Molecular data in R
Phylogeny, evolution & R

Vojtěch Zeisek

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

November 23 to 25, 2016

Outline I

Outline II

Outline III

Outline IV

Outline V

Materials to help you…

- Download the presentation from `https://trapa.cz/sites/default/rcourse/r_mol_data_phylogen.pdf`
- Follow all code we will use at `https://trapa.cz/sites/default/rcourse/course_commands.html`
- Download the script from `https://trapa.cz/sites/default/rcourse/course_commands.r`, use it and write your comments and notes to it during the course
  - Note: Open the R script in some good text editor – showing syntax highlight, line numbers, etc. (NO Windows Notepad); the file is in UTF-8 encoding and with 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 in e.g. RStudio or any other R GUI (see further) and directly work with it
  - Downloaded file must have sufix `*.r`, not `*txt`

What we will and what we will not do…

We will go through…

- Basic introduction into R
- Analyzing phylogeny and evolution and basic theory
    - DNA sequences, SNP, SSRs, AFLP, …
    - NJ, UPGMA, PCoA, DAPC, Bayesian clustering, ML, maximum parsimony, …
        - Character evolution, ancestral state reconstructions, …
        - Alignments
        - Manipulations with trees
- Plotting

- Maps, spatial analysis, …
- Basic creation of scripts
- And more…

We will not dig deep into…

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

About R

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

Graphical user interfaces (GUI)

- Provide more comfortable interface for work with scripts (source code highlight), overview of loaded packages and variables, easier work with figures, …
- RStudio https://www.rstudio.com/ – probably the most common, multiplatform, very powerful
- RKWard https://rkward.kde.org/ – feature very rich, developed mainly for Linux, available also for another operating systems
- R commander (Rcmdr) http://www.rcommander.com/ – multiplatform, not so rich as previous
- Java GUI for R http://rforge.net/JGR/ – Java (multiplatform, but with all Java issues like memory consumption)
- Tinn-R (Windows only) https://sourceforge.net/projects/tinn-r/ and http://nbcgib.uesc.br/lec/software/editores/tinn-r/en
- Pick one you like (from above list or any else) and install it…

RKWard

RStudio

MS Windows & Apple Mac OS X

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

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
- 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

- Install package `build-essential`
- 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 `r.adu.org.za`, see CRAN
  Mirrors at https://cran.r-project.org/

Linux – openSUSE

- See instructions at
  https://cran.r-project.org/bin/linux/suse/
- Add repositories for appropriate version of your distribution
  - http://download.opensuse.org/repositories/devel:/languages:/R:
    /patched/
  - http://download.opensuse.org/repositories/devel:/languages:/R:
    /released/
- Install packages R-base (the R) and rstudio and/or rkward
- Install package patterns-openSUSE-devel_basis for compilation
  of R packages when installing them from R (only some R packages
  are available in repositories)

Sources of R packages

- R CRAN `https://cran.r-project.org/` – main and largest source of R packages (over 9500 packages + many orphaned and archived)
- Bioconductor `https://bioconductor.org/` – mainly bioinformatics packages, genomic data (∼1300 packages)
- R-Forge `https://r-forge.r-project.org/` (over 2000 packages)
- RForge `https://www.rforge.net/` (much smaller)
- And more…
- Some packages are available from more resources
- Same name for function can be used in different packages (there is no central index) – to distinguish them call functions like this: `muscle::read.fasta()` vs. `seqinr::read.fasta()` – call function `read.fasta()` from package `muscle` or `seqinr` (and their parameters can be different…) – see further

First steps in R

Recommended is usage of GUI (RKWard or RStudio)

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

How it works

- General look of R commands:
- `function(argument1="SomeName", argument2=SomeVariable, argument3=8)`
- `ModifiedObject <- SomeFunction(argument1=MyData, argument2=TRUE)`
- 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

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

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

- Name of the package (top left)
- Function name (headline)
- Description
- Usage
- Comments on arguments

- Details
- About author(s)
- References to cite
- Example code

Where we are?

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

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

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

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 meat many of them…
- Functions require particular object types – take care about it

Popular object classes (we are going to use) I

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

Popular object classes (we are going to use) II

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

To get information about content of each data type see

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

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

Conversions among data types I

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

Conversions among data types II

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

Conversions among data types III

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

Basic operations with data I

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

Basic operations with data II

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

Repositories

- Repositories (internet directories full of R packages) can be set via `options(repos=c())` or as `repos` parameter for each `install.packages()` command (slide 31 and onward)
- Repositories doesn't have to be set as global options, e.g. Bioconductor has its own way to manage packages

Installation of packages in GUI

- **RStudio:** set repositories by command from slide 31 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 31. **OK**. Go to menu **Settings | Manage R packages and plugins...**, click to **Install...**, select and install desired packages...

Introduction | R | Data | Basic analysis | DAPC | SNP | Spatial analysis | Structure | Alignment | Trees | Evolution | The end

Installation | Let's start with R | Basic operations in R | Packages for our work

Set repositories

```r
# Basic package installation
install.packages("PackageName") # Case sensitive!
?install.packages # Shows all available parameters (options)
# We will need extra repositories. For Bioconductor keep same version
# as is version of your R installation (3.3 in this case).
options(repos=c("http://cran.at.r-project.org",
  "https://r-forge.r-project.org/", "https:/rforge.net/",
  "https://bioconductor.statistik.tu-dortmund.de/packages/3.3/bioc",
  "https://bioconductor.statistik.tu-dortmund.de/packages/3.3/data/
  annotation", "https://bioconductor.statistik.tu-dortmund.de/
  packages/3.3/data/experiment",
  "https://bioconductor.statistik.tu-dortmund.de/packages/3.3/extra"))
getOption("repos") # Shows actual repositories
options() # Generic function to modify various settings
?options # Gives details
```

- Keep newest version of R and and newest versions of packages!
- Installation of multiple packages may sometimes fail – install then packages in smaller groups or one by one

Install needed packages

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

Install packages without setting the repositories

- If repositories from slide 31 are not set (for any reason), it is possible to install in several steps packages from main repository (CRAN) and from another sources (following slides)
- This is the basic and default the most common usage

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

Install phyloch package

Example of installation of package not available in any repository

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

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

Bioconductor

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

```
1 # Load basic Bioconductor function
2 source("https://bioconductor.org/biocLite.R")
3 ?biocLite # Get help how to use it
4 # Install packages
5 biocLite(c("package1", "package2", "..."))
6 biocLite() # Update Bioconductor packages (within same R version)
7 # Upgrades installed Bioconductor packages to new R release
8 biocLite("BiocUpgrade") # E.g. from R 3.2 to 3.3
```

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

Bioconductor packages for the course

- If repositories are not set via command on slide 31, it is possible to use Bioconductor's own installation method using function `biocLite`
- If Bioconductor repositories are set manually, user must select correct version number
- List of available Bioconductor mirrors is at
  https://bioconductor.org/about/mirrors/

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

Bioconductor and others – differences from another repositories

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

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

Non-R software

- We use several software packages outside R
  - R functions can use this software
  - External software can be used (depending on R package) to create/modify R object, or just as different method for (batch) usage of the software
  - User must install this software manually

- Structure http://pritchardlab.stanford.edu/structure.html (optionally also CLUMPP and distruct – not part of the course)
- MAFFT http://mafft.cbrc.jp/alignment/software/
- MUSCLE http://www.drive5.com/muscle/
- Clustal (W/X; Omega is not used in the course) http://clustal.org/

Our data…

- Work with microsatellites is in most cases (except some methods taking advance from microsatellite mutational nature) same as with presence-absence data and methods can handle both data types in nearly same fashion
  - Examples are shown mainly with microsatellites, but AFLP and another presence-absence $(0/1)$ data are used in same way – try it

- Distance-based methods are same regardless input data on the beginning (microsatellites, AFLP, DNA sequences, …)

- Extraction of SNP from DNA is useful in case of huge datasets – for smaller datasets it is not necessary

Always save your work!

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

Notes about paths to import the data

- Generally, R can accept any local or web location
- Web locations start with `https://`, `http://` or `ftp://`
- Local paths (within one computer) can be absolute or relative
  - Absolute paths start from the top of files hierarchy: on UNIX (Linux, Mac OS X, …) it use to look like `/home/USER/`, on Windows like `C:\`
  - Relative paths start in current directory (so **no** with `/` or `C:`) – in the easiest case the input file is in same directory as is R's working directory – verify by `getwd()` and `dir()` – you then need to specify only the filename
  - 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 any filename parameter
- Some R functions have problems with spaces and special (non-alphanumeric and accented) characters – try to avoid them
- One of the most common source of errors – when the command fails, double check paths (and Internet connection, if applicable)

Population genetics and phylogenetics in R
Microsatellites, AFLP, SNP & sequences

- Now we will use mainly packages adegenet and poppr

- Other important genetic packages: ape, ade4 and pegas

- Dominant/co-dominant marker data of any ploidy level including SSRs, SNP, and AFLP are analyzed in same way

- Most of methods are available for polyploids (although not all)

- Some methods are unavailable for dominant (presence/absence) data

- Mixing of ploidy levels is tricky (but possible) – it doesn't matter when data are encoded as PA, otherwise it is mathematically problematic

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

Load data

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

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

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

Prepare genind object for analysis and load coordinates

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

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

Source data:

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

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

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

Add coordinates to genind and create genpop object

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

Import existing data set from popular software

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

One function rules them all…

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

Import of polyploid microsatellites

- **adegent**, **poppr** and related packages can for most of functions handle any ploidy level (including mixing of ploidy levels, but not for all analysis)

- **polysat** package can handle mixed ploidy levels for microsatellites, but range of methods is limited

As for AFLP, we need two files: the data matrix and individual's populations (it can be combined in one file – next slide)

Triploid microsatellite data:

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

Triploid species of *Taraxacum* sect. *Taraxacum*

How to import polyploid microsatellites

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

Import of AFLP data – background

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

**AFLP or any other
presence/absence data:**

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

AFLP data of *Cardamine amara* group

Individual's populations:

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

Just list of populations for respective
individuals...

- Use any names, just keep one
  word (no spaces) and don't use
  special characters

- Keep names of loci as simple
  as possible, there are some
  issues when they contain dots

- As soon as one line of data =
  one individual, ploidies and
  their mixing doesn't matter

- Not all methods are
  available/meaningful for PA

Import of AFLP data – the code

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

Another data manipulation

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

```
1 genind2df()  # adegenet - export into data frame
2 genind2genalex()  # poppr - export for genalex
3 splitcombine()  # poppr - edits population hierarchy
4 popsub()  # poppr - extracts only selected population(s)
5 clonecorrect()  # poppr - corrects for clones
6 informloci()  # poppr - removes uninformative loci
7 seppop()  # adegenet - separates populations from genind or genlight
8 seploc()  # adegenet - splits genind, genpop or genlight by markers
9 # seppop and seploc return lists of genind objects - for further
10 # analysis using special functions to work on lists (see further)
11 # read manual pages (?...) of the functions before usage
```

Notes about getting data into R

- When importing fragmentation data, we somehow use function `read.table()` – it is important to understand it
- I recommend to use TAB (TSV – tab separated values; encoded as `\t` in R) to separate columns (no quotation marks, no commas)
- When importing microsatellites, all alleles **must** have same number of digits. Separate alleles by "`/`", "`|`" or something similar and correctly specify it in `read.loci()` or `df2genind()`
- Do not use underscores ("`_`") or minuses to name objects in R – only numbers, letters or dots
- `read.loci()` sometimes doesn't work correctly on AFLP or polyploid microsatellites – try `read.table()` instead…
- Genind object (since Adegenet version 2) is able to store mixed ploidy data, but not all analysis are able to handle that…

Import of DNA data I

```r
# Reading FASTA (read.dna() reads also another formats, see ?read.dna)
# Sequences of flu viruses from various years (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 data II

```r
# Convert DNAbin to genind - only polymorphic loci (SNPs) are retained
# When converting DNAbin to genind, the sequences must be aligned!
usflu.genind <- DNAbin2genind(x=usflu.dna, pop=usflu.annot[["year"]])
# read.fasta() from seqinr package reads DNA or AA in FASTA format and
# returns a list (DNAbin is for us now better choice)
usflu.dna3 <- seqinr::read.fasta(file="http://adegenet.r-forge.
  r-project.org/files/usflu.fasta", seqtype="DNA")
class(usflu.dna3)
length(usflu.dna3) # How many sequences we have in the list
usflu.dna3
# Convert into DNAbin class (technically, DNAbin is a list)
class(usflu.dna3) <- "DNAbin"
```

Import sequences from GenBank

```
1  # Importing sequences according to sequence ID
2  # Data from https://www.ncbi.nlm.nih.gov/popset/608602125 (Meles meles)
3  meles.dna <- read.GenBank(c("KJ161355.1", "KJ161354.1", "KJ161353.1",
4    "KJ161352.1", "KJ161351.1", "KJ161350.1", "KJ161349.1", "KJ161348.1",
5    "KJ161347.1", "KJ161346.1", "KJ161345.1", "KJ161344.1", "KJ161343.1",
6    "KJ161342.1", "KJ161341.1", "KJ161340.1", "KJ161339.1", "KJ161338.1",
7    "KJ161337.1", "KJ161336.1", "KJ161335.1", "KJ161334.1", "KJ161333.1",
8    "KJ161332.1", "KJ161331.1", "KJ161330.1", "KJ161329.1", "KJ161328.1"))
9  meles.dna
10 class(meles.dna)
11 # Converts DNAbin to genind - extracts SNP - for large datasets can be
12 # computationally very intensive
13 meles.genind <- DNAbin2genind(meles.dna)
```

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

Query on-line sequence databases

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

Checking SNPs

```
1  # Position of polymorphism within
2  # alignment - snpposi.plot requires
3  # input data in form of matrix
4  snpposi.plot(x=as.matrix(
5    meles.dna), codon=FALSE)
6  # Position of polymorphism within
7  # alignment - differentiating codons
8  snpposi.plot(as.matrix(meles.dna))
9  # When converting to genind object,
10 # only polymorphic loci are kept -
11 # threshold for polymorphism can
12 # be arbitrary
13 meles.genind <- DNAbin2genind(x=
14   meles.dna, polyThres=0.01)
15 # Test of random distribution of SNP
16 snpposi.test(as.matrix(meles.dna))
```
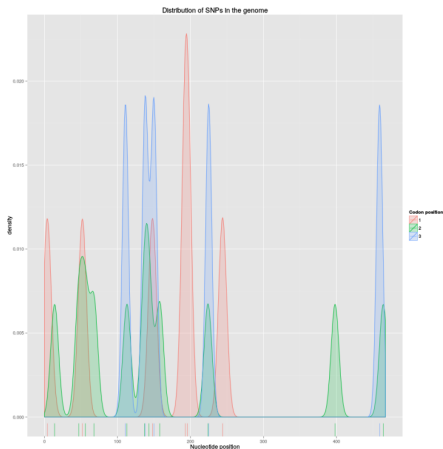
Importing SNP

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

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

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

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

- For small datasets, keep data as genind as it is more information-rich
  – genlight is more efficient for large data

```
1 # Convert genlight to genind (usually not necessary)
2 genind <- df2genind(as.data.frame(genlight)) # Check settings!
3 # Resulting data matrix might be slightly different than when using
4 # DNAbin2genind - depends on used settings
```

Checking sequences

```r
# Nucleotide diversity
pegas::nuc.div(x=meles.dna)
# Base frequencies
ape::base.freq(x=meles.dna)
# GC content
ape::GC.content(x=meles.dna)
# Number of times any dimer/trimer/etc oligomers occur in a sequence
seqinr::count(seq=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
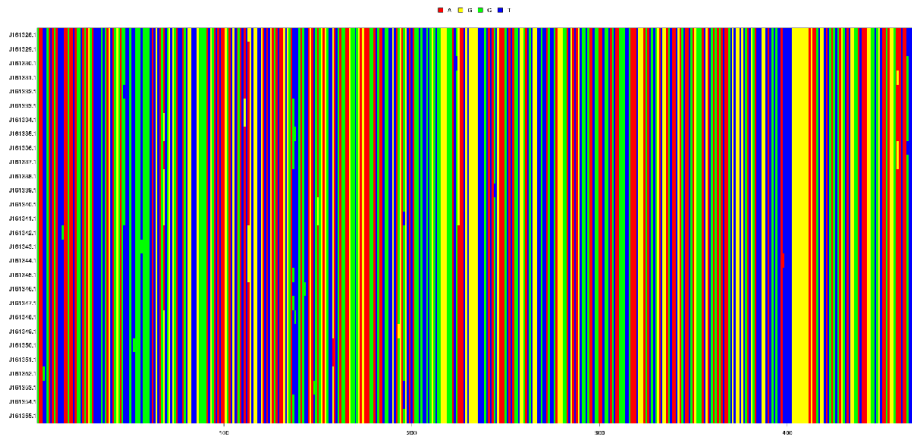
Notes about using genlight (vs. genind)

- 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 – try genind and when work fails because of not enough computer resources, go on with genlight

- Can be quickly converted to genind (not back)

- Use is basically same as when working with genind – but not all functions are able to deal with it (on the other hand, others are 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

Export data

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

Descriptive statistics I

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

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

Descriptive statistics II

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

Graphs from previous two slides
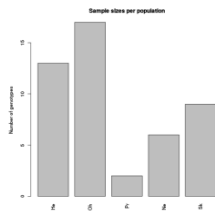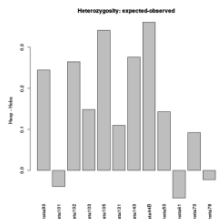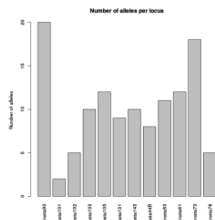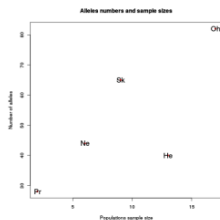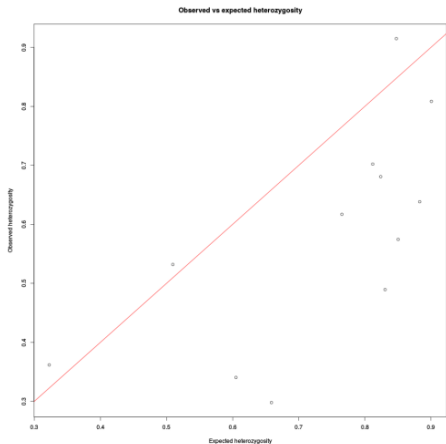
Population statistics by poppr

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

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

- Pop – Population name (Note that "Total" also means "Pooled")
- N – Number of individuals observed
- MLG – Number of multilocus genotypes (MLG) observed
- eMLG – The number of expected MLG at the smallest sample size ≥ 10 based on rarefaction
- SE – Standard error based on eMLG
- H – Shannon-Wiener Index of MLG diversity
- G – Stoddart and Taylor's Index of MLG diversity

- Hexp – Nei's 1978 genotypic diversity (corrected for sample size), or Expected Heterozygosity
- E.5 – Evenness, E5
- Ia – The index of association, IA
- rbarD – The standardized index of association
- See poppr's manual for details
- The outputs are as table in R console and images saved to disk

Departure from Hardy-Weinberg equilibrium

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

Departure from HWE – results

```
hauss.hwe.test
```

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

F-statistics and population theta I

- Functions return tables of F-statistics/thetas values for populations/loci

- The different F-statistics look at different levels of population structure. F_{IT} is the inbreeding coefficient of an individual relative to the total population; F_{IS} is the inbreeding coefficient of an individual relative to the subpopulation and averaging them; and F_{ST} is the effect of subpopulations compared to the total population

- For `Fst`, `fstat` and `theta.msat` the loci object **must** contain population column

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

F-statistics and population theta II

```
1  # multilocus estimators of variance components and F-statistics,
2  # alternative to Fst
3  library(hierfstat)
4  fstat(x=hauss.genind, pop=NULL, fstonly=FALSE)
5              pop          Ind
6  Total 0.1589501 0.2582641
7  pop    0.0000000 0.1180834
8  # Nei's pairwise Fst between all pairs of populations.
9  pairwise.fst(x=hauss.genind, pop=NULL, res.type="matrix", truenames=T)
10            He          Ne          Oh          Pr          Sk
11 He 0.00000000 0.19960826 0.11391904 0.09404571 0.11184561
12 ...        ...         ...         ...         ...         ...
13 # Estimates of population theta according to Kimmel et al. 1998
14 theta.msat(hauss.loci)
15          theta.v     theta.h   theta.x
16 msta93   1287.09037 42.7450000  49.500
17 msta101   105.62800  0.6006928   0.000
18     ...        ...         ...         ...
```

Multi locus genotypes
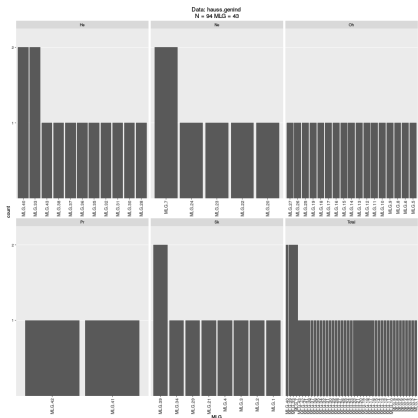

```
1  # Total number of MLGs
2  # (simple value)
3  mlg(gid=hauss.genind, quiet=FALSE)
4  # MLGs shared among populations
5  mlg.crosspop(gid=hauss.genind,
6    df=TRUE, quiet=FALSE)
7  # Detailed view on distribution
8  # of MLGs into populations
9  # (table and/or plot)
10 mlg.table(gid=hauss.genind,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

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

Inbreeding

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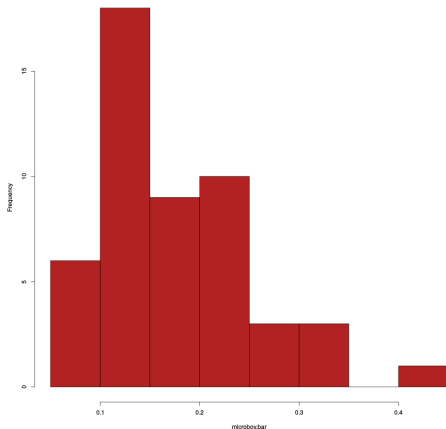
```
1  # Load training data (cattle)
2  data(microbov)
3  # Separate populations of Salers
4  microbov.pops <- seppop(microbov)
5    [["Salers"]]
6  # Calculate the inbreeding
7  microbov.inbr <- inbreeding(x=
8    microbov.pops, N=100)
9  # Check for more settings
10 ?inbreeding
11 # population means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

Basic distances

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

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

## Distances reflecting microsatellite repeats

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

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

Turning distance matrix into Euclidean is controversial...

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

Histograms of Bruvo distance before and after transformation:
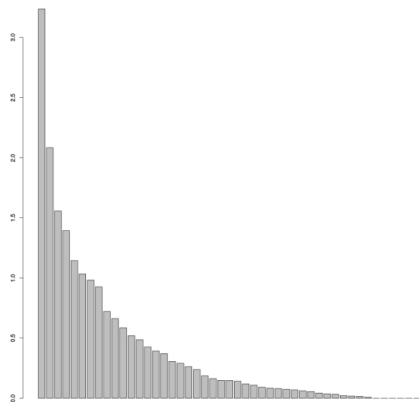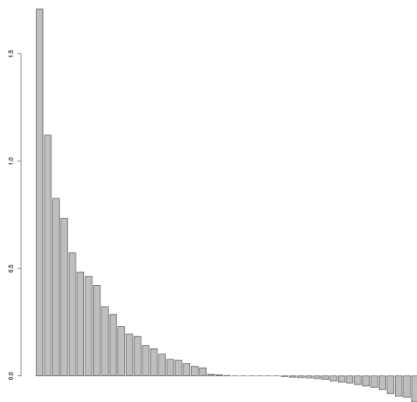
More distances…

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

Import own distance matrix from another software:

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

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

Distances among DNA sequences

The sequences must be aligned before calculating distances among them!

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

Distances and genlight object

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

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

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

Visualize pairwise genetic similarities

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

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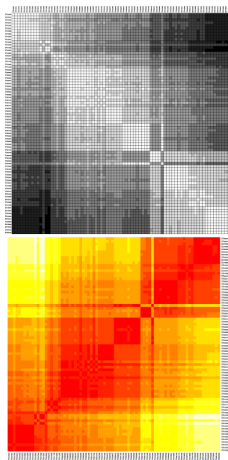
Heatmaps
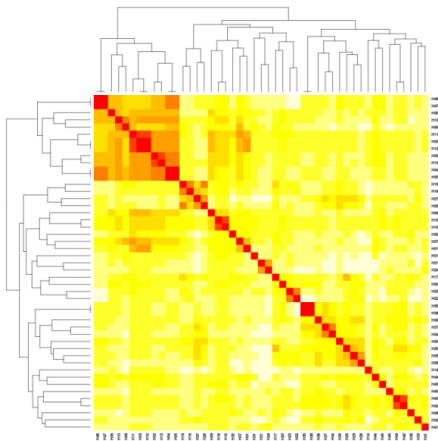

```
1  # Based on various distances
2  heatmap(as.matrix(hauss.dist),
3    symm=TRUE, labRow=rownames(
4    as.matrix(hauss.dist.bruvo)),
5    labCol=colnames(as.matrix(
6    hauss.dist.bruvo)))
7    # hauss.dist doesn't contain
8    # names of individuals - add here
9  heatmap(as.matrix(hauss.dist.pop),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

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

Hierarchical clustering – UPGMA and others
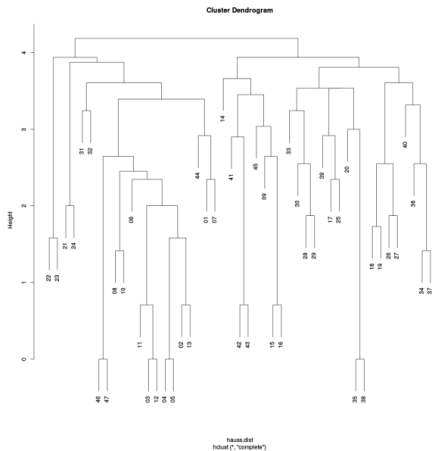

```
1  # According to distance used
2  # see ?hclust for methods
3  plot(hclust(d=hauss.dist,
4    method="complete"))
5  plot(hclust(d=hauss.dist.pop,
6    method="complete"))
7  plot(hclust(d=hauss.dist.
8    bruvo, method="complete"))
```
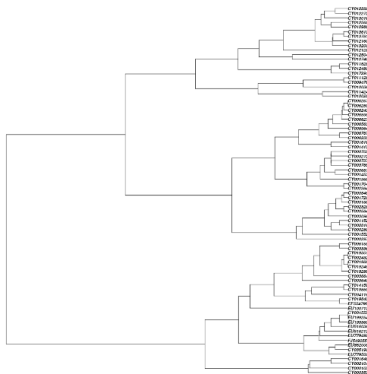
This is very basic function to make dendrogram. There are better possibilities (NJ etc – see slide 85 and onward).

UPGMA and its test

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

UPGMA is not the best choice here…
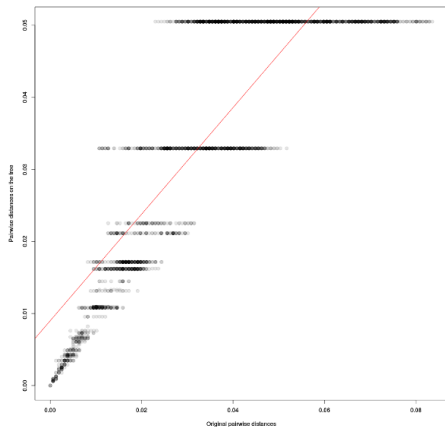
All points in the right graph should be clustered along the red line…

AMOVA

```
1 # From package pegas (doesn't directly show percentage of variance)
2 hauss.pop <- hauss.genind$pop
3 hauss.amova <- pegas::amova(hauss.dist~hauss.pop, data=NULL,
4   nperm=1000, is.squared=TRUE)
5 hauss.amova
6 # Output:
7 $tab
8              SSD       MSD df
9 hauss.pop 15.11916 3.779790  4
10 Error     59.62501 1.419643 42
11 Total     74.74417 1.624873 46
```

- Analysis of molecular variance tests if there are significant differences among populations (and/or another levels)

- Another possibility is `poppr.amova` – for more complicated hierarchy, see `?poppr.amova`

Minimum Spanning Network

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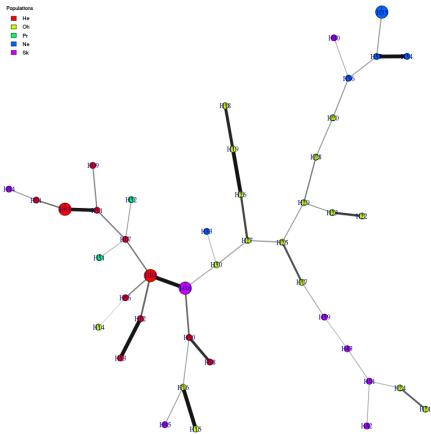

```
1  bruvo.msn(gid=hauss.genind,
2    replen=rep(2, 12), loss=TRUE,
3    palette=rainbow, vertex.label
4    ="inds", gscale=TRUE,
5    wscale=TRUE, showplot=TRUE)
6  ?bruvo.msn
```
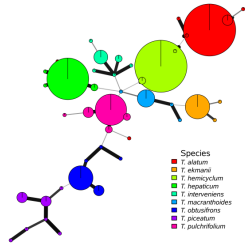
Calculate and test NJ tree

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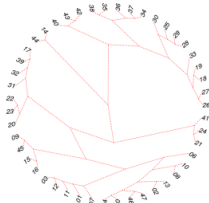
Choose your tree...
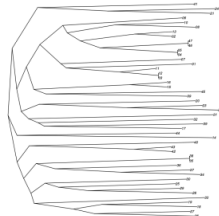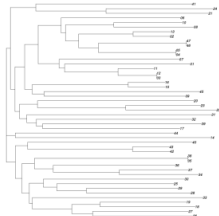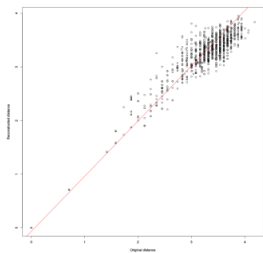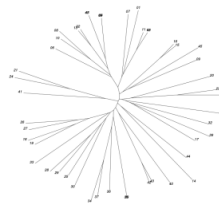
Bootstrap

```r
# boot.phylo() resamples all columns - remove population column first
hauss.loci.nopop <- hauss.loci
hauss.loci.nopop[["population"]] <- NULL
# Calculate the bootstrap
hauss.boot <- boot.phylo(phy=hauss.nj, x=hauss.loci.nopop,
  FUN=function(XXX) nj(dist(loci2genind(XXX))), B=1000)
# boot.phylo returns NUMBER of replicates - NO PERCENTAGE
# Plot the tree
plot.phylo(x=hauss.nj, type="unrooted", main="Neighbour-Joining tree")
# Labels for nodes - bootstrap - see ?nodelabels for graphical settings
nodelabels(text=round(hauss.boot/10))
# Another possibility
?aboot # Package poppr
?boot.phylo
```

Nicer trees

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

Choose your tree…
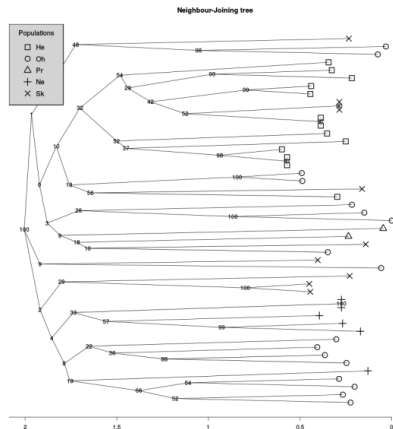
Trees based on Bruvo's distance

Package poppr (bootstrap is incorporated within the function)

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

Choose your tree…
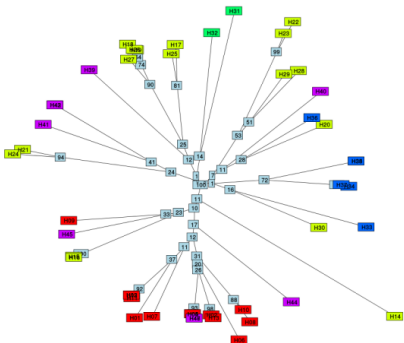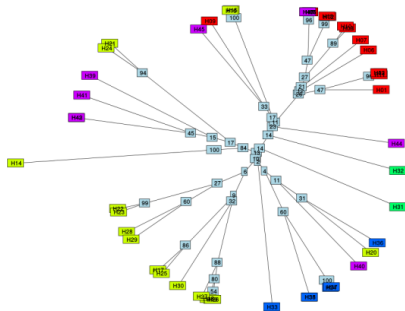
NJ tree based on DNA sequences

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

Root the tree

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

Bootstrap rooted tree

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

Collapse branches with low bootstrap support

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

The trees
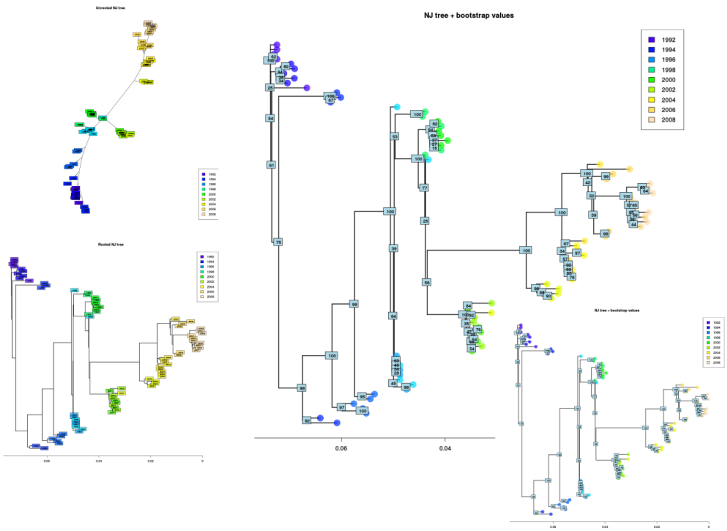
NJ is death. Long live NJ!

- "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

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

PCoA II
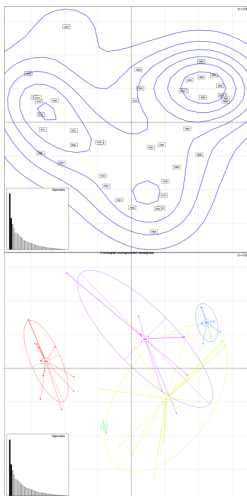

```
1  hauss.pcoa.bruvo <- dudi.pco(d=
2    bruvo.dist(pop=hauss.genind,
3    replen=rep(2, 12)), scannf=F,
4    nf=3)
5  s.class(dfxy=hauss.pcoa.bruvo$li,
6    fac=pop(hauss.genind),
7    col=hauss.pcoa.col)
8  add.scatter.eig(hauss.pcoa.bruvo$
9    eig, posi="bottomright", 3,1,2)
10 # Another possibility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

PCoA – Bruvo and colorplot
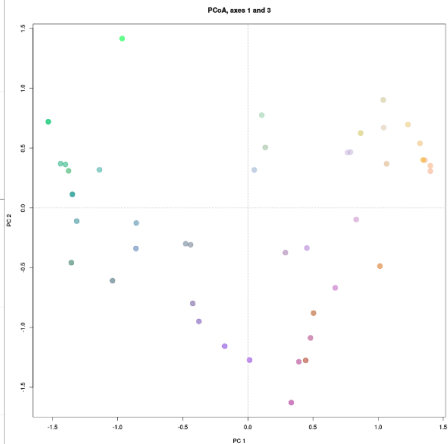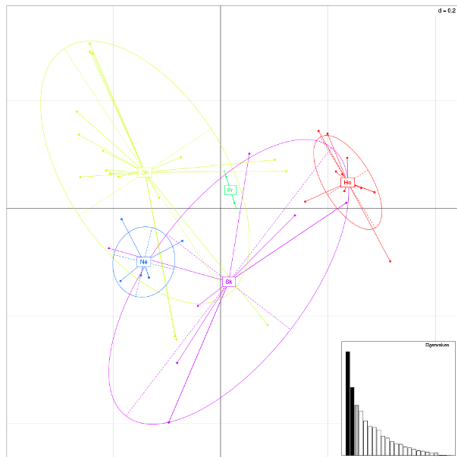
DAPC

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

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

Principal difference between PCA and DA
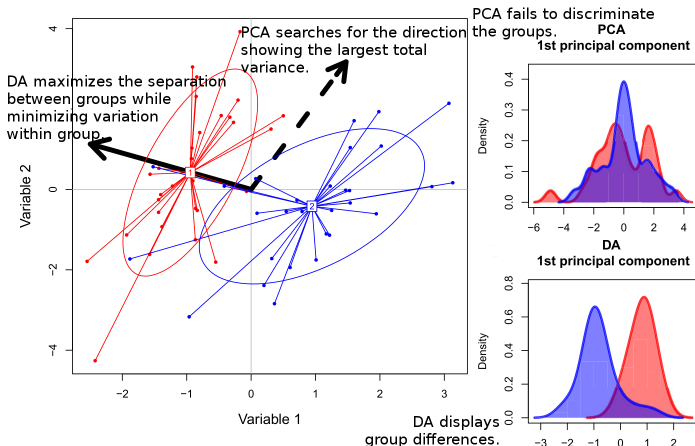
K-find – Bayesian K-means clustering

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

K-find outputs

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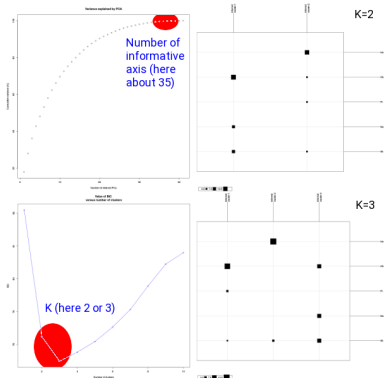

```
1  # See results as text
2  table(pop(hauss.genind),
3    hauss.kfind3$grp)
4  hauss.kfind3
5  # Graph showing table of original
6  # and inferred populations and
7  # assignment of individuals
8  table.value(
9    df=table(pop(hauss.
10   genind), hauss.kfind3$grp),
11   col.lab=paste("Inferred\n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

DAPC code I

```r
## 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
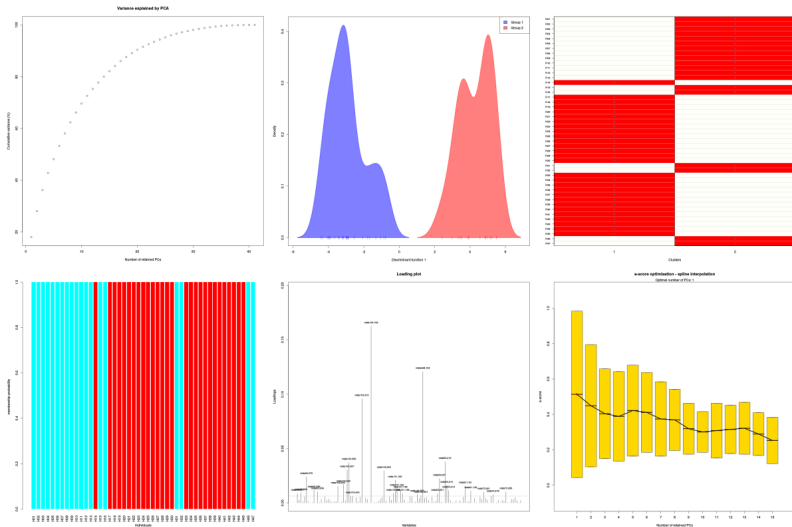
DAPC code II

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

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

DAPC for K=3
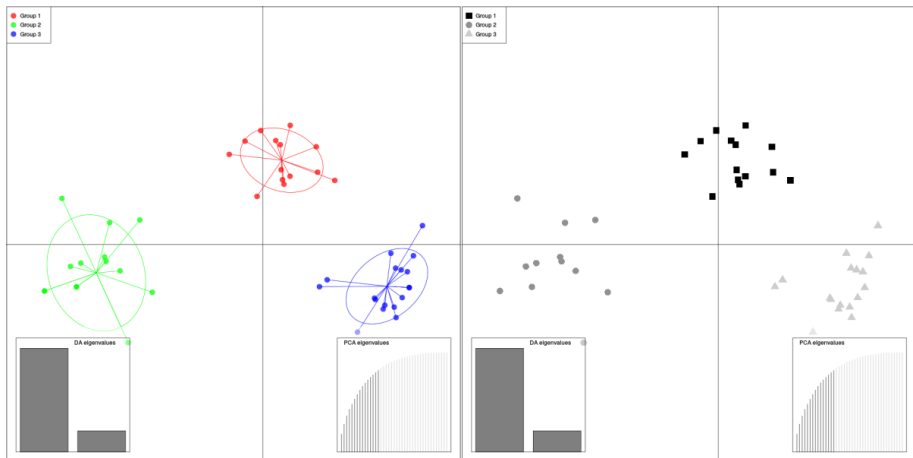
DAPC code III

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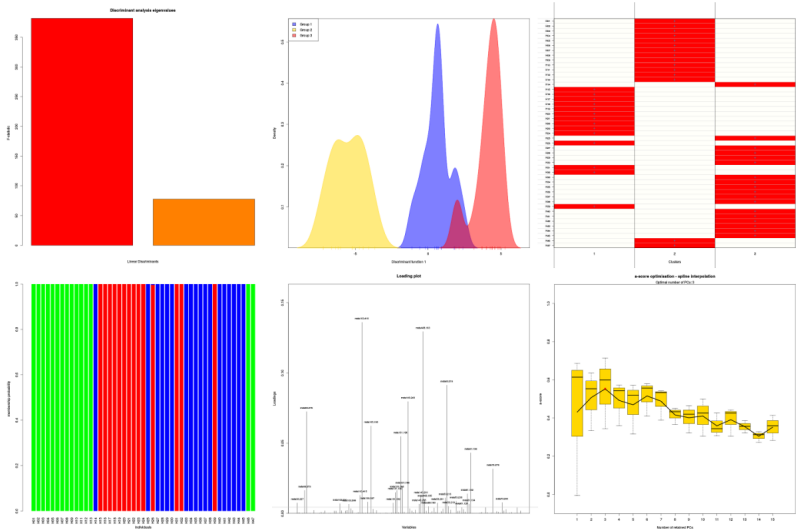
DAPC for K=3, extra information

Another DAPC example
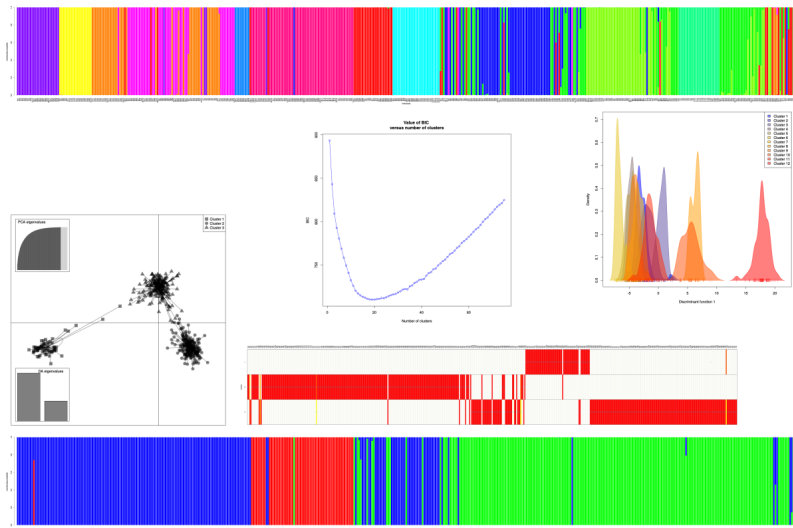
Some functions to work with huge SNP data sets

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

Number of missing values in each locus

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

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

- Lets keep hoping in fast development of computers…

- Note for Windows users: To speed up the processing, `gl*` functions use parallelisation library, which is not available on Windows – add parameter `parallel=FALSE` to be able to use them on Windows

Basic information about SNP: distribution of 2nd allele frequencies, missing data and number of 2nd allele, distribution of allele frequencies, and number of missing values in each locus
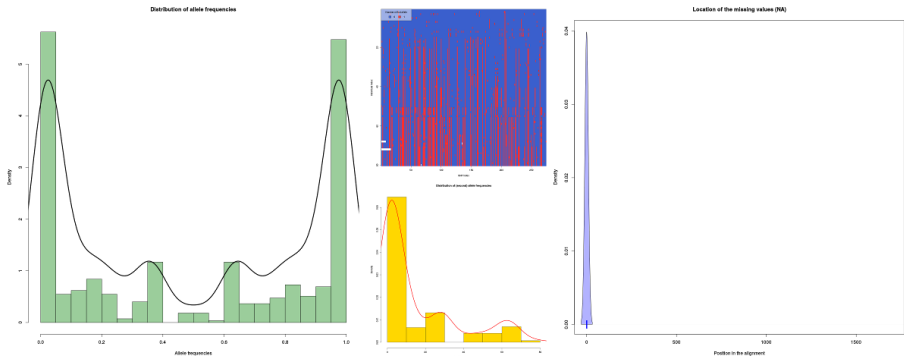
PCA, NJ and genlight objects

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

PCA, NJ and genlight objects
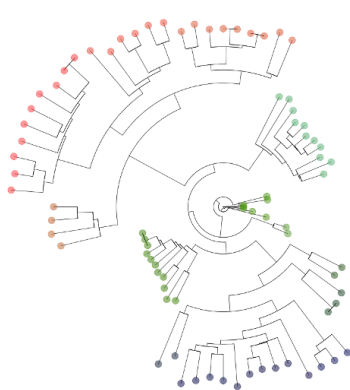
Short overview of spatial genetics (in R)

Basic approaches

- Moran's *I* – several implementations (here as basic autocorrelation index, in sPCA and in Monmonier's algorithm), generally it is autocorrelation coefficient with broader use

- Mantel test – several implementations, popular, although recently criticized as biologically irrelevant, generally correlation of two matrices (here genetic and geographical)

- Bayesian clustering using geographical information as a proxy and showing results in geographical context (here as implemented in Geneland)

- There are unlimited possibilities with connections with GIS software – check specialized courses and literature

Moran's *I*

- "Only" autocorrelation index – no genetic/evolutionary model involved – sometimes criticized as biologically irrelevant mechanism

- This (or similar) approach can be used to test correlation between another characteristics (typically used in ecology)

- Calculations are done according to connectivity network connecting individuals/populations (created by `chooseCN`) – carefully check its options and try several parameters

- Pay attention which hypothesis is tested (i.e. if lower, greater or two sided) – similar to T-test

Calculation of Moran's *I*

```
1 # Load required library
2 library(spdep)
3 # Creates connection network
4 hauss.connectivity <- chooseCN(xy=hauss.genind$other$xy, type=5,
5   d1=0, d2=1, plot.nb=TRUE, result.type="listw", edit.nb=FALSE)
6 hauss.connectivity
7 # Output:
8 Characteristics of weights list object:
9 Neighbor list object:
10 Number of regions: 47
11 Number of nonzero links: 1120
12 Percentage nonzero weights: 50.70167
13 Average number of links: 23.82979
14 Weights style: W
15 Weights constants summary:
16    n    nn S0         S1        S2
17 W 47 2209 47 4.033607 207.037
18 ###
```

Calculation of Moran's *I* II

```
1  # Test of Moran's I for 1st PCoA axis
2  # Results can be checked against permuted values of moran.mc()
3  moran.test(x=hauss.pcoa$li[,1], listw=hauss.connectivity,
4     alternative="greater", randomisation=TRUE)
5  # Output:
6  Moran's I test under randomisation
7  data:  hauss.pcoa$li[, 1]
8  weights: hauss.connectivity
9  Moran I statistic standard deviate = -18.514, p-value = 1
10 alternative hypothesis: greater
11 sample estimates:
12 Moran I statistic        Expectation             Variance
13     -0.5232003724      -0.0217391304         0.0007336276
14 hauss.pcoa1.mctest <- moran.mc(x=hauss.pcoa$li[,1],
15   listw=hauss.connectivity, alternative="greater", nsim=1000)
16 hauss.pcoa1.mctest
17 plot(hauss.pcoa1.mctest) # Plot of densitiy of permutations
18 moran.plot(x=hauss.pcoa$li[,1], listw=hauss.connectivity) # PC plot
```

Moran's I for our 1st axis isn't significant
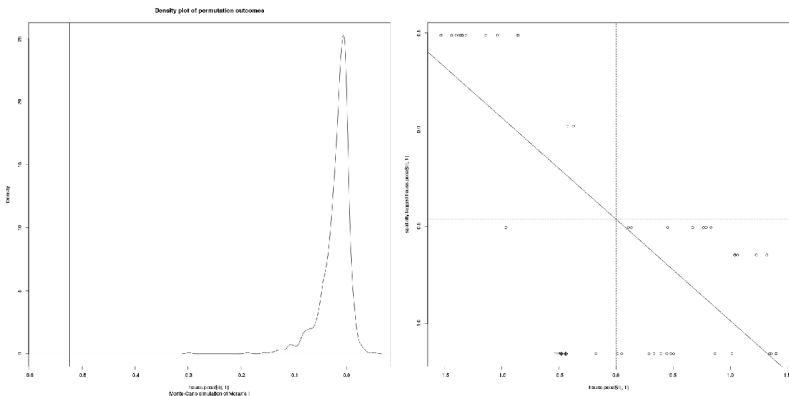
- Tested hypothesis "greater" – **no** significant **positive** autocorrelation
- If testing for hypothesis "less" – significant **negative** autocorrelation
  – individuals are significantly different

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

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

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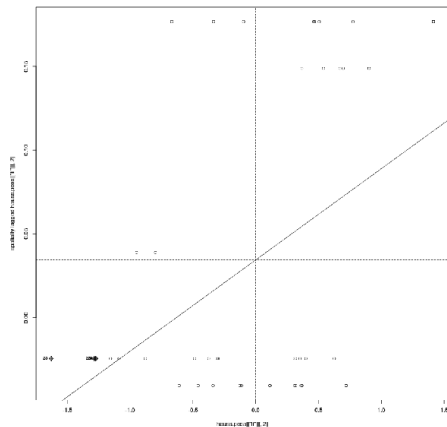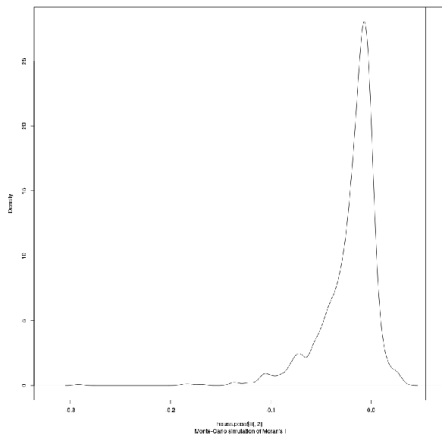
- Tested hypothesis "greater" – there **is** significant positive autocorrelation – individuals are genetically similar over space

Spatial Analysis of Principal Components (sPCA)

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

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

Calculations of sPCA

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

sPCA outputs I
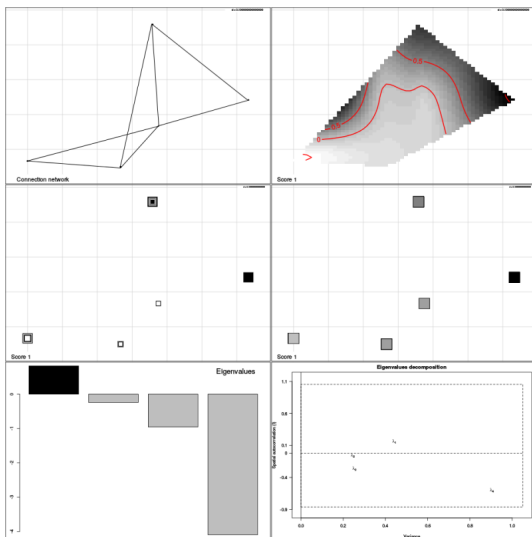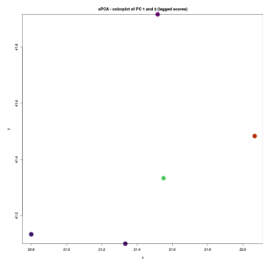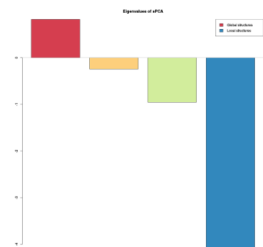
Map of genetic clines

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

sPCA outputs II

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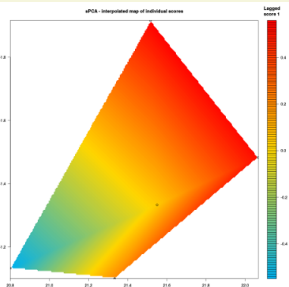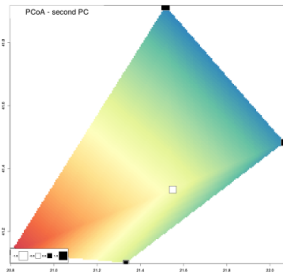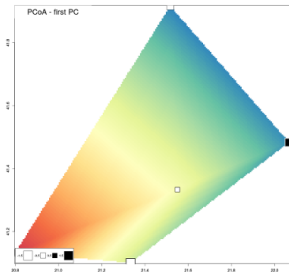
Loading plots – which alleles contribute the most?

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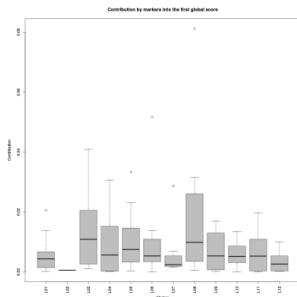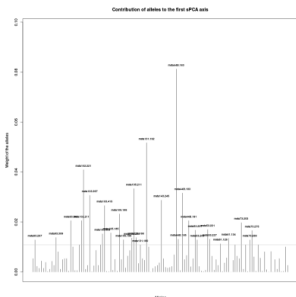
Monmonier's algorithm – genetic boundaries

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

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

Voronoi tessellation

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
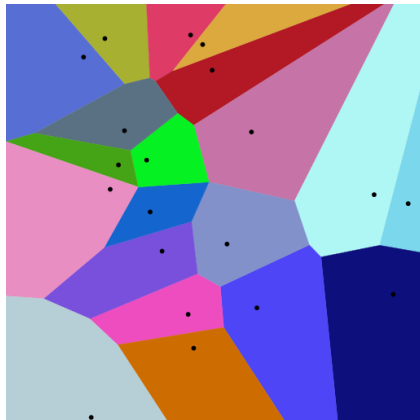
- In simplest case, all points have

Plot genetic boundaries

Monmonier plot

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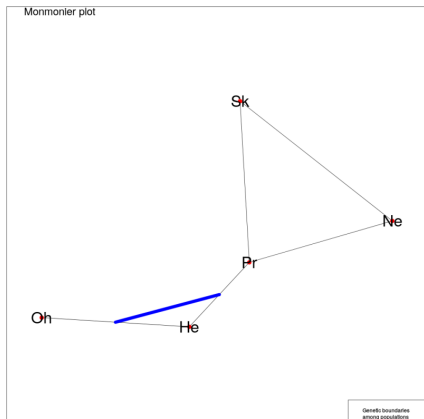
```
1  plot.monmonier(hauss.monmonier,
2    add.arrows=FALSE, bwd=10,
3    sub="Monmonier plot", csub=2)
4  points(hauss.genpop$other$xy,
5    cex=2.5, pch=20, col="red")
6  text(x=hauss.genpop$other$xy$lon,
7    y=hauss.genpop$other$xy$lat,
8    labels=popNames(hauss.genpop),
9    cex=3)
10 legend("bottomright",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

Monmonier notes

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

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

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

Mantel test

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

Mantel test – isolation by distance

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

Mantel test outputs – strongly significant
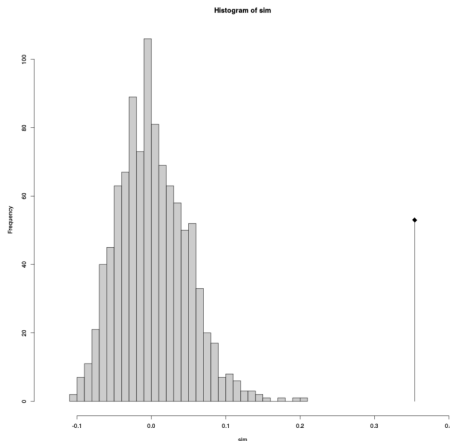

```
1  hauss.mantel # See output
2  Monte-Carlo test
3  Call: mantel.randtest(m1 =
4    hauss.dist, m2 =
5    hauss.gdist, nrepet = 1000)
6  Observation: 0.35409
7  Based on 1000 replicates
8  Simulated p-value: 0.000999001
9  Alternative hypothesis: greater
10   Std.Obs Expectation  Variance
11 7.61967545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

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

Plot of Mantel Correlogram Analysis
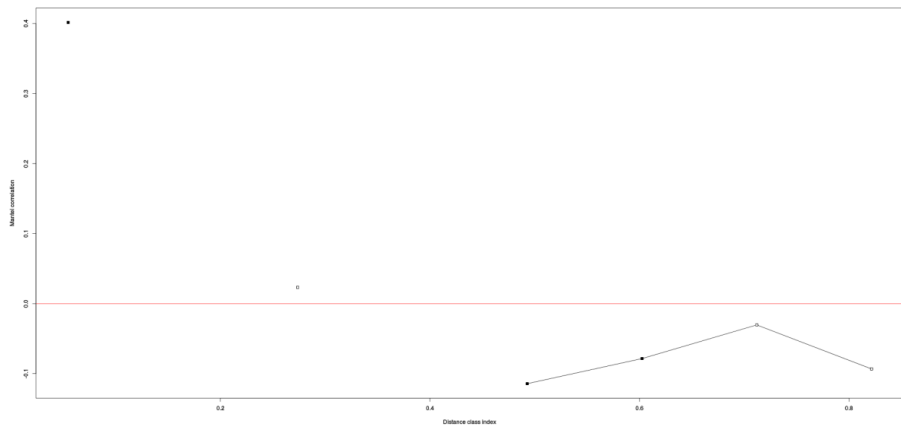
Correlation (genetic similarity) in several distance classes (positive in short distance, negative in long) – see `?mantel.correlog` for details of settings of distance classes

About Geneland

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

```
1  # Load needed libraries
2  library(PBSmapping)
3  library(RandomFields)
4  library(fields)
5  library(spam)
6  library(grid)
7  library(maps)
8  library(tcltk)
9  library(Geneland)
10 # Graphical interface is available
11 # we will use only command line
12 Geneland.GUI()
```

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

Geneland GUI

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

Loading and conversions of coordinates

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

Settings and running MCMC

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

Running MCMC

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

Estimate F_{ST}

```r
# 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

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

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

Produce maps of respective inferred clusters

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

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

Determine which cluster is the best

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

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

MCMC chain, number of clusters and their map
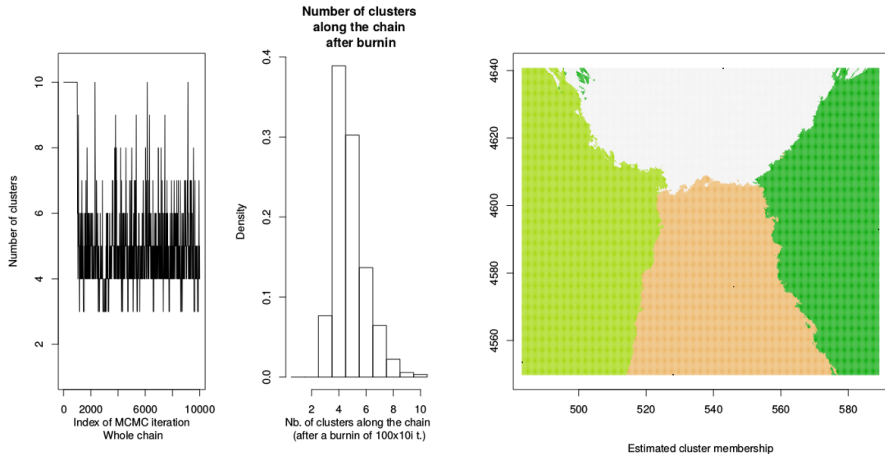
Map of posterior probability of belonging into cluster 1
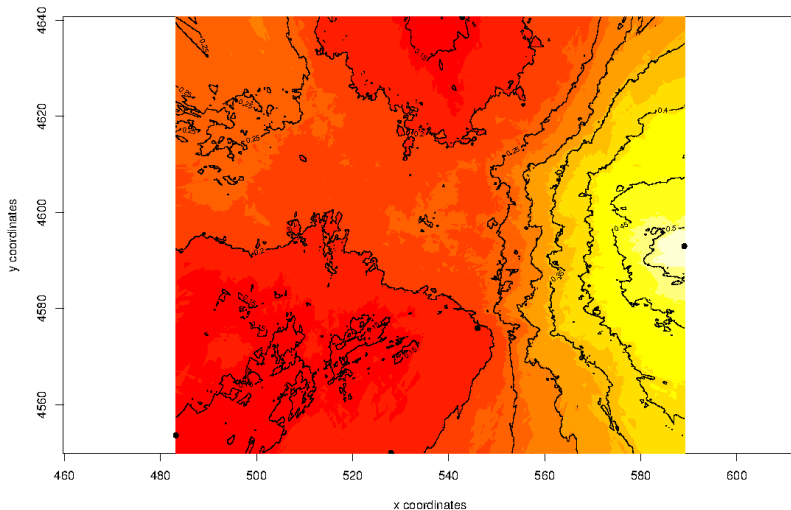
Very basic mapping in R

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

Basic map and Google map

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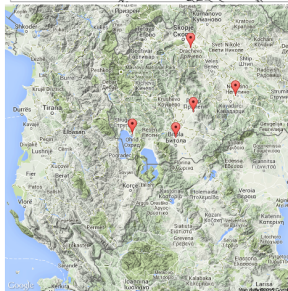
OpenStreetMaps and datasets from mapproj

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

Plotting on SHP files I

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

```r
library(maptools)
# Load SHP file
# Data from http://download.geofabrik.de/europe/macedonia.html
# Directory has to contain also respective DBF and SHX files
# (same name, only different extension)
# There are several functions readShape* - select appropriate
# according to data stored in respective SHP file
macedonia_building <- readShapeLines(fn="macedonia_buildings.shp")
plot(macedonia_building)
macedonia_landuse <- readShapeLines(fn="macedonia_landuse.shp")
plot(macedonia_landuse)
macedonia_natural <- readShapeLines(fn="macedonia_natural.shp")
plot(macedonia_natural)
macedonia_railways <- readShapeLines(fn="macedonia_railways.shp")
plot(macedonia_railways)
macedonia_roads <- readShapeLines(fn="macedonia_roads.shp")
```

Plotting on SHP files II

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

Plotting on SHP files III

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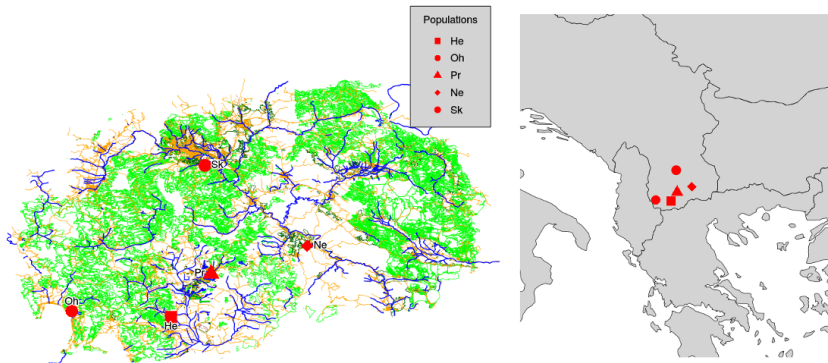
Structure

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

Structure work flow

1. Run multiple runs of Structure
   - User must test several numbers of genetic clusters (K) and test each several times
   - ParallelStructure (see later) can do this

2. Decide which K is the best – explore outputs
   - Plain command line Structure does not help with this
   - There is need for external application reading Structure output, summing them and helping decide which K is the best
   - Structure Harvester or Structure-sum R script (see later) can do this

3. Post process Structure outputs to prepare them for plotting
   - Sort and align Structure outputs
   - Probably most commonly done in CLUMPP

4. Plot the final graphs
   - Can be done in nearly any enough advanced graphical tool (including some R functions), probably the most commonly used is distruct

Running Structure in parallel with R

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

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

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
  - Make it as simple as possible – remove all unneeded columns
  - For population names use subsequent numbers from 1 to number of populations
  - For individual names use only alphanumerical characters

Input files

Joblist file:

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

Input file:

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

Running ParallelStructure

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

- Choose `n_cpu` according to your computer

- `structure_path` points to directory containing Structure binary

- `outpath` should aim to empty directory

- `plot_output=1` will produce plots for all runs

- Check all other settings according to Structure manual and your needs

- Get toy input file
  https://trapa.cz/sites/default/rcourse/hauss_stru.in and
  joblist https://trapa.cz/sites/default/rcourse/joblist.txt

ParallelStructure and Windows

- Authors do not recommend to run ParallelStructure on Windows...

- `parallel_structure()` uses for parallelisation library which is not available on Windows, instead try

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

- It may help to set `n_cpu=1`, but it is only for testing then, not for real work...

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

Post process Structure results – select the best K

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

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

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

  2   results_job_S010_f
  3   results_job_S011_f

  …   …

Run Structure-sum

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

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

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

Results from different data set, not from our toy
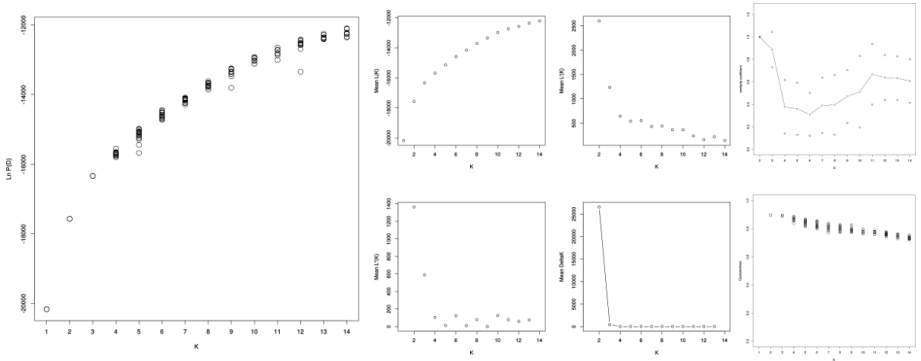
Example of final Structure plot

Drawn by distruct after alignment of Structure outputs by CLUMPP

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

t_12.1.popq

T. alatum T. ekmanii T. hemicyclum T. hepaticum T. interveniens T. macranthoides T. obtusifrons T. piceatum T. pulchrifolium

ALA   EKM   HEM   HEP   INT   MAC   OBT   PIC   PUL
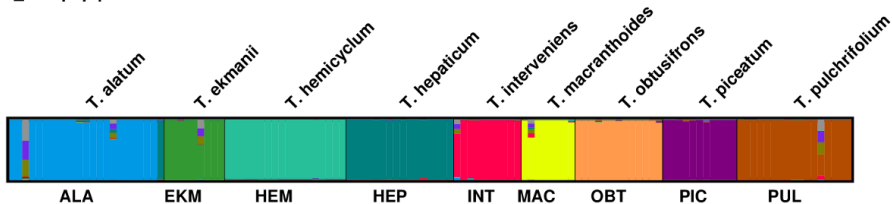
Multiple sequence alignment

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

Multiple sequence alignment with MAFFT

- MUSCLE is available in packages `muscle` and `ape` – first one reads "`*StringSet`" class R objects and writes "`*MultipleAlignment`" R objects; the latter reads and writes object of class "`DNAbin`"

- `ape` also contains functions to use Clustal and T-Coffee – both read and write `DNAbin`

- MAFFT is available from (same author) in packages ips and phyloch – both read and write `DNAbin`

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

Clustal, MUSCLE and T-Coffee from ape

```
1  meles.mafft
2  class(meles.mafft)
3  # read ?clustal and documentation of Clustal, Muscle and T-Coffee
4  # when using them to set correct parameters
5  meles.clustal <- ape::clustal(x=meles.dna, pw.gapopen=10, pw.gapext=0.1,
6    gapopen=10, gapext=0.2, exec="/usr/bin/clustalw2", quiet=FALSE,
7    original.ordering=TRUE) # Change "exec" to fit your path to clustal!
8  meles.muscle <- muscle(x=meles.dna, exec="muscle", quiet=FALSE,
9    original.ordering=TRUE) # Change "exec" to fit your path to muscle!
10 meles.muscle
11 class(meles.muscle)
12 # Plot the alignment - you can select which bases to plot
13 # and/or modify colors
14 image(x=meles.muscle, c("a", "t", "c" ,"g", "n"), col=rainbow(5))
15 # Add grey dotted grid
16 grid(nx=ncol(meles.muscle), ny=nrow(meles.muscle), col="lightgrey")
17 # Remove gaps from alignment - destroy it
18 meles.nogaps <- del.gaps(meles.muscle) # See ?del.gaps for details!
```

Multiple sequence alignment with MUSCLE

Cleaning the alignment

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

Read and write tree and drop tips

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

Extract clades from trees and drop extinct tips

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

Join two trees, rotate tree

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

Ladderize and (un)root the tree

```
1  # Ladderize the tree
2  plot.phylo(hauss.nj)
3  hauss.nj.ladderized <- ladderize(hauss.nj)
4  plot.phylo(hauss.nj.ladderized)
5  # Root the tree
6  plot.phylo(hauss.nj)
7  print.phylo(hauss.nj)
8  hauss.nj.rooted <- root(phy=hauss.nj, outgroup=10)
9  print.phylo(hauss.nj.rooted)
10 plot.phylo(hauss.nj.rooted)
11 # Root the tree interactive
12 plot.phylo(hauss.nj)
13 hauss.nj.rooted <- root(phy=hauss.nj, interactive=TRUE)
14 plot.phylo(hauss.nj.rooted)
15 # unroot the tree
16 unroot()
17 # Check if it is rooted
18 is.rooted()
```

Check tree and compute branch lengths and times

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

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

Topographical distances among matrices I – implementations

- Robinsons-Foulds distance in `phytools::multiRF`
  - The index adds 1 for each difference between pair of trees
  - Well defined only for fully bifurcating trees – if not fulfilled, some results might be misleading
  - Allow comparison of trees created by different methods

- Methods implemented in `ape::dist.topo` allow comparison of trees with polytomies (`method="PH85"`) or use of squared lengths of internal branches (`method="score"`)

- Final matrices are not Euclidean – may be problematic for usage in methods like PCA
  - Test it with `ade4::is.euclid`, can be scaled (forced to became Euclidean) by functions like `quasieuclid` or `cailliez` in `ade4` – carefully, it may damage meaning of the data

Topographical distances among trees II

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

```r
library(gplots)
library(corrplot)
library(phytools)
# Prepare matrix for distances
oxalis.trees.d <- matrix(nrow=length(oxalis.trees),
  ncol=length(oxalis.trees))
# Calculate pairwise topographic distances
for (i in 1:length(oxalis.trees)) {
  for (j in i:length(oxalis.trees)) {
    print(c(i,j))
    oxalis.trees.d[i,j] <- dist.topo(oxalis.trees[[i]],
      oxalis.trees[[j]])
  }
} # dist.topo can compare only two trees in one step... :-(
# Basic information about the distance matrix
dim(oxalis.trees.d)
head.matrix(oxalis.trees.d)
```

Topographical distances among trees III

- Post process the matrix and plot it
- There are several methods for calculating distance matrices among the trees – some take branch lengths into account, some only topology

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

Topographical distances among trees IV

- Calculate Robinsons-Foulds distance matrix among trees and plot it
- `phytools::multiRF` can handle `multiPhylo` objects and directly create matrices (no need to create loops)

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

Topographical distances among trees V – the matrices
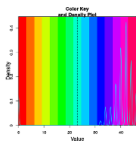
Consensus tree
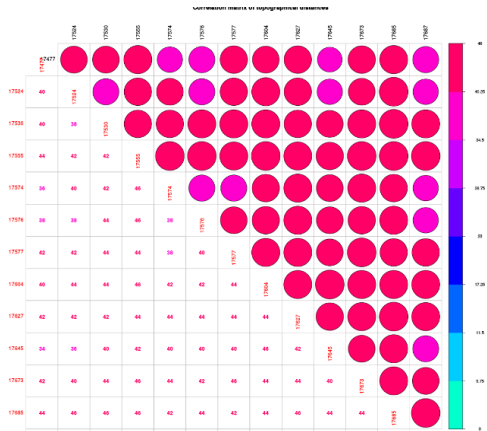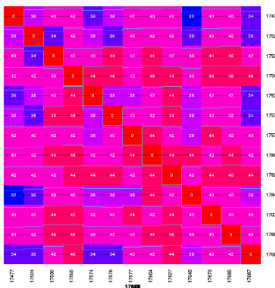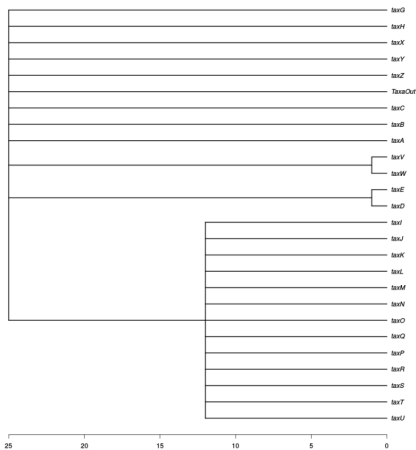

```r
1  # Root all trees
2  oxalis.trees.rooted <- lapply
3    (X=oxalis.trees, FUN=root,
4    "TaxaOut")
5  class(oxalis.trees.rooted) <-
6    "multiPhylo"
7  # Consensus tree (50 % rule)
8  oxalis.tree.con <- consensus
9    (oxalis.trees.rooted, p=0.5,
10   check.labels=TRUE)
11 print.phylo(oxalis.tree.con)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

Species tree – all trees must be ultrametric
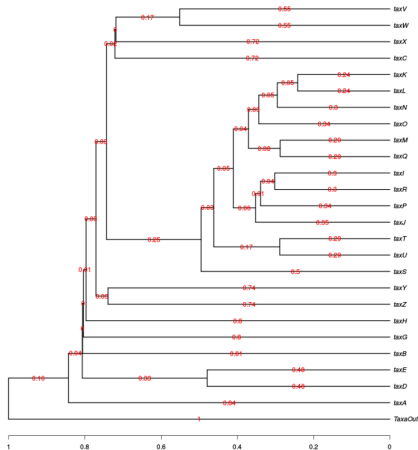

```
1  # Chronos scale trees
2  oxalis.trees.ultra <- lapply
3    (X=oxalis.trees.rooted,
4    FUN=chronos, model="correlated")
5  class(oxalis.trees.ultra) <-
6    "multiPhylo"
7  # Mean distances
8  oxalis.tree.sp.mean <- speciesTree
9    (oxalis.trees.ultra, mean)
10 # Plot the tree
11 plot.phylo(oxalis.tree.sp.mean,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

Parsimony super tree
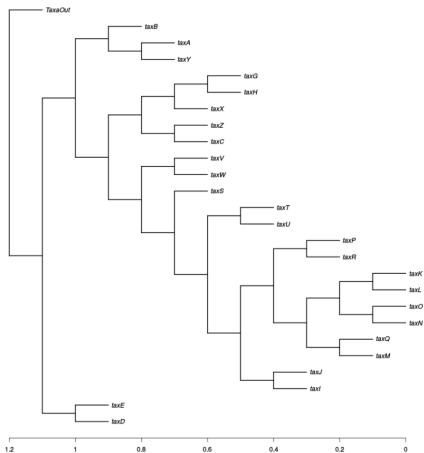

```
1 library(phangorn)
2 oxalis.tree.sp <- superTree(tree=
3   oxalis.trees.rooted, method=
4   "optim.parsimony", rooted=TRUE)
5 print.phylo(oxalis.tree.sp)
6 plot.phylo(oxalis.tree.sp,
7   edge.width=2, label.offset=0.01)
8 axisPhylo(side=1)
```

Density tree

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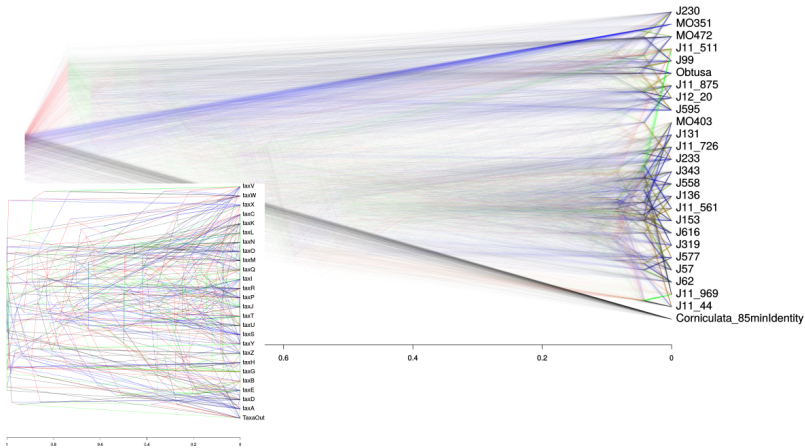
Networks

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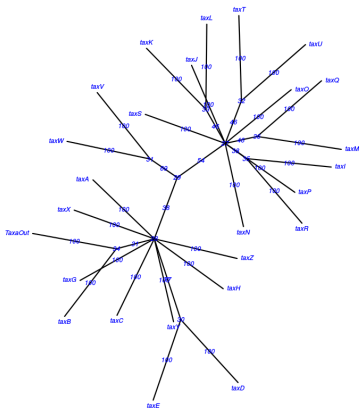


```
1  plot.networx(x=oxalis.tree.net,
2    planar=FALSE, type="2D",
3    use.edge.length=TRUE,
4    show.tip.label=TRUE,
5    show.edge.label=TRUE,
6    show.node.label=TRUE,
7    show.nodes=TRUE,
8    edge.color="black",
9    tip.color="blue")
10 plot.networx(x=oxalis.tree.net,
11   planar=FALSE, type="3D",
12   use.edge.length=TRUE,
13   show.tip.label=TRUE,
14   show.edge.label=TRUE,
15   show.node.label=TRUE,
16   show.nodes=TRUE, edge.color=
17   "black", tip.color="blue")
```

Kronoviz – see all trees on same scale
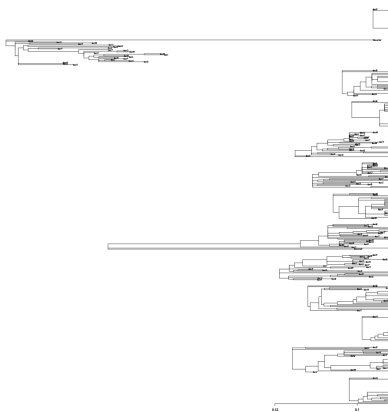

```
1  kronoviz(x=oxalis.trees.rooted,
2    layout=length(oxalis.trees.
3    rooted), horiz=TRUE)
4  # Close graphical device to
5  # cancel division of plotting
6  # device
7  dev.off()
```

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

Introduction | R | Data | Basic analysis | DAPC | SNP | Spatial analysis | Structure | Alignment | Trees | Evolution | The end

Manipulations | Seeing trees in the forest | MP | Comparisons | Notes about plotting the trees

Maximum parsimony

Maximum-parsimony tree of Meles
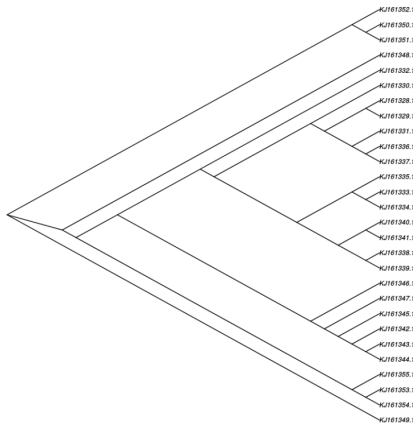
?parsimony # Parsimony details

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

Compare two trees

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

Cophyloplot comparing two trees
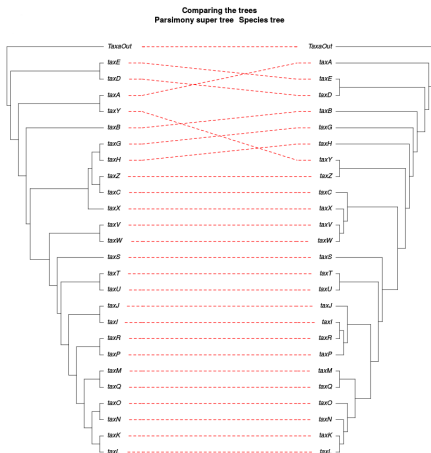
- **`ladderize()`** pre-sorts tips in

the tree – it helps to **`cophyloplot()`** to create better plot

- Automatic plot is usually not perfect – there use to be unneeded crossing lines – **`rotate=TRUE`** is recommended to can fix this manually by clicking to the nodes

- **`cophyloplot()`** has similar parameters like **`plot.phylo()`** – play with it and adjust in graphical editor

Change orientation of plots

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

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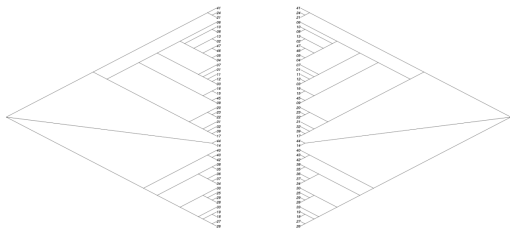
Highlighted labels
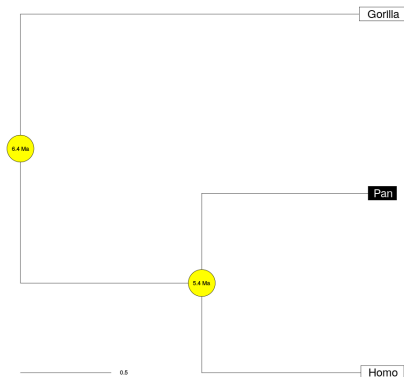

```
1  # Load tree in text format
2  trape <- read.tree(text=
3    "((Homo, Pan), Gorilla);")
4  # Plot the tree
5  plot.phylo(x=trape,
6    show.tip.label=FALSE)
7  # Add colored tip labels
8  tiplabels(trape[["tip.label"]],
9    bg=c("white", "black",
10   "white"), col=c("black",
11   "white", "black"),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

Phylogenetic independent contrast

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

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

PIC and its plotting

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

Plot of PIC (on the tree)
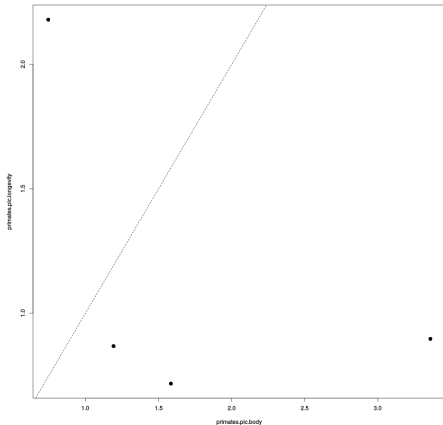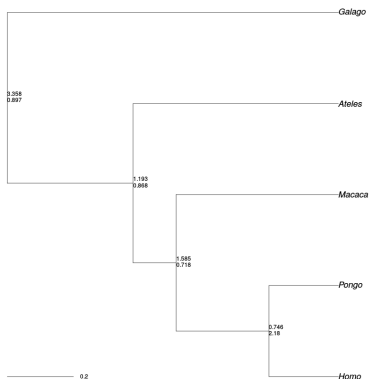
Test it

```
1  lm(formula=primates.pic.longevity~primates.pic.body)
2  Coefficients:
3    (Intercept)   primates.pic.body
4        1.6957              -0.3081
5  # Because PICs have expected mean zero - such linear regressions
6  # should be done through the origin (the intercept is set to zero)
7  lm(formula=primates.pic.longevity~primates.pic.body-1)
8  Coefficients:
9  primates.pic.body
10          0.4319
11 # Permutation procedure to test PIC
12 lmorigin(formula=primates.pic.longevity~primates.pic.body, nperm=1000)
13 Regression through the origin
14 Permutation method = raw data
15 Coefficients and parametric test results
16                 Coefficient Std_error  t-value  Pr(>|t|)
17 primates.pic.body   0.43193    0.28649   1.5077   0.2288
18 F-statistic: 2.273067 on 1 and 3 DF:
19   permutational p-value: 0.2377622
```

Intraspecific variation

- `pic.ortho()` requires list of measurements (vectors) for all taxa – their lengths can differ

- If we have sets of measurements in separated vectors (each vector has measurements for all taxa), we must for each `list` item use `cbind` to join columns and select appropriate line (from 1 to number of taxa)

- In example below, `jitter()` adds random noise

- Other usage is same in previous case…

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

Explanation of the cbind trick

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

Phylogenetic autocorrelation

- Autocorrelation coefficient to quantify whether the distribution of a trait among a set of species is affected or not by their phylogenetic relationships

- In the absence of phylogenetic autocorrelation, the mean expected value of I and its variance are known - it is thus possible to test the null hypothesis of the absence of dependence among observations

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

Testing of Moran's *I*

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

Correlogram to visualize results of phylogenetic autocorrelation analysis

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

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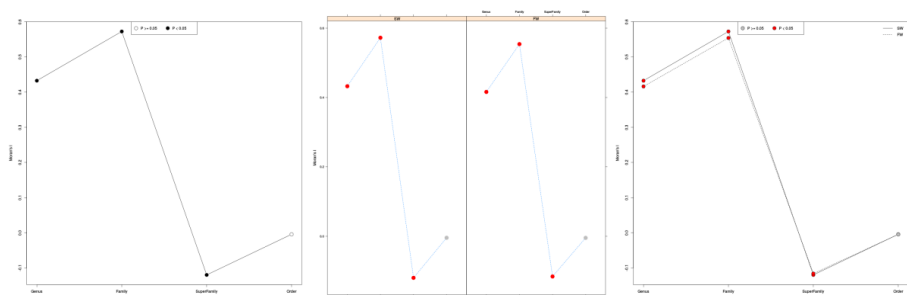
Prepare toy data set (tree)

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

Modified tree
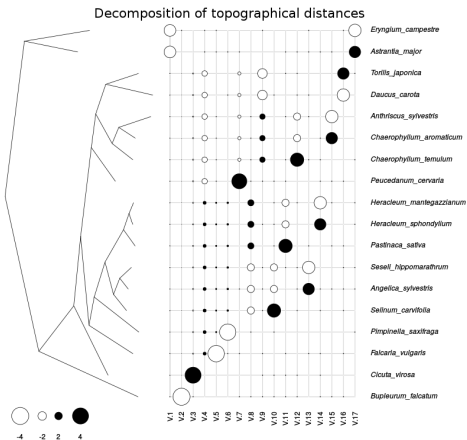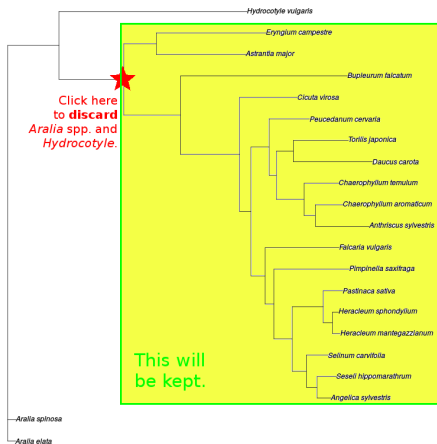

```
1  # Decomposition of topographical distances (right plot)
2  table.phylo4d(x=phylo4d(x=apiaceae.tree, tip.data=treePart
3    (x=apiaceae.tree, result="orthobasis")), treetype="cladogram")
```

Prepare toy data set (the variable)

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

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

Orthonormal decomposition - phylogenetic eigenvector regression

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

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

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

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

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

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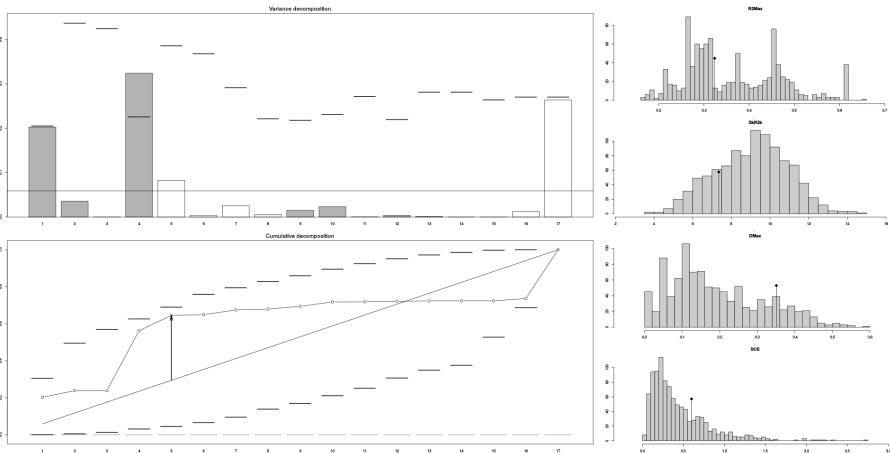
- Observed value is within permutations – no significant phylogenetic signal...

Phylogenetic Generalized Least Squares

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

```r
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

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

Results of PGLS
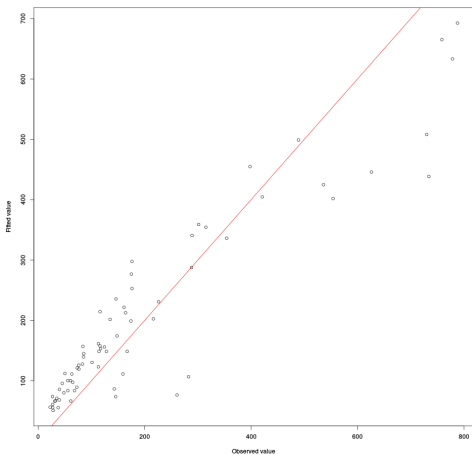
- `pgls()` uses maximum likelihood to test for phylogenetic signal

- The signal is clearly presented

- Usually, tuning the model (possible data transformations and or changing model parameters) is necessary to find the best model – AIC helps

- See caper manual for details

Generalized Estimating Equations

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

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

Not significant in this case…

```
1 Call: compar.gee(formula = primates.longevity ~
2   primates.body, phy = primates.tree)
3 Number of observations: 5
4 Model:
5                     Link: identity
6 Variance to Mean Relation: gaussian
7 QIC: 7.310142
8 Summary of Residuals:
9        Min          1Q      Median          3Q         Max
10 -0.8031302 -0.0132754   0.0999588   0.1988258   0.2862064
11 Coefficients:
12               Estimate       S.E.         t Pr(T > |t|)
13 (Intercept)   1.0670417 0.5838429 1.827618    0.2695894
14 primates.body 0.8497249 0.2157006 3.939372    0.1101432
```

Phylogenetic signal

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

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

- If Blomberg's values of 1 correspond to a Brownian motion process, which implies some degree of phylogenetic signal or conservatism

- K values closer to zero correspond to a random or convergent pattern of evolution, while K values greater than 1 indicate strong phylogenetic signal and conservatism of traits

- Blomberg's K statistic of phylogenetic signal

Analyze multiple traits in once

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

When there are vectors with standard errors of measurements

- Functions for testing of phylogenetic signal do not work with more measurements per taxon
  - Currently, the only possibility is `phylosig()` which is able to work with SE (user must prepare this vector from the data manually)

- `phylosig()` can be used as an alternative to `phylosignal()` – the functions are similar in basic usage

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

Alternative testing for phylogenetic signal with GLM

- It is possible to use intercept-only (`model`/`formula` will be something like `variable ~ 1`, not `variable1 ~ variable2`) GLM to quantify phylogenetic signal in trait

- It is tricky to select the best correlation structure – AIC can help with selections

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

Phylogenetic principal component analysis

PCA corrected for phylogeny

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

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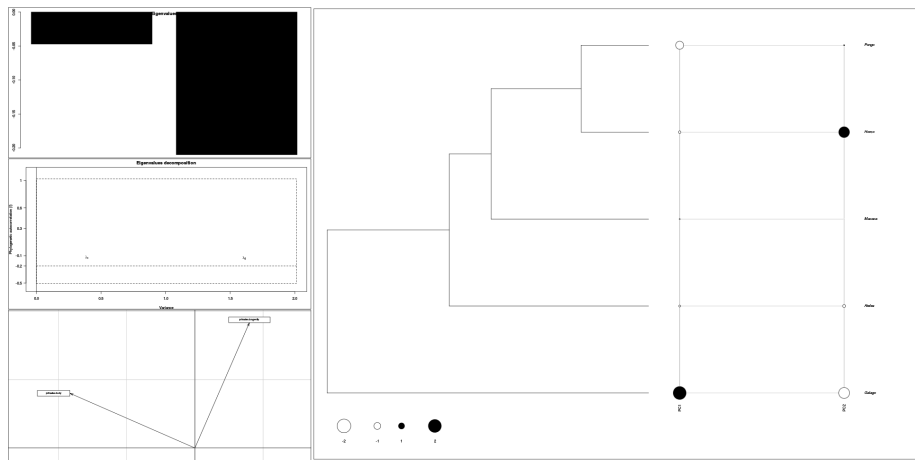
Ancestral state reconstruction

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

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

Ancestral state reconstructions of primates body weights

Another possibilities (package phytools)

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

Bayesian ancestral character estimation

```
 1 primates.body.ace.bayes # Print the output object and check it
 2       gen      sig2       '6         7         8         9'    logLik
 3  [1,]     0 1.391013 2.288170 2.288170 2.288170 2.288170 -13.238593
 4  [2,]   100 1.394484 1.742455 2.248855 2.648808 3.105533  -7.552295
 5  [3,]   200 1.280646 1.501700 2.514334 2.524295 3.251273  -7.565108
 6  [4,]   300 1.230536 1.433547 2.242559 2.593056 2.725938 -10.204376
 7  [5,]   400 1.414644 1.648370 2.178676 2.573255 3.381587  -6.949438
 8  [6,]   500 2.069528 2.205779 2.111277 2.966358 3.361720  -8.461222
 9  [7,]   600 2.314460 2.361329 3.006070 2.995382 3.885636  -8.059215
10  [8,]   700 2.808398 3.119423 3.621859 3.504098 3.736082  -9.159853
11  [9,]   800 3.101082 2.281787 3.497516 2.526587 3.146811 -11.130158
12 [10,]   900 3.110501 2.971506 2.649267 2.913260 4.132872  -9.346940
13 [11,]  1000 2.361981'1.819626 2.674728 2.814608 3.412479' -7.964549
14 # We need node labels (nodes are numbered - here columns "6",
15 # "7", "8" and "9") from the last Bayes generation (here line 11)
16 primates.body.ace.bayes[11,3:6] # Use it for nodelabels()
17        6        7        8        9
18 1.819626 2.674728 2.814608 3.412479
```

Continuous map
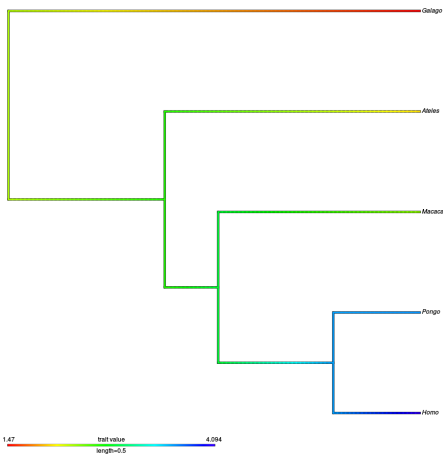

```
 1  library(phytools)
 2  contMap(tree=primates.tree,
 3    x=primates.body)
 4  # Change colors with setMap()
 5  primates.contmap <- setMap(x=
 6    contMap(primates.tree,
 7    primates.body),
 8    colors=c("white", "black"))
 9  plot(primates.contmap)
10  # See ?par for more settings
```
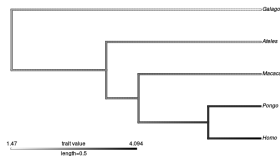
Display more characters on a tree in a table

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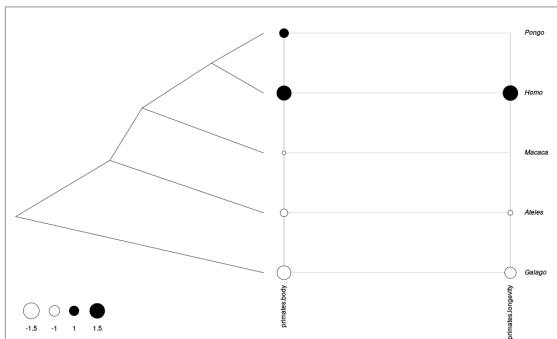
Phenogram

Vertical axis shows character values

```
1 phenogram(tree=primates.tree, x=primates.longevity, fsize=1.2,
2   ftype="i", colors="red", main="Longevity")
3 fancyTree(tree=primates.tree, type="phenogram95", x=primates.longevity,
4   fsize=1.2, ftype="i", main="95-percentile of longevity")
```
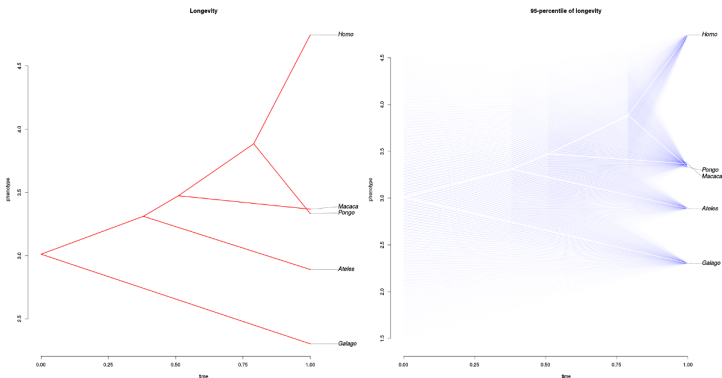
Display 2 continuous characters in space and 3D tree connecting them

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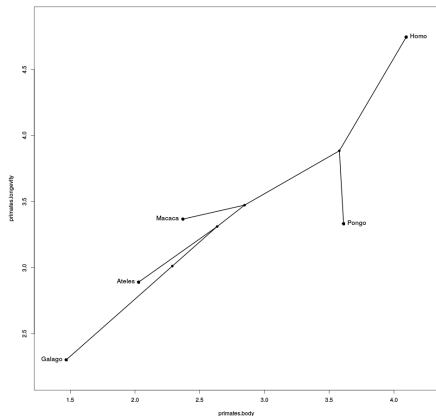
Combine phenograms and ancestral state reconstructions

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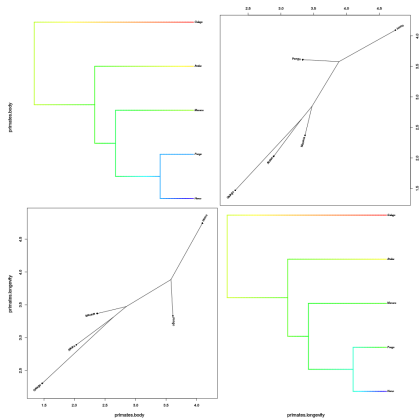
Install package from GitHub

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

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

Simple loop

- Loops repeat one task given number of time
- variable `i` has changing value for every repetition
- It is possible to use variables or numeric output of functions in `from:to` expression

```r
1  # Simplest loop - print value of "i" in each step
2  for (i in 1:5) { print(i) }
3  # In every step modify value of variable "X"
4  X <- 0
5  for (i in 10:1) {
6    print(i)
7    X <- X+i
8    print(X)
9    }
10 # Work on each item of a list object
11 for (L in 1:length(nothofagus.sequences)) {
12   print(length(nothofagus.sequences[[L]])) }
```

Simple function

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

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

R script and its running from command line

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

Citations

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

Where to look for the help I

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

- List of R documentation
  https://cran.r-project.org/manuals.html

- R phylogeny mailing list
  https://stat.ethz.ch/mailman/listinfo/r-sig-phylo

- R genetics mailing list
  https://stat.ethz.ch/mailman/listinfo/r-sig-genetics

- Bioconductor home page https://bioconductor.org/

- Bioconductor support forum
  https://support.bioconductor.org/ and Bioconductor help
  pages https://master.bioconductor.org/help/

Where to look for the help II

- R phylo wiki `http://www.r-phylo.org/wiki/Main_Page`
- R phylogenetics at CRAN
  `https://cran.r-project.org/web/views/Phylogenetics.html`
- Integrated documentation search
  `http://www.rdocumentation.org/`
- RForge package repository `https://r-forge.r-project.org/`
  (with documentation)
- Little Book of R for Bioinformatics
  `https://a-little-book-of-r-for-bioinformatics.`
  `readthedocs.org/en/latest/`
- Little Book of R for Multivariate Analysis
  `https://little-book-of-r-for-multivariate-analysis.`
  `readthedocs.org/en/latest/`

Where to look for the help III

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

Where to look for the help IV

- Phytools http://phytools.org/, its blog http://blog.phytools.org/ and GitHub page https://github.com/liamrevell/phytools

- Poppr documentation http://grunwaldlab.cgrb.oregonstate.edu/primer-population-genetic-analyses-r/installation and forum https://groups.google.com/forum/#!forum/poppr

- ade4 home page http://pbil.univ-lyon1.fr/ADE-4/ade4-html/00Index.html?lang=eng and documentation http://pbil.univ-lyon1.fr/ade4/home.php?lang=eng

- Phangorn resources http://cran.r-project.org/web/packages/phangorn/index.html

- R help mailing list https://stat.ethz.ch/mailman/listinfo/r-help (web interface http://r.789695.n4.nabble.com/)

Where to look for the help V

- R announce mailing list
  https://stat.ethz.ch/mailman/listinfo/r-announce
- R ecology mailing list
  https://stat.ethz.ch/mailman/listinfo/r-sig-ecology
- Books about R
  https://www.r-project.org/doc/bib/R-books.html
- R at StackOverflow StackExchange
  https://stackoverflow.com/questions/tagged/r
- R at CrossValidated StackExchange
  https://stats.stackexchange.com/questions/tagged/r
- The R journal https://journal.r-project.org/
- R-bloggers – aggregation of R blogs http://www.r-bloggers.com/

Where to look for the help VI

- R on The Molecular Ecologist
  http://www.molecularecologist.com/category/software/r/

- R tutorial http://www.r-tutor.com/

- Cookbook for R http://www.cookbook-r.com/

- Spatial R https://sites.google.com/site/spatialr/

- R for open big data http://ropensci.org/

- Statistics with R http://zoonek2.free.fr/UNIX/48_R/all.html

- The R Inferno book http://www.burns-stat.com/documents/books/the-r-inferno/

- Springer R series
  https://www.springer.com/series/6991?detailsPage=titles

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

Where to look for the help VII

- plyr documentation http://plyr.had.co.nz/ – manipulation with data

- Learning R https://learnr.wordpress.com/

- R for Community Ecologists
  http://ecology.msu.montana.edu/labdsv/R/

- Try R on-line course full of adventure anf heroic quests
  http://tryr.codeschool.com/

- Quick-R learning resource http://statmethods.net/

- R manual and help search http://finzi.psych.upenn.edu/

- Biostars – general bioinformatics forum
  https://www.biostars.org/

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

Where to look for the help VIII

- Do not hesitate to ask on the forum or contact author of package with which you have problem, preferably through some public forum or mailing list, they usually respond quickly and helpfully…

- Uncle Google is your friend (*"how to XXX in R"*)…

- R packages commonly contain vignettes (tutorials) – list them by `vignette()` and load selected by `vignette("VignetteName")`

- List available training datasets from various R packages by `data()` and load selected by `data(DatasetName)`

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

- **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

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

Packages we used… I
We used following packages – but not all functions – explore them for more possibilities

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

Packages we used… II

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

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

Packages we used… III

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

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

Packages we used… IV
We used following packages – but not all functions – explore them for more possibilities

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

Packages we used… V

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

- seqinr: exploratory data analysis and data visualization for biological sequence

- seqLogo: sequence logos for DNA sequence alignments

- sos: searches contributed R packages

- sp: classes and methods for spatial data

- spdep: spatial dependence: weighting schemes, statistics and models

- TeachingDemos: demonstrations for teaching and learning

- vegan: community ecology

- XVector: representation and manpulation of external sequences

Another interesting packages (we did not use)… I
For your own explorations…

- **adhoc**: ad hoc distance thresholds for DNA barcoding identification

- **apex**: analysis of multiple gene data

- **apTreeshape**: analysis of phylogenetic tree topologies

- **BAMMtools**: analyzing and visualizing complex macroevolutionary dynamics on phylogenetic trees

- **bayou**: Bayesian fitting of Ornstein-Uhlenbeck models to phylogenies

- **betapart**: partitioning beta diversity into turnover and nestedness components

- **Biodem**: biodemography

- **BioGeoBEARS**: probabilistic inference of both historical biogeography as well as comparison of different models of range evolution

Another interesting packages (we did not use)… II

For your own explorations…

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

Another interesting packages (we did not use)… III
For your own explorations…

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

Another interesting packages (we did not use)… IV

For your own explorations…

- ggtree: visualization and annotation of phylogenetic trees

- HardyWeinberg: statistical tests and graphics for HWE

- HMPTrees: models, compares, and visualizes populations of taxonomic tree objects

- hwde: models and tests for departure from HWE and independence between loci

- HWxtest: tests whether a set of genotype counts fits the HW expectations

- HyPhy: macroevolutionary phylogentic analysis of species trees and gene trees

- iteRates: iterates through a phylogenetic tree to identify regions of rate variation

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

For your own explorations...

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

Another interesting packages (we did not use)… VI

For your own explorations…

- **mvMORPH**: multivariate comparative tools for fitting evolutionary models to morphometric data

- **ouch**: Ornstein-Uhlenbeck models for evolution along a phylogenetic tree

- **OUwie**: analysis of evolutionary rates in an OU framework

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

- **paleotree**: paleontological and phylogenetic analyses of evolution

- **paleoTS**: analyze paleontological time-series

- **pastis**: phylogenetic assembly with soft taxonomic inferences

- **PBD**: protracted birth-death model of diversification

- **PCPS**: principal coordinates of phylogenetic structure

Another interesting packages (we did not use)… VII

For your own explorations…

- **phyclust**: phylogenetic clustering

- **phyloclim**: integrating phylogenetics and climatic niche modeling

- **PHYLOGR**: manipulation and analysis of phylogenetically simulated data sets and phylogenetically based analyses using GLS

- **phyloland**: models a space colonization process mapped onto a phylogeny

- **phylolm**: phylogenetic linear models and phylogenetic generalized linear models

- **phyloTop**: calculating and viewing topological properties of phylogenetic trees

- **phylotools**: supermatrix for DNA barcodes using different genes

- **plyr**: splitting, applying and combining data

Another interesting packages (we did not use)… VIII
For your own explorations…

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

Another interesting packages (we did not use)… IX

For your own explorations…

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

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

For your own explorations...

- TESS: simulation of reconstructed phylogenetic trees under tree-wide time-heterogeneous birth-death processes and estimation of diversification parameters under the same model

- treebase: discovery, access and manipulation of TreeBASE phylogenies

- TreePar: estimating birth and death rates based on phylogenies

- TreeSim: simulating phylogenetic trees

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

Orientation in so many packages...

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

The methods are over

- We went in more or less details through plenty of methods to work with molecular data to analyze phylogeny, population genetics, evolution and so on in R

- There are many more methods to try…

- It is nearly impossible to go in reasonable time through all relevant R tools – a lot of space for you

The end
Our course is over…

…I hope it was helpful for You…

…any feedback is welcomed…

…happy *R* hacking…

… any final questions?

Typesetting using X∃LATEX on openSUSE GNU/Linux, November 23, 2016.