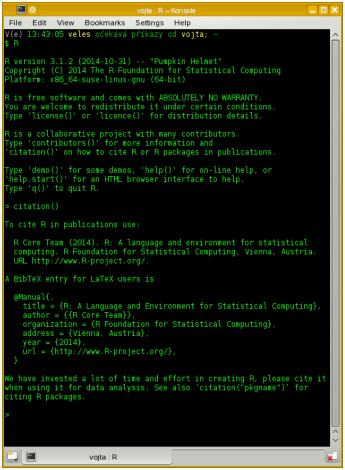


https://journal.r-project.org/archive/2019-1/cran.pdf

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



How it works

- General look of R commands:

```
function(argument1="SomeName", argument2=SomeVariable, argument3=8)
ModifiedObject <- SomeFunction(argument1=MyData, argument2=TRUE)
```

- New/modified object (with data, ...) is on the left: " `<-` " says to insert result of the function `SomeFunction` on the right into the object `ModifiedObject` on the left
- Functions have various parameters/arguments (in brackets, separated by comas): `argument=ItsValue`
- Arguments are named — if you keep order, no need to name them:

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

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

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

- Some arguments are required, some optional

Get help in R

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

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

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

Where we are?

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

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

- In Windows (**File | working directory**) or in RStudio (**Session | Set working directory**) set it in menu or by above command
- R saves history of commands into file `.Rhistory` file within working directory (by default hidden in Linux/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
- RStudio and RKWard help with this very much

Importance of working directory

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

Types of objects

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

Popular object classes (we are going to use) I

- `AAbin` —stores amino acid sequences (aligned or not)
- `alignment` — aligned sequences (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
- `phylo` — phylogenetic information, typically trees
- `phylo4` — derived from phylo (more data), S4 instead of S3
- `SNPbin` — stores large SNP data for single genome
- `spca` — results of sPCA

Popular object classes (we are going to use) III

- `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

Conversions among data types I

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

Conversions among data types II

From	To	Command	Package
treeshape	phylo	`as.phylo(x)`	apTreeshape
splits	phylo	`as.phylo(x)`	phangorn
splits	networx	`as.networx(x)`	phangorn
evonet	phylo	`as.phylo(x)`	ape
evonet	networx	`as.networx(x)`	ape
evonet	network	`as.network(x)`	ape
evonet	igraph	`as.igraph(x)`	ape
haploNet	network	`as.network(x)`	pegas
haploNet	igraph	`as.igraph(x)`	pegas
hclust	phylo	`as.phylo(x)`	ape
hclust	dendrogram	`as.dendrogram(x)`	stats

Conversions among data types III

From	To	Command	Package
DNAbin	character	`as.character(x)`	ape
DNAbin	alignment	`as.alignment(x)`	ape
DNAbin	phyDat	`as.phyDat(x)`	phangorn
DNAbin	genind	`DNAbin2genind(x)`	adegenet
character	DNAbin	`as.DNAbin(x)`	ape
character	loci	`as.loci(x)`	pegas
alignment	DNAbin	`as.DNAbin(x)`	ape
alignment	phyDat	`as.phyDat(x)`	phangorn
alignment	character	`as.matrix(x)`	seqinr
alignment	genind	`alignment2genind(x)`	adegenet

Conversions among data types IV

From	To	Command	Package
phyDat	DNAbin	`as.DNAbin(x)`	phangorn
phyDat	character	`as.character(x)`	phangorn
loci	genind	`loci2genind(x)`	pegas
loci	data frame	`class(x) <- "data.frame"`	—
genind	loci	`genind2loci(x)`	pegas
genind	genpop	`genind2genpop(x)`	adegenet
genind	df	`genind2df(x)`	adegenet
data frame	phyDat	`as.phyDat(x)`	phangorn
data frame	loci	`as.loci(x)`	pegas
data frame	genind	`df2genind(x)`	adegenet
matrix	phyDat	`as.phyDat(x)`	phangorn

Conversions among data types V

From	To	Command	Package
vcfR	chromR	`vcfR2chromR(x)`	vcfR
vcfR	genind	`vcfR2genind(x)`	vcfR
vcfR	migrate	`vcfR2migrate(x)`	vcfR
vcfR	loci	`vcfR2loci(x)`	vcfR
vcfR	tidy	`vcfR2tidy(x)`	vcfR
vcfR	DNAbin	`vcfR2DNAbin(x)`	vcfR
vcfR	genlight	`vcfR2genlight(x)`	vcfR

Basic operations with data I

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

```
x <- c(5, 6, 7, 8, 9) # Creates vector (see also ?rep)
x # Print "x" content
c() # Is generic function to concatenate objects into new one
length(x) # Length of the object - for matrices and DF use dim()
str(x) # Information about structure of the object
mode(x) # Gets type of storage mode of the object
class(x) # Shows class of the object
x[2] # Shows second element of the object
x <- x[-5] # Removes fifth element
y <- matrix(data=5:20, nrow=4, ncol=4) # Creates a matrix
is.matrix(y) # Is it matrix? Try is.<TAB><TAB>
# TAB key shows available functions and objects starting by typed text
y # Prints the matrix
y[,2] # Prints second column
y[3,] # Prints third row
```

Basic operations with data II

```
y[4,3] # Prints element from fourth row and third column
x <- y[2,] # Replaces "x" by second row of "y" (no warning)
rm(x) # Deletes x (no warning)
y[,1:3] # Prints first through third column of the matrix
y[3,] <- rep(x=20, each=4) # Replaces third line by value of 20
y[y==20] <- 10 # If value of y's element is 20, replace it by 10
summary(y) # Basic statistics - according to columns
colnames(y) <- c("A", "B", "C", "D") # Set column names
# Objects and functions are without quotation marks; files and text with
colnames(y) # Prints column names, use rownames() in very same way
y[,"C"] # Prints column C (R is case sensitive!)
t(y) # Transposes the matrix
y <- as.data.frame(y) # Turns into DF (see other functions as.*)
y[y==17] <- "NA" # Removes values of 17 (NA = not available = missing)
y$B # Gets variable B of data frame y ($ works similarly in S3 objects)
```

Basic operations with data III

```
# When loading saved project, you have to load again libraries and
# scripts (see further), data objects are restored
save(list=ls(), file="test.RData") # Saves all objects during the work
load("test.RData") # Loads saved R environment with all objects
# Use to edit matrices, data frames, functions, ...
fix(y)
```

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 49 and onward)
- Repositories doesn't have to be set as global options, e.g. Bioconductor 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 49 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 49. **OK**. Go to menu **Settings | Manage R packages and plugins...**, click to **Install...**, select and install desired packages...

Theory for packages and their management

- Standalone plain R doesn't have enough tools for most of scientific disciplines — only basic methods and tools for programmers, including for package management
- Users/developers contribute by making extra packages extending computational possibilities — one of biggest R advantages — it then has unlimited possibilities
- R has infrastructure for maintaining (for developers) and installing (for users) packages — the CRAN repository
- For various reasons, some people build their own infrastructures to maintain and install R packages — compatible with R, bud separated
- User has basically two options
  1. Set all repositories in R and use basic commands to install packages (slide 49)
  2. Specify non-CRAN repository every time installing from it (e.g. slide 54) or use special tools (e.g. for Bioconductor — slide 55)

Set repositories

```
# Basic package installation
install.packages("PackageName") # Case sensitive!
?install.packages # Shows all available parameters (options)
getOption("repos") # Shows actual repositories
options(repos=c("https://mirrors.nic.cz/R/",
  "https://r-forge.r-project.org/", "https://rforge.net/"))
options() # Generic function to modify various settings
?options # Gives details
```

- Keep newest version of R and and newest versions of packages!
- Installation of multiple packages may sometimes fail — install then packages in smaller groups or one by one — 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 3.5 to 3.6), user must reinstall all packages

Install packages

- If repositories from slide 49 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 3.5 to 3.6), all packages must be reinstalled

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

Install packages needed for the course

```
# Install packages
# Installation of multiple packages may sometimes fail - install then
# packages in smaller groups or one by one
install.packages(pkgs=c("ade4", "adegenet", "adegraphics", "adephylo",
  "akima", "ape", "BiocManager", "caper", "corrplot", "devtools",
  "gee", "geiger", "ggplot2", "gplots", "hierfstat", "ips", "kdetrees",
  "lattice", "mapdata", "mapplots", "mapproj", "maps", "maptools",
  "nlme", "PBSmapping", "pegas", "phangorn", "philentropy", "phylobase",
  "phytools", "picante", "plotrix", "poppr", "raster", "rgdal",
  "RgoogleMaps", "Rmpi", "rworldmap", "rworldxtra", "seqinr",
  "shapefiles", "snow", "sos", "sp", "spdep", "splancs", "StAMPP",
  "TeachingDemos", "tripack", "vcfR", "vegan"),
  repos="https://mirrors.nic.cz/R/", dependencies="Imports")
# Regularly update installed packages
update.packages(ask=FALSE)
# Upgrade all packages e.g. from R 3.5 to 3.6
install.packages(pkgs=installed.packages())
```

Install phyloch package

Example of installation of package not available in any repository

- Check http://www.christophheibl.de/Rpackages.html
- Package phyloch is similar to ips from same author (but some functions behave differently) — both are great for usage of external applications within R, ips seems to develop more and phyloch will probably be deprecated...

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

Install Geneland package

- Since version 4, not in CRAN anymore, check manual and https://i-pri.org/special/Biostatistics/Software/Geneland/

```
# Install package Geneland (since version 4 not available in CRAN)
# It is possible to specify direct path to package source
install.packages("https://i-pri.org/special/Biostatistics/Software/
  Geneland/distrib/Geneland_4.0.8.tar.gz", repos=NULL, type="source")
# If above command fails on Windows, try
install.packages("https://i-pri.org/special/Biostatistics/Software/
  Geneland/distrib/Geneland_4.0.8.zip", repos=NULL)
# Other packages used when using Geneland
# Needed is PBSmapping or mapproj for conversion of coordinates
# GUI uses for parallelisation snow and Rmpi
# RgoogleMaps (requires rgdal) can be used to plot Geneland output
# on top of Google map, maptools, shapefiles and tripack on GIS layer
install.packages(pkgs=c("PBSmapping", "mapproj", "rgdal", "RgoogleMaps",
  "Rmpi", "sp", "maptools", "shapefiles", "snow", "tripack"),
  repos="https://mirrors.nic.cz/R/", dependencies="Imports")
```

Bioconductor and others — differences from another repositories

Bioconductor has its own installation method

```
# Standard installation
install.packages(pkgs=c("adegenet", "poppr", "phytools"))
update.packages(ask=FALSE) # Update packages
# Installation from custom repository (example for our course)
install.packages(pkgs="ParallelStructure",
  repos="https://r-forge.r-project.org/")
?install.packages # See help for details
# Install CRAN package BiocManager used to manage Bioconductor packages
if (!requireNamespace("BiocManager"))
  install.packages("BiocManager")
BiocManager::install() # Update installed packages
BiocManager::install()
# Install Bioconductor packages
BiocManager::install("muscle") # Simplest usage
BiocManager::install(pkgs=c("Biostrings", "muscle"))
?BiocManager::install # See more options
```

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** (recommended by Bioconductor):

```
# Install CRAN package BiocManager used to manage Bioconductor packages
if (!requireNamespace("BiocManager"))
  install.packages("BiocManager")
BiocManager::install()
# Update installed packages
BiocManager::install()
# Search for Bioconductor packages
BiocManager::available() # List everything
?BiocManager::available # See options
```

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

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 112)
- Gdal https://trac.osgeo.org/gdal/wiki/DownloadingGdalBinaries
  - Recommended for conversion of coordinates for Geneland (from slide 241), loading of SHP files, etc.
- GIMP https://www.gimp.org/
  - Image manipulation, free open-source alternative to products of Adobe or Corel
  - Optional, one of possibilities to view and edit output graphics from R
- Inkscape https://inkscape.org/

Non-R software II

- 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 112)
- 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 112)
- proj https://proj4.org/download.html
  - Required for conversion of coordinates for Geneland (from slide 241)

Non-R software III

- Structure
  https://web.stanford.edu/group/pritchardlab/structure.html
  (optionally also CLUMPP and distruct — not part of the course)
  - De facto standard for analysis of population genetic structure (from slide 205)
  - 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

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) 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
  - 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 will also use data provided by the teacher and/or his colleagues

```
data() # List data available in currently loaded packages
# List data available in all installed packages (can be very long)
data(package=rownames(installed.packages()))
# Load selected data set; for example phylogeny and species traits
# of shorebirds from package caper (we will use it much later)
data(shorebird, package="caper")
# Optional method (load respective library and then data)
library(caper) # Library containing (among others) desired data
data(shorebird) # Loading the data
?shorebird # See content of the dataset # or ?caper::shorebird
?adegenet::microbov
?adegenet::rupica
?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
  - Modified subset of triploid microsatellite data of several species of *Taraxacum* sect. *Taraxacum* (Asteraceae) — *T. alatum*, *T. ekmanii*, *T. hemicyclum* and *T. hepaticum* from central and northern Europe by Kirschner et al 2016
  - Small subset of population AFLP data of *Cardamine amara* (Brassicaceae) from Europe by Karol Marhold and his team
  - Small subset of non-synonymous SNPs from ASY3 gene (required for meiosis) from diploids and tetraploids of *Arabidopsis arenosa* (Brassicaceae) from central and northern Europe by Magdalena Bohutínská and her colleagues
  - Small subset of trees and taxa from phylogeny of *Oxalis* spp. (Oxalidaceae) from South Africa (the Cape region) by Schmickl et al. 2016
  - Partial mitochondrial sequence of mm18 control region of *Meles meles* (Mustelidae) from Frantz et al 2014 downloaded from GenBank

Our data... II

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

- Work with microsatellites is in most cases (except some methods taking advance from microsatellite mutational nature) same as with presence-absence data and methods can handle both data types in nearly same fashion

Our data... III

- Examples are shown mainly with microsatellites, but AFLP and another presence-absence (1/0) data are used in same way — try it
- Distance-based methods are same regardless input data on the beginning (microsatellites, AFLP, DNA sequences, ...)
- Extraction of SNP from DNA is useful in case of huge data sets — for smaller data sets it is not necessary
- 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)
  - Compressed version of SAM (see below)
  - Suffix `*.bam`
- **CSV** (Comma Separated Values)
  - "One sheet of Excel"
  - Common format to store data (traits, coordinates, ...), similar to TSV
  - Columns (cells) are separated by commas, cells are commonly bordered by quotation marks — it is important to check structure before import into R (and very it after import)
  - Suffix usually `*.csv`
- **FASTA**
  - Simple and popular text format to store genetic sequences (DNA, RNA, proteins)
  - **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 `>`

Input/output data formats II

Representative selection

- Suffix usually `*.fasta` (generic, also `*.fas`, `*.fa`, `*.seq`, `*.fsa`), `*.fna` (nucleic acid), `*.ffn` (nucleotides, coding regions), `*.faa` (amino acids), `*.frn` (non-coding RNA)

- **FASTQ**
  - Text based format to store sequences (mainly nucleotides)
  - Every record consists of **4 lines**: (**1**) sequence ID (with possible description) starting with `@`, (**2**) the sequence (`ATCG...`), (**3**) `+` optionally followed by the same ID as line 1, and (**4**) quality values for nucleotides from line 2
  - Common format for output of modern high throughput sequencing machines (e.g. Illumina)
  - Commonly compressed by `gzip` (`*.gz`), sometimes by other compression application
  - Suffix usually `*.fastq`, `*.fq`, `*.fastq.gz`, `*.fq.gz`, ...
- **NEWICK**

Input/output data formats III

Representative selection

- Every line contains one tree represented by brackets, optionally with numbers (separated by `:`) labeling nodes and/or branches (bootstrap supports, likelihoods, branch lengths, ages, ...)
- E.g. `(A,B,(C,D)E)F;` or `(A:0.1,B:0.2,(C:0.3,D:0.4):0.5);`
- Suffix usually `*.newick`, `*.nwk`, `*.tre`, ...

- **NEXUS**
  - Popular plain text format used by software like Mesquite, MrBayes, PAUP*, SplitsTree, ...
  - Structure can be complex, is divided into blocks containing e.g. sequences, trees (in NEWICK format), distance matrix, fragmentation data, networks (e.g. for SplitsTree), MrBayes commands, traits, ...
  - Several variants, sometimes problems with interoperability
  - Suffix usually `*.nexus`, `*.nex`, `*.nxs`
- **SAM** (Sequence Alignment Map)
  - Text-based format for storing biological sequences aligned to a reference sequence

Input/output data formats IV

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`

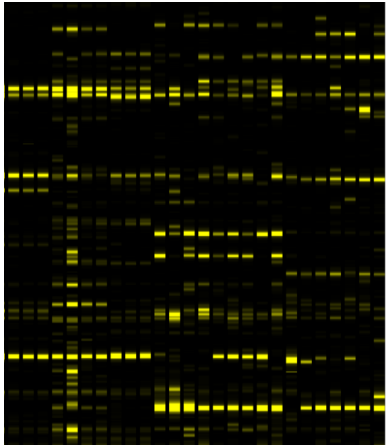
- **TXT**
  - Plain text file can contain content of all listed file formats (except binary BAM) — suffix (`*.txt`) is not really reliable...
  - Technically, all listed plain text formats belong also to this category

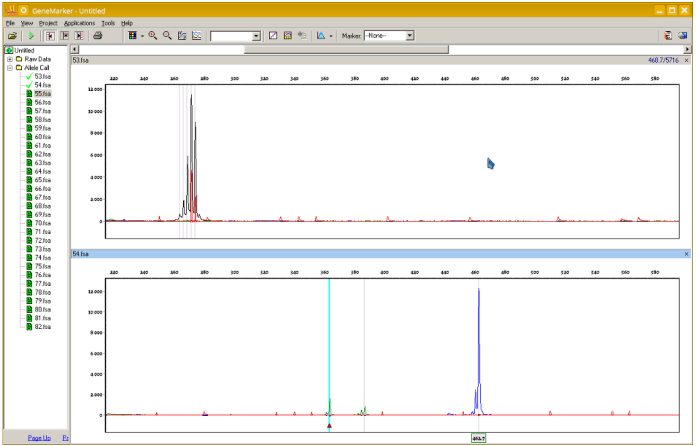
Input/output data formats V

Representative selection

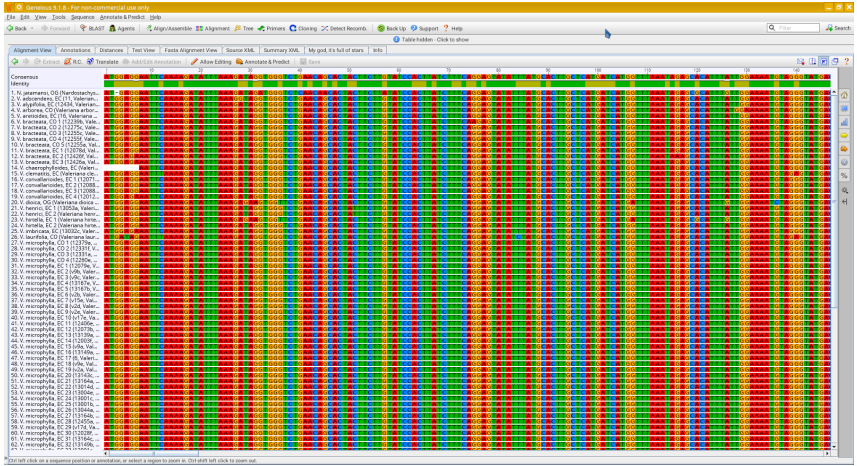
- **VCF** (Variant Call Format)
  - Do not confuse with vCard (`*.vcf`, `*.vcard`) storing virtual business cards and address books
  - Bioinformatics plain text format storing gene variants, annotations, quality data and more information
  - Used by software like Bcftools, GATK, Picard, VCFtools, ...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

Examples of data and formats I — AFLP (presence/absence) gel and microsatellites from sequencing machine





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



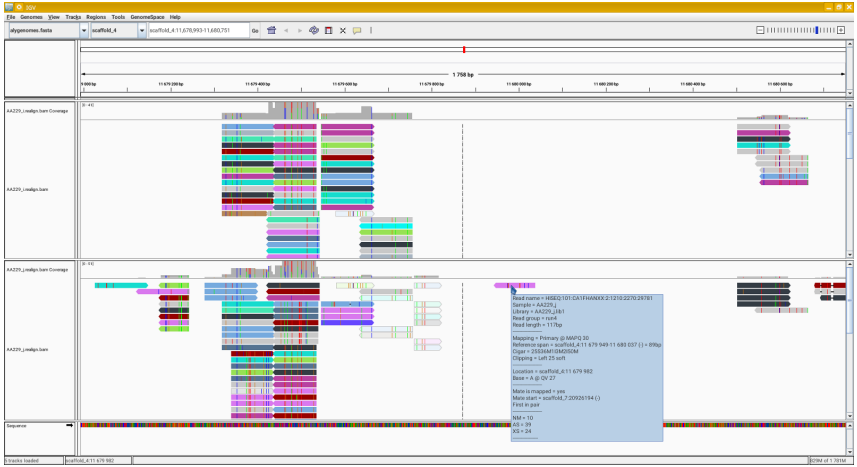
Examples of data and formats III — FASTA and FASTQ sequences in text view

```
> CY013200
atgaagactatcattgctttgagctacattttatgtctggttttcgctcaaaaacttcccctagtgaaaaca
ggaaatgacaacagcacagcaacgctgtgcctgggacaccatgcagtgccaaacggaacgatcacgaatgat
> CY013781
atgaagactatcattgctttgagctacattttatgtctggttttcgctcaaaaacttccccaaattgaagtg
... # FASTA continues...
```

```
@7001425F:141:CAV4JANXX:3:1104:1496:1976 1:N:0:GTGTCC
NCGATTCATTTTAGCAATTAGACGTGAAGGTCTCTTGATGAAAGACACTAACGAACTCTTTCCTTGGACACC
+
#<<BBFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
@7001425F:141:CAV4JANXX:3:1104:1250:1987 1:N:0:GTGTCC
TCGATTCACTGAACTGAATGTCCGACAAACTTTAGTTTGTCGTTTCTACCTCACCAAAGTTCGGAGCTTCGA
+
... # FASTQ continues...
```

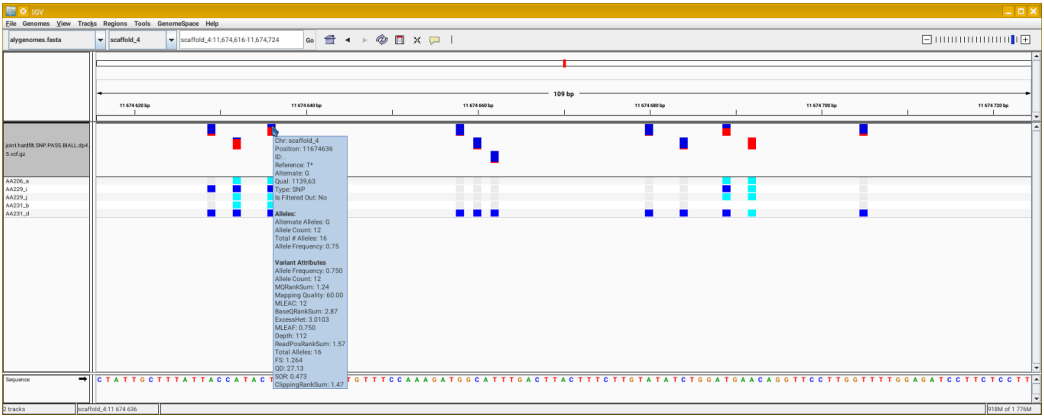
Examples of data and formats IV — BAM displayed in IGV

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

```
... # Header
##ALT=<ID=NON_REF,Description="Represents any possible alternative al...
##FILTER=<ID=DP_4,Description="DP < 4">
... # Information about data stored
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in geno...
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency, for ...
... # Information about chromosomes etc.
##contig=<ID=scaffold_1,length=33132539>
##contig=<ID=scaffold_2,length=19320864>
... # The data (variants of nucleotides)
#CHROM  POS      ID       REF     ALT     QUAL     FILTER  INFO     FORMA...
scaffold_4       22846289         .       C       A       91.52    PASS ...
scaffold_4       22846291         .       C       T       325.58   PASS ...
...AC=1;AF=1.057e-03;AN=946;BaseQRankSum=0.825;ClippingRankSum=0.118;...
...DP=4046;ExcessHet=3.0302;FS=0.000;InbreedingCoeff=-0.0096;MQ=79.24...
... # 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 357

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

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

Load data

```
# Source data
pop  msta93 msta101 msta102 msta103 msta105 msta131 ...
H01  He 269/269 198/198 221/223 419/419 197/197 196/196 ...
H02  He 275/283 198/198 221/223 419/419 193/193 168/190 ...
... ...     ...     ...     ...     ...     ...     ... ...
# Loading the data
# Load training data (Taraxacum haussknechtii from Macedonia)
hauss.loci <- read.loci(file="https://soubory.trapa.cz/rcourse/
  haussknechtii_ssrs.txt", header=TRUE, loci.sep="\t", allele.sep="/",
  col.pop=2, col.loci=3:14, row.names=1) # \t means TAB key
hauss.loci # Data control
print(hauss.loci, details=TRUE)
```

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

Prepare genind object for analysis and load coordinates

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

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

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

Add coordinates to genind and create genpop object

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

Import existing data set from popular software

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

One function rules them all...

All those functions (including `read.loci()` and `read.csv()`) are only modifications of **`read.table()`**. You can use it directly to import any data. Look at `?read.table` and play with it. Take care about parameters. Does the table use quotes to mark cell (e.g. `quote="\""`)? How are columns separated (e.g. `sep="\t"`)? Is there a header with names of populations/loci/whatever (`header=T/F`)? What is decimal separator (e.g. `dec="."`)? Are there row names (used typically as names of individuals; e.g. `row.names=1`)? Check data after import!

Import of polyploid microsatellites

- `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:

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

Triploid species of *Taraxacum* sect. *Taraxacum*

How to import polyploid microsatellites

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

Import of AFLP data — background

Two files — AFLP data with individual names, and populations

AFLP or any other presence/absence data:

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

AFLP data of *Cardamine amara* group

Individual's populations:

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

Just list of populations for respective individuals...

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

Import of AFLP data — the code

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

Another data manipulation

- SNPs can be into genind imported in same way as AFLP (PA)

```
genind2df() # adegenet - export into data frame
genind2genalex() # poppr - export for genalex
splitcombine() # poppr - edits population hierarchy
popsub() # poppr - extracts only selected population(s)
clonecorrect() # poppr - corrects for clones
informloci() # poppr - removes uninformative loci
seppop() # adegenet - separates populations from genind or genlight
seploc() # adegenet - splits genind, genpop or genlight by markers
alleles2loci() # pegas - transforms a matrix of alleles into "loci"
# seppop and seploc return lists of genind objects - for further
# analysis using special functions to work on lists (see further)
# read manual pages (?...) of the functions before usage
```

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

Notes about getting data into R

- When importing fragmentation data, we somehow use function `read.table()` — it is important to understand it
- I recommend to use TAB (TSV — tab separated values; encoded as `\t` in R) to separate columns (no quotation marks, no commas)
- When importing microsatellites, all alleles **must** have same number of digits. Separate alleles by "`/`", "`|`" or something similar and correctly specify it in `read.loci()` or `df2genind()` (or read the data with `read.table()`, convert into matrix and use `alleles2loci()`)
- **Do not use** underscores ("`_`") or minuses ("`-`") to name objects in R — only numbers, Latin letters or dots
- `read.loci()` sometimes doesn't work correctly on AFLP or polyploid microsatellites — try `read.table()` instead...
- Genind object is able to store mixed ploidy data, but not all analysis are able to handle it

Import of DNA sequence data I

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

Import of DNA sequence data II

```
# Convert DNAbin to genind - only polymorphic loci (SNPs) are retained
# When converting DNAbin to genind, the sequences must be aligned!
usflu.genind <- DNAbin2genind(x=usflu.dna, pop=usflu.annot[["year"]])
usflu.genind # Check it
# Read sequence data in NEXUS
read.nexus.data(file="sequences.nex")
```

- RNA or protein sequences can be handed in same way — see `?read.dna` and `?read.FASTA`
- For sequences there is R class `DNAbin`, for protein `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/608602125 (*Meles meles*, phylogeographical study of Frantz et al. 2014)

```
# Importing sequences according to sequence ID
meles.dna <- read.GenBank(c("KJ161355.1", "KJ161354.1", "KJ161353.1",
  "KJ161352.1", "KJ161351.1", "KJ161350.1", "KJ161349.1", "KJ161348.1",
  "KJ161347.1", "KJ161346.1", "KJ161345.1", "KJ161344.1", "KJ161343.1",
  "KJ161342.1", "KJ161341.1", "KJ161340.1", "KJ161339.1", "KJ161338.1",
  "KJ161337.1", "KJ161336.1", "KJ161335.1", "KJ161334.1", "KJ161333.1",
  "KJ161332.1", "KJ161331.1", "KJ161330.1", "KJ161329.1", "KJ161328.1"))
meles.dna
class(meles.dna)
# Converts DNAbin to genind - extracts SNP - for large data sets can be
# computationally very intensive
meles.genind <- DNAbin2genind(meles.dna)
meles.genind # Check it
```

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

Query on-line sequence databases

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

Importing SNP

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

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

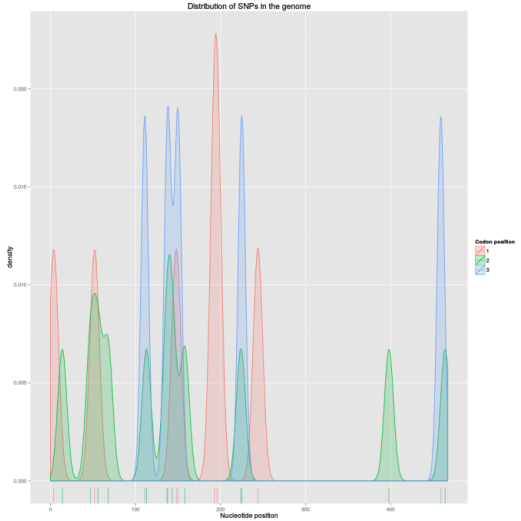
- Extracting SNP from alignments reads FASTA alignments and keep only SNPs. The method is relatively efficient even for large data sets with several genomes:

```
usflu.genlight <- fasta2genlight
  (file="http://adegenet.r-forge.r-project.org/files/usflu.fasta",
  quiet=FALSE, saveNbAlleles=TRUE)
usflu.genlight # See genlight
?fasta2genlight # Function has several options to speed up reading
# 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

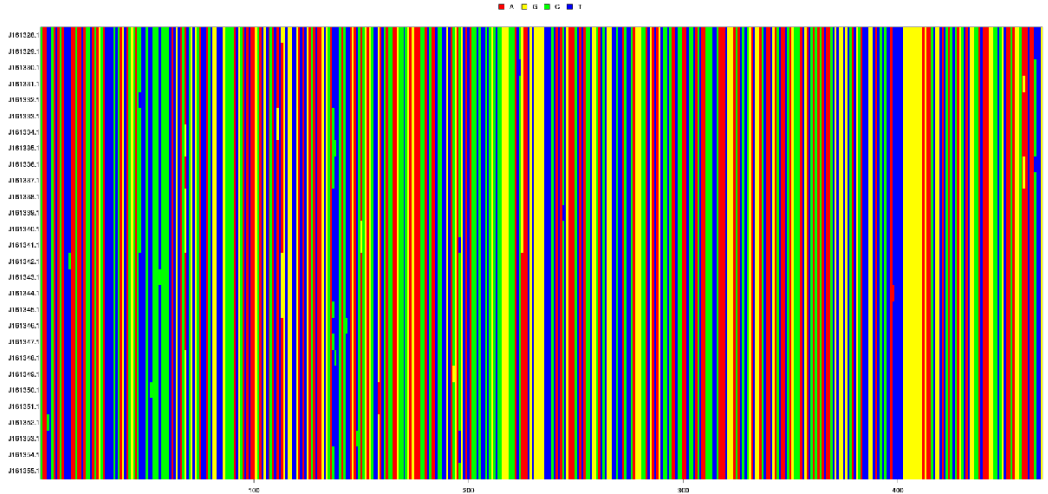
```
# Position of polymorphism within
# alignment - snpposi.plot requi-
# res input data in form of matrix
snpposi.plot(x=as.matrix(
  meles.dna), codon=FALSE)
# Position of polymorphism within
# alignment-differentiating codons
snpposi.plot(as.matrix(meles.dna))
# When converting DNAbin to genind
# only polymorphic loci are kept -
# threshold for polymorphism can
# be arbitrary (polyThres=...)
meles.genind <- DNAbin2genind(x=
  meles.dna, polyThres=0.01)
meles.genind # See it
```



Checking sequences

```
# Test is distribution of SNPs is random (1000 permutations)
snpposi.test(as.matrix(meles.dna))
pegas::nuc.div(x=meles.dna) # Nucleotide diversity
ape::base.freq(x=meles.dna) # Base frequencies
ape::GC.content(x=meles.dna) # GC content
# Number of times any dimer/trimer/etc oligomers occur in a sequence
# Note: count() requires single sequence as DNAbin/character
seqinr::count(seq=as.character.DNAbin(meles.dna[["KJ161328.1"]]),
  wordsize=3)
# View sequences - all must be of the same length
image(x=usflu.dna, c("a", "t", "c" ,"g", "n"), col=rainbow(5))
# Function "image" requires as input matrix
# So that sequences must be of same length
image(x=as.matrix(meles.dna), c("a", "t", "c" ,"g", "n"), col=rainbow(5))
# Direct function to display the sequences
image.DNAbin(x=usflu.dna)
image.DNAbin(x=as.matrix(meles.dna))
```

Meles sequences



Notes about using genlight (vs. genind)

- Genlight is "just" version of more common genind object to store large data sets with (nearly) complete multiple genomes
- "Large" is tricky — there is no easy criterion (roughly, genind is inefficient since dozens or hundreds thousands of SNPs) — try genind and when work fails because of not enough computer resources, go on with genlight
- Use is basically same as when working with genind — but not all functions are able to deal with it (on the other hand, others are optimized to work well on large data sets)
- SNPbin is version of genind/genlight to store one large genome — serves basically as storage, no need to deal with it
- Genlight as well as genind allow varying ploidy level
- Functions working with genlight use to use parallelisation to speed up operations — this commonly doesn't work properly on MS Windows

Reading VCF

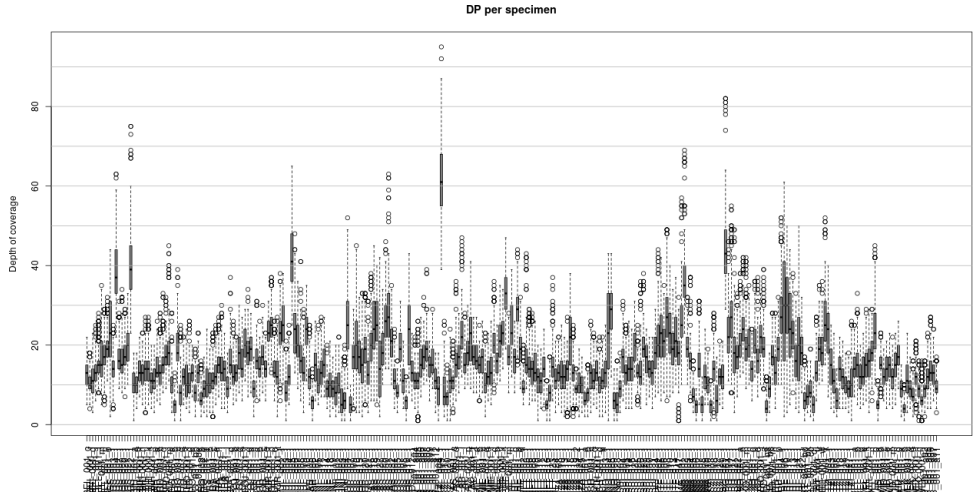
- Download input file from
  https://soubory.trapa.cz/rcourse/arabidopsis.vcf.gz
- Non-synonymous SNPs from ASY3 gene (required for meiosis) from diploids and tetraploids of *Arabidopsis arenosa* from central and northern Europe
- Package vcfR has functions to manipulate and explore VCF

```
library(vcfR) # Required library
# Pick up downloaded file 'arabidopsis.vcf.gz' from the disk
arabidopsis.vcf <- read.vcfR(file=file.choose())
# File choose dialog can open in background - search for it :-)
# Or directly load remote file
arabidopsis.vcf <- read.vcfR
  (file="https://soubory.trapa.cz/rcourse/arabidopsis.vcf.gz")
# It returns object of class vcfR-class
?read.vcfR # See more import options
?pegas::read.vcf # This one returns list of objects loci and data.frame
```

Checking VCF I

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

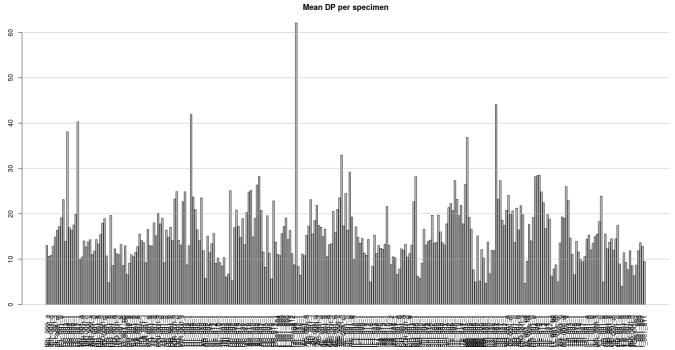
DP per specimen



Checking VCF II

Bar plot of mean DP

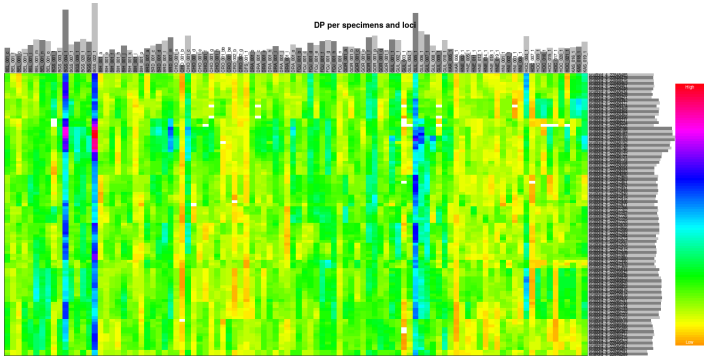
```
barplot(apply(X=arabidopsis.vcf.dp, MARGIN=2, FUN=mean, na.rm=TRUE), las=3)
title("Mean DP per specimen")
abline(h=seq(from=0, to=60, by=10), col="#b3b3b3")
```



Checking VCF III

Heat map of DP

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

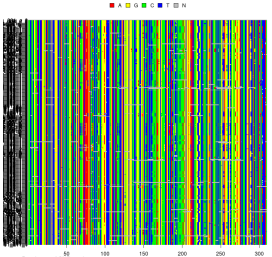


Convert VCF into genind, genlight and loci

```
# Genind
arabidopsis.genind <- vcfR2genind(x=arabidopsis.vcf)
arabidopsis.genind # Check it
nInd(arabidopsis.genind)
indNames(arabidopsis.genind)
nLoc(arabidopsis.genind)
locNames(arabidopsis.genind)
# Genlight (suitable for huge data, not required now)
# On Linux/macOS and with large data use higher n.cores
arabidopsis.genlight <- vcfR2genlight(x=arabidopsis.vcf, n.cores=1)
arabidopsis.genlight # Check it
# Loci
arabidopsis.loci <- vcfR2loci(x=arabidopsis.vcf)
arabidopsis.loci # Check it
print(x=arabidopsis.loci, details=TRUE)
```

Convert VCF into DNAbin

```
# There are various options how
# to process variants in VCF
?vcfR2DNAbin
arabidopsis.dnabin <- vcfR2DNAbin
  (x=arabidopsis.vcf, consensus=
  FALSE, extract.haps=TRUE,
  unphased_as_NA=FALSE)
# Check it
arabidopsis.dnabin
dim(arabidopsis.dnabin)
as.character.DNAbin
  (arabidopsis.dnabin[1:15,1:12])
image.DNAbin(arabidopsis.dnabin)
snpposi.plot.DNAbin
  (arabidopsis.dnabin)
snpposi.test.DNAbin
  (arabidopsis.dnabin)
```



Remove non-biallelic loci

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

```
# Create vector of loci to keep
arabidopsis.genind.keep <- nAll(arabidopsis.genind) < 3
# See the vector of loci
arabidopsis.genind.keep
# Keep only passing loci
arabidopsis.genind.bial <-
  arabidopsis.genind[loc=arabidopsis.genind.keep]
# See result
arabidopsis.genind.bial
# Filtration of missing data
?poppr::missingno # Various options...
```

Export data

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


```
# Go back to the original working directory
# Go to YOUR OWN directory, same as on beginning
setwd("/home/vojta/dokumenty/vyuka/r_mol_data/examples/")
```

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...
- R packages use common alignment software: MAFFT, MUSCLE, Clustal, ...
  - User must install this software manually — R is just using external applications (in the examples shown)

Multiple sequence alignment with MAFFT

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

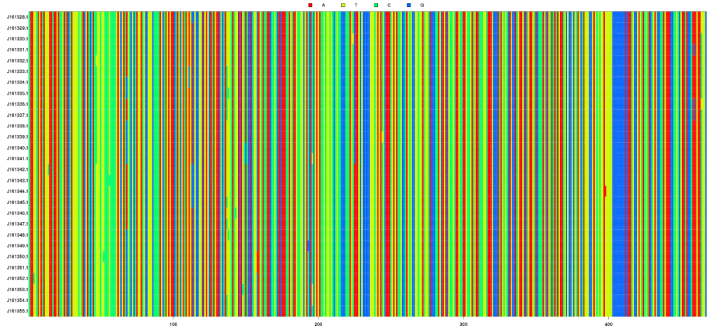
```
library(ips)
# Requires path to MAFFT binary - set it according to your installation
# read ?mafft and mafft's documentation
# Change "exec" to fit your path to mafft!
meles.mafft <- mafft(x=meles.dna, method="localpair", maxiterate=100,
  options="--adjustdirection", exec="/usr/bin/mafft")
```

Clustal, MUSCLE and T-Coffee from ape

```
meles.mafft
class(meles.mafft)
# read ?clustal and documentation of Clustal, Muscle and T-Coffee
# when using them to set correct parameters
meles.clustal <- ape::clustal(x=meles.dna, pw.gapopen=10, pw.gapext=0.1,
  gapopen=10, gapext=0.2, exec="/usr/bin/clustalw2", quiet=FALSE,
  original.ordering=TRUE) # Change "exec" to fit your path to clustal!
meles.clustal
class(meles.clustal)
meles.muscle <- muscle(x=meles.dna, exec="muscle", quiet=FALSE,
  original.ordering=TRUE) # Change "exec" to fit your path to muscle!
meles.muscle
class(meles.muscle)
?muscle::muscle # See option in muscle package
# Remove gaps from alignment - destroy it
meles.nogaps <- del.gaps(meles.muscle)
?del.gaps # See for details
```

Multiple sequence alignment with MUSCLE

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



Align multiple genes

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

```
# Create a list of DNAbin objects to process
multialign <- list(meles.dna, usflu.dna, usflu.dna2)
# See it
multialign
class(multialign)
lapply(X=multialign, FUN=class)
# Do the alignment
multialign.aln <- lapply(X=multialign, FUN=ips::mafft, method="localpair",
maxiterate=100, exec="/usr/bin/mafft")
  # Change "exec" to fit your path to mafft!
# See result
multialign.aln
multialign.aln[[1]]
lapply(X=multialign.aln, FUN=class)
```

Align multiple genes in parallel

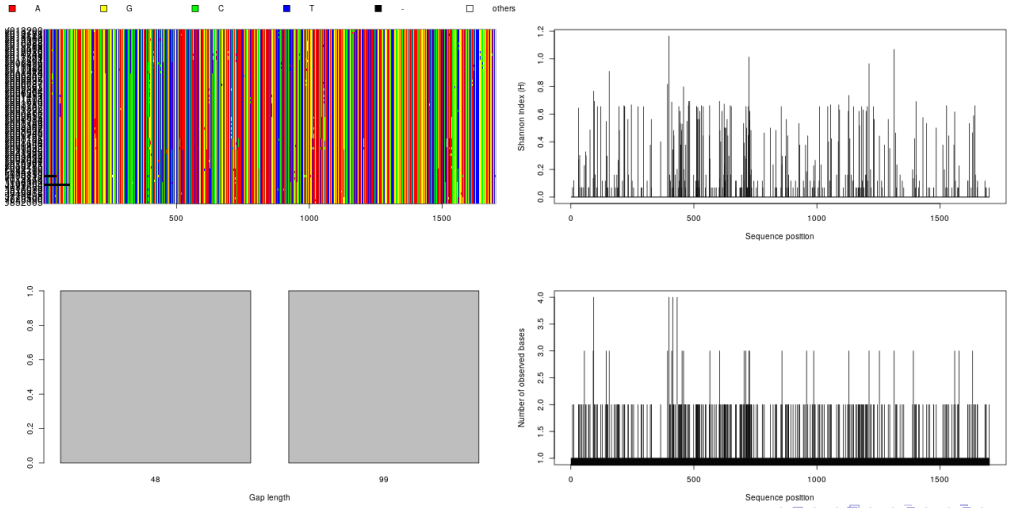
- There are plenty of implementations of parallelisation and using of computer clusters, see https://CRAN.R-project.org/web/views/HighPerformanceComputing.html

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

Checking the alignment

```
# Shortcut for plotting alignment
image.DNAbin(x=meles.mafft)
# Display aligned sequences with gaps
image.DNAbin(x=usflu.dna)
# Check the alignment
checkAlignment(x=usflu.dna, check.gaps=TRUE, plot=TRUE, what=1:4)
checkAlignment(x=as.matrix.DNAbin(x=meles.dna), check.gaps=TRUE,
  plot=TRUE, what=1:4)
?checkAlignment # See details
# DNAbin can be technically list or matrix - some functions require
# list, some matrix, some can handle both - check manual and if needed,
# use:
as.matrix.DNAbin()
as.list.DNAbin()
# Matrix makes sense only for alignments, list for any import
# (sequences do no have to have same lengths)
```

Checking the alignment



Cleaning the alignment

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

Cleaning the alignment

```
# Delete every alignment position having at least 20% of missing data
usflu.dna.ng <- del.colgapsonly(x=usflu.dna.ng, threshold=0.2,
  freq.only=FALSE)
?ape::del.colgapsonly # See help page for details
# Display the result
image.DNAbin(x=usflu.dna.ng)
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 ∼10–15% 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

Basic analysis

Introductory overview of statistics and methods I

- Selected method depends on data type, question to answer, ...
  - Check assumptions and requirements of the methods before usage
  - Think if the method answers your question
- **Population-genetic indices** — from slide 129
  - 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 147
  - It is crucial to select appropriate distance method for given data type
  - Usually require the distance matrix to be Euclidean
  - Distance matrix has one single number (index) for each pair of comparisons (individuals, populations) — rough

Introductory overview of statistics and methods II

- Generally, the matrices describe pairwise similarities among the individuals/populations
- Distance-based methods are phenetic
  - Based on similarity (described by the matrix), not on any (evolutionary) model
  - The matrix based on genetic data is supposed to well reflect the genetic similarity, thus real relationships among individuals/populations
- **Hierarchical clustering** — from slide 160
  - 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 167
  - A tree starting from the two most similar individuals and connecting in the next steps next and next the most similar individual
  - In some cases artificially chains individuals
  - Several methods try to improve it — slide 181
- **Principal Coordinates Analysis (PCoA)** — from slide 182

Introductory overview of statistics and methods III

- 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 166
  - 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?
  - 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

Introductory overview of statistics and methods IV

- 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 274
  - 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)
  - 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

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

- **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)
    - Usually, ∼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 ∼10–25% of steps

Introductory overview of statistics and methods VI

- 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

Descriptive statistics I

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

```
# Get summary - names and sizes of populations,
# heterozygosity, some info about loci
hauss.summ <- summary(hauss.genind)
# Plot expected vs. observed heterozygosity it looks like big difference
plot(x=hauss.summ$Hexp, y=hauss.summ$Hobs,
  main="Observed vs expected heterozygosity",
  xlab="Expected heterozygosity", ylab="Observed heterozygosity")
abline(0, 1, col="red")
# Bartlett's K-squared test of difference
# between observed and expected heterozygosity - not significant
bartlett.test(list(hauss.summ$Hexp, hauss.summ$Hobs))
            Bartlett test of homogeneity of variances
data:  list(hauss.summ$Hexp, hauss.summ$Hobs)
Bartlett's K-squared = 0.069894, df = 1, p-value = 0.7915
```

Descriptive statistics II

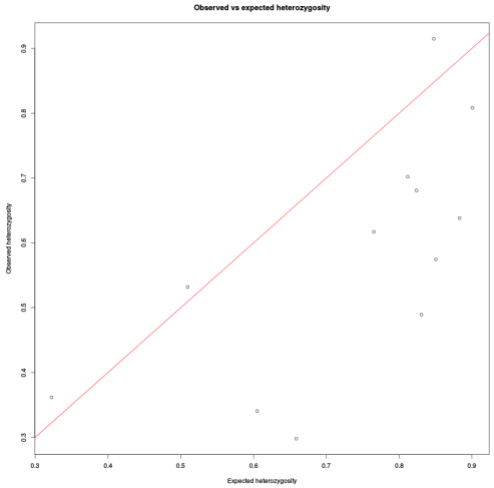
- `t.test` and `bartlett.test` require data to have normal distribution — if the condition is not met, it is necessary to use some weaker non-parametric test (`kruskal.test`, `wilcox.test`, ...)
- See respective manual pages for details
- `shapiro.test()` tests the normality of given vector

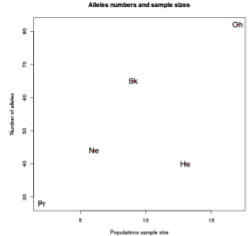
```
# T-test of difference between observed and expected heterozygosity
t.test(x=hauss.summ$Hexp, y=hauss.summ$Hobs, paired=TRUE, var.equal=T)
            Paired t-test
data:  hauss.summ$Hexp and hauss.summ$Hobs
t = 3.5622, df = 11, p-value = 0.004456 # strongly significant
alternative hypothesis: true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

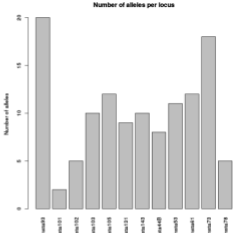
Descriptive statistics III

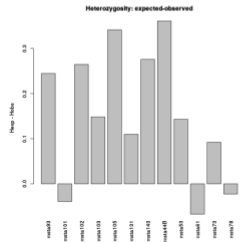
```
# Create pane with some information
par(mfrow=c(2, 2)) # Divide graphical devices into 4 smaller spaces
# Plot alleles number vs. population sizes
plot(x=hauss.summ$n.by.pop, y=hauss.summ$pop.nall, xlab="Populations
  sample size", ylab="Number of alleles", main="Alleles numbers and
  sample sizes",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

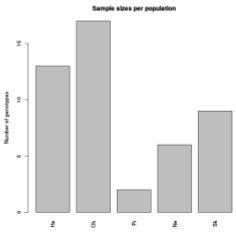
Graphs from previous slides











Population statistics by poppr()

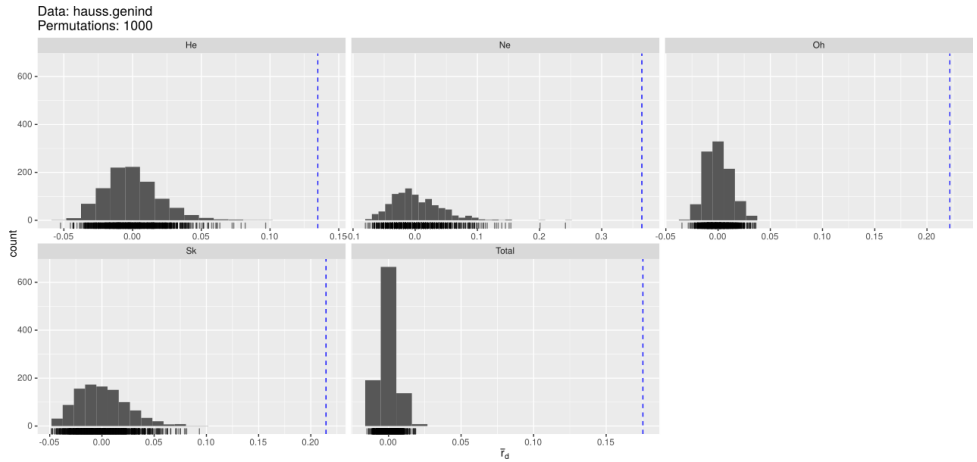
- `poppr()` is central function of `poppr` package calculating plenty of population genetic indices

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

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

Histograms of simulations of rbarD for each population

The populations are significantly far from being clonal



Population statistics returned by poppr() I

Too much to choose from?

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

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

Population statistics returned by poppr() II

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

Population statistics returned by poppr() III

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

Population statistics returned by poppr() IV

  - The index of association is a calculation based on the ratio of the variance of the raw number of differences between individuals and the sum of those variances over each locus
  - It as the observed variance over the expected variance — if they are the same, then the index is zero (=prevailing clonal reproduction) after subtracting one — it rises with with increasing differences
- `p.Ia` — P-value for `Ia` from the number of reshuffling indicated in `sample`
- `rbarD` — Standardized Index of Association for each population (see `?ia`) — corrected for higher number of loci not to rise so steeply
- `p.rD` — P-value for `rbarD` from the number of reshuffles indicated in `sample`
- See poppr's manual and `vignette("algo", package="poppr")` for details

Departure from Hardy-Weinberg equilibrium

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

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

Departure from HWE — results per locus

`hauss.hwe.test`

	chi^2	df	Pr(chi^2 >)	Pr.exact
msta93	383.5519728	190	3.552714e-15	0.000
msta101	0.6927242	1	4.052393e-01	0.657
msta102	83.0741964	10	1.250111e-13	0.000
msta103	77.1819098	45	1.998865e-03	0.000
...	...	...	...	...

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

F-statistics I

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

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

F-statistics II

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

F-statistics for mixed ploidy data I

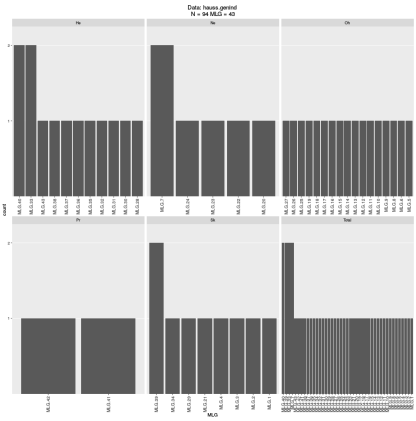
```
# stamppFst requires population factor in genlight (here, population
# code consists of first three letters of individual's name)
indNames(arabidopsis.genlight)
pop(arabidopsis.genlight) <- substr(x=indNames(arabidopsis.genlight),
  start=1, stop=3)
pop(arabidopsis.genlight) # Check it
popNames(arabidopsis.genlight)
arabidopsis.fst <- StAMPP::stamppFst(geno=arabidopsis.genlight,
  nboots=100, percent=95, nclusters=1)
# For large data use higher nclusters to parallelize calculations
?StAMPP::stamppFst # See method details
arabidopsis.fst[["Fsts"]] # Matrix of Fst among populations
arabidopsis.fst[["Pvalues"]] # Matrix of P values
# Save results - open in spreadsheet (e.g. LibreOffice Calc)
write.table(x=arabidopsis.fst[["Fsts"]], file="arabidopsis_fst.tsv",
  quote=FALSE, sep="\t")
```

F-statistics for mixed ploidy data II

- 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

```
# Correlation plot of pairwise Fst
library(corrplot)
corrplot(corr=arabidopsis.fst[["Fsts"]], method="circle", type="lower",
  col=funky(15), title="Correlation matrix of Fst among populations",
  is.corr=FALSE, diag=FALSE, outline=TRUE, order="alphabet", tl.pos="lt",
  tl.col="black")
?corrplot # See for more options
# Display in similar way also another Fst tables
```

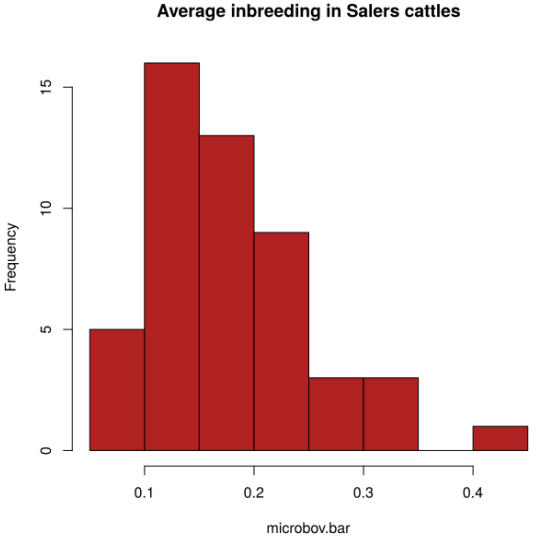
Multi locus genotypes




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

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

Inbreeding




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

Basic distances

```
# See ?dist.gene for details about methods of this distance
hauss.dist.g <- dist.gene(x=hauss.genind@tab, method="pairwise")
# Euclidean distance for individuals (plain ordinary distance matrix)
hauss.dist <- dist(x=hauss.genind, method="euclidean", diag=T, upper=T)
# Nei's distance (not Euclidean) for populations (other methods are
# available, see ?dist.genpop)
hauss.dist.pop <- dist.genpop(x=hauss.genpop, method=1, diag=T, upper=T)
# Test if it is Euclidean
is.euclid(hauss.dist.pop, plot=TRUE, print=TRUE, tol=1e-10) # FALSE = No
# Turn to be Euclidean
hauss.dist.pop <- cailliez(distmat=hauss.dist.pop, print=FALSE, tol=1e-07,
  cor.zero=TRUE)
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

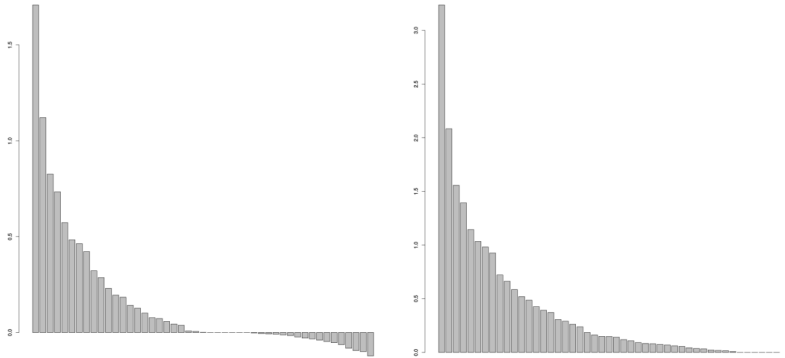
```
# Bruvo's distances weighting SSRs repeats - take care about replen
# parameter - requires repetition length for every SSRs locus
hauss.dist.bruvo <- bruvo.dist(pop=hauss.genind, replen=rep(2, 12),
  loss=TRUE)
# Test if it is Euclidean
is.euclid(hauss.dist.bruvo, plot=TRUE, print=TRUE, tol=1e-10)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

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

Import own distance matrix from another software:

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

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

Different distances have different use case and outputs...

Different distances available in package poppr

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

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

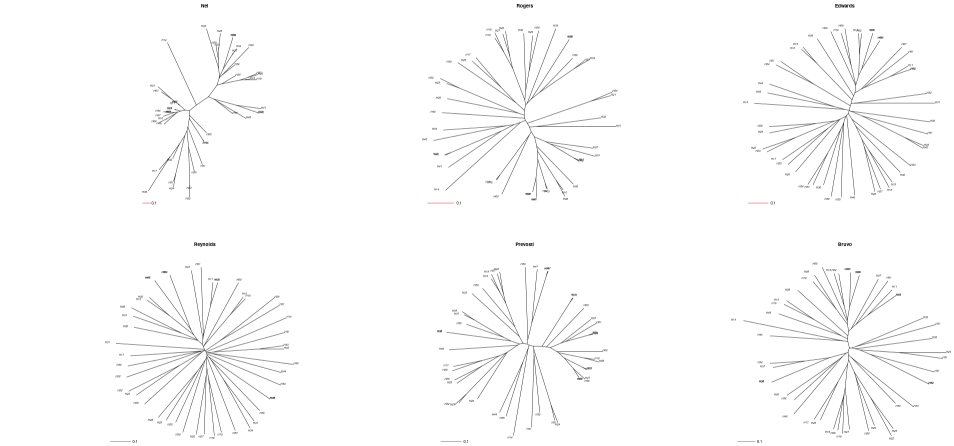
[1]Rogers (1972): Measures of genetic similarity and genetic distances. Pp. 145-153 of Studies in Genetics. University of Texas Publishers

Comparison of different matrices

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

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...

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

Distances and genlight object

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

```
usflu.dists.l <- seploc(usflu.genlight, n.block=10, parallel=FALSE)
class(usflu.dists.l)
usflu.dists <- lapply(X=usflu.dists.l, FUN=function(D) dist(as.matrix(D)))
class(usflu.dists)
names(usflu.dists)
class(usflu.dists[[1]])
usflu.distr <- Reduce(f="+", x=usflu.dists)
class(usflu.distr)
usflu.distr
# It is possible to use just basic dist function on whole genlight object
# (might require a lot of RAM)
usflu.distg <- dist(as.matrix(usflu.genlight))
```

Rationale of this approach is to save resources when dividing whole data set into smaller blocks — useful for huge data, not for all of the cases

Distances in mixed-ploidy data sets I

```
# Load required library
library(StAMPP)
# stamppNeisD requires population factor in genlight Nei's 1972 distance
# between individuals (use pop=TRUE to calculate among populations)
arabidopsis.dist <- stamppNeisD(geno=arabidopsis.genlight, pop=FALSE)
# Check it
head(arabidopsis.dist)
dim(arabidopsis.dist)
class(arabidopsis.dist)
# The same on population level
arabidopsis.dist.pop <- stamppNeisD(geno=arabidopsis.genlight, pop=TRUE)
# Check it
head(arabidopsis.dist.pop)
dim(arabidopsis.dist.pop)
class(arabidopsis.dist.pop)
```

Distances in mixed-ploidy data sets II

```
# Export the distance matrix as Phylip format for usage in
# external software (e.g. SplitsTree)
stamppPhylip(distance.mat=arabidopsis.dist, file="arabidopsis_dist.txt")
# Genomic relationship matrix
?stamppGmatrix # Method details
arabidopsis.genomat <- stamppGmatrix(geno=arabidopsis.genlight)
# Check it
head(arabidopsis.genomat)
dim(arabidopsis.genomat)
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

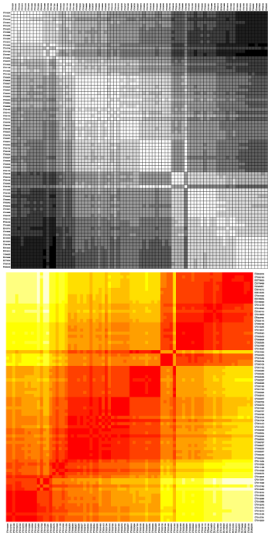
Over 40 distances from philentropy package

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

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

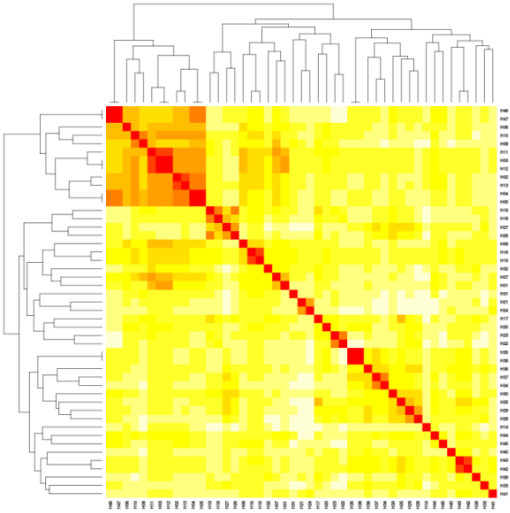
Visualize pairwise genetic similarities

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



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

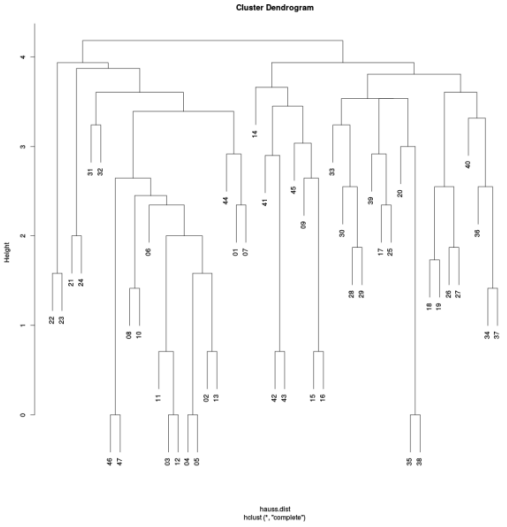
Heat maps



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

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

Hierarchical clustering — UPGMA and others



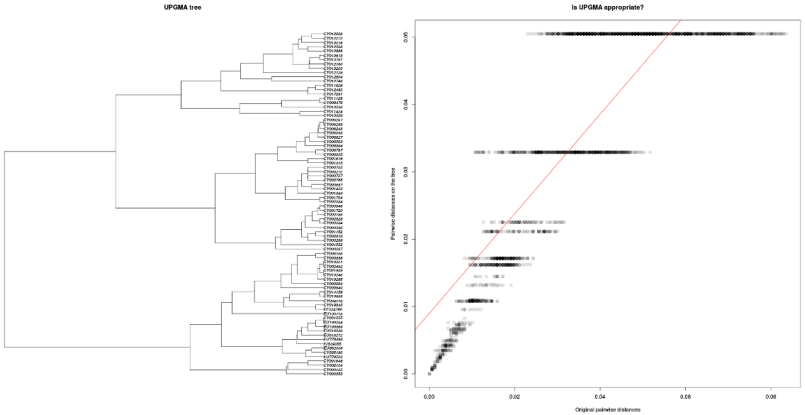

```
# According to distance used
# see ?hclust for methods
plot(hclust(d=hauss.dist,
  method="complete"))
plot(hclust(d=hauss.dist.pop,
  method="complete"))
plot(hclust(d=hauss.dist.bruvo,
  method="complete"))
```

- This is very basic function to make dendrogram
- There are better possibilities (NJ etc — see slide 167 and onward)

UPGMA and its test

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

UPGMA is not the best choice here...



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

AMOVA I

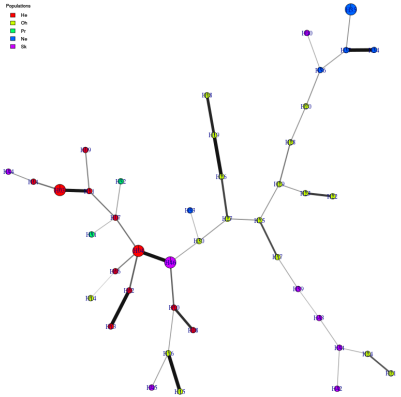
- Analysis of molecular variance tests if there are significant differences among populations (and/or another levels)
- Some implementations can partition variance into various levels
- `pegas::amova` returns a table of sums of square deviations (`SSD`), mean square deviations (`MSD`), and the number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

AMOVA II

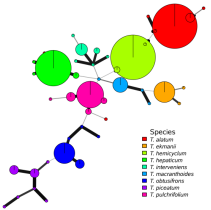
```
hauss.pop <- pop(hauss.genind)
hauss.amova <- pegas::amova(hauss.dist~hauss.pop, data=NULL,
  nperm=1000, is.squared=TRUE)
# See results
hauss.amova
...
                SSD      MSD df
hauss.pop  30.71923 7.679809  4
Error     119.58100 2.847167 42
Total     150.30023 3.267396 46
...
Variance components:
           sigma2 P.value
hauss.pop 0.55738       0 # From here we can calculate the percentage
Error     2.84717
...
```

Minimum Spanning Network

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




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

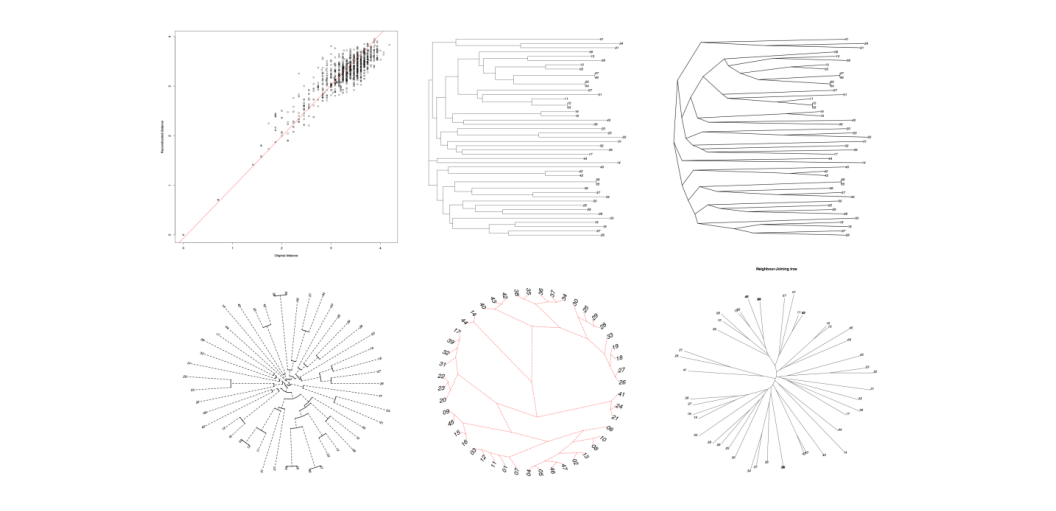



```
?bruvo.msn # See details...
```

Calculate and test NJ tree

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

Choose your tree...



Bootstrap

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

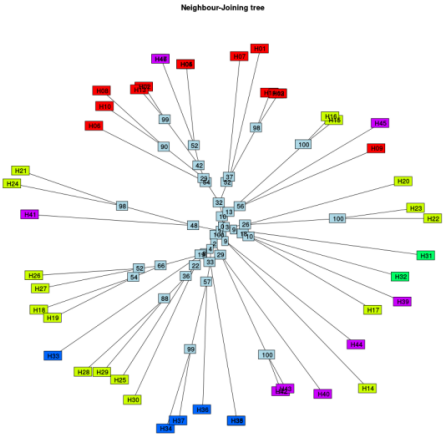
Plotting bootstrap and nicer trees

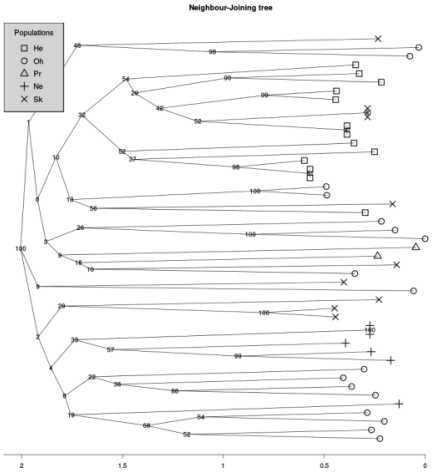
```
# Plot the tree, explicitly display node labels
plot.phylo(x=hauss.aboot, show.node.label=TRUE)
?aboot # Package poppr
## Plot a nice tree with colored tips
plot.phylo(x=hauss.nj, type="unr", show.tip=F, edge.width=3, main="NJ")
# Labels for nodes - bootstrap - see ?nodelabels for graphical settings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

Nicer trees

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

Choose your tree...



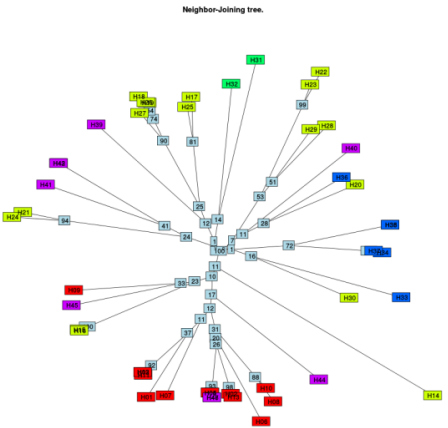


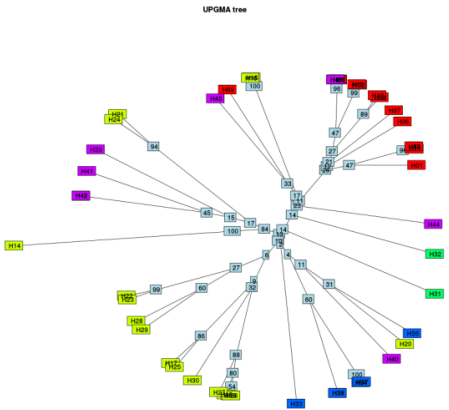
Trees based on Bruvo's distance

Package poppr (bootstrap is incorporated within the function)

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

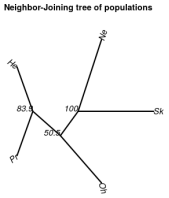
Choose your tree...





NJ tree of populations

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



NJ tree based on DNA sequences

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

Root the tree

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

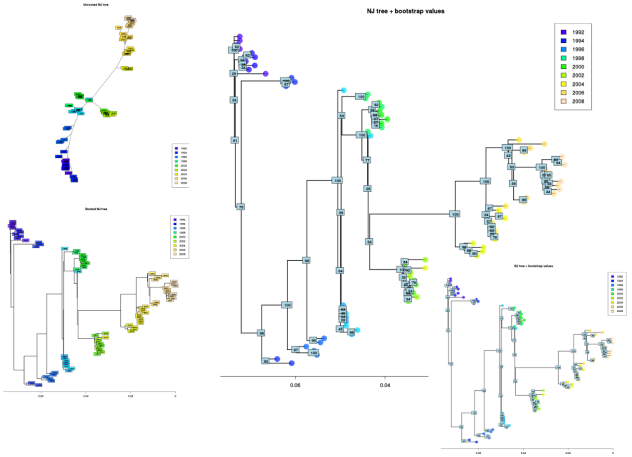
Bootstrap rooted tree

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

Collapse branches with low bootstrap support

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

The trees


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

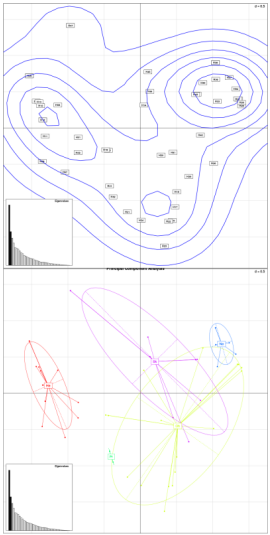
NJ is death. Long live NJ!

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

PCoA I

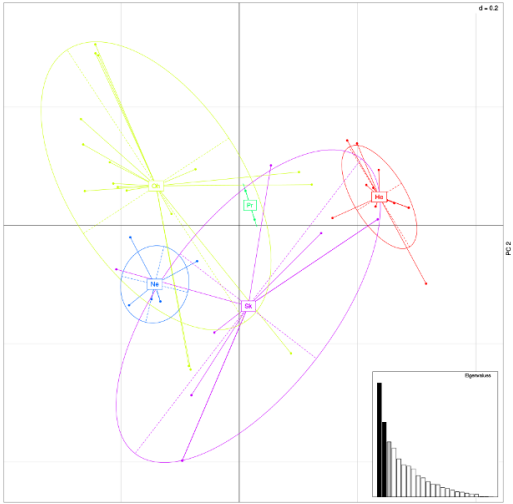
```
hauss.pcoa <- dudi.pco(d=dist(x=scaleGen(x=hauss.genind, center=TRUE,
  scale=FALSE, truenames=TRUE), method="euclidean"), scannf=FALSE, nf=3)
s.label(dfxy=hauss.pcoa$li, clabel=0.75) # Basic display
# To plot different axes use for example dfxy=hauss.pcoa$li[c(2, 3)]
s.kde2d(dfxy=hauss.pcoa$li,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

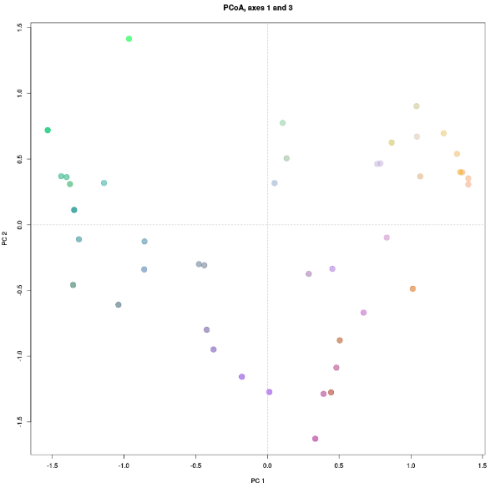
PCoA II



```r
# Based on Bruvo's distance
hauss.pcoa.bruvo <- dudi.pco(d=
  bruvo.dist(pop=hauss.genind,
  replen=rep(2, 12)),
  scannf=FALSE, nf=3)
s.class(dfxy=hauss.pcoa.bruvo$li,
  fac=pop(hauss.genind),
  col=hauss.pcoa.col)
add.scatter.eig(hauss.pcoa.bruvo$
  eig, posi="bottomright", 3,1,2)
# Another possibility for colored
# plot (see ?colorplot for details)
colorplot(xy=hauss.pcoa$li[c(1,2)],
  X=hauss.pcoa$li, transp=TRUE,
  cex=3, xlab="PC 1", ylab="PC 2")
title(main="PCoA, axes 1 and 2")
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

Special functions to work with huge SNP data sets

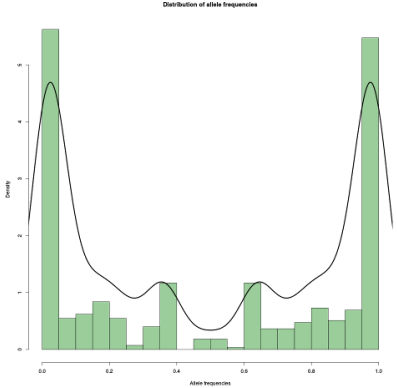
```
# Plot of missing data (white) and number of 2nd alleles
glPlot(x=usflu.genlight, legend=TRUE, posi="topleft")
# Sum of the number of second allele in each SNP
usflu.freq <- glSum(usflu.genlight)
# Plot distribution of (second) allele frequencies
hist(x=usflu.freq, proba=TRUE, col="gold", xlab="Allele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

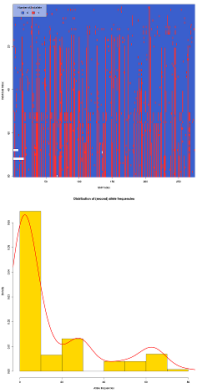
Number of missing values in each locus

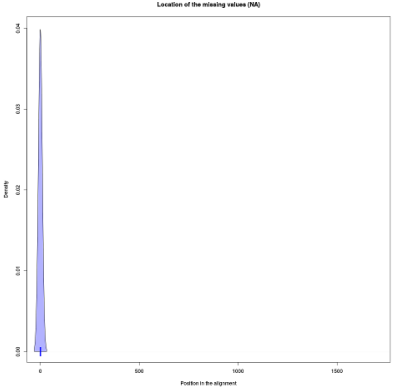
```
# Play with bw parameter to get optimal image
usflu.na.density <- density(glNA(usflu.genlight), bw=10)
# Set range of xlim parameter from 0 to the length of original alignment
plot(x=usflu.na.density, type="n", xlab="Position in the alignment",
  main="Location of the missing values (NA)", xlim=c(0,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

- Those tools are designed mainly for situation when having multiple (nearly) complete genomes — not needed for smaller 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





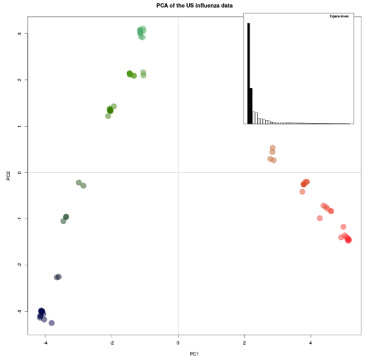


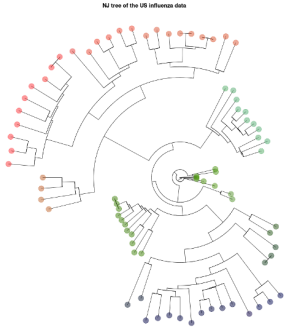
PCA, NJ and genlight objects

```
usflu.pca <- glPca(x=usflu.genlight, center=TRUE, scale=FALSE,
  loadings=TRUE) # Select number of retained PC axes, about 10 here
scatter.glPca(x=usflu.pca, posi="bottomright") # Plot PCA
title("PCA of the US influenza data")
# Loading plot - contribution of variables to the pattern observed
loadingplot.glPca(x=usflu.pca)
colorplot(usflu.pca$scores, usflu.pca$scores, transp=TRUE, cex=4) # Cols
title("PCA of the US influenza data")
abline(h=0, v=0, col="gray")
add.scatter.eig(usflu.pca[["eig"]][1:40], 2, 1, 2, posi="topright",
  inset=0.05, ratio=0.3)
usflu.tree.genlight <- nj(dist(as.matrix(usflu.genlight))) # Get the tree
# Plot colored phylogenetic tree
plot.phylo(x=usflu.tree.genlight, type="fan", show.tip=FALSE)
tiplabels(pch=20, col=num2col(usflu.annot[["year"]], col.pal=usflu.pal),
  cex=4)
title("NJ tree of the US influenza data")
```

PCA, NJ and genlight objects

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





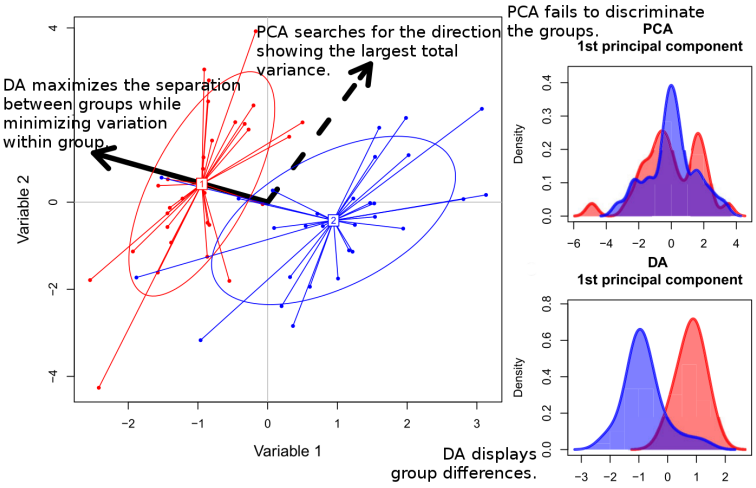
Discriminant Analysis of Principal components

DAPC

- Discriminant Analysis of Principal components (Jombart et al. 2010)
- Runs K-means Bayesian clustering on data transformed with PCA (reduces number of variables, speeds up process)
- Finally it runs discriminant analysis (DA) to maximize differences among groups
- Various modes of displaying of results — "Structure-like", "PCA-like" and more
- More information at `http://adegenet.r-forge.r-project.org/` and `adegenetTutorial("dapc")`
- If following commands would seem too complicated to you, try web interface by this command:

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

Principal difference between PCA and DA

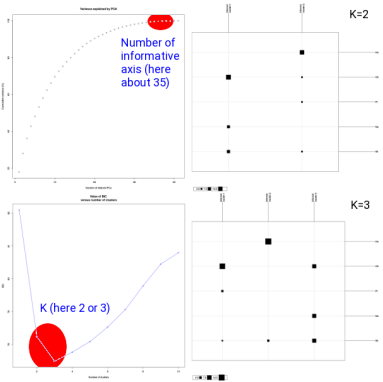


K-find — Bayesian K-means clustering

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

K-find outputs

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

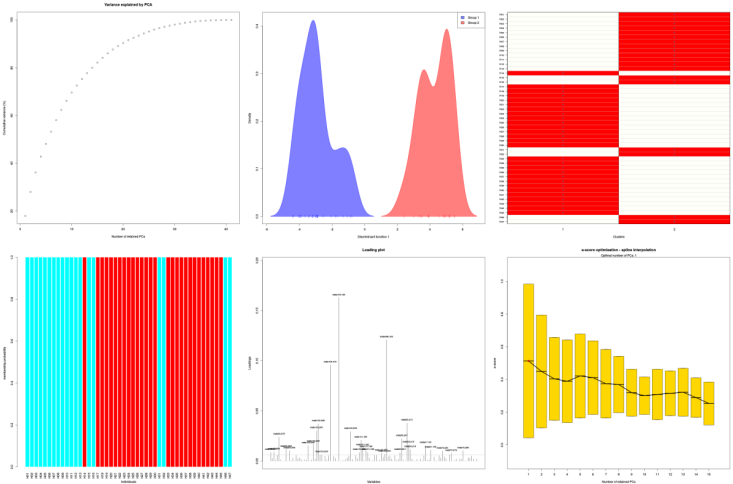



```
# See results as text
table(pop(hauss.genind),
  hauss.kfind3$grp)
hauss.kfind3
# Graph showing table of original
# and inferred populations and
# assignment of individuals
table.value(
  df=table(pop(hauss.
  genind), hauss.kfind3$grp),
  col.lab=paste("Inferred\n
  cluster",
  1:length(hauss.kfind3$size)),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

DAPC code I

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

DAPC for K=2

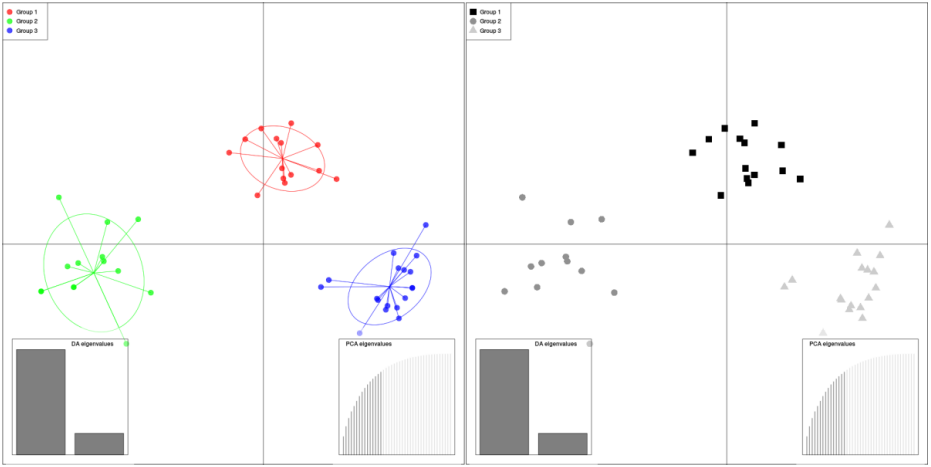

Variance explained by PCA
Cumulative variance (%)
Number of retained PCs
Group 1
Group 2
Density
Discriminant function 1
Clusters
Membership probability
Individuals
Loading plot
Loadings
Variables
a-score optimization - spline interpolation
Optimal number of PCs
a-score
Number of retained PCs

DAPC code II

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

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

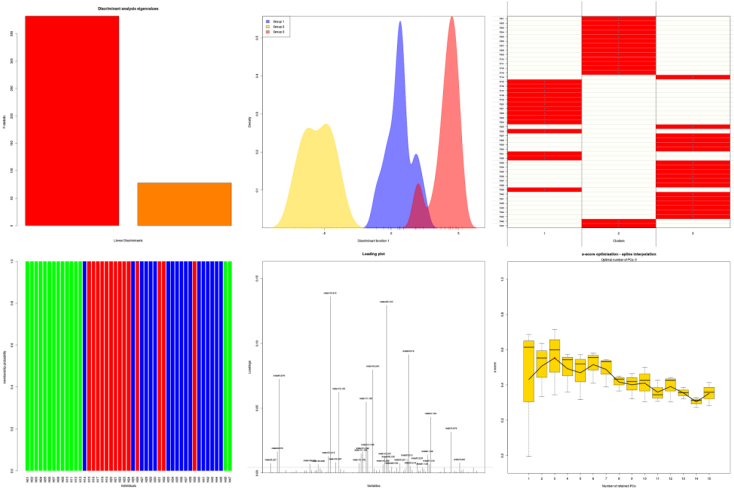
DAPC for K=3


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

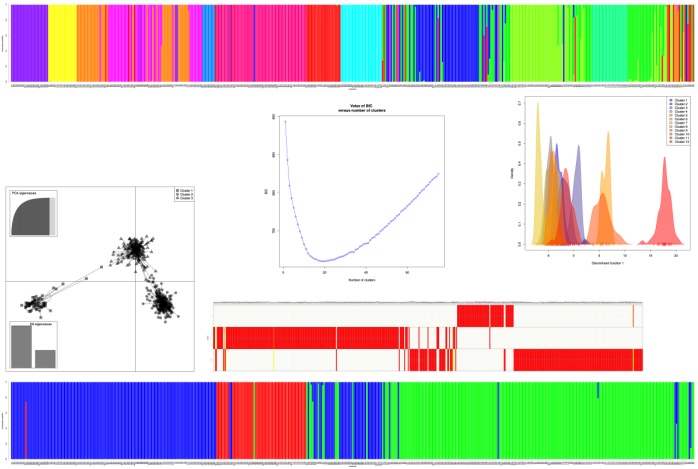
DAPC code III

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

DAPC for K=3, extra information



Another DAPC example



Bayesian clustering with Structure

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`)

Structure work flow

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

Running Structure in parallel with R

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

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

Preparing for ParallelStructure

Within working "structure" directory you need

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

Input files

Joblist file:

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

Input file:

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

Running ParallelStructure

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

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

ParallelStructure and Windows

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

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

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

Post process Structure results — select the best K

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

```
# Load the script
source("https://soubory.trapa.cz/rcourse/structure-sum-2011.r")
# Create new directory with result files results_job_*_f and set
# working directory accordingly
setwd("/home/vojta/dokumenty/vyuka/r_mol_data/examples/structure
  /structure_sum/")
```

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

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

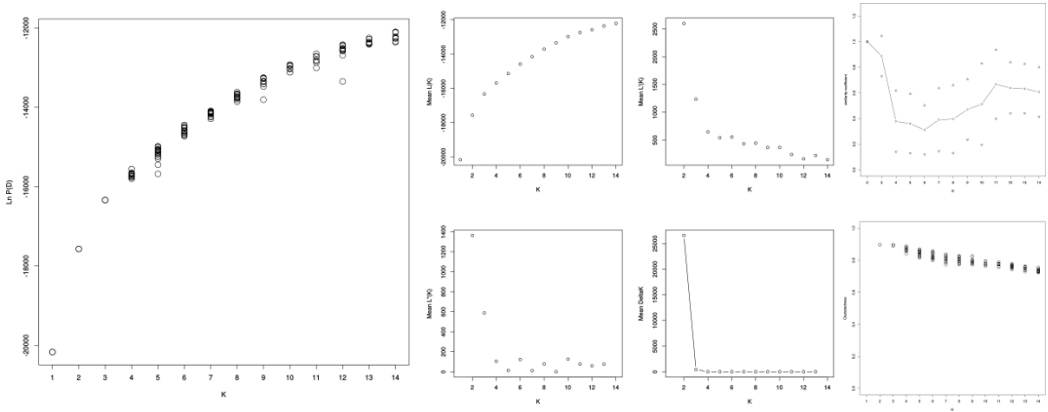
Run Structure-sum

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

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

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

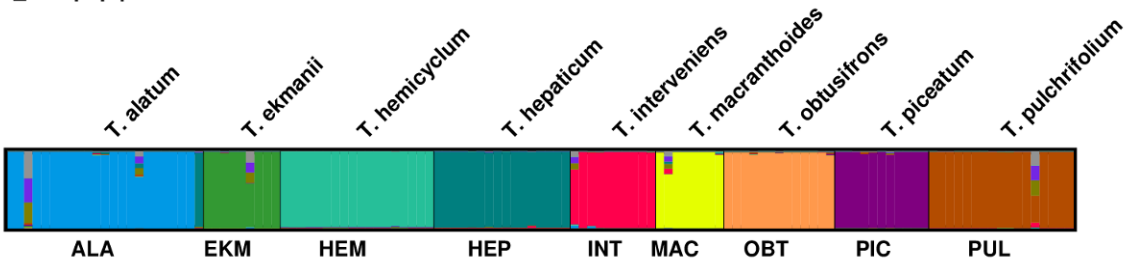
Results from different data set, not from our toy



Example of final Structure plot

Drawn by distruct after alignment of Structure outputs by CLUMPP

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



Spatial analysis and genetic data

Correlation of genes and space, spatial structure of genotypes

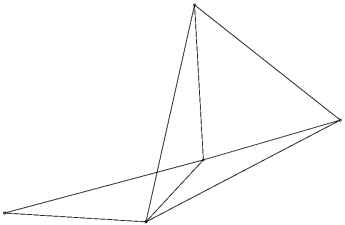
Short overview of spatial genetics (in R)

Basic approaches

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

Moran's *I*

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



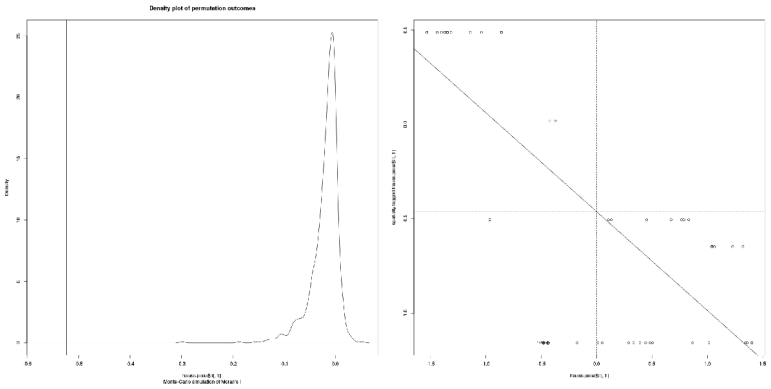
Calculation of Moran's *I* I

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

Calculation of Moran's *I* II

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

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

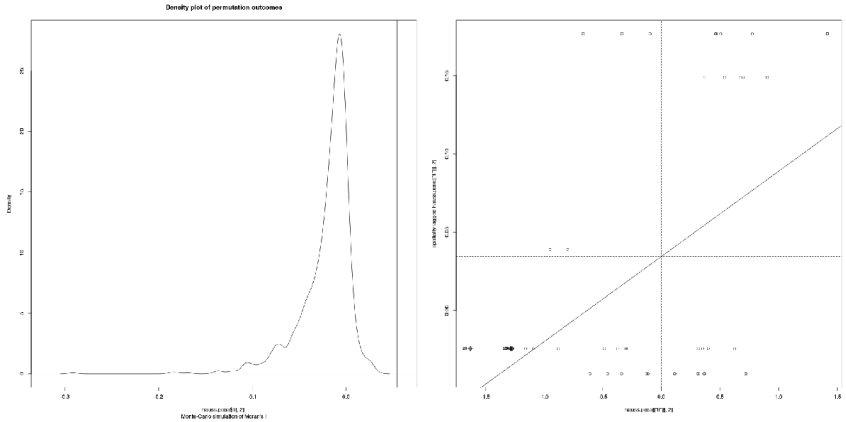


- Tested hypothesis "greater" — **no** significant **positive** autocorrelation
- If testing for hypothesis "less" — significant **negative** autocorrelation — individuals are significantly different

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

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

Second axis is surprisingly significant



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

Spatial Analysis of Principal Components (sPCA)

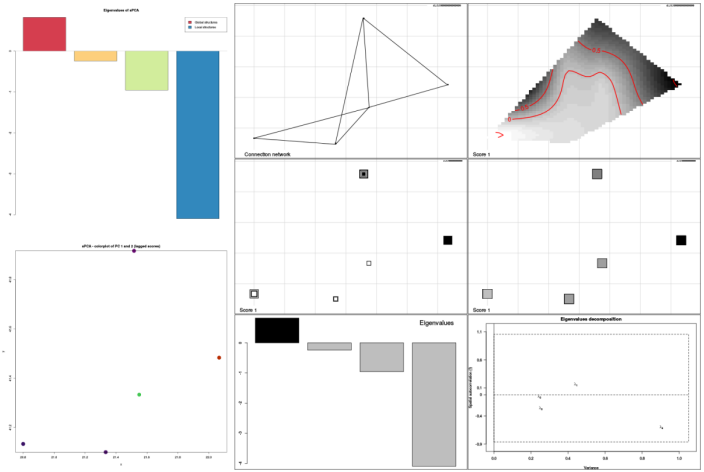
- Implemented in adegenet, see `adegenetTutorial("spca")`
- Analyzes matrix of relative allele frequencies of genotypes/populations and spatial weighting matrix
- The geographical matrix is usually (as for Moran's *I*) created by `chooseCN()` — creates connectivity network among entities (genotypes/populations)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

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

Calculations of sPCA

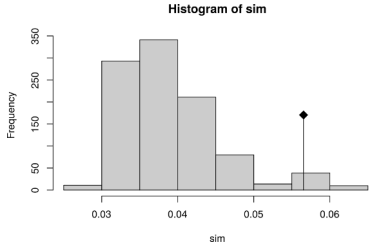
```
hauss.spca <- spca(obj=hauss.genind, cn=hauss.connectivity,
  scale=TRUE, scannf=TRUE)
# Plot eigenvalues of sPCA - global vs. local structure
barplot(height=hauss.spca$eig, main="Eigenvalues of sPCA",
  col=spectral(length(hauss.spca$eig)))
legend("topright", fill=spectral(2), leg=c("Global structures",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

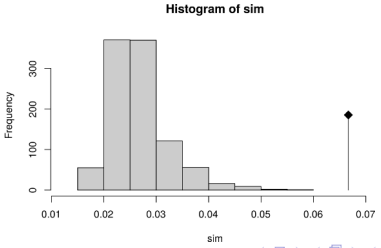
sPCA outputs I



Test if global/local structure is significant

```
hauss.spca.glo <- global.rtest(X=hauss.genind$tab, listw=hauss.spca$lw,
  nperm=999)
hauss.spca.glo
plot(hauss.spca.glo)
hauss.spca.loc <- local.rtest(X=hauss.genind$tab, listw=hauss.spca$lw,
  nperm=999)
hauss.spca.loc
plot(hauss.spca.loc)
```



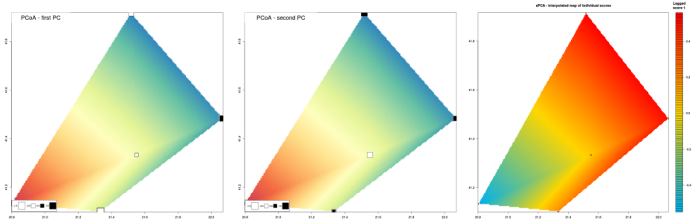


Map of genetic clines

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

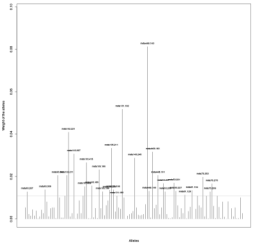
sPCA outputs II

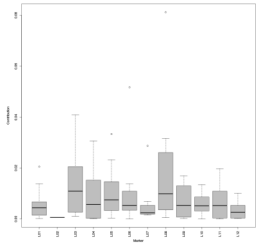
```
# Interpolated lagged score on a map
hauss.spca.annot <- function() {
  title("sPCA - interpolated map of individual scores")
  points(other(hauss.genind)$xy[,1], other(hauss.genind)$xy[,2])
  }
filled.contour(hauss.spca.temp, color.pal=colorRampPalette(
  lightseasun(100)), pch=20, nlevels=100,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



Loading plots — which alleles contribute the most?

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



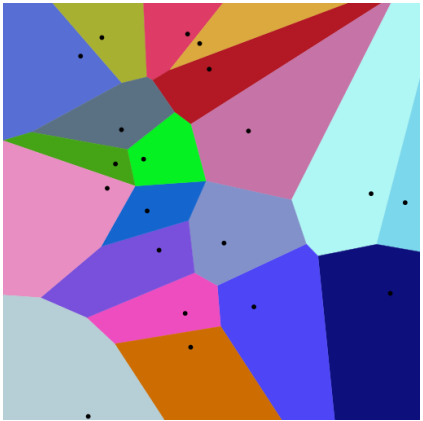


Monmonier's algorithm — genetic boundaries

- 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

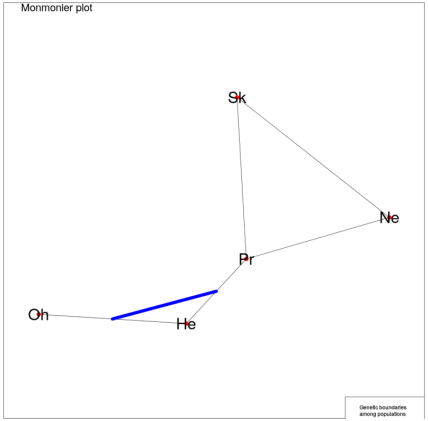
```
# Calculates Monmonier's function (for threshold use 'd')
hauss.monmonier <- monmonier(xy=hauss.genpop$other$xy, dist=
  dist(hauss.genpop@tab), cn=chooseCN(hauss.genpop$other$xy,
  ask=FALSE, type=2, plot.nb=FALSE, edit.nb=FALSE), nrun=1)
coords.monmonier(hauss.monmonier) # See result as text
```

Voronoi tessellation



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

Plot genetic boundaries




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

Monmonier notes

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

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

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

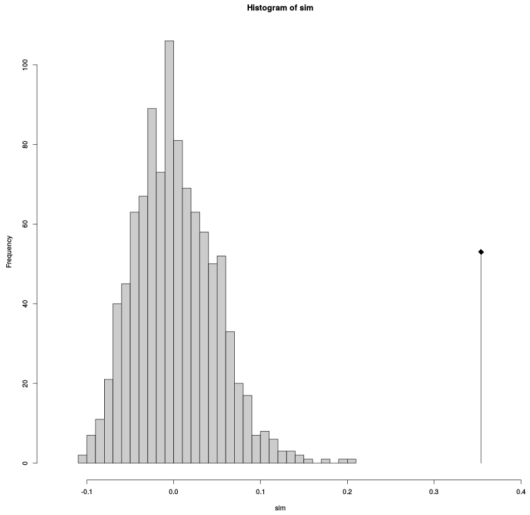
Mantel test

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

Mantel test — isolation by distance

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

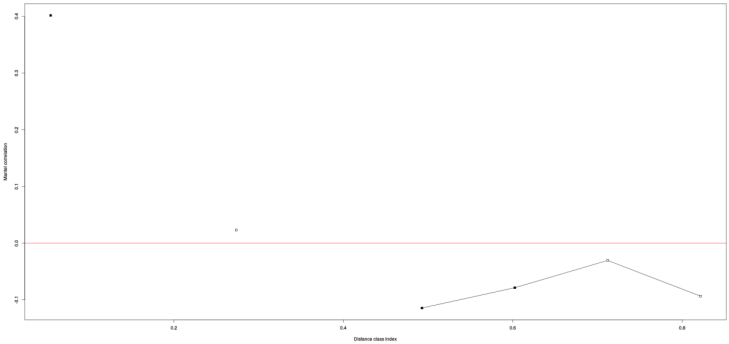
Mantel test outputs — strongly significant




```
hauss.mantel # See output
Monte-Carlo test
Call: mantel.randtest(m1 =
  hauss.dist, m2 =
  hauss.gdist, nrepet = 1000)
Observation: 0.35409
Based on 1000 replicates
Simulated p-value: 0.000999001
Alternative hypothesis: greater
  Std.Obs Expectation  Variance
7.61967545 0.001687140 0.0021389
```

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

Plot of Mantel Correlogram Analysis



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

Mantel correlogram — text output

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

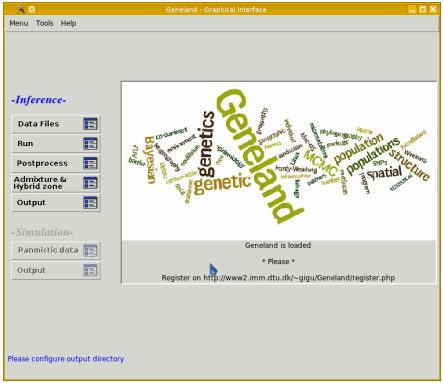
About Geneland

- For installation see slide 53
- 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/

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

Geneland GUI

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



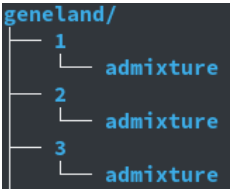
Loading and conversions of coordinates

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

Before running MCMC

- Monte Carlo Markov Chains (MCMC) require usually millions of generations (iterations, `nit`) to find optimal solution
- Beginning (∼10–20%) of the steps (`burnin`) use to be very unstable and useless for following 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 $1000–10000^{th}$ generation
- Denser sampling produces smoother data, but can consume too much disk space...

Directory structure for Geneland:



Settings and running MCMC

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

Running MCMC

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

Estimate F_{ST}

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

MCMC inference under the admixture model

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

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

Produce maps of respective inferred clusters

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

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

Estimate frequencies of null alleles

```
hauss.geneland.fna <- list()
for (hauss.geneland.irun in 1:hauss.geneland.nrun) {
  hauss.geneland.path.mcmc <- paste("geneland/", hauss.geneland.irun,
    "/", sep="")
  # Estimation
  hauss.geneland.fna[[hauss.geneland.irun]] <-
    EstimateFreqNA(path.mcmc=hauss.geneland.path.mcmc)
  }
# See output
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

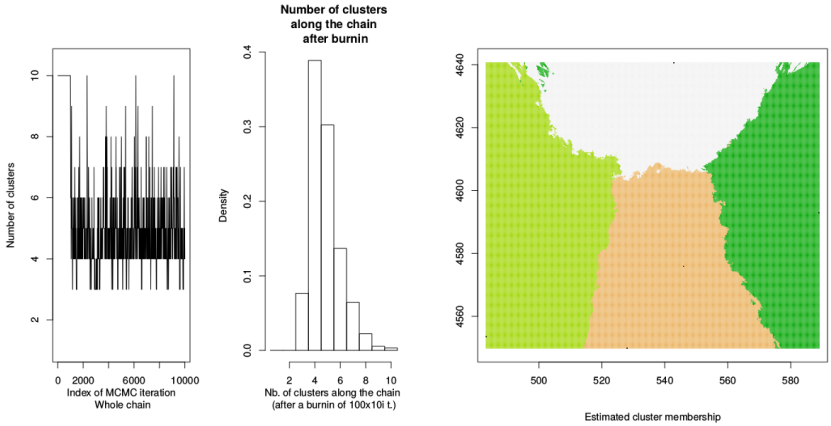
Determine which run is the best

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

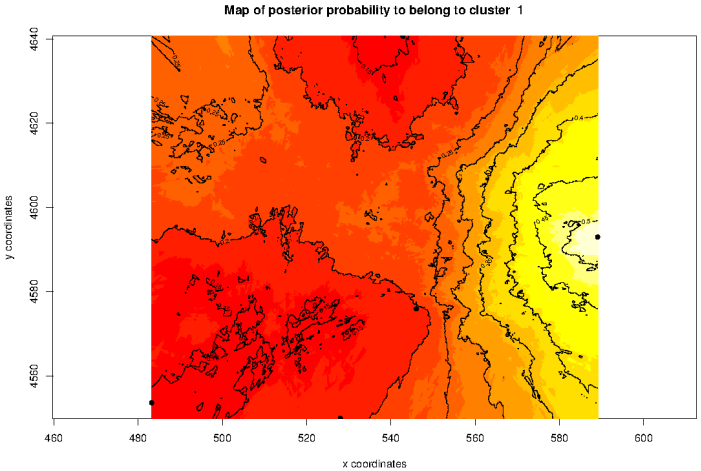
- We will use figures and F_{ST} outputs only from the best run
- It is useful to keep all runs especially for comparison if there are different solutions with similar posterior probability

MCMC chain, number of clusters and their map

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



Map of posterior probability of belonging into cluster 1



When using Geneland, remember...

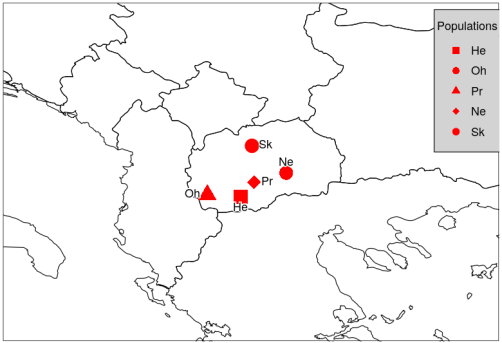
- Within `MCMC()`, there must be at least hundreds thousands or millions of generations (`nit`) and appropriate sampling (thousands or higher, `thinning` — not to fill whole disk)
- To analyze geo-referenced data with a non-spatial prior set in `MCMC()` `spatial=FALSE`
- To analyze non-spatial data remove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

Very basic mapping in R

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

Basic map

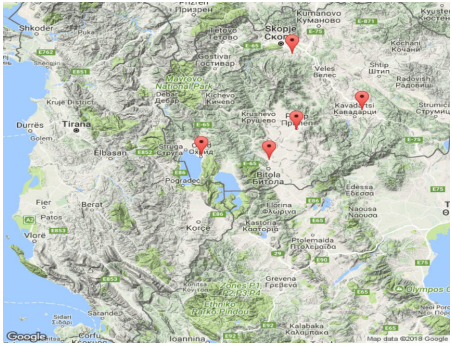
```
# Insert legend
legend(x="topright", inset=1/50, legend=c("He", "Oh", "Pr", "Ne", "Sk"),
  col="red", border="black", pch=15:19, pt.cex=2, bty="o", bg="lightgrey",
  box.lwd=1.5, cex=1.5, title="Populations")
```



Google map

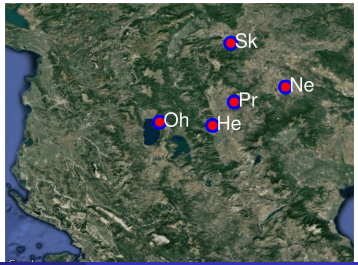
```
# Google map is produced into a
# file. Parameter markers contain
# data frame with coordinates and
# possibly with more information
hauss.gmap <- GetMap(center=c(lat=
  41, lon=21), size=c(640, 640),
  destfile="gmap.png", zoom=8,
  markers=hauss.coord, maptype=
  "terrain", API_console_key="XXX")
# Plot the map
PlotOnStaticMap(MyMap=hauss.gmap)
```

- Google recently started to require API key, it doesn't work without it...
- See documentation and more options, it can use plenty of map resources (Bing maps, ...)
- Many on-line map services do require (paid) API key...



More options for Google map

```
hauss.gmap2 <- GetMap(center=c(
  lat=41, lon=21), size=c(640,
  640), destfile="gmap2.png",
  zoom=8, maptype="satellite",
  API_console_key="XXX")
PlotOnStaticMap(MyMap=hauss.gmap2,
  lat=hauss.genpop@other$xy[["lat"
  ]], lon=hauss.genpop@other$xy
  [["lon"]], FUN=points, pch=19,
  col="blue", cex=5)
PlotOnStaticMap(MyMap=hauss.gmap2,
  lat=hauss.genpop@other$xy[["lat"
  ]], lon=hauss.genpop@other$xy
  [["lon"]], add=TRUE, FUN=points,
  pch=19, col="red", cex=3)
PlotOnStaticMap(MyMap=hauss.gmap2,
  lat=hauss.genpop@other$xy[["lat"
  ]], lon=hauss.genpop@other$xy
  [["lon"]], add=TRUE, FUN=text,
  labels=as.vector(popNames(
  hauss.genind)), pos=4, cex=3,
  col="white")
# Google maps have their own
# internal scaling, adding of
# points by standard functions
# will not work correctly
```

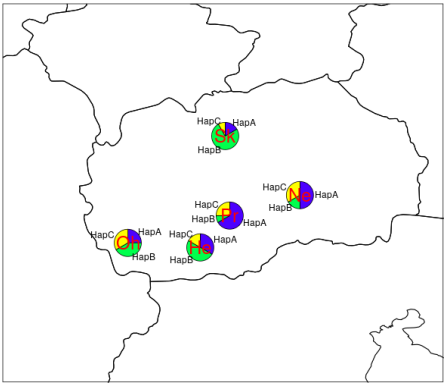


Adding pie charts to map I

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

Adding pie charts to map II

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

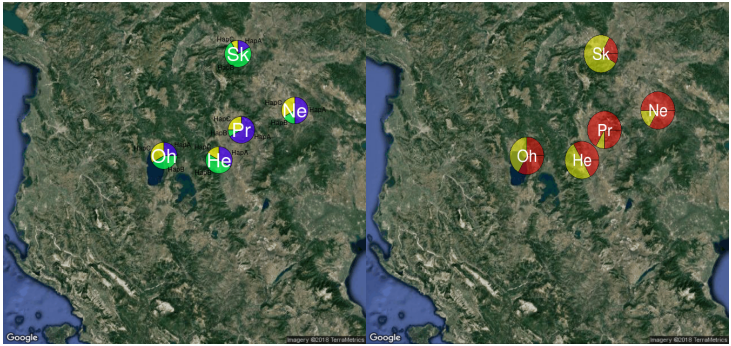


Pie charts on Google map I

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

Pie charts on Google map II

```
# Add population text labels
PlotOnStaticMap(MyMap=hauss.gmap2, lat=hauss.genpop@other$xy[["lat"]],
  lon=hauss.genpop@other$xy[["lon"]], add=TRUE, FUN=text,
  labels=as.vector(popNames(hauss.genind)), cex=2.5, col="white")
```



Datasets from mapproj

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

Plotting on SHP files I

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

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

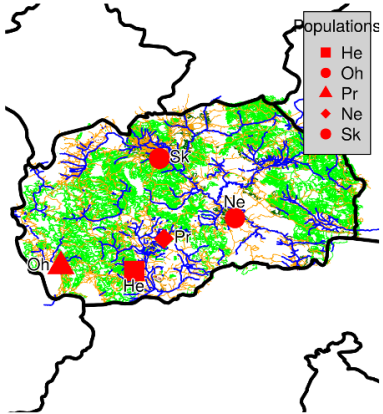
Plotting on SHP files II

```
macedonia_landuse <- readShapeLines(fn="macedonia_landuse.shp")
plot(macedonia_landuse)
macedonia_natural <- readShapeLines(fn="macedonia_natural.shp")
plot(macedonia_natural)
macedonia_railways <- readShapeLines(fn="macedonia_railways.shp")
plot(macedonia_railways)
macedonia_roads <- readShapeLines(fn="macedonia_roads.shp")
plot(macedonia_roads)
macedonia_waterways <- readShapeLines(fn="macedonia_waterways.shp")
plot(macedonia_waterways)
# Plot all layers into single image, add more information
plot(macedonia_building)
plot(macedonia_landuse, add=TRUE, col="darkgreen", fill=TRUE)
plot(macedonia_natural, add=TRUE, col="green", fill=TRUE)
plot(macedonia_railways, add=TRUE, col="brown", lty="dotted")
plot(macedonia_roads, add=TRUE, col="orange")
plot(macedonia_waterways, add=TRUE, col="blue", lwd=2) # Ends on next slide
```

Plotting on SHP files III

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

Plotting on SHP files IV and maps from mapproj



Manipulation, display and analysis of sets of trees

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

Read and write tree and drop tips

```
# Read trees in NEWICK format - single or multiple tree(s)
oxalis.trees<-read.tree(file="https://soubory.trapa.cz/rcourse/oxalis.nwk")
summary(oxalis.trees)
length(oxalis.trees)
names(oxalis.trees)
# Export trees in NEWICK format
write.tree(phy=oxalis.trees, file="trees.nwk")
oxalis.trees[[1]][["tip.label"]] # See tip labels
# Drop a tip from multiPhylo
plot.multiPhylo(x=oxalis.trees)
oxalis.trees.drop<-lapply(X=oxalis.trees,FUN=drop.tip,tip="O._callosa_S15")
class(oxalis.trees.drop) <- "multiPhylo"
plot.multiPhylo(x=oxalis.trees.drop)
plot.phylo(hauss.nj) # Drop a tip from single tree
hauss.nj[["tip.label"]] # See tip labels (and numbers)
hauss.nj.drop <- drop.tip(phy=hauss.nj, tip=47)
plot.phylo(hauss.nj.drop)
```

Extract clades from trees and drop extinct tips

```
# Interactively extract tree
plot.phylo(hauss.nj) # Plot source tree
nodelabels() # See node labels (numbers) - needed for some tasks
# Select clade to extract by clicking on it
hauss.nj.extracted <- extract.clade(phy=hauss.nj, interactive=TRUE)
# See new extracted tree
plot.phylo(hauss.nj.extracted)
# Non-interactively extract tree
hauss.nj.extracted <- extract.clade(phy=hauss.nj,node=60,interactive=FALSE)
# See new extracted tree
plot.phylo(hauss.nj.extracted)
# Drop "extinct" tips - those who don't reach end the tree tolerance is
# respective to the used metrics
plot.phylo(hauss.nj)
axisPhylo()
hauss.nj.fossil <- drop.fossil(phy=hauss.nj, tol=0.4)
plot.phylo(hauss.nj.fossil)
```

Join two trees, rotate tree

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

Ladderize and (un)root the tree

```
plot.phylo(hauss.nj) # Ladderize the tree
hauss.nj.ladderized <- ladderize(hauss.nj)
plot.phylo(hauss.nj.ladderized)
# Root the tree
plot.phylo(hauss.nj)
print.phylo(hauss.nj)
# resolve.root=TRUE ensures root will be bifurcating (needed here)
# (without this parameter it sometimes doesn't work)
hauss.nj.rooted <- root.phylo(phy=hauss.nj, resolve.root=TRUE, outgroup=10)
print.phylo(hauss.nj.rooted)
plot.phylo(hauss.nj.rooted)
# Root the tree interactive
plot.phylo(hauss.nj)
hauss.nj.rooted <- root.phylo(phy=hauss.nj, interactive=TRUE)
plot.phylo(hauss.nj.rooted)
unroot.phylo() # Unroot the tree
is.rooted() # Check if it is rooted
```

Check tree and compute branch lengths and times

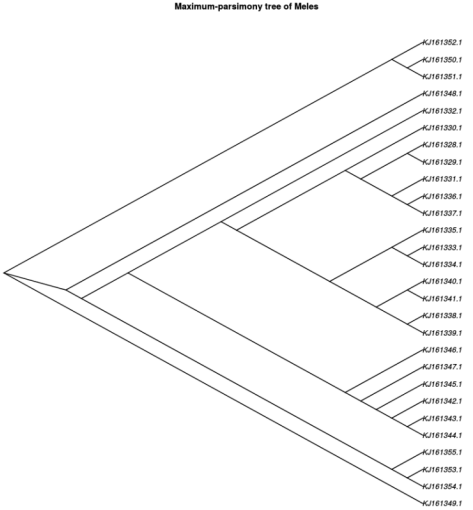
```
# Check if the tree is ultrametric - is variance of distances of all tips
# to node 0? It is required for some analysis
is.ultrametric()
# Make tree ultrametric
?chronos # Check it for mode how to calculate the lengths
# chronos has more uses - it is mainly used for dating
# Compute branch lengths for trees without branch lengths
?compute.brlen # Check it for mode how to calculate the lengths
# Computes the branch lengths of a tree giving its branching
# times (aka node ages or heights)
?compute.brtime # Check it for mode how to calculate the lengths
```

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

Maximum parsimony — theory

- Maximum parsimony finds optimal topology of the phylogenetic tree by minimizing of the total number of character-state changes
- It minimizes homoplasy (convergent evolution, parallel evolution, evolutionary reversals)
- Very simple criterion, easy to score the tree, but not to find it — exhaustive search to explore all possible trees is realistic until ∼9 taxa, branch-and-bound swapping (guaranteeing finding the best tree) until ∼20 taxa, for more heuristic search is needed — it doesn't always guarantee to find the most probable tree
- To speed up calculations, initial tree (usually NJ — slide 167) is used to start the search
- With rising performance of computers, it uses to be replaced maximum likelihood or Bayesian methods

Maximum parsimony — code and result




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

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 became 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

Topographical distances among trees II

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

```
library(gplots)
library(corrplot)
library(phytools)
# Compute matrix of topological distances among phylogenetic trees
?dist.topo # See details of available computing methods
oxalis.trees.d <- dist.topo(x=oxalis.trees, method="score")
# Basic information about the distance matrix
dim(as.matrix(oxalis.trees.d))
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...

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

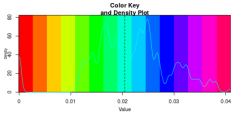
Topographical distances among trees IV

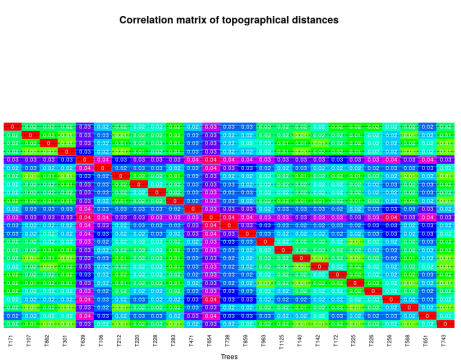
Calculate Robinsons-Foulds distance matrix among trees and plot it

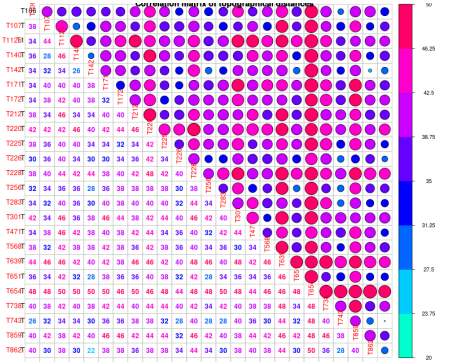
- `phytools::multiRF` can handle `multiPhylo` objects and directly create matrices (no need to create loops)

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

Topographical distances among trees V — the matrices





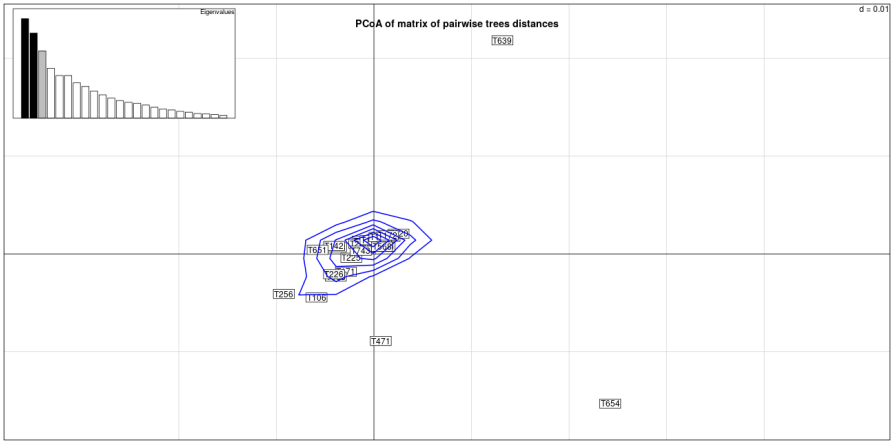


PCoA from distance matrices of topographical differences among trees — the code

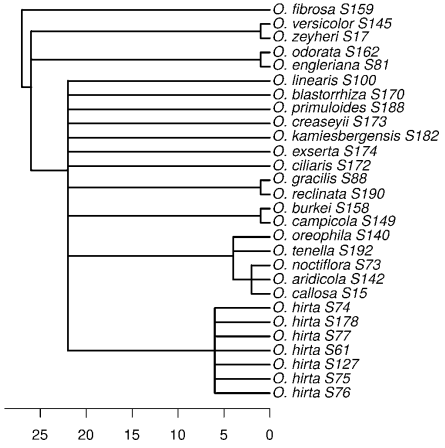
PC plots help to identify outliers — trees with noticeably different topology

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

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



Consensus tree




```
# Root all trees
oxalis.trees.rooted <-
  root.multiPhylo(phy=oxalis.trees,
  outgroup="O._fibrosa_S159",
  resolve.root=TRUE)
# Consensus tree (50 % rule)
oxalis.tree.con <- consensus
  (oxalis.trees.rooted, p=0.5,
  check.labels=TRUE)
print.phylo(oxalis.tree.con)
# Plot the tree
plot.phylo(oxalis.tree.con,
  edge.width=2, label.offset=0.3)
axisPhylo(side=1)
# What a nice tree... :-P
```

kdetrees — identification of outlying trees I

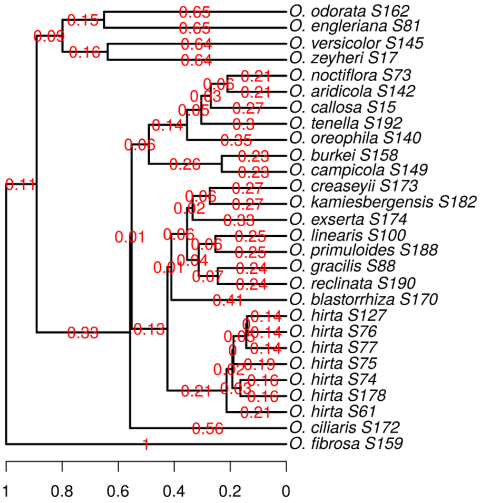
- Distance-based method of identification of trees with significantly different topology
- Function `kdetrees` has plenty of options...
- Parameter `k` sets threshold for trees to be removed — it requires repeated running with different `k` (and plotting the figures) to decide which trees to remove and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

```
# Load library
library(kdetrees)
?kdetrees # See options
# Run main function - play with parameter k
oxalis.kde <- kdetrees(trees=oxalis.trees, k=0.4, distance="dissimilarity",
  topo.only=FALSE, greedy=TRUE)
# See results
oxalis.kde
plot(oxalis.kde)
hist(oxalis.kde)
```

kdetrees — identification of outlying trees II

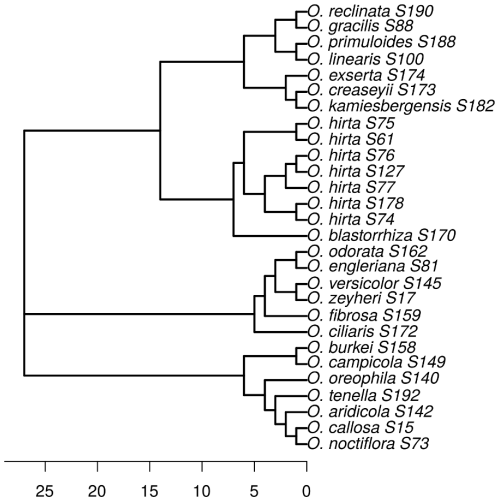
```
# See removed trees
plot.multiPhylo(oxalis.kde[["outliers"]])
# Save removed trees
write.tree(phy=oxalis.kde[["outliers"]], file="oxalis_trees_outliers.nwk")
# Save kdetrees report
write.table(x=as.data.frame(x=oxalis.kde), file="oxalis_trees_scores.tsv",
  quote=FALSE, sep="\t")
# Extract passing trees
oxalis.trees.good <- oxalis.trees[names(oxalis.trees) %in%
  names(oxalis.kde[["outliers"]]) == FALSE]
oxalis.trees.good
# Save passing trees
write.tree(phy=oxalis.trees.good, file="trees_good.nwk")
```

Species tree — all trees must be ultrametric




```
# Chronos scale trees
oxalis.trees.ultra <- lapply
  (X=oxalis.trees.rooted,
  FUN=chronos, model="correlated")
class(oxalis.trees.ultra) <-
  "multiPhylo"
# Mean distances
oxalis.tree.sp.mean <- speciesTree
  (oxalis.trees.ultra, mean)
# Plot the tree
plot.phylo(oxalis.tree.sp.mean,
  edge.width=2, label.offset=0,01)
edgelabels(text=round(oxalis.
  tree.sp.mean[["edge.length"]],
  digits=2), frame="none",
  col="red", bg="none")
axisPhylo(side=1)
```

Parsimony super tree

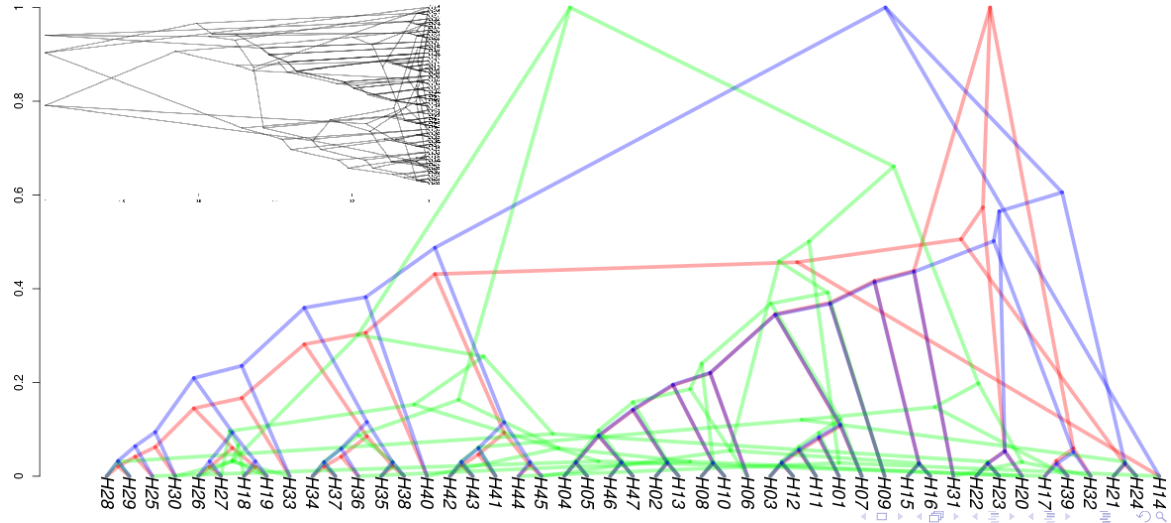



```
library(phytools)
class(oxalis.trees.rooted) <-
  "multiPhylo"
oxalis.tree.sp <- mrp.supertree
  (tree=oxalis.trees.rooted,
  method="optim.parsimony",
  rooted=TRUE)
print.phylo(oxalis.tree.sp)
plot.phylo(oxalis.tree.sp,
  edge.width=2, label.offset=0.01)
axisPhylo(side=1)
# Similar function
?phangorn::superTree
# Coalescence model to handle
# multiple individuals per species
?phangorn::coalSpeciesTree
```

Density tree I

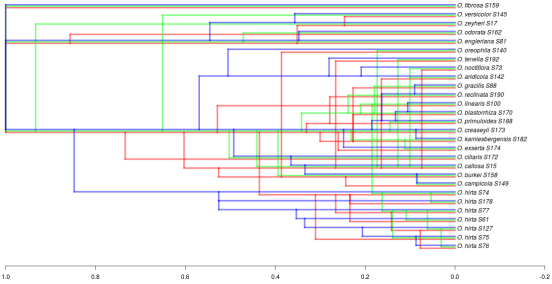
```
# Prepare list of trees to show
hauss.nj.trees <- list(hauss.nj, hauss.nj.bruvo, hauss.nj.rooted)
hauss.nj.trees <- lapply(X=hauss.nj.trees, FUN=compute.brlen)
hauss.nj.trees <- lapply(X=hauss.nj.trees, FUN=chronos)
class(hauss.nj.trees) <- "multiPhylo"
# The trees should be (otherwise plotting works,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

Density tree II



Density tree III

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



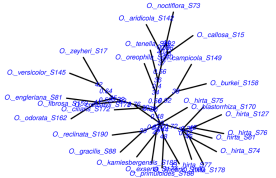
Kronoviz — see all trees on same scale

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

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

Networks

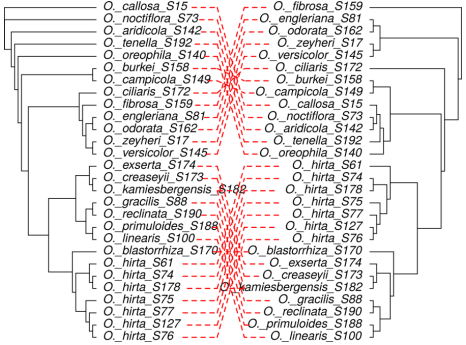
```
library(phangorn)
oxalis.tree.net <- consensusNet(oxalis.trees.rooted, prob=0.25)
plot(x=oxalis.tree.net, planar=FALSE, type="2D", use.edge.length=TRUE,
  show.tip.label=TRUE, show.edge.label=TRUE, show.node.label=TRUE,
  show.nodes=TRUE, edge.color="black", tip.color="blue") # 2D
plot(x=oxalis.tree.net,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



Compare two trees

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

Cophyloplot comparing two trees

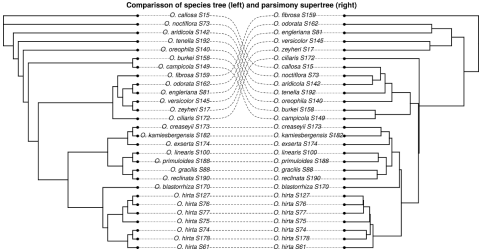


- `ladderize()` pre-sorts tips in the tree — it can help to `cophyloplot()` to reduce crossings

- `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

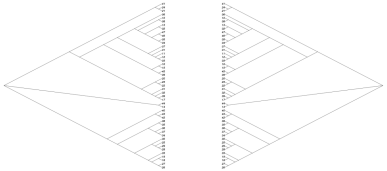
```
?cophylo # See options
# Prepare the object for plotting
oxalis.cophylo <- cophylo(tr1=oxalis.tree.sp, tr2=oxalis.tree.sp.mean,
  assoc=(cbind(sort(oxalis.tree.sp$tip.label),
  sort(oxalis.tree.sp$tip.label))), rotate=TRUE)
plot.cophylo(x=oxalis.cophylo, lwd=2, link.type="curved") # Plot it
title("Comparison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



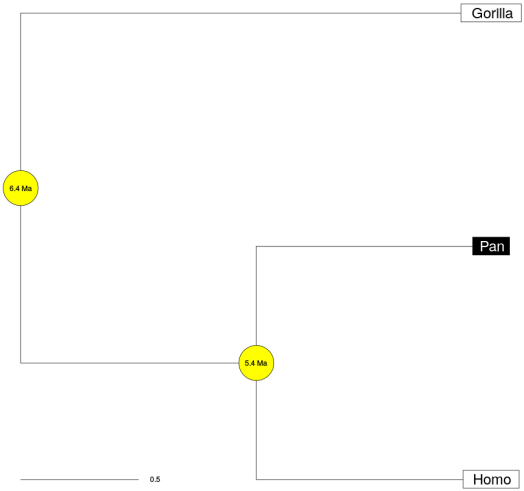
Change orientation of plots

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

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



Highlighted labels




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

Reconstruction of evolution of traits

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

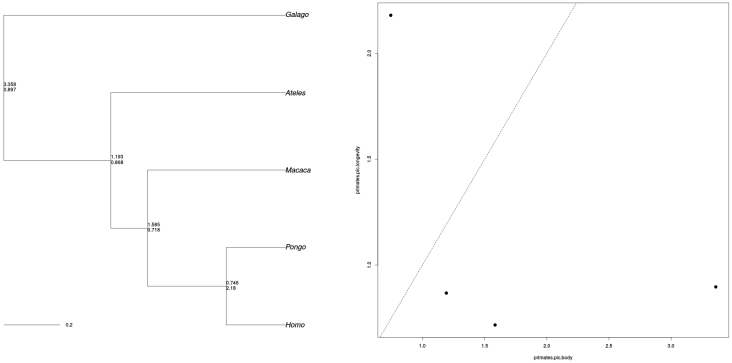
```
# Prepare the data # Body mass of primates
primates.body <- c(4.09434, 3.61092, 2.37024, 2.02815, 1.46968)
# Longevity of primates
primates.longevity <- c(4.74493, 3.3322, 3.3673, 2.89037, 2.30259)
# Add names to the values
names(primates.body) <- names(primates.longevity) <- c("Homo", "Pongo",
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

PIC and its plotting

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

Plot of PIC (on the tree)

```
# Testing if there is positive correlation
cor.test(x=primates.pic.body, y=primates.pic.longevity,
  alternative="greater", method="pearson")
```



Test it

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

Intraspecific variation

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

```
# Preparing tips with fake random variable values
primates.pic.var <- list(cbind(primates.body, jitter(primates.body),
  jitter(primates.body))[1,], cbind(primates.body, jitter(primates.
  body), jitter(primates.body))[2,], cbind(primates.body, jitter(
  primates.body), jitter(primates.body))[3,], cbind(primates.body,
  jitter(primates.body), jitter(primates.body))[4,], cbind(
  primates.body, jitter(primates.body), jitter(primates.body))[5,])
```

Explanation of the cbind trick

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

Calculation of phylogenetic independent contrast with intraspecific variation

```
# Check list of numeric vectors - required as input for pic.ortho()
primates.pic.var
primates.pic.var[[1]]
class(primates.pic.var)
class(primates.pic.var[[1]])
# Calculate for one character - this must be done for at least one more
# character and correlation of PICs must be tested
primates.pic.ortho <- pic.ortho(x=primates.pic.var, phy=primates.tree,
  var.contrasts=FALSE, intra=FALSE)
primates.pic.var
```

- Resulting vector (here `primates.pic.ortho`) can then be used in absolutely same way as output of classical `pic()` — testing by `lmorigin()` (slide 305) etc.

Phylogenetic autocorrelation

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

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

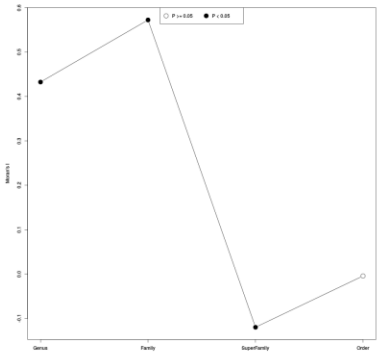
Testing of Moran's *I*

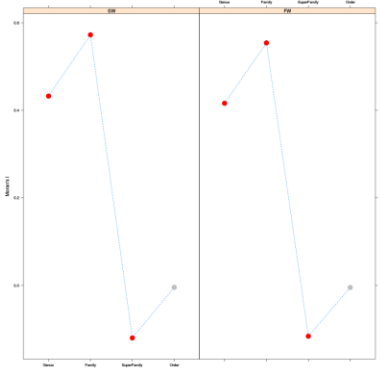
```
# Calculate Moran's I
# Slightly significant positive phylogenetic correlation among body mass
Moran.I(x=primates.body, weight=primates.weights, alternative="greater")
# Positive, but non-significant
Moran.I(x=primates.longevity, weight=primates.weights,
  alternative="greater")
# Test of Moran's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

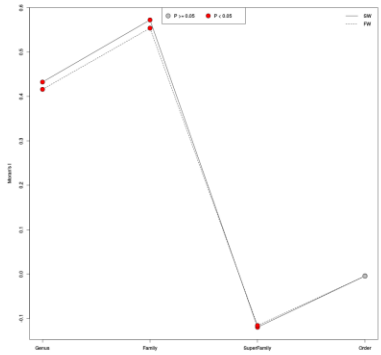
Correlogram to visualize results of phylogenetic autocorrelation analysis

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



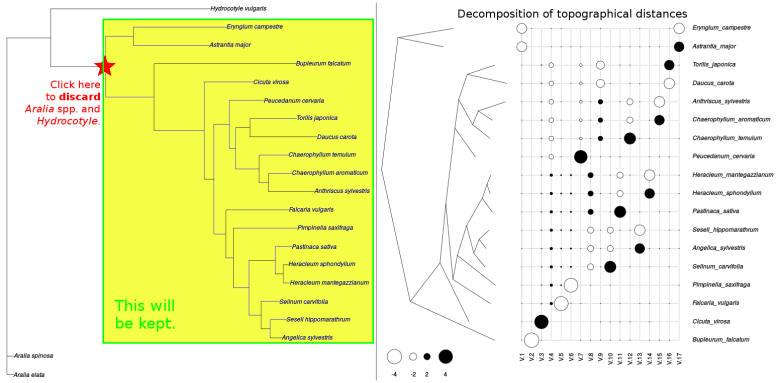




Prepare toy data set (tree)

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

Modified tree




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

Prepare toy data set (the variable)

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

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

Orthonormal decomposition - phylogenetic eigenvector regression

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

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

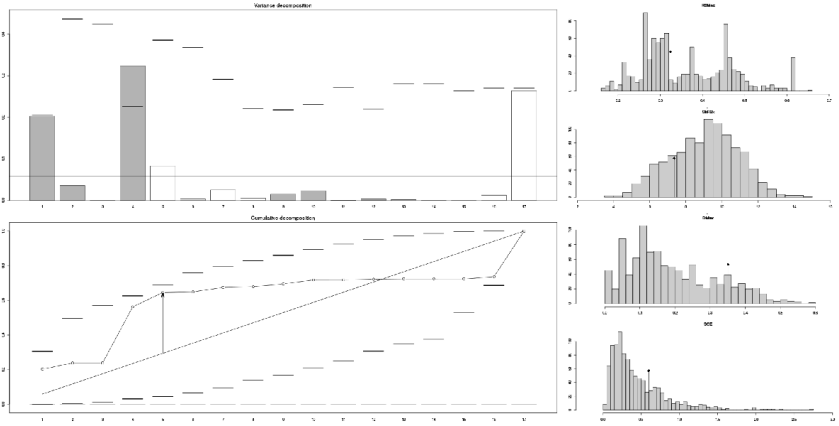
```
              Df   Sum Sq  Mean Sq F value Pr(>F)
as.matrix...   2 0.063689 0.031845  2.2275 0.1422
Residuals     15 0.214443 0.014296
```

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

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

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

Orthogram



- Observed value is within permutations — no significant inertia of the trait to phylogeny...

Phylogenetic Generalized Least Squares

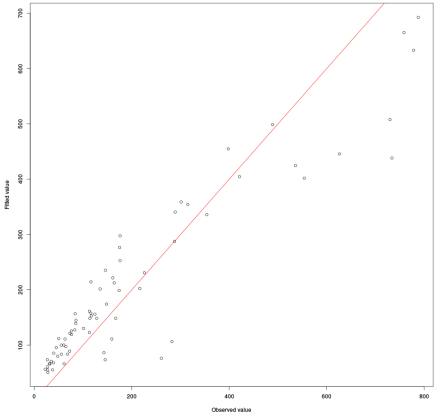
- Model-based testing if there is significant correlation between two traits (after removing the phylogenetic component)
- `nlme::gls` fits a linear model using generalized least squares
- Functions `corBlomberg`, `corBrownian`, `corMartins` and `corPagel` from `ape` package create correlation matrix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

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

Implementation in caper package

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

Results of PGLS



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

Generalized Estimating Equations

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

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

Not significant in this case...

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

Phylogenetic signal

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

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

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

Analyze multiple traits in once

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

When there are vectors with standard errors of measurements

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

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

Alternative testing for phylogenetic signal with GLM

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

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

Phylogenetic principal component analysis

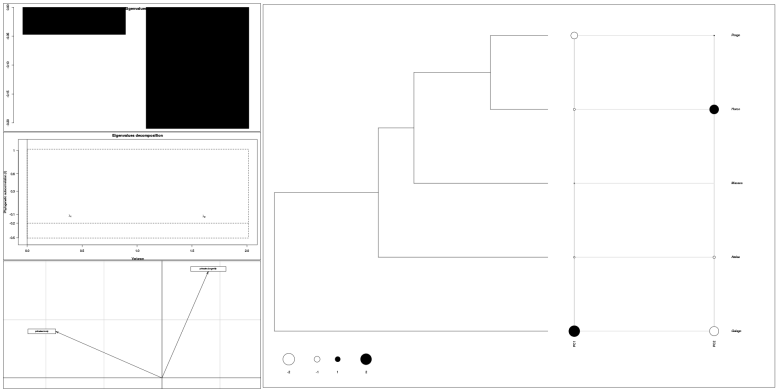
PCA corrected for phylogeny

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

Phylogenetic principal component analysis — the code

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

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



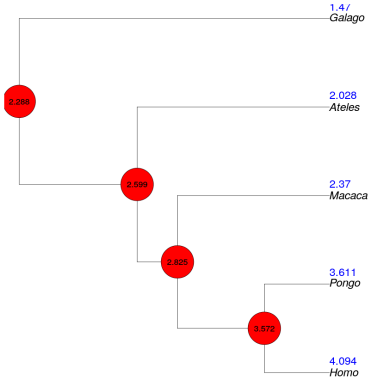
Ancestral state reconstruction

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

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

Ancestral state reconstructions of primates body weights

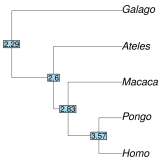
- ACE returns long numbers — truncate them by e.g. `round(x=…, digits=3)` (`x` is vector with ACE values)



Another possibilities (package phytools)

```
library(phytools)
# More possibilities
plot.phylo(x=primates.tree,
  edge.width=2, cex=2)
# ML estimation of a continuous trait
# can compute confidence interval
nodelabels(fastAnc(tree=
  primates.tree, x=primates.body))
# ACE for Brownian evolution with
# directional trend
plot.phylo(x=primates.tree,
  edge.width=2, cex=2)
nodelabels(anc.trend(tree=
  primates.tree, x=primates.body,
  maxit=100000)$ace)
# ACE for Brownian evolution using
# likelihood
```

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



Bayesian ancestral character estimation I

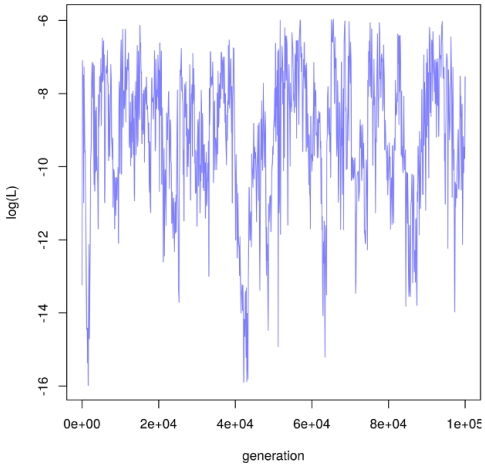
```
primates.body.ace.bayes <- anc.Bayes(tree=primates.tree, x=primates.body,
  ngen=100000) # Use more MCMC generations
primates.body.ace.bayes
# Get end of ancestral states from Bayesian posterior distribution (it
# should converge to certain values)
tail(primates.body.ace.bayes[["mcmc"]])
primates.body.ace.bayes[["mcmc"]][1001,3:6]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

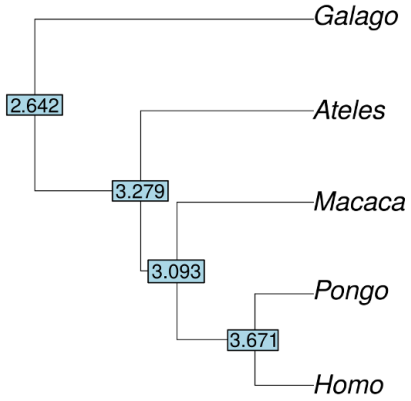
Bayesian ancestral character estimation II

```
primates.body.ace.bayes # See the values
       6        7        8        9
2.418557 2.569484 2.844268 3.584153
tail(primates.body.ace.bayes[["mcmc"]]) # See end of the table
        gen     sig2        6        7        8        9     logLik
 ...    ...      ...      ...      ...      ...      ...      ...
 999  99800 1.960194 4.129754 3.196746 3.310278 3.425686 -9.780220
1000  99900 1.636984 3.556224 2.701105 2.720756 2.965226 -9.770242
1001 100000 1.588710'2.641937 3.279439 3.093420 3.670807' -7.538937
primates.body.ace.bayes[["mcmc"]][1001,3:6] # See end of the table
           6        7        8        9
1001 2.641937 3.279439 3.09342 3.670807
colMeans(primates.body.ace.bayes[["mcmc"]][201:nrow(primates.body.ace.
  bayes[["mcmc"]]),as.character(1:primates.tree$Nnode+length
  (primates.tree$tip.label))]) # Get means of ancestral states
       6        7        8        9
2.418557 2.569484 2.844268 3.584153
```

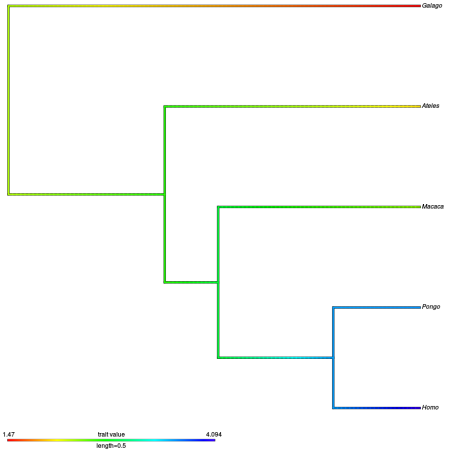
Bayesian ancestral character estimation III

Likelihood of Bayesian posterior probability and the tree with reconstructions

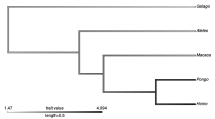




Continuous map

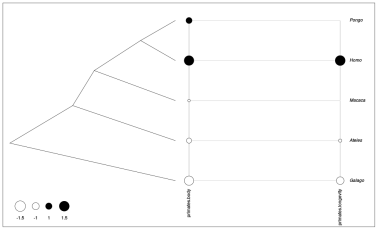



```
library(phytools)
contMap(tree=primates.tree,
  x=primates.body)
# Change colors with setMap()
primates.contmap <- setMap(x=
  contMap(primates.tree,
  primates.body),
  colors=c("white", "black"))
plot(primates.contmap)
# See ?par for more settings
```



Display more characters on a tree in a table

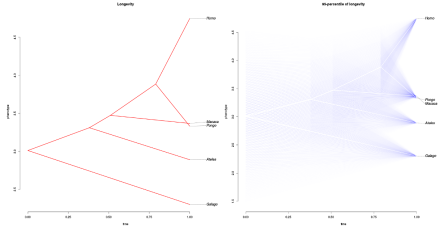
```
library(adephylo)
table.phylo4d(x=phylo4d(x=primates.tree, tip.data=as.data.frame
  (cbind(primates.body, primates.longevity))), treetype="cladogram",
  symbol="circles", scale=FALSE, ratio.tree=0.5)
table.phylo4d(x=phylo4d(x=shorebird.tree, tip.data=shorebird.data),
  treetype="cladogram", symbol="circles",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



Phenogram

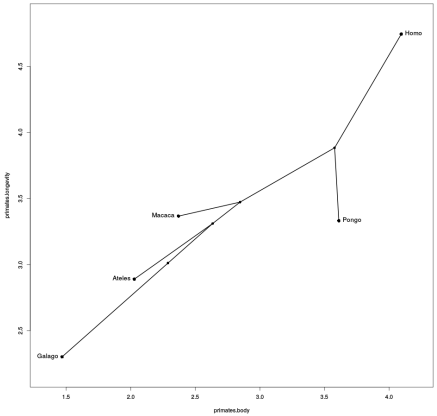
Vertical axis shows character values

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



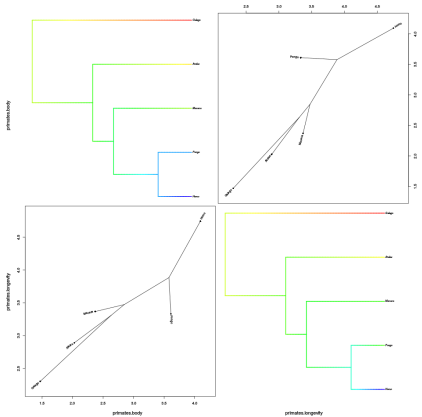
Display 2 continuous characters in space and 3D tree connecting them

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



Combine phenograms and ancestral state reconstructions

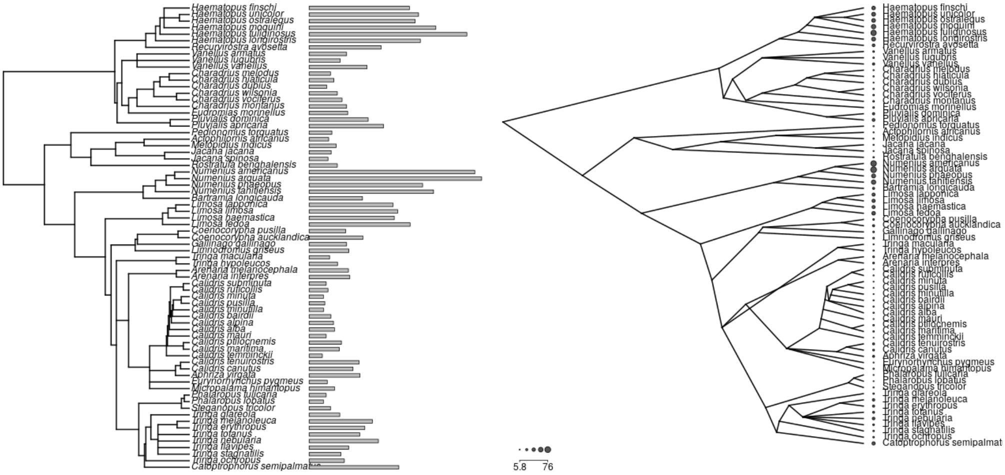
```
# 2 characters on 2 axis
fancyTree(tree=primates.tree,
  type="scattergram",
  X=cbind(primates.body,
  primates.longevity),
  res=500, ftype="i")
# See manuals for more settings
?fancyTree
?phenogram
?phylomorphospace
?phylomorphospace3d
?contMap
?setMap
?par
```



Plotting traits on trees — code

```r
# Load training data to display
data(shorebird, package="caper")
?caper::shorebird # See information about the data
library(phytools) # Load library
# Prepare named vector of values to plot
shorebird.toplot <- shorebird.data[["Egg.Mass"]]
names(shorebird.toplot) <- rownames(shorebird.data)
shorebird.toplot # See it
# See options for plotting functions
?plotTree.wBars
?dotTree
?plotSimmap
# Plot the trees
plotTree.wBars(tree=shorebird.tree, x=shorebird.toplot,
  tip.labels=TRUE)
dotTree(tree=shorebird.tree, x=shorebird.toplot, tip.labels=TRUE,
  type="cladogram")
```

Plotting traits on trees — plots



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

Direct saving of plots to disk

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

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

Graphical packages

- Basic plotting functions in R are very limited...
  - The usage is simple, but anything more complicated requires extensive coding (plenty of examples were shown in the course)...
  - It can be tricky to get desired figure — some magic use to be needed...
- There are plenty of graphical packages
- Advanced functions we used internally by used packages are lattice (web), gplot and ggplot2 (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`, `mfcol`, `mfrow`, ...) — see help pages...
- Most important low-level functions are `points`, `lines`, `text`, `abline`, `legend`, `axis`, `axes`, `arrows`, `box` — see help pages...

Install package from GitHub

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

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

R script and its running from command line 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

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

Simple loop — for cycles

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

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

For and while loops

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

If-else branching I

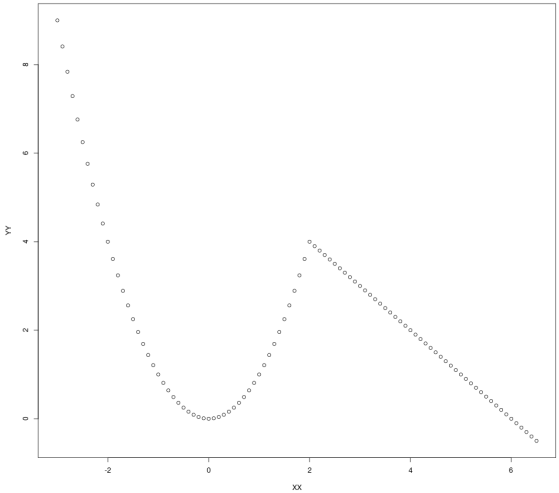
- Basic method of branching the code — **if** the condition is met, **then** one branch is followed, **else** — in any other case — the other branch of the code is executed
- `else` part can be missing — the code is executed **only if** the condition is met

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

If-else branching II

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

Output of the if-else branching example



Most common problems and their solutions I

- Something was not found (object, function file, ...)
  - Check spelling of all methods, parameters, etc.
  - Check all paths (slide 78)
  - 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 193) — 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 361 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 361 and onward)
  - Generally, follow news for your topic (appropriate mailing list, ...)

How to ask for help I

- Never ever ask simple silly lazy questions you can quickly find in manual or web
- People on mailing lists and forums respond volunteerly in their spare free time — do not waste it — be polite, brief and informative
- Be as specific and exact as possible
  - Write exactly what you did ("It doesn't work!" is useless...)
  - Copy/paste your commands and their output, especially error messages — they are keys to solve the problem
  - Try to search web for the error messages (or their parts)
  - Try to provide minimal working example — add at least part of your data (if applicable) so that the problem is reproducible
  - Specify version(s) of R/packages, operating system and/or another important details — authors will commonly insist on newest versions: add outputs of `sessionInfo()` and `packageVersion("PackageName")`
- **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 359.

- 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 http://lists.r-forge.r-project.org/cgi-bin/mailman/listinfo/adegenet-forum and GitHub page https://github.com/thibautjombart/adegenet/wiki
- Poppr forum https://groups.google.com/forum/#!forum/poppr
- R help mailing list https://stat.ethz.ch/mailman/listinfo/r-help (web interface http://r.789695.n4.nabble.com/)
- R announce mailing list https://stat.ethz.ch/mailman/listinfo/r-announce
- R ecology mailing list https://stat.ethz.ch/mailman/listinfo/r-sig-ecology

Where to look for the help III

- R at StackOverflow StackExchange (for programmers)
  https://stackoverflow.com/questions/tagged/r
- R at CrossValidated StackExchange (for stasticians, mathematicians, etc.)
  https://stats.stackexchange.com/questions/tagged/r
- Biostars — general bioinformatics forum https://www.biostars.org/
- Biology — general forum about biology at StackExchange
  https://biology.stackexchange.com/
- Do not hesitate to ask on the forum or contact author of package with which you have problem, preferably through some public forum or mailing list, they usually respond quickly and helpfully... — they wish their packages to be working and useful
- Uncle Google is your friend here (*"how to XXX in R"*)...

Citations

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

Further reading

The most important books for our topics


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


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


Paurush Praveen Sinha
Bioinformatics with R Cookbook
Packt Publishing, 2014


Anthony R. Ives
Mixed and Phylogenetic Models: A Conceptual Introduction to Correlated Data
Leanpub, 2018
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 phylo wiki 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/
- RForge package repository https://r-forge.r-project.org/ (with documentation)

Learning resources II

- 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/
- Little Book of R for Time Series
  https://a-little-book-of-r-for-time-series.readthedocs.org/en/latest/

Learning resources III

- Adegenet web http://adegenet.r-forge.r-project.org/ and GitHub page https://github.com/thibautjombart/adegenet/wiki
- APE home page http://ape-package.ird.fr/
- Information and manual about pegas http://ape-package.ird.fr/pegas.html
- Phytools http://phytools.org/, its blog http://blog.phytools.org/ and GitHub page https://github.com/liamrevell/phytools
- Poppr documentation https://grunwaldlab.github.io/poppr/reference/poppr-package.html
- Population Genetics in R https://popgen.nescent.org/ by Kamvar et al

Learning resources IV

- ade4 home page http://pbil.univ-lyon1.fr/ADE-4/ade4-html/00Index.html?lang=eng and documentation http://pbil.univ-lyon1.fr/ade4/home.php?lang=eng
- Phangorn resources https://CRAN.R-project.org/package=phangorn
- The R journal https://journal.r-project.org/
- 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/
- R on The Molecular Ecologist https://www.molecularecologist.com/category/software/r/

Learning resources V

- 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
- 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
- ggplot2 (the most powerful graphical library used by many packages) information
  https://ggplot2.tidyverse.org/

Learning resources VI

- 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"*)...
- 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
- rgdal: bindings to the Geospatial Data Abstraction Library and access to projection/transformation operations library
- RgoogleMaps: interface to query the Google server for static maps and uses the map as a background image to overlay plots

Packages we used... V

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

- Rmpi: interface (wrapper) to MPI (used for parallel processing)
- rworldmap: mapping global data (and extra data in rworldxtra)
- 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
- StAMPP: statistical analysis of mixed ploidy populations

Packages we used... VI

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

- 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
- 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...

- diveRsity: 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...

- 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 phylogentic 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
- jaatha: simulation-based maximum likelihood parameter estimation

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

For your own explorations...

- 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
- MonoPhy: visualization and exploration of monophyletic clades on a tree

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

For your own explorations...

- 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
- paleoPhylo: assess how speciation, extinction and character change contribute to biodiversity

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

For your own explorations...

- 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
- Phybase: read, write, manipulate, simulate, estimate, and summarize phylogenetic trees (gene trees and species trees)
- phyclust: phylogenetic clustering

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

For your own explorations...

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

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

For your own explorations...

- polyfreqs: Gibbs sampling algorithm to perform Bayesian inference on biallelic SNP frequencies, genotypes and heterozygosity in a population of autopolyploids
- polysat: polyploid microsatellite analysis
- 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
- rMaps: interactive maps
- rphast: interface to PHAST software for comparative genomics

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

For your own explorations...

- 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
- 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

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

For your own explorations...

- 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
- surface: fitting Hansen models to investigate convergent evolution

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
- 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
- treespace: exploration of distributions of phylogenetic trees

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

For your own explorations...

- 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 XeLaTeX on openSUSE GNU/Linux, January 26, 2020.