An introduction to the programming language for environmental researchers

NIVA: Norges ledende kompetansesenter på vannmiljø
Abstract

The seminar "An introduction to the programming language R for environmental researchers" was arranged at NIVA in 2006. The objective was to help NIVA researchers get familiar with R, because it is freely available, under rapid development, and widely used by environmental scientists world-wide. This report contains the presentations, examples and exercises from the seminar. The main components of the seminar were: (1) Introduction: the R language, scripts, importing data, plotting, etc. (2) From linear models to GAM: linear regression, generalised linear models (GLM), generalised additive models (GAM), model selection. (3) Time series: trends, seasons, autocorrelation, structural changes. (4) Multivariate models: the package "vegan", community analysis, non-metric multidimensional scaling (NMDS), canonical correspondence analysis (CCA), constrained ordination. (5) Data visualization: colour gradients, 3-D GAM plots, lattice. The data and scripts used in the examples are available upon request to the authors.

4 keywords, Norwegian
1. Programmeringsverktøy
2. Statistisk analyse
3. Matematisk modellering
4. Visualisering

4 keywords, English
1. Programming tool
2. Statistical analysis
3. Mathematical modelling
4. Visualization

Jannicke Moe
Project manager

Unn Hilde Refseth
Research manager

Harsha Ratnaweera
Strategy Director

ISBN 978-82-577-5259-0
NIVA internal project

An introduction to the programming language R

for environmental researchers
Preface

The seminar "An introduction to the programming language R for environmental researchers" was arranged for researchers at NIVA in 2006. Many types of statistical software are currently in use at NIVA. If more researchers used the same software, we would have a common platform for sharing scripts, data and experiences, which would facilitate collaboration. There are many good types of software available for statistical analysis, but the object-oriented programming software R has three unbeatable benefits.

1) It is freely available and can be downloaded from Internet (http://www.r-project.org), no licence or dongles are required
2) It is under rapid development and the list of available tools ("packages") contributed from users is growing steadily; many cutting-edge methods are available (e.g. GAMs and smoothers, quantile regression, mixed-effect models, community analysis, breakpoint detectors)
3) There is a large and expanding scientific community of R users, with newsletters and web fora for questions and discussions.

Moreover, this software includes all common statistical methods as well as all options for organisation of data. Therefore, all operations from formatting of data to analysis and plotting can be done within one script. This also means that all steps of data treatment and analysis can easily be communicated between researchers via email, which is an enormous benefit when collaborating internationally. However, since R is completely text-based, researchers usually need some introductory training to get familiar with this programme.

The examples given in this report are based on research problems and data provided by participants (Per Stålnacke, Heleen de Wit, and Hilde Trannum). The data and the script files are available upon request (jannicke.moe@niva.no, robert.ptacnik@niva.no).

Oslo, 20.12.2007

Jannicke Moe
## Contents

1. **INTRODUCTION** 8  
1.1 The award-winning S language 8  
1.1.1 The R language and environment 8  
1.1.2 Differences between R and S-plus 8  
1.1.3 Installing R 8  
1.1.4 A first encounter with R 9  
1.2 Object-oriented programming in R 9  
1.2.1 Data frames in R 9  
1.2.2 Importing data into R 9  
1.2.3 Basic R graphics 10  
1.2.4 Scripts in R 11  
1.2.5 Control structures in R 11  
1.2.6 User-defined functions in R 11  
1.2.7 Getting help 11  
1.3 Example 1: Plot snow cover data 12  
1.4 Example 2: Explore and plot water chemistry data 18  

2. **FROM LINEAR MODELS TO GAM** 31  
2.1 Linear models: what sort of test should I use? 31  
2.2 Linear models in R: some useful commands 31  
2.3 Formula syntax in R – general rules 32  
2.4 Generalised linear models (GLM) 33  
2.5 "Families" of distributions in R 33  
2.6 Generalised Additive Models (GAM) 34  
2.7 Model selection 35  
2.7.1 Model selection for LM: ANOVA table 35  
2.7.2 Model selection for GLM: deviance analysis 36  
2.7.3 Akaike’s information criterion 36  
2.7.4 Automatic stepwise selection 36  
2.7.5 Model selection by cross validation 37  
2.8 Example 3: Nitrate concentrations in the river Storgama 37  
2.8.1 Script 1: Import and formatting of dates. 37  
2.8.2 Script 2: Linear models. 40  
2.8.3 Script 3: Generalised linear models. 44  
2.8.4 Script 4: Generalised additive models. 49  

3. **TIME SERIES** 53  
3.1 Trends, seasons and autocorrelation 53  
3.1.1 Trend analysis by regression 55  
3.1.2 Make time-series object 58  
3.1.3 Trend analysis by non-parametric trend test 61  
3.1.4 Trend analysis by seasonal decomposition 61  
3.1.5 Autocorrelation 63  
3.2 Structural changes in time series 66
4. MULTIVARIATE MODELS IN VEGAN

4.1 Multivariate analyses: Examples
4.1.1 Example 4: Basics in vegan
4.1.2 Example 5: Applications of NMDS & CCA to benthos data
4.1.3 Example 6: constrained ordination. Extracting indicator values from an ordination.

5. DATA VISUALIZATION AND LATTICE (Robert Ptacnik)

Tables
Table 1. Basic linear models for different combinations of predictor variables, with corresponding model formula in R
Table 2. Families of distributions applicable for Generalised Linear Models in R

Figures
Figure 1. Plots for the data set snow.cover. Left panel: points; right panel: lines.
Figure 2. Plots for the data set snow.cover, shown as anomalies (deviations from the mean).
Figure 3. Plots for the data set snow.cover: absolute values (upper) and anomalies (lower).
Figure 4. Histogram for NIVA chemistry data: number of observations per Fylke (county).
Figure 5. Histogram for NIVA chemistry data: number of observations per Fylke (county), with bins as integers.
Figure 6. Histogram for NIVA chemistry data: number of observations per Fylke (county), with Fylke as integer.
Figure 7. Barplot of NIVA chemistry data: number of observations per Fylke.
Figure 8. Boxplot of NIVA chemistry data: Chloride per Fylke.
Figure 9. Boxplot of NIVA chemistry data: Chloride per Fylke (excluding Oslo and Vestfold).
Figure 10. Boxplot for all variables of the NIVA chemistry data.
Figure 11. Scatterplot matrix for all variables of the NIVA chemistry data.
Figure 12. Scatterplot matrix with smoothed functions for all variables of the NIVA chemistry data.
Figure 13. Biplot of principal components of major ions, showing the contribution from sea salts (Na and Cl) and weathering (Ca and HCO₃⁻).
Figure 14. Relations of major ions to chloride, compared to the standard composition of seawater.
Figure 15. HCO₃⁻ is definitely a major determinant of ANC.
Figure 16. Measured vs. predicted specific conductivity (as the sum of the equivalent conductivities of the individual major ions).
Figure 17. Histogram for of differences between measured and predicted conductivities, showing two possible outliers.
Figure 18. The contribution of H⁺ to specific conductivity is highest when ANC is low.
Figure 19. Binary data: linear regression (left panel) versus logistic regression (right panel).
Figure 20. Regression with linear model (LM; left panel) versus generalised additive model (GAM; right panel).
Figure 21. Relationship between LM, AM, GLM and GAM.
Figure 22. Time-series plots for Storgama.
Figure 23. Pairwise plots of parameters from Storgama.
Figure 24. Data series NO3_N from Storgama.
Figure 25. Barplot for estimated coefficients per month for Storgama.
Figure 26. Linear regression of pH vs. NO3 in Storgama.
Figure 27. Linear regression of binary transformed pH vs. NO3 in Storgama.
Figure 28. Linear regression (red) and logistic regression (green) of binary transformed pH vs. NO3 in Storgama.
Figure 29. Generalised additive model (GAM) regression of pH vs. NO3 in Storgama.
Figure 30. Generalised additive model (GAM) regression of pH vs. NO3 in Storgama, with high and low degree of smoothing.
Figure 31. Barplot for estimated coefficients per month for Storgama.
Figure 32. Each chemical variable in the data set glomma plotted against time.
Figure 33. Trend estimations for TN from Glomma. Upper panel: Effect of month on TN, estimated by GAM. Middle panel: linear effect of year on TN. Lower panel: comparison of data and model prediction.
Figure 34. Trend estimations for TN from Glomma. Upper panel: Effect of month on TN, estimated by GAM. Middle panel: effect of year on TN, estimated by GAM. Lower panel: comparison of data and model prediction.
Figure 35. Time-series plot for the data set glomma.
Figure 36. Time-series plot of time-series object glomma.ts.
Figure 37. Seasonal decomposition of time-series object glomma.ts.
Figure 38. Seasonal decomposition of time-series object glomma.ts, with more smoothing (left panel) and less smoothing (right panel).
Figure 39. Autocorrelation plots for dataset glomma (upper panel) and for time-series object glomma.ts.
Figure 40. Autocorrelation plots for remainders (residuals) from AR-model of dataset glomma (upper panel); for the estimated temporal trend (middle panel); and for the seasonal effect (lower panel).
Figure 41. Number of missing TotP values per year, in the data from Ringkøbingfjord.
Figure 42. Time-series plot of data from Ringkøbingfjord.
Figure 43. Breakpoint analyses for data from Ringkøbingfjord.
Figure 44. Breakpoint analysis for relationship between Total P and chlorophyll from Ringkøbingfjord.
Figure 45. Non-metric multidimensional scaling (NMDS) for the varespec-data.
Figure 46. PCA of same data as in Figure 45.
Figure 47. NMDS with surface fits for the environmental variables ‘Al’ (red) and ‘Humdepth’ (green).
Figure 48. Histogram of no. of sites per taxon.
Figure 49. Histogram of no. of sites per taxon.
Figure 50. Command pairs: pair wise scatterplots of environmental variables.
Figure 51. NMDS of contaminants.
Figure 52. NMDS for all species.
Figure 53. Procrustes rotation comparing the NMDS of species and contaminants.
Figure 54. Various ordinations and procrustes errors between overfitted CCA and unconstrained PCA (lower left).
Figure 55. Number of species per site plotted against different environmental factors.
Figure 56. Pair-wise scatterplots of environmental variables.
Figure 57. PCA of the phytoplankton matrix.
Figure 58. PCA of the phytoplankton matrix. The colorcode illustrates the Chlorophyll-a concentration (dark=low, light=high).
Figure 59. CCA of the phytoplankton data with three predictors.
Figure 60. CCA against Chlorophyll-a where the effects of CA and Depth are partialled out.

Figure 61. Dotchart illustrating the species optima.
Figure 62. The calculated trophic scores plotted against the chlorophyll-a concentration.
Figure 63. Histogram of the species’ ‘goodness’ in the CCA.
Figure 64. As Figure above, but for selected species only.
Figure 65. Example for how to code a color gradient. The y-values range from 0-30 and code the corresponding cyan-magenta colors: cm.colors(30).
Figure 66. Different default color gradients on grey background. (cm.colors(), heat.colors(), rainbow()).
Figure 67. Mean depth plotted against total phosphorus, and showing chlorophyll-a as color gradient. The labels were produced using the ‘identify’ function.
Figure 68. 3-dimentional interpolation of the data shown in Figure 67, showing the third variable (here chl-a) as contour lines (function contour()).
Figure 69. Example for the filled.contour() function.
Figure 70. A scatterplot can be difficult to read when too many dots are plotted together.
Figure 71. 2-dim density estimation of the data shown in Figure 70.
Figure 72. A density plot, indicating the number of observations along a gradient.
Figure 73. Implementation of the density function in the lattice package. The data are split by countries.
Figure 74. A boxplot produced with the lattice command bwplot.
Figure 75. Standard scatterplot
Figure 76. xyplot produces scatterplots (and others, see ?xyplot). Countries are grouped by a color-code.
Figure 77. xyplot. Data split by countries into sub-panels.
Figure 78. xyplot. Combining split-and group function.
Figure 79. xyplot. Two or more variables can be plotted together using a function argument in the command.
Figure 80. Example of a barchart plot.
Figure 81. The dotplot function.
1. INTRODUCTION

Tom Andersen

1.1 The award-winning S language

The origin of R is S, an object-oriented language for statistical computing. This is used for “Programming with data” - it is thus a whole language, not just a statistical package. S was developed in AT&T labs (now Lucent Technologies) by Richard A. Becker, John M. Chambers, and Allan R. Wilks. The 1998 ACM Software Systems Award was given to John M. Chambers for the S language. The reason was that Chambers's work has “forever altered the way people analyze, visualize, and manipulate data...”

S-plus is a value-added version of S sold by Insightful Corporation (formerly MathSoft, Inc.). S-plus has a GUI (graphical user interface) and add-on packages.

1.1.1 The R language and environment

R is a General Public License implementation of S. It was initiated by Ross Ithaka and Robert Gentleman. So R may stand for “Ross & Robert” – supposedly. Or “R is before S” (in the alphabet).

R is freely downloadable and available for several platforms: Unix/Linux, MacOS, Windows. It has an active user community and many add-on packages

1.1.2 Differences between R and S-plus

The rationale behind R is that: “R should make it easier to detect programming errors, while at the same time being as compatible as possible with S”.

S-plus stores objects as separate files, whereas R objects are stored internally. R is generally faster than S-plus, especially to load. But - R objects are lost if R crashes! You should save R workspace to avoid this.

It is important to be aware of certain differences: (the meaning will become clear later...):

- Powers in formulae must be in “insulated” in R
  S-plus: \( y \sim x + x^2 \)
  R: \( y \sim x + I(x^2) \)
- R uses different default contrasts in (generalized) linear models
  S-plus: Helmert contrasts
  R: Treatment contrasts

1.1.3 Installing R

R can be installed without administrator privileges! Go to http://cran.r-project.org/ (CRAN = Comprehensive R Archive Network). Select a Precompiled Binary Distribution: Linux, MacOS, or Windows.

Windows installation:
- Select “Windows (95 or later)”
- Select “Base distribution”
- Download and run “rw2011.exe” (or the latest version)

Packages can be installed from the menu Packages -> Install package(s). Installation from local zip files is possible, but NOT recommended. When installed, a package can be loaded to the workspace by require() or by library().
1.1.4 A first encounter with R

Create a directory (folder) on your home area called “R-seminar” (for example).
Start R.
Select from the menu: File -> Change dir...
Select the directory you just created. All files will now be loaded from and saved to this directory.

Type “demo(graphics)” in the “R console” window to see some capabilities of R. Press enter (return) until the demo ends. Notice what happens in the console and graphics windows.

R returns answers to arithmetical expressions typed in the console window.
– Arithmetic operators (+-*/)
– Transcendental functions: sin, cos, exp, log, etc.
– Etc.

Expressions can include named variables.
Expressions can be assigned to variables. R uses a special assignment operator: "<-"

Remember:
– R uses Anglo-American decimal separator (period, not comma).
– R is case-sensitive (e.g. ANC ≠ anc)
– Lines starting with # are ignored (comments).

1.2 Object-oriented programming in R

Objects in R can be lists. Lists are in turn ordered collections of objects.
Arrays are lists of objects of a single type: Numerical, character, or logical.
List components can be named:
> names(object)
How to construct a list:
> list(object1, object2, ...)
List components can be accessed in different ways:
> object$component
> object[index]
> attach(object)
> detach(object)

1.2.1 Data frames in R

Data tables are usually arranged as the object type data.frame, with different variables in columns and subjects/sites/samples in rows. A data frame is a list of data arrays of same length, but not necessarily of same type (continuous, categorical etc.). Both rows and columns of a data frame can be named. The elements can be accessed by names, or by indexing.
Whole column:
> frame$column.name
> frame[, "column.name"]
> frame[, column.number]
Single element:
> frame[row.number, ]$column.name
> frame["row.name", "column.name"]
> frame[row.number, column.number]

1.2.2 Importing data into R
In order to read a data file, you need to set the correct search path first. With the command window in front, select from the menu:

File -> Change dir...

and navigate to the directory where your file is located.

Alternatively, you can use the console to tell R where to look for and store files and objects.

Where to look for files:
> choose.dir()

Where to put objects (optional):
> setwd()

Within the brackets you must write the whole path with double back-slashes and quotes, e.g.:
> choose.dir("C:\R-seminar\Day1\")

A data table from an ASCII file or other text can be read into a data frame by read.table():
> data.frame <- read.table("datafile.txt", header = TRUE)
header = TRUE: use first row as variable names.

Alternatively, you can specify the path directly in the read command. This can be useful if you want to read files from different locations in the same script. (NB: this only tells R where to read files from, not where to store files). For example:
> DATA <- read.table("C:\R-seminar\Day1\data.txt", header=T)
(Here you must write the name of your own path. Note that R uses double backslashes!)

This version can be even further generalised (for easier transfer to new scripts):
> path <- "C:\R-seminar\Day1\"
> file <- "data.txt"
> DATA <- read.table(paste(path, file, sep=""), header=T)
paste() is a useful function for combining text. Here paste() is used to combine the path and file names. (sep=" " is default, so one must say sep="" to avoid the space.)

When using the read.table() function, each row in the data table must have the same number of elements. The elements can be separated e.g. by tabs (sep="\t") or by spaces (sep=" "). Missing values are coded as “NA” (Not Available).

Proprietary file formats like Excel are not supported. Excel worksheet should be saved as tab-delimited text files, then they can be imported with read.table().

1.2.3 Basic R graphics

Make a scatter plot:
> plot(...)
Add points to a plot:
> points(...)
Add lines to a plot:
> lines(...)
Change plot parameters:
> par(...)
Add a straight line to a plot:
> abline(...)

Notice: Smoothing functions (section 2.6) are great to reveal trends in your data. But there are many ways to use them and they can give quite different results, so they should be used with caution.

Make a histogram:
> hist(...)
1.2.4 Scripts in R
A script is a sequence of R commands collected in a text file. There are several benefits of using script files:
– You can build up an analysis
– You can easily change settings and re-run the analysis
– You can make a new analysis or analyse new data by modifying an old analysis
– You can document and recall the steps in your analysis
– You can share models, analyses and plotting routines with colleagues

Make a script file by selecting from the menu: File -> New script. You can run a script by either:
– Typing `source("scriptfile.R")` in the console window
– Selecting text and right-click: Run selection
– Selecting text and pressing ctrl-R

1.2.5 Control structures in R
When you want a part of the script to execute only under certain conditions:
```r
if (conditions) {execute commands}
```
```r
> if (x < 0) {x <- 0}
```
Other possibilities with `if`:
```r
if(test) {yes} else {no}
```
```r
ifelse(test, yes, no)
```

When you want a part of the script to execute many times:
```r
for (index) {execute commands}
```
```r
> for (i in 1:10) {print(10^i)}
```
For larger operations, functions like `apply()` are more efficient than `for()` loops (see example p. 67).

Notice: There are different types of brackets:
() for grouping expressions
[] for indexing
{} for grouping statements

1.2.6 User-defined functions in R
When you have a piece of code that you will use repeatedly, you should make a function:
```r
function(input...) {commands... return(output...)}
```

Executing a function script creates a function object. The object is executed when the function is called.

1.2.7 Getting help
R has an extensive set of help files in html format. However, R (and S-plus) help is basically written by programmers for programmers... Manuals are available from inside R:
– Menu Help -> Manuals (in PDF)
– `help.start()` in console window, opens general HTML help in web browser

Help on commands, functions calls, etc from the console window:
```r
> help(plot)  # or
> ?plot  # help window on plot function
```
Syntax, parameters, examples, related topics:
```r
> help.search("plot")  # all help windows containing "plot"
> help(help.search)  # for options
> help(package = "vegan")  # help for an installed package
```
A package must be installed, before you can search for help on the package.
1.3 Example 1: Plot snow cover data

The first example is a time series of snow cover in Asia in the period 1970-1979.

We can specify an integer range by the colon (:) operator

```r
> year <- 1970:1979
> year
```

The `c()` operator constructs a list from its arguments. Arguments can again be lists (lists of lists = matrices, etc)

```r
> snow.cover <- c(6.5, 12.0, 14.9, 10.0, 10.7, 7.9, 21.9, 12.5, 14.5, 9.2)
> snow.cover
[1]  6.5 12.0 14.9 10.0 10.7  7.9 21.9 12.5 14.5  9.2
```

Notice periods (.) are allowed in variable names while e.g. underscores (_) are NOT recommended (cf. MS Access export files). Notice also that variable names are case sensitive

```r
> Snow.cover
Error: object "Snow.cover" not found
```

Now let's make a simple plot of the data with the `plot()` command (Figure 1, left panel).

```r
> plot(year, snow.cover)
```

![Figure 1. Plots for the data set snow.cover. Left panel: points; right panel: lines.](image)

OK, but kind of dull - aren't there any glitziness-knobs to turn? Let's get some help on this:

```r
> help(plot)
```

A line plot would perhaps be more informative? (Figure 1, right panel)

```r
> plot(year, snow.cover, type = "l")
```

Notice that we use the "<-" operator to assign objects but we use the "=" operator to change function call parameters from default values.

Let's put in some more informative text:

```r
> plot(year, snow.cover, type = "l",
+ main = "Asia snow cover 1970-1979",
+ ylab = "Snow cover (mill. sq.km)"")
```
Not for the faint of heart: superscripts in legends. Note that the `expression()` function expects individual terms to be separated by asterisks (*)

```r
plot(year,snow.cover, type = "l",
+ main = "Asia snow cover 1970-1979",
+ ylab = expression("Snow cover (" * 10^6 * " * km^2 * ")"))
```

Check `help(plotmath)` for more info on plotting mathematical symbols.

Now, what about line styles? Nothing about this in the help for `plot()`. Many graphics settings are documented in the catch-all function `par()` even parameters which are normally used in `plot()`
```
plot(year,snow.cover, type = "l",
+ col = "red", lwd = 4,
+ main = "Asia snow cover 1970-1979",
+ ylab = expression("Snow cover (" * 10^6 * " * km^2 * ")"))
```

Notice that colors can be specified in several ways:

```
# "red" = "#FF0000" = rgb(255,0,0)
```

The function `colors()` gives a (long) list of predefined colors.

Suppose we make many different plot of this type then we could make it into a function
```
snow.plot(t,x,s) with parameters t = years, x = snow cover, s = legend
snow.plot <- function(t,x,s) {
+ plot(t,x, type = "l", col = "red", lwd = 4, main = s,
+ xlab = "", ylab = expression("Snow cover (" * 10^6 * " * km^2 * ")"))
+ }
```

The curly brackets {} enclose what the function actually does. Notice that we need to execute the function definition for the `snow.plot` function to exist in memory. Type the function name in the command window to check that it's there
```
snow.plot.
```

We can produce the same plot as before by calling our new function:
```
snow.plot(year,snow.cover,"Asia snow cover 1970-1979")
```

We can add data to an existing plot with functions `points()` and `lines()`
```
plot(year,snow.cover, type = "l", col = "red", lwd = 4,
+ main = "Asia snow cover 1970-1979",
+ ylab = expression("Snow cover (" * 10^6 * " * km^2 * ")"))
```

```
abline(h = mean(snow.cover), lty = 2, lwd = 2, col = "blue")
```

```
abline(h = 0, lty = 2, lwd = 2, col = "blue")
```

```
maybe it would look better with y axis symmetrical around 0? Parameters settings for axis limits are for some reason not documented in `plot()` but in `plot.default()` ???
```
```
> help(plot.default)
```

Ahhh... of course, `ylim = ...
```

Notice that `xlim` and `ylim` expect 2-element lists as arguments, which we construct with the `c()` operator.
We can make a function for this plot as well. But we should make sure that the function will work with any data set, not just those within the range -11 to 11

```r
snow.anomaly.plot <- function(t,x,s) {
  z <- x - mean(x)
  plot(t,z, type = "l", col = "red", lwd = 4, main = s,
       ylim = c(-1, 1) * (1.05 * max(abs(z))),
       xlab = "", ylab = expression("Snow cover anomaly (" * 10^6 * " * km^2 * ")"))
  abline(h = 0, lty = 2, lwd = 2, col = "blue")
}
```

Now check if it works as intended (**Figure 2**):

```r
c > snow.anomaly.plot(year,snow.cover,"Asia snow cover anomaly 1970-1979")
```

**Figure 2.** Plots for the data set snow.cover, shown as anomalies (deviations from the mean)

What if we want to have several graphs in the same window?

Again, this is accomplished with the `par()` function

```r
mfrow = Multiple figures, row-wise (siebling of mfcol)
mfrow expects number of rows and columns in a 2-element list
```

```r
c > par(mfrow = c(2,1))  # 2 rows of figures with 1 each
```

Now we can plot both absolute snow cover and anomalies below each other (**Figure 3**):

```r
c > snow.plot(year,snow.cover,"Asia snow cover 1970-1979")
c > snow.anomaly.plot(year,snow.cover,"Asia snow cover anomaly 1970-1979")
```
It is good programming practice to restore default settings:

```r
> par(mfrow = c(1,1))
```

We can also save the whole graphics environment and restore it afterwards.

```r
> oldpar <- par(mfrow = c(2,1))
> snow.plot(year,snow.cover,"Asia snow cover 1970-1979")
> snow.anomaly.plot(year,snow.cover,"Asia snow cover anomaly 1970-1979")
> par(oldpar)
```

Function `ls()` (unix heritage) gives us a list of the objects that currently exist in the R workspace

```r
> ls()
```

If you see more than 5 objects then these are probably leftovers from previous R sessions. Let's continue with a clean slate... We can clear the workspace by either selecting

*Misc -> Remove all objects*

(you need to have to console window in front for this menu to show), or you can do this mystical incantation:

```r
> rm(list=ls(all=TRUE))
```

Remember that R operates exclusively on data structures in memory (while e.g. S-plus operates on files). This means that all our current variables die if R crashes (happens sometimes...). Which is another good reason to work from scripts and save them often!

Entering data in a script with the `c()` function is normally not a good idea except for very small data sets. Typically you will have data in a file of some kind, which you want to read into R.

R does not read proprietary file formats like MS Excel directly. This means that data must be exported to a delimited text file first. Select menu *File -> Save as...*, and choose file type Text (Tab delimited)

In order to read a data file, you need to set the correct search path first. With the command window in front, select

*File -> Change dir...*

and navigate to the directory where your file is located.
To check that you're actually in the right spot, you can view the content of your current working directory by writing (as in ancient DOS):

```r
> dir()
```

If you see the file name Asia.snow.cover.txt in the working directory you can read it by using the function read.table():

```r
> asia.snow <- read.table("Asia.snow.cover.txt", header=TRUE)
```

If there is no error message you should see a new object in your workspace:

```r
> ls()
[1] "asia.snow"
```

The `asia.snow` object will show its content if we write its name. The generic function `names()` gives a list of an object's attributes:

```r
> asia.snow
   year snow.cover
 1  1970        6.5
 2  1971       12.0
 3  1972       14.9
 4  1973       10.0
 5  1974       10.7
 6  1975        7.9
 7  1976       21.9
 8  1977       12.5
 9  1978       14.5
 10 1979        9.2
```

```r
> names(asia.snow)
[1] "year"  "snow.cover"
```

`asia.snow` is an object called a dataframe, which is a rectangular table with individually named columns that can be of different data types. We can inspect a dataframe in a (primitive) spreadsheet view:

```r
> fix(asia.snow)
```

We can inspect the contents of any object with the `str()` function. The `summary()` function gives information about the content of an object. An object can also show itself graphically though the `plot()` function. Most objects should have generic functions like `print()`, `plot()`, `summary()`

```r
> str(asia.snow)
'data.frame': 10 obs. of  2 variables:
$ snow.cover: num 6.5 12 14.9 10 10.7 7.9 21.9 12.5 14.5 9.2
```

```r
> summary(asia.snow)
    year snow.cover
      Min. :1970 Min. : 6.50
1st Qu.:1972 1st Qu.: 9.40
Median :1975 Median :11.35
Mean :1975 Mean :12.01
3rd Qu.:1977 3rd Qu.:14.00
Max. :1979 Max. :21.90
```

```r
> plot(asia.snow)
```

The latter gives the same plot as we got earlier from

```r
> plot(year,snow.cover)  # (See Figure 1)
```

But what happens if we try this now? We get an error message. This is because the variable "year" is currently not visible outside the dataframe "asia.snow"

We can make the attributes of a data frame visible in 3 ways:
- Accessing attributes of an object explicitly with the `$` operator
- Accessing individual table columns with the `[]` operator
- Make all attributes accessible with the `attach()` function
We can specify a specific attribute with the $ operator.

```r
> plot(asia.snow$year, asia.snow$snow.cover)
```

Individual cells in a dataframe can be addressed by bracket indexing `[row,column]`. Entire rows or columns are addressed by `[row, ]` or `[ ,column]`. Double brackets `[[index]]` can be used to access whole columns of a dataframe:

```r
> plot(asia.snow[[1]], asia.snow[[2]])
```

We can put all attributes into the search path with `attach()`. It is good programming practice to `detach()` an object when you don't need it anymore (to avoid confusing variables with same name):

```r
> attach(asia.snow)
> plot(year, snow.cover)
> detach(asia.snow)
```

In the `read.table()` call we set the parameter `header = TRUE` because we wanted the first row to be treated specially as variable names. What happens if we use the default (`header = FALSE`)?

```r
> asia.snow <- read.table("Asia.snow.cover.txt")
> str(asia.snow)
'data.frame': 11 obs. of 2 variables:
$ V1: Factor w/ 11 levels "1970","1971",...:
$ V2: Factor w/ 11 levels "10.0","10.7",...:
```

Now we get 2 variables with generic names V1 and V2, while our variable names appear in the first row of the table. Since there is at least 1 text field in each column, both become factor (nominal) variables. The `summary()` of a factor variable is just an alphabethically sorted list the number of occurrences of each unique string (factor level).

Let's take a closer look at the documentation for `read.table()`

```r
> help(read.table)
```

There are actually quite a few knobs to turn in this function. Of particular relevance to non-anglo-americans is the possibility to handle other the default decimal separator `dec = "."`

The file `asia.snow.cover.no.txt` uses the official Norwegian decimal separator (,), which the default setting would interpret as text:

```r
> asia.snow <- read.table("Asia.snow.cover.no.txt", header = TRUE)
> str(asia.snow)
'data.frame': 10 obs. of 2 variables:
$ snow.cover: Factor w/ 10 levels "10.0","10.7",...:
```

The file is read correctly with the setting `dec = ","`

```r
> asia.snow <- read.table("Asia.snow.cover.no.txt", header = TRUE, dec = ",")
> str(asia.snow)
'data.frame': 10 obs. of 2 variables:
$ snow.cover: num 6.5 12.0 14.9 10 10.7 7.9 21.9 12.5 14.5 9.2
```

`read.table()` is not very forgiving with empty cells or unequal number of cells per row (remember: no blanks in variable or factor level names, concatenate of pad with periods; e.g. "snow.cover"). Missing values must be flagged with the special symbol "NA" (not applicable). Delete empty rows: they make the table more readable for you, but not for R...

```r
> asia.snow <- read.table("Asia.snow.cover.NA.txt", header = TRUE)
> plot(asia.snow, type = "l", col = "red", lwd = 4)
```

We can remove all rows with missing values with the `na.omit()` function:

```r
> plot(na.omit(asia.snow), type = "l", col = "red", lwd = 4)
```
We can also assemble a dataframe by hand, from any set of vector objects of the same size. This is most useful for quantities derived from the original input data, e.g. by transformations:

```r
> year <- 1970:1979
> snow.cover <- c(6.5,12.0,14.9,10.0,10.7,7.9,21.9,12.5,14.5,9.2)
> asia.snow <- data.frame(year, snow.cover)
> rm(year, snow.cover)
> plot(asia.snow)
```

The Asian snow cover data is more than just a table, it is an ordered table - a time series. So maybe we should have taken this into consideration from the start? R has a particular object class for time series:

```r
> help(ts)
```

Time-Series Objects

Description:
The function 'ts' is used to create time-series objects. 'as.ts' and 'is.ts' coerce an object to a time-series and test whether an object is a time series.

So, let's make a ts object instead. Time series have their own plot method with line as default.

```r
> snow.cover <- c(6.5,12.0,14.9,10.0,10.7,7.9,21.9,12.5,14.5,9.2)
> asia.snow <- ts(snow.cover, start = 1970)
> plot(asia.snow)
```

What more can we do with time series objects?

```r
> help.search("time series")
```

`lag.plot()` - that looks interesting? What does it do?

```r
> help(lag.plot)
```

Time Series Lag Plots

Description:
Plot time series against lagged versions of themselves.

OK, let's try it:

```r
> lag.plot(asia.snow)
```

Neat, but maybe not too useful for this data set? We will get back to time series objects later in the course, though.

**1.4 Example 2: Explore and plot water chemistry data**

Water chemistry data from NIVA's national lake survey 1995: subset of 716 lakes from southern Norway.
- Unit for Ca, Mg, Na, K, Cl, SO\(_4\): mg/L
- Unit for HCO\(_3\): meq/L,
- Unit for NO\(_3\): µg/L

```r
> ion.data <- read.table("N716ion.txt", header = TRUE)
```

Did we get all the rows and columns that should have been there?

```r
> dim(ion.data)
[1] 716 10
```

List variable names to get an overview:

```r
> names(ion.data)
[1] "FYLKE"  "Y20"    "Ca"     "Mg"     "Na"     "K"     "HCO3"  "Cl"
[9] "SO4"    "NO3"
```
Show summary statistics:
> summary(ion.data)

  FYLKE
     Min.   : 1.000
     1st Qu.: 6.000
     Median : 9.000
     Mean   : 9.356
     3rd Qu.:12.000
     Max.   :16.000

We can use a stem-and-leaf plot get an overview of the distribution of samples among regions (FYLKE)
> stem(ion.data$FYLKE)

The decimal point is at the |

  1 | 00000000000
  2 | 00000000000
  3 | 000
  4 | 00000000000000000000000000000000000000000000000000000000000000000000
  5 | 00000000000000000000000000000000000000000000000000000000000000000000
  6 | 00000000000000000000000000000000000000000000000000000000000000000000
  7 | 0000
  8 | 00000000000000000000000000000000000000000000000000000000000000000000
  9 | 0000000000000000000000000000000000000000000000000000000000000000
 10 | 000000000000000000000000000000000000000000000000000000000000
 11 | 00000000000000000000000000000000000000000000000000000000000000000000
 12 | 00000000000000000000000000000000000000000000000000000000000000000000
 13 | 00000000000000000000000000000000000000000000000000000000000000000000
 14 | 00000000000000000000000000000000000000000000000000000000000000000000
 15 | 000000000000000000000000000000000000000000000000000000000000
 16 | 00000000000000000000000000000000000000000000000000000000000000000000

FYLKE > 16 is missing since this is a S. Norway subset.
FYLKE 13 is missing (used to be Bergen).
FYLKE 3 (Oslo) and 7 (Vestfold) have the lowest number of samples.

How would this work out with a histogram (Figure 4)?
> hist(ion.data$FYLKE)

Figure 4. Histogram for NIVA chemistry data: number of observations per Fylke (county).
There's something wrong with the break placement. Maybe we should define the breaks explicitly? Let each bin bracket the integers from 0 to 20 (Figure 5):

```r
> hist(ion.data$FYLKE, breaks=-0.5:20.5)
```

![Histogram of ion.data$FYLKE](image)

**Figure 5.** Histogram for NIVA chemistry data: number of observations per Fylke (county), with bins as integers.

What type of variable is FYLKE anyway?

```r
> str(ion.data$FYLKE)
int [1:716] 1 1 1 1 1 1 1 1 1 ... 
```

Maybe FYLKE should be considered a factor variable? Now, how do we make factor variable?

```r
> ?factor
```

We overwrite the original with the new factor variable

```r
> ion.data$FYLKE <- as.factor(ion.data$FYLKE)
```

Check the result: FYLKE should now be factor

```r
> str(ion.data$FYLKE)
Factor w/ 15 levels "1","2","3","4",...: 1 1 1 1 1 1 1 1 1 ... 
```

Maybe we could change the levels to something more meaningful? What options do we have to do this?

```r
> ?levels
```

Ahhh... Of course. We just replace the levels with a list of same length.

```r
> levels(ion.data$FYLKE) <- c("Øf","Ah","Os","He","Op","Bu","Vf","Te","AA","VA","Ro","Ho","SF","MR","ST")
```

Let's see how this worked

```r
> summary(ion.data$FYLKE) 
Øf Ah Os He Op Bu Vf Te AA VA Ro Ho SF MR ST
12 13 3 50 74 62 4 79 64 60 69 79 76 33 38
```

But what happens if we now want to do the same histogram as before?

```r
> hist(ion.data$FYLKE)
Error in hist.default(ion.data$FYLKE) : 'x' must be numeric
```

This gives an error message because FYLKE is no longer numerical.

We can recover the numerical ranks of factors with the function `as.integer()`

```r
> hist(as.integer(ion.data$FYLKE), breaks=-0.5:20.5)  # Figure 6
```
Figure 6. Histogram for NIVA chemistry data: number of observations per Fylke (county), with Fylke as integer.

Ups! - only almost recovered. Notice FYLKE 13 is no longer missing because levels have been remapped from 1,2,...,12,14,15,16 to 1,2,...,12,13,14,15. So maybe this was not the way to do it? Well, what we actually need is to produce bars with values given by `summary(FYLKE)`. Is there something called

```r
> ?barplot
```

Yes, that seems to do the job (Figure 7).

```r
> barplot(summary(ion.data$FYLKE))
```

Figure 7. Barplot of NIVA chemistry data: number of observations per Fylke.

Now that we got a regional factor variable, can we then do regional boxplots?

```r
> ?boxplot
```

Let's attach the data frame to make the coding more readable

```r
> attach(ion.data)
```
Yes, it says we can use a "\textit{y ~ group}" formula to make grouped boxplots. For example chloride grouped by region (expand the windows to all legends)

\begin{verbatim}
> boxplot(Cl ~ FYLKE)  # Figure 8
\end{verbatim}

\textbf{Figure 8.} Boxplot of NIVA chemistry data: Chloride per Fylke.

Hmmm... Oslo and Vestfold seems to break a pattern here. These were also the smallest regions with the lowest number of samples. Is there a way to exclude these regions?

\begin{verbatim}
> ?which

So we need a logical statement which is \texttt{TRUE} except for Oslo and Vestfold. Remember that R uses the following logical operators: "\(!\)" (NOT), "\&" (AND), "\|" (OR). Notice that R uses symbols "\(\text{==}\)" for EQUAL and "\(\text{!=}\)" for NOT EQUAL.

\begin{verbatim}
> not.Os.Vf <- which((FYLKE != "Os") & (FYLKE != "Vf"))
> boxplot(Cl[not.Os.Vf] ~ FYLKE[not.Os.Vf])
\end{verbatim}

\textbf{Figure 9.} Boxplot of NIVA chemistry data: Chloride per Fylke (excluding Oslo and Vestfold).

Now we see much clearer the contrast between inland and coastal regions. We could also have accomplished this with the subset option in \texttt{boxplot()}

\begin{verbatim}
> boxplot(Cl ~ FYLKE, subset = not.Os.Vf)
\end{verbatim}
Now we can get ambitious - Let's make the same regional boxplot for all 9 variables in our data set, arranged in a 3 X 3 matrix.

```r
> par(mfrow = c(3,3))
> for (i in 2:10) {
+   boxplot(ion.data[[i]] ~ FYLKE, subset = not.Os.Vf)
+ }
> par(mfrow = c(1,1))  # Reset par()
```

Almost there - except it would be nice to see what plot is which parameter. The function `colnames()` will give us the variable names in a data frame (Figure 10).

```r
> par(mfrow = c(3,3))
> for (i in 2:10) {
+   boxplot(ion.data[[i]] ~ FYLKE, subset = not.Os.Vf,
+            main = colnames(ion.data)[i])
+ }
> par(mfrow = c(1,1))  # Reset par()
```

![Figure 10. Boxplot for all variables of the NIVA chemistry data.](image)

Water chemistry data are for historical reasons often reported in wildly different units. Let's transform all variables to charge equivalents

```r
#    {µeq / L} = {charge} * 1000 * {mg / L} / {mol.weight}
> Ca.eq   <- 2 * 1000 *   Ca / 40.078  # mg/L  -> µeq/L
> Mg.eq   <- 2 * 1000 *   Mg / 24.3050  # mg/L  -> µeq/L
> Na.eq   <- 1 * 1000 *   Na / 22.989770 # mg/L  -> µeq/L
> K.eq    <- 1 * 1000 *    K / 39.0983  # mg/L  -> µeq/L
> HCO3.eq <- 1 * 1000 * HCO3 / 1   # meq/L -> µeq/L
> Cl.eq   <- 1 * 1000 *   Cl / 35.453  # mg/L  -> µeq/L
> SO4.eq  <- 2 * 1000 *  SO4 / 96.0626  # mg/L  -> µeq/L
> NO3.eq  <- 1 *    1 *  NO3 / 14.0067  # µg/L  -> µeq/L

# We can then collect all our new variables in a new dataframe
> ion.eq <- data.frame(Ca.eq,Mg.eq,Na.eq,K.eq,HCO3.eq,Cl.eq,SO4.eq,NO3.eq)
```

Now can `detach()` the original data and delete intermediate variables. Again, this is good programming practice.
The default `plot()` method for a dataframe is a scatterplot matrix (Figure 11).

> detach(ion.data)
> rm(Ca.eq, Mg.eq, Na.eq, K.eq, HCO3.eq, Cl.eq, SO4.eq, NO3.eq)

Figure 11. Scatterplot matrix for all variables of the NIVA chemistry data.

The actual function for scatterplot matrices is called `pairs()`

`?pairs`

The part about panel functions looks interesting. Are there any predefined ones?

`> help.search("panel")`

`panel.smooth` looks good, let's try that one (Figure 12).

`> pairs(ion.eq,panel=panel.smooth)`
Log transformation reveals the chemists’ weird ideas about detection limits. Some variables, e.g. HCO3.eq or Cl.eq, show quantization at the detection limit. Notice also that the smooth trend depends on which variable is x and which is y. Compare e.g. Ca.eq ~ HCO3.eq and HCO3.eq ~ Ca.eq.

There seems to be more correlation structure in this data set than what can be captured by bivariate relations. What options do we have for principal components in R?

> help.search("pca")

At least 2 different functions: `prcomp()` and `princomp()`. Let’s try one of them.

> princomp(ion.eq)

Error in cov.wt(z) : 'x' must contain finite values only

Which gives an error message, probably because our data has missing values.
> summary(ion.eq)

<table>
<thead>
<tr>
<th></th>
<th>Ca.eq</th>
<th>Mg.eq</th>
<th>Na.eq</th>
<th>K.eq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>1.996</td>
<td>0.7406</td>
<td>3.045</td>
<td>0.2558</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>17.341</td>
<td>9.0516</td>
<td>18.1033</td>
<td>2.0461</td>
</tr>
<tr>
<td>Median</td>
<td>40.172</td>
<td>37.843</td>
<td>37.843</td>
<td>3.8365</td>
</tr>
<tr>
<td>Mean</td>
<td>74.952</td>
<td>30.0360</td>
<td>73.860</td>
<td>6.3809</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>92.320</td>
<td>38.6752</td>
<td>82.319</td>
<td>7.4172</td>
</tr>
<tr>
<td>Max.</td>
<td>883.278</td>
<td>234.5196</td>
<td>904.750</td>
<td>80.8219</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>HCO3.eq</th>
<th>Cl.eq</th>
<th>SO4.eq</th>
<th>NO3.eq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>2.00</td>
<td>2.821</td>
<td>2.082</td>
<td>0.00714</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>32.00</td>
<td>14.103</td>
<td>22.902</td>
<td>0.92813</td>
</tr>
<tr>
<td>Median</td>
<td>48.00</td>
<td>35.394</td>
<td>35.394</td>
<td>4.06948</td>
</tr>
<tr>
<td>Mean</td>
<td>79.35</td>
<td>75.814</td>
<td>46.234</td>
<td>5.72822</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>82.00</td>
<td>85.324</td>
<td>58.295</td>
<td>7.56781</td>
</tr>
<tr>
<td>Max.</td>
<td>864.00</td>
<td>789.778</td>
<td>303.968</td>
<td>73.53624</td>
</tr>
<tr>
<td>NA's</td>
<td>5.00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Yes, there's the culprit - HCO3.eq contains 5 missing values. Let's make a selection variable for this subset.

```r
> not.missing <- which(!is.na(ion.eq$HCO3.eq))
```

Now try again

```r
> p <- princomp(ion.eq, subset = not.missing)
```

Notice that the default for `princomp()` is `cor = FALSE`. That is, principal components on the covariance matrix, which is appropriate in our case since all variables are same unit.

p is now a princomp object which can show itself in various ways

```r
> summary(p)

Importance of components:
                        Comp.1       Comp.2       Comp.3       Comp.4
Standard deviation    163.7245863 124.3512080  28.7724113  16.809869114
Proportion of Variance 0.6139391   0.3541584  0.01896051  0.006471816
Cumulative Proportion  0.6139391   0.9680975  0.98705802  0.993529839
                        Comp.5       Comp.6       Comp.7       Comp.8
Standard deviation    13.842010127  6.5386169015 5.6127370499  4.0794439211
Proportion of Variance 0.004388295  0.0009791952 0.0007215176  0.0003811531
Cumulative Proportion  0.997918134  0.9988973294 0.9996188469 1.0000000000
```

So, 97% of the variance is contained in the first 2 PCA axes. Which means we should get a good representation by a biplot of the first 2 axes. Can R do biplots for us?

```r
?help.search("biplot")
```

`biplot.princomp()` seems to be what we're looking for...

```r
> biplot.princomp(p)

Error: could not find function "biplot.princomp"
```

Which gives an error message on my computer... While the generic `biplot()` seems to work OK (Figure 13).

```r
> biplot(p)
```
Figure 13. Biplot of principal components of major ions, showing the contribution from sea salts (Na and Cl) and weathering (Ca and HCO₃)

The biplot seems to show very clearly the 2 major sources of ions: Na + Cl from sea salts and Ca + HCO₃ from weathering. We can look in more detail at the sea water contribution under the standard assumption that all chloride is sea salts and that other ions accompany Cl according to the standard composition of sea water (Figure 14).

```r
> attach(ion.eq)
> par(mfrow=c(2,3))
> plot(Cl.eq, Ca.eq); abline(0,0.037,col="red")
> plot(Cl.eq, Mg.eq); abline(0,0.193,col="red")
> plot(Cl.eq, Na.eq); abline(0,0.852,col="red")
> plot(Cl.eq, K.eq); abline(0,0.018,col="red")
> plot(Cl.eq, HCO3.eq); abline(0,0.004,col="red")
> plot(Cl.eq, SO4.eq); abline(0,0.103,col="red")
> par(mfrow=c(1,1))
> detach(ion.eq)
```
Figure 14. Relations of major ions to chloride, compared to the standard composition of seawater

Since our data are already in charge equivalents, we can easily calculate the acid neutralizing capacity (ANC) as the difference between base cations and strong acid anions. We will expect ANC to be closely related to alkalinity.

```r
> attach(ion.eq)
> ANC <- Ca.eq + Mg.eq + Na.eq + K.eq - Cl.eq - SO4.eq - NO3.eq
> plot(ANC,HCO3.eq)
> detach(ion.eq)
```

Figure 15. HCO₃ is definitely a major determinant of ANC.
Another quantity that is easily calculated from charge equivalents is the equivalent conductivities of individual ions. If molar conductivities are given as (S/m) / (mol/L)) at 20° C then equivalent conductivity = {mol.cond} * {µeq/L} / 1000 = {mS/m}

\[
\begin{align*}
\text{Ca.ec} & \leftarrow 5.4 \times \frac{\text{Ca.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{Mg.ec} & \leftarrow 4.8 \times \frac{\text{Mg.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{Na.ec} & \leftarrow 4.5 \times \frac{\text{Na.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{K.ec} & \leftarrow 6.7 \times \frac{\text{K.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{HCO3.ec} & \leftarrow 4.1 \times \frac{\text{HCO3.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{Cl.ec} & \leftarrow 6.8 \times \frac{\text{Cl.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{SO4.ec} & \leftarrow 7.2 \times \frac{\text{SO4.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{NO3.ec} & \leftarrow 8.4 \times \frac{\text{NO3.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m}
\end{align*}
\]

attach(ion.eq)

\[
\text{Ca.ec} \leftarrow 5.4 \times \frac{\text{Ca.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{Mg.ec} \leftarrow 4.8 \times \frac{\text{Mg.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{Na.ec} \leftarrow 4.5 \times \frac{\text{Na.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{K.ec} \leftarrow 6.7 \times \frac{\text{K.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{HCO3.ec} \leftarrow 4.1 \times \frac{\text{HCO3.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{Cl.ec} \leftarrow 6.8 \times \frac{\text{Cl.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{SO4.ec} \leftarrow 7.2 \times \frac{\text{SO4.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m} \\
\text{NO3.ec} \leftarrow 8.4 \times \frac{\text{NO3.eq}}{1000} \quad \mu\text{eq/L} \rightarrow \text{mS/m}
\]

detach(ion.eq)

Predicted specific conductivity is then just the sum of the equivalent conductivities of the individual major ions (Figure 16).

\[
\text{K20.pred} \leftarrow \text{Ca.ec} + \text{Mg.ec} + \text{Na.ec} + \text{K.ec} + \text{HCO3.ec} + \text{Cl.ec} + \text{SO4.ec} + \text{NO3.ec}
\]

plot(K20.pred, ion.data$K20)

abline(0,1)

**Figure 16.** Measured vs. predicted specific conductivity (as the sum of the equivalent conductivities of the individual major ions)

There are at least 4 possible outlier candidates in this plot 2 with much higher measured conductivity than predicted, two of them with somewhat lower conductivity than predicted. Let's look at them in more detail (Figure 17).

\[
\text{K20.diff} \leftarrow \text{ion.data}\$\text{K20} - \text{K20.pred}
\]

hist(K20.diff, breaks=50)
Figure 17. Histogram for of differences between measured and predicted conductivities, showing two possible outliers

Since our data set does not include the cation with the by far highest equivalent conductivity (H+ 32 (S/m)/(mol/L)), we would expect the specific conductivity to be underpredicted in acid lakes (Figure 18).

> plot(ANC,K20.diff)

Figure 18. The contribution of H+ to specific conductivity is highest when ANC is low
2. FROM LINEAR MODELS TO GAM

Jannicke Moe

Topics:
A data import problem: dates format (script: dag2_script1_import_Storgama.txt)
Linear models: regression, ANOVA etc. (script: dag2_script2_LM_Storgama.txt)
Generalised linear models: logistic regression etc. (script: dag2_script3_GLM_Storgama.txt)
Generalised additive models: smooth splines etc. (script: dag2_script4_GAM_Storgama.txt)
Model selection – some issues (no script)

2.1 Linear models: what sort of test should I use?

Let’s say you have one response variable (continuous, normally distributed) and various possible predictor variables (Table 1. Basic linear models for different combinations of predictor variables, with corresponding model formula in R.). In excel, you must choose the right kind of test from a menu, depending on the type of data you have. In R, on the other hand, you can use a standard test formulation, and the test result will depend on the type of data you put into it. The basic formula is:

> \text{lm}(y \sim x)

This stands for: \( y = b_0 + b_1x + \text{residuals} \)

Table 1. Basic linear models for different combinations of predictor variables, with corresponding model formula in R.

<table>
<thead>
<tr>
<th>Predictor:</th>
<th>1 predictor variable</th>
<th>2 or more predictor variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continuous</td>
<td>Linear regression: ( \text{lm}(y \sim x) )</td>
<td>Multiple regression ( \text{lm}(y \sim x_1 + x_2 + \ldots) )</td>
</tr>
<tr>
<td>Categorical</td>
<td>t-test ( \text{lm}(y \sim x) )</td>
<td>ANOVA ( \text{lm}(y \sim x_1 + x_2 + \ldots) )</td>
</tr>
<tr>
<td>Combination</td>
<td>-</td>
<td>ANCOVA ( \text{lm}(y \sim x_1 + x_2 + \ldots) )</td>
</tr>
</tbody>
</table>

2.2 Linear models in R: some useful commands

> fit <- \text{lm}(y \sim x_1 + x_2)  # Store the model as the object “fit”

Diagnostic plots (good idea to set \text{par}(mfrow=c(2,3))!):

> \text{plot}(fit)

What is stored in the fit?

> names(fit)

The most commonly used info:

> summary(fit)

Here you find the names that R gives the summary components, so that you can extract them and e.g use them on plots:

> names(summaryfit)
Examples:
> summary(fit)$adj.r.squared  # gives you the R^2 value
> summary(fit.1)$coef  # this component turns out to be a matrix
> dimnames(summary(fit.1)$coef) # a matrix has “dimnames” rather than “names”
> summary(fit)$coef["x1", "t value"]

Elements of the fitted object can be accessed just like columns in a dataframe:
> fit$coefficients  # gives you the coefficients
> fit$coef  # any unambiguous abbreviation can be used

Extractor functions are made to extract certain information, you should use these rather than the object components.
> coef(fit)  # same as fit$coef - usually...
> fitted(fit) # fitted values
> resid(fit) # residuals
> predict(fit) # predicted values
> predict(fit, se.fit=T)  # ...with standard error
> deviance(fit)

Alternative to lm() exist for certain types of linear models. For example, comparison of two groups can also be done with the function t.test(y ~ x), but the function lm(y ~ x) is simpler. ANOVA can also be done with the function aov():
> fit <- aov(y ~ x1 + x2)  # same fit as lm(),
> anova(fit)  # gives you an ANOVA table

The function aov() seems to do the same as lm(), but it is also possible to combine fixed and random effects. This is applicable for balanced design only. For unbalanced design, and for more complicated models, you should use the function lme() (linear mixed-effects models, in the package "nlme")

2.3 Formula syntax in R – general rules

The following information is extracted from the help file on formulae:
> ?formula

Y ~ F  Response variable Y is modeled as F, where F may include other terms
Fa + Fb  Include both Fa and Fb in the model
Fa - Fb  Include all of Fa in the model, except what is in Fb
Fa : Fb  The interaction between Fa and Fb
Fa * Fb  Fa + Fb + Fa : Fb
Fb %in% Fa  Fb is nested within Fa
Fa / Fb  Fa + Fb %in% Fa
I(F^m)  All terms in F crossed to order m
.  in update(): the previous set
.  in other formulae: all variables (except the response variable(s) Y)

A model with no intercept (going through the origin) can be specified as:
Y ~ F - 1 or
Y ~ 0 + F

While formulae usually involve just variable and factor names, they can also involve arithmetic expressions. The formula ‘log(y) ~ a + log(x)’ is quite legal. When such arithmetic expressions involve operators which are also used symbolically in model formulae, there can be confusion between arithmetic and symbolic operator use. To avoid this confusion, the function ‘I()’ can be used to bracket those portions of a model formula where the operators are used in their arithmetic sense. For example, in the formula ‘y ~ a + I(b+c)’, the term ‘b+c’ is to be interpreted as the sum of ‘b’ and ‘c’.
2.4 Generalised linear models (GLM)

What if the y variable is not suitable for linear models? E.g. the y variable is not normally distributed but 0/1, proportions, or Poisson-distributed (e.g. counts). The framework for analysing linear models can be generalised using a link function g()

\[
\begin{align*}
\text{LM:} & \quad y = b_0 + b_1 x_1 + b_2 x_2 + \ldots \\
\text{GLM:} & \quad g(y) = b_0 + b_1 x_1 + b_2 x_2 + \ldots 
\end{align*}
\]

The link function transforms the y variable into something that can be modelled as a linear combination of predictor variables. Ordinary linear models are then a "special case" where the link function = identity.

A typical example of a GLM is logistic regression. The y variable is then binary distributed, such as 0/1. A linear model is obviously not appropriate (Figure 19) - a sigmoid curve is better.

We can use the link function "logit" to obtain a linear link between the predictor and response variables:

\[
\logit(y) = \log(y / (1-y))
\]

\(\logit(y)\) has range \((-\infty, \infty)\), and can therefore be modelled as linear combination of the predictor variables.

\[
\log(y/(1-y)) = b_0 + b_1 x
\]

Back-transformation gives:

\[
y = \frac{\exp(b_0 + b_1 x)}{1 + \exp(b_0 + b_1 x)}
\]

**Figure 19.** Binary data: linear regression (left panel) versus logistic regression (right panel).

A logit transformation cannot be done directly on the data, when the response is binary 0/1. In R, the transformation is done implicitly by the function glm(). Instead of

```r
> lm(y ~ x)
```

we write

```r
> glm(y ~ x, family = binomial(link = logit))
```

2.5 ”Families” of distributions in R

What sort link (variance functions) can the different families of distributions use, within the GLM framework? The following table summarises the suitable pairings:
Table 2. Families of distributions applicable for Generalised Linear Models in R.

<table>
<thead>
<tr>
<th></th>
<th>binomial</th>
<th>gaussian</th>
<th>Gamma</th>
<th>inverse</th>
<th>poisson</th>
<th>quasi</th>
</tr>
</thead>
<tbody>
<tr>
<td>logit</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>probit</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>cloglog</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>identity</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inverse</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>log</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/mu^2</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sqrt</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If you have a non-linear relationship for which none of the GLMs seems suitable, then you can specify your own non-linear formula by non-linear least-squares regression, `nls()`. You must specify parameters directly in the formula, and suggest start values (`start=list(...)`).

As an example we can use the backtransformed version of logit(y):

```r
> nls(y ~ exp(b0 + b1*x)/(1 + exp(b0 + b1*x)), start=list(b0=0, b1=0) )
```

### 2.6 Generalised Additive Models (GAM)

GLM allows non-linear models within a linear framework, but we must still specify a parametric model. **Additive Models (AM)**, on the other hand, are not restricted to linear combination of predictor variables. AMs allow you to **add** various *functions of predictor variables*.

**LM:** \( y = b_0 + b_1 x_1 + b_2 x_2 + b_3 (x_2^2) + \ldots \)

**AM:** \( y = b_0 + f_1(x_1) + f_2(x_2) + \ldots \)

Just like linear models, AMs can also be **generalised**, using a link function \( g() \):

**GAM:** \( g(y) = b_0 + f_1(x_1) + f_2(x_2) + \ldots \)

In GAM the functions \( f() \) can be **non-parametric**, e.g. loess (locally weighted regression), or so-called splines. In R we can use splines with the function \( s() \).

```r
> gam(y ~ s(x))
```

Non-parametric regression can give you nice flexible curves, but not conventional parameter estimates. An example of a data set where a GAM may be more suitable than a linear model is shown in Figure 20.

![Figure 20. Regression with linear model (LM; left panel) versus generalised additive model (GAM; right panel).](image-url)
An important consideration when using GAM is how "smooth" your estimated curves should be. If the curves are too smooth, you may lose important details in the data structure. On the other hand, if you allow the curves to get too "wiggly", then the estimated curve may become too specific for your current data set, and be less useful for more general interpretations. The function `gam()` in the package "basis" (and in S-plus) uses 4 degrees of freedom as default. R also has a package "mgcv" with a somewhat different function `gam()`. In this version, degrees of freedom are optimised by cross-validation together with the model fit. We recommend that you use the mgcv version of gam, because of this built-in optimisation of the smoothing.

Figure 21 summarises the relationship between linear models, additive models, and generalised linear and additive models.

![Diagram summarising relationships between LM, AM, GLM, and GAM](image)

**Figure 21.** Relationship between LM, AM, GLM and GAM.

### 2.7 Model selection

An important consideration for all types of models is how many explanatory variables you should include in your model. If there are too few variables, the model will not be able to explain much of the variation. On the other hand, if there are too many variables, then the model will be too specific for the current data set. A guideline for model selection is: when is a more complicated model significantly better than a simpler model?

Various criteria are developed for model selection, including the following:
- Compare how much variation is explained ($R^2$)
- Compare how much variation is left (Mallow’s Cp, deviance,…)
- Compare criteria based on log-likelihood (AIC, BIC)
- Compare ability to predict new data: cross-validation

In R, some approaches use a mix of criteria. The type of model you have may restrict the choice of approach and criteria. Some examples are given below.

#### 2.7.1 Model selection for LM: ANOVA table

A simple example: Model 1 is a subset of Model 2.

```r
> x1 <- 1:10
> y  <- x1^3
> fit.1 <- lm(y ~ x1)  # Adjusted R-squared: 0.8446
> fit.2 <- lm(y ~ x1 + I(x1^2))   # Adjusted R-squared: 0.9963
```
Fit 2 is obviously better than fit 1, but is it significantly better?

```r
> anova(fit.1, fit.2)
Analysis of Variance Table
Model 1: y ~ x1
Model 2: y ~ x1 + I(x1^2)

Res.Df RSS Df Sum of Sq      F    Pr(>F)
1  8 146837
2  7   3089  1    143748 325.77 3.961e-07 ***
```

The low p-value of this test means that model 2 is significantly better.

Model selection by `anova()` is restricted to cases where one or more models are subsets of a more complete model. Note that rows with missing values should be removed before model fitting, to obtain the same number of observations for all models. When applying `anova()` to glm objects, the function does not return p-values directly, but it returns deviances from which a p-value can be calculated.

### 2.7.2 Model selection for GLM: deviance analysis

We use the same simple example as above.

```r
> fit.1 <- glm(y ~ x1)
> fit.2 <- glm(y ~ x1 + I(x1^2))
> deviance(fit.1)
[1] 146836.8
> deviance(fit.2)
[1] 3088.8
```

Again, fit.2 is obviously better than fit 1, but is it significantly better? The difference in deviance is \( \chi^2 \)-distributed with 1 degree of freedom (=difference in number of parameters). Hence, we can test the significance of the difference in deviance with a \( \chi^2 \) test.

```r
> 1 - pchisq(deviance(fit.1) - deviance(fit.2), df=1)
[1] 0
```

The difference is significant (p < 0.05).

This kind of test is restricted to cases where deviance can be calculated, and where models are subsets. Deviance is available also for GAMs, but number of parameters is not defined in the same way as for GLMs, so it is not straightforward to use this test for GAMs.

### 2.7.3 Akaike's information criterion

Akaike's information criterion (AIC) is calculated as

\[-2 \times \text{log-likelihood} + k \times \text{npar},\]

where npar = the number of parameters in the fitted model, and k = penalty per parameter. The function for AIC in R is

```r
> AIC(object, ..., k=2)
```

AIC requires that the log-likelihood function can be easily calculated (which is the case for e.g. GLM, but not for `nls` (non-linear models)), and that the model has parameters in the usual sense (which is not the case for GAM). The models do not have to be subsets of each other, but they must use the same response variable.

### 2.7.4 Automatic stepwise selection

The function `step()` can help you select the best model from a full model, by automatically adding or removing terms in a stepwise fashion. This function uses the criteria AIC and Mallow’s Cp (the total square errors).
Consider a full model with interactions:

```r
> fit <- lm(y ~ (x1 + x2 + x3 + x4)^2)
```

For objects of type lm and glm, you can use

```r
> step(fit)
```

For a wider range of object classes (in package "MASS"), you can use:

```r
> stepAIC(fit)
```

### 2.7.5 Model selection by cross validation

Cross validation is a model selection approach that is applicable for all types of models. The procedure is generally as follows.

1. Specify a model (e.g. full model)
2. Exclude a subset of data (e.g. 1/10): $x_{\text{excl}}, y_{\text{excl}}$
3. Estimate the parameters with the remaining data: $x_{\text{incl}}, y_{\text{incl}}$
4. Use the $x_{\text{excl}}$ as input in the model parameterised by $x_{\text{incl}}, y_{\text{incl}}$ and predict $y_{\text{pred}}$
5. Compare the $y_{\text{pred}}$ with the real $y_{\text{excl}}$; calculate the squared differences
6. Repeat for each subset of the data
7. Sum the calculated squared differences. This gives the CV score for this model
8. Repeat CV calculation for each model

A drawback with this approach is that there is no general rule for defining a significant difference between CV scores.

### 2.8 Example 3: Nitrate concentrations in the river Storgama

Request from participant (Heleen de Wit): I wish to import a data file so that I can plot different variables versus time, and do time-series analyses. I wish to check if the water nitrate concentration shows a temporal trend, or some pattern, and if there is a relationship with the water TOC (total organic carbon).

#### 2.8.1 Script 1: Import and formatting of dates.

First we try to read the data as they are. I've saved a copy the excel file Storgama_0.xls (first sheet) as a text file "Storgama_0.txt". To tell R where the files are, you can change directory from the menu (File -> Change dir...).

```r
> DATA <- read.table("Storgama_0.txt", header=T)
```

Why call the object "DATA", not very informative? A benefit is that it's easier to adjust and re-use the script to new datasets.

```r
> names(DATA)
[1] "STID"  "STCOD"  "NAME"  "Date"  "pH"    "KOND"  "TOC"  "TOTN"  "NO3.N"  
[10] "NH4.N" "ECa"  "ECl"  "ENa"  "EMg"  "ESO4"  "EK"  "ENO3"  "EALK1"

> is.factor(DATA$Date) # Dates are formatted as levels - not so useful.
[1] TRUE
```

Can we try to change them into numeric?

```r
> as.numeric(DATA$Date)[1:10]# Look at the first 10
[1] 855 1217 1543 149 149 149 149 149 149 149
```
Even less useful. Let's check helpfiles.

> ?date
Gives us today's date :(  
> help.search("date")
format.Date(base) sounds useful  
> ?format.Date  
> as.Date(DATA$Date)[1:10]
Totally useless - or something wrong with the format

So maybe we should check what sort of format R actually requires  
From helpfile "as.Date":

The default formats follow the rules of the ISO 8601 international standard which expresses a day as "YMD".

NB: sometimes the most useful information is found not in the description, but in the examples.  
I found this at the bottom of the helpfile:

dates <- c("02/27/92", "02/27/92", "01/14/92", "02/28/92", "02/01/92")  
as.Date(dates, "%m/%d/%y")
HWI has format "%d/%m/%y" - I guess

> as.Date(DATA$Date, "%d/%m/%y")[1:10]
[6] "2019-09-03" "2019-09-03" "2019-09-03" "2019-09-03" "2019-09-03"
Now the data series starts in 2020 so either someone has faked these data, or we still have a format problem...

Let's go back to the excel file and see if we find some solution there. It turns out, some of the cells have hidden time values after the date values. Things like this are quite common and can create a lot of trouble when you work with other people's excel files. One solution is to split dates into separate columns for year, month etc. in excel, with these excel function:  
=year(), =month(), =day(), =hour(), =minute(), =second().
and combine these together again in R with the function ISODatetimex().  
I've done this in "Storgama_1.txt".

> file <- "Storgama_1.txt"  
> DATA <- read.table(paste(path, file, sep=""), header=T)
> DATA$iso.date <- ISOdate(DATA$year, DATA$month, DATA$day)

Why attach the new vector to the dataset, couldn't it just be separate vector?  
It can be easier to select rows etc. for all variables if it's in the dataset.

> oldpar <- par(mfrow = c(3,2))  
> for (i in 18:23){
+ plot(DATA$iso.date, DATA[[i]], type="l", ylab = names(DATA)[i])
+ }  
> par(oldpar)

This plot produces Figure 22.
Figure 22. Time-series plots for Storgama.

NB: The rest of the script is optional - not required for subsequent scripts
Alternatively, we can try to import the dates in a date format that R understands.

In "Storgama_1.txt" I've formatted the column "Date": removed everything after the date (replaced "**" by "" in excel), and formatted the date like "%d/%m/%y" (Format cells -> Number -> Date)
Does it now accept the dates as dates? Let's compare with the original version

> DATA$Date_original[1:5]
1556 Levels: 01/02/1979 01/02/1993 01/02/1995 01/02/1999 ... 31/12/1985
> DATA$Date[1:5]
[1] 17/07/74 24/07/74 31/07/74 03/09/74 03/09/74
1556 Levels: 01/02/79 01/02/93 01/02/95 01/02/99 01/03/04 01/03/82 ... 31/12/85
Both versions are still read as factors

What if we now try to persuade R to read these as numeric dates:
> as.Date(DATA$Date_original, "%d/%m/%y")[1:5]
> as.Date(DATA$Date, "%d/%m/%y")[1:5]

Looks much better! But what about those "", is this vector numeric anyway?
> is.numeric(as.Date(DATA$Date, "%d/%m/%y"))
[1] TRUE

:-)
We'll change this permanently in the dataset
> DATA$Date <- as.Date(DATA$Date, "%d/%m/%y")
NB: There is a very similar function in another package, which does something slightly different!

```r
> ?as.Date (package:base)
  Functions to convert between character representations and objects of class "Date" representing calendar dates.
> ?as.date (package:survival)
  Converts any of the following character forms to a Julian date:
    8/31/56, 8-31-1956, 31 8 56, 083156, 31Aug56, or August 31 1956.
This does NOT use the standard format "1956-08-31", which the base package uses...
```

When you search for useful functions, it's proably a good idea to select those from the most common packages, if possible.

Other potentially useful functions for date conversion in package "survival":

```r
> library(survival)
Loading required package: splines
> julian(as.Date("2006-05-02"), origin=as.Date("1960-01-01")) # Default origin "1970-01-01"
[1] 16923
attr(,"origin")
[1] "1960-01-01"
> date.mdy(16923)# uses origin "1960-01-01"
$month
[1] 5

$day
[1] 2

$year

> mdy.date(5, 2, 2006)
[1] 2May2006
```

### 2.8.2 Script 2: Linear models.

Let's look at the data:

```r
> plot(DATA[,12:16])
```
This gives pairwise plots of selected parameters (Figure 23). (This is not recommended for all parameter columns at once).

Figure 23. Pairwise plots of parameters from Storgama.

We want to look at certain parameters against time. Make a continuous numeric vector of time, with unit year (with decimals) as.numeric() gives date as no. of seconds since 1970-01-01.

```r
> DATA$iso.year <- as.numeric(DATA$iso.date)/(365*24*60*60) + 1970
```

"Trend analysis" tries to describe a temporal trend and at the same time account for autocorrelation in the data. "Time-series analysis" in R is a method/framework that requires data with regular intervals (see ?ts), and which will require some reformatting of the data in this case. We will get back to these issues later. For now, we'll ignore the temporal correlation and treat the data as independent, and start with ordinary regression analyses.

First we can "attach" a dataframe: then the columns of the dataframe will be directly accessible, so that we can skip the "DATA$", and wrote "NO3.N" instead of "DATA$NO3.N"

```r
> is.object(NO3.N)# No object with this name
Error: object "NO3.N" not found
> attach(DATA)
> is.object(NO3.N)
[1] FALSE
```

Now the column is "visible" for R, although it's not a proper object.

Does nitrate decrease with time? (Figure 24).

```r
> windows()
> plot(iso.date, NO3.N, type="l")
```
Figure 24. Data series NO3.N from Storgama.

Regression of NO3 against time:

```r
> fit.1 <- lm(NO3.N ~ iso.year)
> summary(fit.1)
```

Call:
```
lm(formula = NO3.N ~ iso.year)
```

Residuals:
```
  Min      1Q  Median      3Q     Max
-168.17  -78.49  -25.71   47.57  884.64
```

Coefficients:
```
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 8607.475    680.358   12.65   <2e-16 ***
iso.year      -4.269      0.342  -12.48   <2e-16 ***
```

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 121.2 on 1550 degrees of freedom
Multiple R-Squared: 0.09135,    Adjusted R-squared: 0.09077
F-statistic: 155.8 on 1 and 1550 DF,  p-value: < 2.2e-16

Very significant indeed, but has low adjusted R2 (0.09).

Can seasonal variation explain some of the variation? Use month as a categorical factor. This model is an ANCOVA (Analysis of covariance).

```r
> fit.2 <- lm(NO3.N ~ iso.year + as.factor(month))
> summary(fit.2)
```

Coefficients:
```
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 7193.1363   519.0880  13.857  < 2e-16 ***
iso.year     -3.5373     0.2609 -13.557  < 2e-16 ***
as.factor(month)2   -0.1264    11.7585  -0.011 0.991423
as.factor(month)3    35.0309    11.5731   3.027 0.002511 **
as.factor(month)4   124.8507    10.9909  11.359  < 2e-16 ***
as.factor(month)5   -60.3850    10.9800  -5.500 4.45e-08 ***
as.factor(month)6 -143.1046    11.5944 -12.343 < 2e-16 ***
as.factor(month)7   -2.8566    11.5966  -0.246 0.805462
```
Look at coefficients for each month (Figure 25). The argument \[-(1:2)\] excludes element 1:2.

```
> barplot(c(0,coef(fit.2)[-(1:2)]), names.arg=1:12)
```

**Figure 25.** Barplot for estimated coefficients per month for Storgama.

Is there an interaction between year and month? This means that effect of time can be different for each month.

```
> fit.3 <- lm(NO3.N ~ iso.year * as.factor(month))
> summary(fit.3)
```

```
Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          4094.0268  1870.6059   2.189 0.0288 *
iso.year             -1.9794     0.9404  -2.105 0.0355 *
as.factor(month)2    -279.6590  2631.3218  -0.106 0.9154
as.factor(month)3    7380.5148  2600.8831   2.838 0.0046 **
as.factor(month)4    15150.6343  2391.2350   6.336 3.10e-10 ***
as.factor(month)5    4852.8377  2383.9214   2.036 0.0420 *
as.factor(month)6    1255.9275  2551.1398   0.493 0.6202
as.factor(month)7    2583.3170  2870.0494   0.900 0.3682
as.factor(month)8    370.1818  2523.3984   0.147 0.8834
as.factor(month)9    3715.8048  2483.1261   1.496 0.1348
as.factor(month)10   1687.2379  2556.7416   0.660 0.5094
as.factor(month)11   2773.3034  2560.6876   1.083 0.2790
as.factor(month)12   -3715.8048  2483.1261  -1.496 0.1348
iso.year:as.factor(month)2   0.1401     1.3226   0.106 0.9156
iso.year:as.factor(month)3   -3.6920     1.3073  -2.824 0.0048 **
iso.year:as.factor(month)4   -7.5590     1.2025  -6.286 4.24e-10 ***
iso.year:as.factor(month)5   -2.4709     1.1989  -2.061 0.0395 *
iso.year:as.factor(month)6    1.2651     1.2824   0.987 0.3240
iso.year:as.factor(month)7    1.5273     1.4166   1.078 0.2811
iso.year:as.factor(month)8    1.2283     1.4415   0.852 0.3943
iso.year:as.factor(month)9   -0.2452     1.2685  -0.193 0.8468
iso.year:as.factor(month)10   -1.9073     1.2482  -1.528 0.1267
iso.year:as.factor(month)11   -0.8700     1.2850  -0.677 0.4985
iso.year:as.factor(month)12   -1.3957     1.2870  -1.084 0.2783
```

---

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 88.58 on 1528 degrees of freedom
Multiple R-Squared: 0.5213, Adjusted R-squared: 0.5141
F-statistic: 72.35 on 23 and 1528 DF, p-value: < 2.2e-16
The increased R2 value is probably not worth all the extra parameters.

Does NO3 have a relationship with TOC?
> fit.4 <- lm(NO3.N ~ iso.year + as.factor(month) + TOC)
> summary(fit.4)
Coefficients:
       Estimate Std. Error t value Pr(>|t|)
TOC       1.0069     1.9195   0.525    0.600
---
Multiple R-Squared: 0.581, Adjusted R-squared: 0.5755
F-statistic: 105.6 on 13 and 990 DF, p-value: < 2.2e-16
Effect of TOC is not significant (although adjusted R2 is slightly higher).

What if we look at relationship with TOC only?
> fit.5 <- lm(NO3.N ~ TOC)
> summary(fit.5)
Coefficients:
       Estimate Std. Error t value Pr(>|t|)
(Intercept) 135.487     10.981  12.339  < 2e-16 ***
TOC         -9.680      2.209  -4.382 1.30e-05 ***
---
Multiple R-Squared: 0.01881, Adjusted R-squared: 0.01783
F-statistic: 19.2 on 1 and 1002 DF, p-value: 1.298e-05
The effect of TOC only is significant, but has a very low R2.

Now what if we try to put year back again:
> fit.6 <- lm(NO3.N ~ iso.year + TOC)
> summary(fit.6)
Coefficients:
       Estimate Std. Error t value Pr(>|t|)
(Intercept) 6822.1445  1048.2457   6.508 1.20e-10 ***
iso.year    -3.3674     0.5279  -6.379 2.72e-10 ***
TOC         -3.1584     2.3955  -1.318    0.188
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 87.41 on 1001 degrees of freedom
Multiple R-Squared: 0.05714, Adjusted R-squared: 0.05525
F-statistic: 30.33 on 2 and 1001 DF, p-value: 1.627e-13
Then TOC is no longer significant.

Obviously we need a more systematic way to selecting the best model! We will address this issue later, but now let's look at some alternatives to the linear model.

> detach(DATA)  # Good modelling practice to clean up workspace

2.8.3 Script 3: Generalised linear models.

We want to describe relationship NO3.N and pH in Storgama (Figure 26). (This relationship is clearer that between TOC and NO3.N)

> attach(DATA)
> par(mfrow=c(2,2))  # Make panel for 2x2 plots
> plot(NO3.N, pH, xlab="NO3 (ug/L N)", ylab="pH")

First do a linear regression.
> fit.lm <- lm(pH ~ NO3.N)
> abline(coef(fit.lm), col="blue")
We would like to add some summary stats from `summary(fit.lm)` on the plot: R², p-value. How do we extract these values?

```
> names(fit.lm)
[1] "coefficients" "residuals"   "effects"     "rank"
[5] "fitted.values" "assign"     "qr"          "df.residual"
[9] "na.action"   "xlevels"    "call"         "terms"
[13] "model"
> names(summary(fit.lm))
[1] "call"          "terms"         "residuals"     "coefficients"
[5] "aliased"       "sigma"         "df"            "r.squared"
[9] "adj.r.squared" "fstatistic"  "cov.unscaled"
> dimnames(coef(summary(fit.lm)))
[[1]]
[1] "(Intercept)" "NO3.N"
[[2]]
[1] "Estimate"   "Std. Error" "t value"   "Pr(>|t|)"
```

Add title (can also use `main()` for this):

```
> mtext("LM", side=3, line=.5, col="blue")
```

We can use the function `paste()` to combine summary stats values with text on the plot.

```
> mtext(paste("R²=", round(summary(fit.lm)$adj.r.squared,2), sep=""),
+ side=3, line=-2, adj=.5, col="blue")
```

The argument `line=` gives distance from the box, `adj=` gives adjustment left/right.

If the p-value is below 0.001, we’ll just write "p < 0.001".

```
> pvalue <- coef(summary(fit.lm))[2,"Pr(>|t|)"
+ ] # extract p-value
if (pvalue >= 0.001) {
   + mtext(paste("p=" , pvalue, sep=""), side=3, line=-2, adj=1, col="blue")
}
```

The curly brackets mark beginning and end of a statement. This is necessary if the statement is written over more than one line. Here the test failed, therefore nothing happened.

```
> if (pvalue < 0.001)
+ mtext("p<0.001", side=3, line=-2, adj=0.99, col="blue")
```

The test passed, and so the text was added.

PS: It should be possible to write this test more elegantly with the function `ifelse(test, yes, no)` or `if(test) {yes} else {no}`

We also want to add confidence intervals for the mean. The function `predict.lm(object, interval="confidence")` gives a matrix with columns `fit`, `lwr`, `upr` (lower and upper confidence intervals) for the mean.

Alternatively for the response, see `?predict.lm`.

Make a sequence of x's for which we will predict the response:

```
> new.x  <- data.frame(NO3.N = seq(min(NO3.N, na.rm=T), max(NO3.N, na.rm=T),
+ length=50)
> pred.lm <- predict.lm(fit.lm, interval="confidence", newdata=new.x)
> is.matrix(pred.lm)
[1] TRUE
> is.data.frame(pred.lm)
[1] FALSE
```

Matrices can behave a bit differently from data.frames:

- They don't have "names" for columns, but "colnames"
- They also have "dimnames", which is a list containing "rownames" (= dimnames[[1]]) and "colnames" (= dimnames[[2]])
- Length() for a data.frame is no. of columns, but length() for a matrix is no. of elements (=nrow*ncol)
- You can't select the columns of a matrix by `DATA$columnname`, but by `DATA[,"columnname"]`
- `dim()`, `nrow()`, `ncol()` work the same way for matrices and data.frames.
> lines(new.x$NO3.N, pred.lm[,"fit"], col="blue")
This gives the same as line as abline(coef(fit.lm)).
> lines(new.x$NO3.N, pred.lm[,"lwr"], col="blue", lty=2)
> lines(new.x$NO3.N, pred.lm[,"upr"], col="blue", lty=2)

Figure 26. Linear regression of pH vs. NO3 in Storgama

Now let's say we're only interested in whether pH is above or below a boundary value of 4.5
> boundary  # OK to use this name?
Error: object "boundary" not found
> boundary <- 4.5

Change pH data into binary 0/1, below/above boundary. First make empty vector (not really necessary
here, but in other cases it's useful).
> DATA$pH.bin <- rep(NA, nrow(DATA))

Identify those rows with pH above the boundary, with a logical test:
test <- DATA$pH > boundary
(Why use object "boundary" in stead of the value 4.5 directly? - In case we want to change the value
later, then it needs to be updated in only one place.)
This gives a vector "test" with components TRUE, FALSE, NA
> DATA$pH.bin <- as.numeric(test)
Here, as.numeric() translates these components to 1, 0, NA, respectively. We could also make a
more compact formulation:
> DATA$pH.bin <- as.numeric(DATA$pH > boundary)

Note that columns added to a data.frame AFTER it is attached, will not be available directly. So we'll
detach and attach the data.frame again.
> detach(DATA)
> attach(DATA)

> plot(NO3.N, pH.bin, xlab="NO3 (ug/L N)", ylab=paste("pH >", boundary))
How will a linear regression work now (Figure 27)?

```r
> fit.lm <- lm(pH.bin ~ NO3.N)
> pred.lm <- predict.lm(fit.lm, newdata=new.x, interval="confidence")
> lines(new.x$NO3.N, pred.lm[,"fit"], col="red")
> lines(new.x$NO3.N, pred.lm[,"lwr"], col="red", lty=2)
> lines(new.x$NO3.N, pred.lm[,"upr"], col="red", lty=2)
> mtext(paste("R^2=" , round(summary(fit.lm)$adj.r.squared,2), sep=""),
+ side=3, line=-2, adj=.5, col="red")
> pvalue <- coef(summary(fit.lm))[2,"Pr(>|t|)"],# extract p-value
> if (pvalue >= 0.001) {
+ mtext(paste("p=" , pvalue, sep=""), side=3, line=-2, adj=1, col="red")
> if (pvalue < 0.001) {
+ mtext("p<0.001", side=3, line=-2, adj=0.99, col="red")
```

Figure 27. Linear regression of binary transformed pH vs. NO3 in Storgama.

The linear model is significant, but common sense shows that an LM is not appropriate. Look at the range of fitted values compared to the range of original values:

```r
> range(pH.bin)
[1] NA NA
> range(pH.bin, na.rm=TRUE)
# na.rm=T can also be used with min(), max(), mean(), stddev(), etc.
[1] 0 1
> range(fitted(fit.lm))
[1] -0.5725272  0.9426986
```

LOGISTIC REGRESSION is the solution here: this method deals with binary response variable. R can also do logistic regression with the response variable in other formats, e.g. proportions. See ?glm

```r
> fit.glm <- glm(pH.bin ~ NO3.N, family=binomial)
The default is: family=binomial(link=logit)
Note that for GLM, predict() or predict.glm() gives us predicted y values on the logit-transformed scale (-Inf,Inf)....
> range(predict(fit.glm))
[1] -6.536935 2.391123
...while fitted() gives us fitted y values back-transformed to (0,1) scale.
> range(fitted(fit.glm))
[1] 0.001446826 0.916147870
```

The function predict.glm() doesn't seem to give confidence intervals directly, like predict.lm() does. But we will still get the standard errors, and can add them ourselves to plot confidence intervals.

```r
> pred.glm <- predict.glm(fit.glm, newdata=new.x, se.fit=T)
> y.lower <- pred.glm$fit - pred.glm$se.fit
> y.upper <- pred.glm$fit + pred.glm$se.fit
```
Now we must back-transform the predicted values to (0,1) scale before plotting them (Figure 28).

We'll make a little function to help us with this.

```r
> backtrans <- function(x) {
+   exp(x)/(1 + exp(x))# inverse function of logit(y) = log(y)/log(1-y)
+ }
> lines(new.x$NO3.N, backtrans(pred.glm$fit), col="green")
> lines(new.x$NO3.N, backtrans(y.lower), col="green", lty=2)
> lines(new.x$NO3.N, backtrans(y.upper), col="green", lty=2)
> mtext("GLM: Logistic reg.", side=3, line=.5, col="green")
> pvalue <- coef(summary(fit.glm))[2,"Pr(>|z|)"] # extract p-value
> if (pvalue >= 0.001) {
+   mtext(paste("p=", pvalue, sep=""), side=3, line=-3, adj=1, col="green")
+ }
> if (pvalue < 0.001) {
+   mtext("p<0.001", side=3, line=-3, adj=0.99, col="green")
+ }
```

As it turns out, plotting of the logistic regression prediction could be done simpler. The argument `type="response"` gives predicted probabilities \( p(y) \) (on scale 0:1), instead of the default log(odds) = \( \log(p(y)/(1-p(y))) \) (on scale \( -\infty:\infty \))

```r
> pred.glm <- predict.glm(fit.glm, newdata=new.x, se.fit=T, type="response")
> y.lower <- pred.glm$fit - pred.glm$se.fit
> y.upper <- pred.glm$fit + pred.glm$se.fit
> lines(new.x$NO3.N, y.lower, col="blue", lty=2)
> lines(new.x$NO3.N, y.upper, col="blue", lty=2)
```

The command `summary(fit.glm)` doesn't give any R2 value for a glm object. But a "pseudo R2" can be calculate from the deviance:

```r
> nres <- length(residuals(fit.glm))
> R2 <- (1 - exp((fit.glm$deviance -
+ fit.glm$null.deviance)/nres))/(1 - exp(- fit.glm$null.deviance/nres))
> mtext(paste("R2=", round(R2,2), sep=""), side=3, line=-3, adj=.5, col="green")
```

But for model selection with GLMs we'll use the deviance directly, rather than R2. Compare what you get with `anova(fit.lm)` and `anova(fit.glm)`: 

---

Figure 28. Linear regression (red) and logistic regression (green) of binary transformed pH vs. NO3 in Storgama.
> anova(fit.lm)      # gives Analysis of variance
Analysis of Variance Table
Response: pH.bin
   Df  Sum Sq Mean Sq F value Pr(>F)
NO3.N  1  54.303  54.303 390.28  < 2.2e-16 ***
Residuals 1550 215.666 0.139
> anova(fit.glm)    # gives Analysis of deviance
Analysis of Deviance Table
Model: binomial, link: logit
Response: pH.bin
Terms added sequentially (first to last)
   Df Deviance Resid. Df Resid. Dev
NULL                     1551    1651.97
NO3.N  1   287.28      1550    1364.69

A statistical theorem says that we can use the difference in deviance between "null model" and "full model" (given in the the anova table) to test if the parameter(s) in the full model is significant.

> dev.null <- anova(fit.glm)["NULL", "Resid. Dev"]
> dev.full <- anova(fit.glm)["NO3.N", "Resid. Dev"]
> 1 - pchisq(dev.null - dev.full, df=1)# df = difference in number of parameters
[1] 0

Actually, we can also ask for this test directly within anova():

> anova(fit.glm, test="Chi")
Analysis of Deviance Table
Model: binomial, link: logit
Response: pH.bin
Terms added sequentially (first to last)
   Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                     1551    1651.97
NO3.N  1   287.28      1550    1364.69 1.950e-64

> detach(DATA)

2.8.4 Script 4: Generalised additive models.

Let's go back to our original y variable.

attach(DATA)
plot(NO3.N, pH, xlab="NO3 (ug/L N)", ylab="pH")
fit.lm <- lm(pH ~ NO3.N)

Are we certain that a linear model is the best choice? Does pH really decrease lineary with NO3, no lower boundary? We could try various polynomial functions etc., but a more efficient solution is to use GAM for explorative data analysis (when you have "enough" data). See Figure 29.

> library(mgcv)
This is mgcv 1.3-16

> fit.gam <- gam(pH ~ s(NO3.N))
The argument s() is a non-parametric, flexible "spline function"

> summary(fit.gam)
Family: gaussian
Link function: identity
Formula: pH ~ s(NO3.N)
Parametric coefficients:
  Estimate Std. Error t value Pr(>|t|)
(Intercept) 4.698061  0.005334  880.7  <2e-16 ***
---
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Approximate significance of smooth terms:

<table>
<thead>
<tr>
<th>edf</th>
<th>Est.rank</th>
<th>F</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.652</td>
<td>9.000</td>
<td>51.89</td>
<td>&lt;2e-16 ***</td>
</tr>
</tbody>
</table>

---

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) = 0.228   Deviance explained = 23.2%
GCV score = 0.044412   Scale est. = 0.044165  n = 1552

The command `summary(fit.gam)` gives us R2 and p-value, but with slightly different names than in `summary(fit.lm)`.

```r
> names(summary(fit.gam))
[1] "p.coeff"       "se"            "p.t"           "p.pv"
[5] "residual.df"   "m"             "chi.sq"        "s.pv"
[9] "scale"         "r.sq"          "family"        "formula"
[13] "n"             "dev.expl"      "edf"           "dispersion"
[17] "pTerms.pv"     "pTerms.chi.sq" "pTerms.df"   "cov.unscaled"
[21] "cov.scaled"    "p.table"       "pTerms.table" "s.table"
[25] "gcv"
```

The GAM curve gives a better description of the relationship between variables (Figure 29). The CI are also more flexible. E.g. for NO3 > 600, you can't really conclude from the data whether pH increases or decreases.

**Figure 29.** Generalised additive model (GAM) regression of pH vs. NO3 in Storgama.

The GAM curve gives a better description of the relationship between variables (Figure 29). The CI are also more flexible. E.g. for NO3 > 600, you can't really conclude from the data whether pH increases or decreases.
"GCV" stands for generalised cross-validation - a method for automatic optimisation of smoothness together with model fitting. Estimated degrees of freedom is

```r
> summary(fit.gam)$edf
[1] 7.651956
```

But we can allow it to be more wiggly or more smooth manually. The `gam()` argument "gamma=" multiplies the "penalty" for wiggyness. So lower gamma means more wiggly (Figure 30, left panel); higher gamma means more smooth (Figure 30, right panel).

```r
> plot(NO3.N, pH, xlab="NO3 (ug/L N)", ylab="pH")
> fit.gam.2 <- gam(pH ~ s(NO3.N), gamma=.01)
> y.lower <- pred.gam$fit - pred.gam$se.fit
> y.upper <- pred.gam$fit + pred.gam$se.fit
> lines(new.x$NO3.N, y.lower, col="purple", lty=2)
> lines(new.x$NO3.N, y.upper, col="purple", lty=2)
> mtext("gamma=.01", side=3, line=.5, col="purple", adj=0)
```

Not so much difference from the optimised curve

```r
> fit.gam.3 <- gam(pH ~ s(NO3.N), gamma=100)
> y.lower <- pred.gam$fit - pred.gam$se.fit
> y.upper <- pred.gam$fit + pred.gam$se.fit
> lines(new.x$NO3.N, y.lower, col="turquoise", lty=2)
> lines(new.x$NO3.N, y.upper, col="turquoise", lty=2)
> mtext("gamma=100", side=3, line=.5, col="turquoise", adj=1)
```

Almost linear fit.

```r
> mtext("gamma=100", side=3, line=.5, col="turquoise", adj=1)
> mtext(paste("R2=" , round(summary(fit.gam.3)$r.sq, 2), sep=""),
+ side=3, line=-2, adj=.5, col="purple")
> pvalue <- summary(fit.gam.3)$s.pv# p-value for spline function
> if (pvalue >= 0.001) {
+ mtext(paste("p=" , pvalue, sep=""), side=3, line=-2, adj=1, col="purple")
+ }
> if (pvalue < 0.001) {
+ mtext("p<0.001", side=3, line=-2, adj=0.99, col="purple")
+ }
```
Figure 30. Generalised additive model (GAM) regression of pH vs. NO3 in Storgama, with high and low degree of smoothing.

detach(DATA)
3. TIME SERIES

3.1 Trends, seasons and autocorrelation

Jannicke Moe

The data are a series on nutrients etc. in River Glomma (from Per G. Stålnacke).

Question: is there a temporal trend in TN (total nitrogen)?

```r
> glomma.na <- read.delim("PGS_Glomma_TN.txt")
> names(glomma.na)
[1] "year" "month" "Q" "TN" "SPM" "TP" "Cu" "Pb"
```

The series consists of 13 years x 12 months, with some months lacking observations. Some of the functions we will use do not accept NAs, so we will replace these by other values.

```r
> (na.rows <- which(is.na(glomma.na$TN)))
[1]  74 109 124 133 144 152 156
```

Make a new data.frame, where we will replace NAs with numbers:

```r
> glomma <- glomma.na
```

We'll use the mean of all remaining years for that month (assuming there is no long-term trend):

```r
> for (i in na.rows) {   # i runs through rows with NAs
+  for (j in 3:8) {   # j runs through columns with variables
+    glomma[i,j] <- mean(glomma[glomma$month==glomma$month[i], j], na.rm=T)
+  }
+ }
```

For larger operations, functions like `apply()` are more efficient than `for()` loops. Below I will run some functions with both `glomma` and `glomma.na`, to test where NAs are accepted.

```r
> plot(glomma)
```

It is hard to spot any trend in Figure 31 (against year or month).
Figure 31. Pair-wise scatterplots for the data set glomma.

To see the trends more clearly, plot each physical/chemical variable (cols 3-8) against time (Figure 32).

```R
> par(mfrow=c(6,1), oma=c(3,3,2,1), mar=c(1,2,1,1))
> for (i in 3:8) {
+   plot(glomma.na[,i], type="l", col=i)
+   mtext(names(glomma.na)[i], side=2, outer=F, line=3)
+ }
> mtext("Time (months)", side=1, outer=T, line=1)
```
Figure 32. Each chemical variable in the data set glomma plotted against time.

3.1.1 Trend analysis by regression

We can start with a simple regression for TN (although this is not recommended by statisticians, since it invalidates some assumptions).

```r
> windows()
> par(mfrow=c(2,1))
```

Fit TN against time sequence counting from 1:end (unit months)

```r
> fit.1 <- lm(glomma$TN ~ seq(1:nrow(glomma)))
```

Here I added `seq()` because `lm()` got confused by the `:`.

```r
> plot(glomma$TN, type="l") # See Figure 33
> abline(coef(fit.1), col="red")
```

We can decompose the model into year and month (month as a categorical covariate), with year.no starting at 1:

```r
> glomma$year.no <- glomma$year - min(glomma$year) + 1
> fit.2 <- lm(glomma$TN ~ glomma$year.no + as.factor(glomma$month))
> summary(fit.2)
```

```r
Call:
  lm(formula = glomma$TN ~ glomma$year.no + as.factor(glomma$month))

Residuals:
  Min      1Q  Median      3Q     Max
-289.92  -95.05  -32.65   64.42  679.93

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)  
(Intercept)       637.253     52.026  12.249  < 2e-16 ***
glomma$year.no   3.574       3.532   1.012  0.313235
```

```r
> plot(1:150, glomma$TN, type="l")
> abline(coef(fit.1), col="red")
> abline(coef(fit.2), col="blue")
```
Summer months give strongest seasonal effect, as could be expected.

We obtain the slope per year from `coef(fit.2)[2]`. Divide this by 12 to get the slope per month, as in the plot.

```R
> abline(a=coef(fit.2)[1], b=coef(fit.2)[2]/12, col="green")
```

This gives a much better fit (check R2 and p values), but there is still no significant trend per year.

In stead of having 12 different parameter estimates for the months, we can use a non-linear fit with splines in GAM.

```R
> library(mgcv)
> fit.3 <- gam(glomma$TN ~ glomma$year.no + s(glomma$month))
> windows()
> par(mfrow=c(3,1))
```

Look at the effect of months (Figure 33, upper panel). Note that this plot is centered around zero on y-axis.

```R
> plot(fit.3)
```

Also look at the linear trend (Figure 33, middle panel). Here the unit is years.

```R
> plot(unique(glomma$year.no), coef(fit.3)[2]*unique(glomma$year.no), type="l", + ylim=c(-200,200))
```

Standard deviations can be added in a cumbersome way:

```R
> lines(unique(glomma$year.no), + (coef(fit.3)[2] + summary(fit.3)$se[2])*unique(glomma$year.no), lty=2)
> lines(unique(glomma$year.no), + (coef(fit.3)[2] - summary(fit.3)$se[2])*unique(glomma$year.no), lty=2)
```

Compare the fitted model with the data (Figure 33, lower panel):

```R
> plot(glomma$TN, type="l")
> lines(fitted(fit.3), type="l", col="red")
```
Figure 33. Trend estimations for TN from Glomma. Upper panel: Effect of month on TN, estimated by GAM. Middle panel: linear effect of year on TN. Lower panel: comparison of data and model prediction.

Of course, the trend can also be modelled as splines instead of as a line (Figure 34).

```r
> fit.4 <- gam(glomma$TN ~ s(glomma$month) + s(glomma$year.no))
> windows()
> par(mfrow=c(3,1))
> plot(fit.4)  # Plots estimate for each predictor variable in separate plots
> # NB: Here R asks for a "return", therefore I've added an empty line.
> plot(glomma$TN, type="l")
> lines(fitted(fit.4), col="red")
```

The trend is not significantly non-linear, and not significantly != zero, according to this plot.
Next we'll make a so-called time-series object, which enables us to do some "proper" time-series analyses. These methods accept serial correlation, and can do operations like decomposition into trend and season more efficiently than regression. In return they demand strictly regular series (equal time intervals), and can be unfriendly towards NAs.

3.1.2 Make time-series object

Time-series plot (Figure 35):
```r
> plot.ts(glomma[,3:8], nc=1)  # PS. Note difference from the command plot()
> plot.ts(glomma.na[,3:8], nc=1)  # Here NAs are accepted
```
Figure 35. Time-series plot for the data set glomma.

But R doesn’t know yet that glomma is a time series:

```R
> is.ts(glomma)
[1] FALSE
```
… so we’ll make it into a time series.

```R
> glomma.ts <- ts(glomma[,3:8], start=c(1990, 1), frequency=12,
+     names=names(glomma[3:8]))
```

The time variables are now defined by `start=` and `frequency=`. R assumes that `frequency=12` means months. Here: `start=c(year, month)`, but could be defined with other time units. NB: the time series must have regular intervals! (We can add NAs to obtain this.)

```R
> glomma.na.ts <- ts(glomma.na[,3:8], start=c(1990, 1), frequency=12,
+     names=names(glomma.na[3:8]))
> is.ts(glomma.na.ts)
[1] TRUE
```

Now glomma.ts is no longer a data.frame but a matrix (unfortunately),

```R
> is.data_frame(glomma.ts)
[1] FALSE
> is.matrix(glomma.ts)
[1] TRUE
```
… so glomma.ts doesn’t have "names" (`$col.name`), but "dimnames" [,"col.name"].

```R
> glomma.ts$TN
NULL
```
The rownames and colnames of `glomma.ts[, "TN"]` are not readily available. Note that time variables are not columns in the matrix `glomma.ts`, they are just stored as the time series' attributes.

```r
> attributes(glomma.ts)  # tsp gives start, end, frequency
$dim
[1] 156  6

$dimnames
$dimnames[[1]]
NULL
$dimnames[[2]]
[1] "Q" "TN" "SPM" "TP" "Cu" "Pb"

$tsp
[1] 1 156  1

$class
[1] "mts" "ts"
```

Plot the time-series object (Figure 36):

```r
> plot.ts(glomma.ts, nc=1)
```

![glomma.ts](image)

**Figure 36.** Time-series plot of time-series object `glomma.ts`.  

---

> glomma.ts[, "TN"]

Time Series:
Start = 1
End = 156
Frequency = 1

<table>
<thead>
<tr>
<th></th>
<th>564.0000</th>
<th>774.0000</th>
<th>680.0000</th>
<th>610.0000</th>
<th>355.0000</th>
<th>374.0000</th>
<th>444.0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>329.0000</td>
<td>321.0000</td>
<td>431.0000</td>
<td>471.0000</td>
<td>483.0000</td>
<td>991.0000</td>
<td>517.0000</td>
</tr>
<tr>
<td>15</td>
<td>796.0000</td>
<td>730.0000</td>
<td>528.0000</td>
<td>333.0000</td>
<td>428.0000</td>
<td>381.0000</td>
<td>378.0000</td>
</tr>
</tbody>
</table>

[etc.]
Notice difference on x-axis compared to Figure 35, where we have used the same function `plot.ts()` but plotted the data frame `glomma` rather than the time-series object `glomma.ts`. On the other hand, the ordinary `plot()` function used on a ts object will also behave like the function `plot.ts()`.

```r
> plot(glomma.ts, nc=1)
```

### 3.1.3 Trend analysis by non-parametric trend test

Mann-Kendall is a simple non-parametric test (not making assumptions about data distribution), checking whether there is a monotonous trend (not necessarily linear).

```r
> library(Kendall)
> MannKendall(glomma$TN)
tau = 0.0662, 2-sided p-value =0.22136
```

We can try to account for seasonal variation. This test requires a time-series object. Here, seasons are implicit (given by the time series' frequency).

```r
> SeasonalMannKendall(glomma.ts[,"TN"])
tau = 0.140, 2-sided pvalue =0.020947
```

Now it’s significant, but less significant than the regression test (compare with `summary(fit.2)`), as it should be.

### 3.1.4 Trend analysis by seasonal decomposition

Now that we have a ts object, we can decompose it into trend, season, and residuals. The function `stl()` does this with loess smoothing. Let's try this decomposition with TN.

```r
> TN.stl <- stl(glomma.ts[,"TN"], s.window="periodic")
s.window = span for seasonal extraction, t.window = for trend extraction, set by R if not given
> plot(TN.stl, col.range="yellow", main=paste("Seasonal decomposition, t.window=", + TN.stl$win[2])
```
Figure 37. Seasonal decomposition of time-series object glomma.ts.

The yellow bars indicate the difference in scale for the different components. NAs are apparently not welcome here:

```r
> TN.stl <- stl(glomma.na.ts[, "TN"], s.window="periodic", na.action=na.exclude)
Error in stl(glomma.na.ts[, "TN"], s.window = "periodic", na.action = na.exclude) :  
  series is not periodic or has less than two periods
> TN.stl <- stl(glomma.na.ts[, "TN"], s.window="periodic", na.action=na.omit)
Error in na.omit.ts(as.ts(x)) : time series contains internal NAs
> TN.stl <- stl(glomma.na.ts[, "TN"], s.window="periodic", na.action=na.pass)
Error in stl(glomma.na.ts[, "TN"], s.window = "periodic", na.action = na.pass) :  
  NA/NaN/Inf in foreign function call (arg 1)
```

We can make the the trend estimation more smooth or less smooth (Figure 38)

```r
windows()
plot(stl(glomma.ts[, "TN"], s.window="periodic", t.window=35), col.range="yellow",  
     main="Seasonal decomposition, t.window=35")
windows()
plot(stl(glomma.ts[, "TN"], s.window="periodic", t.window=7), col.range="yellow",  
     main="Seasonal decomposition, t.window=7")
```
Figure 38. Seasonal decomposition of time-series object glomma.ts, with more smoothing (left panel) and less smoothing (right panel).

Which month gives peak TN, according to our seasonal decomposition?
> which.max(TN.stl$time.series[1:12])  # Which month during first 12 months?
[1] 12
Which are the first and second highest peak?
> rev(order(TN.stl$time.series[1:12]))[1:2]
[1] 12  4
Consider: is the double peak in seasonal effect reasonable?

3.1.5 Autocorrelation

Time-series data are often correlated in time (a reason why regression should not be used). We can check the autocorrelation as a function of number of lags (here: months)
> windows()
> par(mfrow=c(3,1))
> acf(glomma[,"TN"])
# Original data.frame: unit is months
> acf(glomma.ts[,"TN")
# Time-series object: unit is years
Figure 39. Autocorrelation plots for dataset glomma (upper panel) and for time-series object glomma.ts.

The ACF plot (Figure 39) indicates significant autocorrelation for lag around 11-13 months, which should not be a surprise.

Autocorrelation can also be tested with an autoregressive model \texttt{ar()}, which uses AIC to select number of significant lags

\begin{verbatim}
> ar(glomma.ts[, "TN"])
Call:
ar(x = glomma.ts[, "TN"])
Coefficients:
  1        2        3        4        5        6        7        8
0.1018   0.0847  -0.0421  -0.0393  -0.1454  -0.0476  -0.1188  -0.0045
  9       10       11       12       13
0.1404  -0.1090   0.0524   0.2029   0.1704
Order selected 13  sigma^2 estimated as  32336
\end{verbatim}

Here, the estimated order of 13 lags corresponds to 13 months. Let's try also the time series with NA's.

\begin{verbatim}
> ar(glomma.na.ts[, "TN"], na.action=na.exclude)
Call:
ar(x = glomma.na.ts[, "TN"], na.action = na.exclude)
Coefficients:
  1        2        3        4        5        6
0.1738   0.0041   0.0175  -0.1133  -0.1111  -0.1777
  7       8
-0.1500  -0.1777
Order selected 7  sigma^2 estimated as  36083
\end{verbatim}

The NA version gives a different result, so the function \texttt{ar()} does not seem robust to missing values...

We can also test the residuals from the seasonal decomposition. We have seen the residuals in the \texttt{stl} plot so we know they exist, but how do we get the numbers?

\begin{verbatim}
> names(TN.stl)
[1] "time.series" "weights" "call" "win" "deg"
[6] "jump" "inner" "outer"
\end{verbatim}
Check what "time.series" contains:
> names(TN.stl$time.series)
NULL
When names() does not work, it can be useful to try dimnames()
> dimnames(TN.stl$time.series)
[[1]]
NULL
[[2]]
[1] "seasonal" "trend" "remainder"

> windows()
> par(mfrow=c(3,1))
> acf(glomma.ts[, "TN"])
> acf(TN.stl$time.series[, "remainder"]]) # (unit is years)
> ar(TN.stl$time.series[, "remainder"])

Call:
ar(x = TN.stl$time.series[, "remainder"])

Coefficients:
     1      2      3      4      5      6      7      8
-0.3867 -0.3873 -0.4804 -0.4980 -0.5335 -0.4663 -0.5522 -0.4707
      9 10   11    12    13    14    15    16
-0.2449 -0.5266 -0.4503 -0.3780 -0.2023 -0.2896 -0.2729 -0.0976
   17 18
-0.1862 -0.2192
Order selected 18  sigma^2 estimated as  16490

Now there is not much autocorrelation left, as expected (Figure 40, upper panel).

What about the trend?
> acf(TN.stl$time.series[, "trend"])
> ar(TN.stl$time.series[, "trend"])

Call:
ar(x = TN.stl$time.series[, "trend"])

Coefficients:
     1      2      3
1.4052 -0.1569 -0.2988
Order selected 3  sigma^2 estimated as  59.95

There is strong autocorrelation is in the trend (Figure 40, middle panel), as expected.

The estimated seasonal effect will of course also have a strong autocorrelation (Figure 40, lower panel).
> acf(TN.stl$time.series[, "seasonal"])

NIVA 5524-2007
Figure 40. Autocorrelation plots for remainders (residuals) from AR-model of dataset glomma (upper panel); for the estimated temporal trend (middle panel); and for the seasonal effect (lower panel).

There is a large framework for time-series analysis called ARIMA, if interested see ?arima

3.2 Structural changes in time series

Tom Andersen

Data set from Ringkøbing fjord, Denmark (DMU). The data are from 1980-2004, averaged to monthly means.

> ring1 <- read.table("Ringkøbing monthly TS.txt", header=TRUE)
> names(ring1)
[1] "Year"  "Month"  "Nsamp"  "Temp"  "Sal"  "Transp"  "Chla"  "TotP"
[9] "PO4"  "TotN"  "NO3"
> summary(ring1)

Year          Month            Nsamp            Temp
Min.   :1980   Min.   : 1.000   Min.   :0.000   Min.   : 0.000
1st Qu.:1986   1st Qu.: 4.000   1st Qu.:1.000   1st Qu.: 4.200
Median :1992   Median : 7.000   Median :2.000   Median : 9.800
3rd Qu.:1998   3rd Qu.:10.000   3rd Qu.:3.000   3rd Qu.:15.300
Max.   :2004   Max.   :12.000   Max.   :5.000   Max.   :22.030
(etc.)
There are a lot of missing values, especially for Chla. Where are they? Let's make a function to count missing values

```r
> count.na <- function(x) { sum(is.na(x)) }
```

Let's test our function, first columnwise.

```r
> apply(ring1,2,count.na)
    Year Month Nsamp   Temp   Sal Transp   Chla TotP   PO4   TotN   NO3
0     0     0     33     32     31     64     26     45     26     26

OK, this gives the same results as `summary()`.

```r
> attach(ring1)
> TotP.missing <- tapply(TotP, Year, count.na)
> plot(1980:2004,TotP.missing) # Figure 41
```

![Figure 41. Number of missing TotP values per year, in the data from Ringkøbingfjord.](image)

Notice the difference between `apply()` and `tapply()` (as commented below):

```r
> apply(ring1, 2, count.na) # = number of missing in each column of ring1
    Year Month Nsamp   Temp   Sal Transp   Chla TotP   PO4   TotN   NO3
0     0     0     33     32     31     64     26     45     26     26

> tapply(TotP, Year,count.na) # = number of missing TotP in each year
  0    6    3    2    3    3    4    0    0    0    1    0    0    0    0    0
  1    0    0    0    0    0    0    0    0
```

> detach(ring1)

Most missing values are in the years before 1988. Let's make a subset of the years 1988-2004:

```r
> ring1.1988 <- subset(ring1, Year > 1987)
```

Now, let's make a time-series (ts) object out of this. We skip the first 3 columns containing time information, since time is implicit in the ts object definition.

```r
> ring1.ts <- ts(ring1.1988[4:11], frequency=12, start=c(1988,1))
> plot(ring1.ts) # Figure 42
```
Figure 42. Time-series plot of data from Ringkøbingfjord.

Something very special seems to have happened in 1995! In the summer of 1995 the outlet of Ringkøbing fjord was widened so that the seawater exchange increased. The resulting increase in salinity allowed filter-feeding clams to establish, leading to a fast decrease in chlorophyll.

Let’s use the strucchange package to investigate this regime shift. (Notice that strucchange depends on other packages, therefore installation from local zip file is not recommended.)

> require(strucchange)
Loading required package: strucchange
Loading required package: zoo
Loading required package: sandwich
[1] TRUE

Fstats() calculates the F-statistic for all possible breakpoints of a linear model. This significance of the identified breakpoint(s) can be tested with sctest().

Let’s test if there is a change in the mean of some variables. The model formula for x = constant is x ~ 1.

> par(mfrow=c(2,2)) # Figure 43

Temperature:
> fs.temp <- Fstats(Temp ~ 1, data=ring1.ts)
> plot(fs.temp, main="Temperature")
> lines(breakpoints(fs.temp))
> sctest(fs.temp)
sup.F = 1.1378, p-value = 0.982

Breakpoint(s) are detected, but not significant.
Salinity:
> fs.Sal <- Fstats(Sal ~ 1, data=ring1.ts)
> plot(fs.Sal, main="Salinity")
> lines(breakpoints(fs.Sal))
> sctest(fs.Sal)
sup.F = 47.0139, p-value = 3.677e-10
Highly significant breakpoint in March 1995

Chlorophyll a:
> fs.Chla <- Fstats(Chla ~ 1, data=ring1.ts)
> plot(fs.Chla, main="Chlorophyll")
> lines(breakpoints(fs.Chla))
> sctest(fs.Chla)
sup.F = 240.5308, p-value < 2.2e-16
Highly significant breakpoint in October 1995

> fs.TotP <- Fstats(TotP ~ 1, data=ring1.ts)
> plot(fs.TotP, main="Total P")
> lines(breakpoints(fs.TotP))
> sctest(fs.TotP)
sup.F = 122.1825, p-value < 2.2e-16
# Highly significant breakpoint in November 1995

Figure 43. Breakpoint analyses for data from Ringkøbingfjord.

Now let's try a more complicated model: There were significant breakpoints in both the Chla and TotP time series. Was there also a change in the relationship between them? (Figure 44)
> fs.Chla.TotP <- Fstats(log10(Chla) ~ log10(TotP), data=ring1.ts)
> plot(fs.Chla.TotP, main="Chlorophyll vs. Total P")
> lines(breakpoints(fs.Chla.TotP))
> sctest(fs.Chla.TotP)
sup.F = 148.9023, p-value < 2.2e-16
Yes, highly significant, but half a year later (July 1996).
We can visualize the two regression lines in a plot (Figure 44):

```r
> bp <- fs.Chla.TotP$breakpoint
> plot(log10(Chla) ~ log10(TotP), data = ring1)
> points(log10(Chla) ~ log10(TotP), data = ring1, subset = 1:bp, col = 2, pch = 19)
> abline(lm(log10(Chla) ~ log10(TotP), data = ring1, subset = 1:bp), col = 2)
> abline(lm(log10(Chla) ~ log10(TotP), data = ring1, subset = -(1:bp)))
```

**Figure 44.** Breakpoint analysis for relationship between Total P and chlorophyll from Ringkøbingfjord.

So, the relationship between Chla and TotP became stronger after the establishment of benthic filter feeders. The reason is possibly that Chla yield per unit TotP was light limited before mussel invasion, and because mussel removal of TotP makes the TotP gradient longer. The 3 outliers (2 before, 1 after) are probably winter values. Maybe the next step would be to look for a seasonal component?
4. MULTIVARIATE MODELS IN VEGAN

Robert Ptacnik

Community analysis using vegan

• Basic terms in community analysis
  - metric vs. non-metric methods
  - constrained vs. unconstrained methods

• When to choose which approach

• Preparation of data

• Procrustes rotation

• Extract species optima
What is ordination?

Reduce dimensionality in multivariate data

-> organize your data such that overriding gradients become visible ('gradient analysis')

(From http://ordination.okstate.edu/index.html)

Constrained vs. unconstrained ordination

Unconstrained: Obtain distribution from species' information.

-> Find best ordination with respect to community data.

Constrained: Force analysis to build axes based on environmental variables.

-> See how data can be organized based on environmental data.
### Ordination results: 2 types of scores

#### Species scores

<table>
<thead>
<tr>
<th>Species</th>
<th>CCA1</th>
<th>CA1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudokephyron</td>
<td>-0.5554</td>
<td>0.002153</td>
</tr>
<tr>
<td>Cyclotella</td>
<td>0.04614</td>
<td>-1.49453</td>
</tr>
<tr>
<td>Bitrichia</td>
<td>-0.93988</td>
<td>0.15188</td>
</tr>
<tr>
<td>Mallomonas</td>
<td>0.21731</td>
<td>0.019096</td>
</tr>
<tr>
<td>Peridinium</td>
<td>0.049038</td>
<td>0.01634</td>
</tr>
<tr>
<td>Oocystis</td>
<td>-0.15936</td>
<td>0.15644</td>
</tr>
<tr>
<td>Chromulina</td>
<td>-0.32073</td>
<td>0.37128</td>
</tr>
<tr>
<td>Monoraphidium</td>
<td>-0.09795</td>
<td>-0.04223</td>
</tr>
<tr>
<td>Dinobryon</td>
<td>-0.26206</td>
<td>0.013426</td>
</tr>
<tr>
<td>Gymnodinium</td>
<td>-0.15916</td>
<td>-0.04174</td>
</tr>
<tr>
<td>Ochromonas</td>
<td>-0.03809</td>
<td>0.42055</td>
</tr>
<tr>
<td>Katablepharis</td>
<td>0.112106</td>
<td>-0.15999</td>
</tr>
<tr>
<td>Rhodomonas</td>
<td>0.148267</td>
<td>-0.06726</td>
</tr>
<tr>
<td>Cryptomonas</td>
<td>0.475516</td>
<td>0.160346</td>
</tr>
<tr>
<td>Ochromonas (sp.)</td>
<td>-0.25189</td>
<td>0.079738</td>
</tr>
<tr>
<td>Others</td>
<td>-0.1951</td>
<td>0.01077</td>
</tr>
</tbody>
</table>

#### Site scores

<table>
<thead>
<tr>
<th>Site</th>
<th>CCA1</th>
<th>CA1</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_1886</td>
<td>-3.65402</td>
<td>0.660388</td>
</tr>
<tr>
<td>NO_3230</td>
<td>-2.76736</td>
<td>0.175686</td>
</tr>
<tr>
<td>NO_1517</td>
<td>-0.72224</td>
<td>0.149224</td>
</tr>
<tr>
<td>NO_3251</td>
<td>-3.43771</td>
<td>-0.0739</td>
</tr>
<tr>
<td>NO_270</td>
<td>-3.93701</td>
<td>-0.0054</td>
</tr>
<tr>
<td>NO_2009</td>
<td>-1.38994</td>
<td>-3.7422</td>
</tr>
<tr>
<td>NO_4401</td>
<td>-1.14979</td>
<td>-1.48201</td>
</tr>
<tr>
<td>NO_4495</td>
<td>-3.54723</td>
<td>0.990243</td>
</tr>
<tr>
<td>NO_4516</td>
<td>-2.39429</td>
<td>0.204494</td>
</tr>
<tr>
<td>NO_5083</td>
<td>-1.08829</td>
<td>0.80939</td>
</tr>
</tbody>
</table>

![CCA1 vs CCA2 plot](image-url)
In the process of ordination, axes are calculated which represent the (dis-) similarity of samples and species, with most similar observations most close to each other. The maximum variation that can be explained in a one-dimensional space is translated into the first ordination axis (CCA1 below). Next, as much of the remaining variation that can be explained (again in a one-dimensional space) is being projected onto the second axis, and so on.

How much of the variability can be explained by the 1st, 2nd, 3rd... axis? ->‘inertia’.

In case of constrained ordination the inertia is split into constrained and unconstrained component.

<table>
<thead>
<tr>
<th>Inertia Rank</th>
<th>Total</th>
<th>0.80328</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Constrained</td>
<td>0.05322</td>
</tr>
<tr>
<td></td>
<td>Unconstrained</td>
<td>0.75006</td>
</tr>
</tbody>
</table>

Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
CCA1 0.05322

Eigenvalues for unconstrained axes:
CA1 0.122600
CA2 0.111737

Significance of results can be tested with permutation test.
### Types of ordination

<table>
<thead>
<tr>
<th>Axes are...</th>
<th>Unconstrained</th>
<th>Constrained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method for scores..</td>
<td>CA (DCA)</td>
<td>CCA</td>
</tr>
<tr>
<td>Chi² distances weighted averaging</td>
<td>Correspondence analysis</td>
<td>Canonical correspondence analysis</td>
</tr>
<tr>
<td>Euclidean distances least squares</td>
<td>PCA</td>
<td>RDA</td>
</tr>
<tr>
<td>Dissimilarity matrix rank-based</td>
<td>NMDS</td>
<td>-</td>
</tr>
<tr>
<td>Non-metric multidimensional scaling</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Constrained vs. unconstrained ordination

**Pros and Cons**

**Unconstrained:** Obtain distribution from species' information

**Constrained:** Force analysis to build axes based on environmental variables

**Advantage:** 'let the community tell you how sites are related to each other'

Species can be more easily associated with environmental data

**Disadvantage:** Analysis more time-consuming; potential problems with large datasets (NMDS);

The sites are grouped based on their true variation

**Disadvantage:** Only the variation that can be related to env. variables will be visible
### Types of ordination - when to choose which

<table>
<thead>
<tr>
<th>Method</th>
<th>Unconstrained</th>
<th>Constrained</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>weighted averaging</strong></td>
<td>CA</td>
<td>CCA</td>
</tr>
<tr>
<td></td>
<td>many species have zero observations</td>
<td></td>
</tr>
<tr>
<td><strong>least squares</strong></td>
<td>PCA</td>
<td>RDA</td>
</tr>
<tr>
<td></td>
<td>gradual differences in community</td>
<td></td>
</tr>
<tr>
<td><strong>rank-based</strong></td>
<td>NMDS</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>binary data (pres./abs.)</td>
<td></td>
</tr>
</tbody>
</table>

### Types of ordination - commands in vegan

<table>
<thead>
<tr>
<th>Method</th>
<th>Unconstrained</th>
<th>Constrained</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>weighted averaging</strong></td>
<td>CA (spec)</td>
<td>CCA (spec, env)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>least squares</strong></td>
<td>PCA (spec)</td>
<td>RDA (spec, env)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>rank-based</strong></td>
<td>NMDS (spec)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

spec: species matrix  
env: environmental predictors
Constrained vs. unconstrained ordination

Two approaches

- start with unconstrained analysis (NMDS, CA, PCA)
  ```r```
  ordi <- cca(resp)
```
- start with constrained analysis (CCA, RDA)
  ```r```
  cca(resp, env)
```

- fit environmental variables to ordination (incl. permutation test)
  ```r```
  envfit(ordi, env, perm=1000)
```
- improve model with formula interface
  ```r```
  cca(resp~Fac1+Fac2*(Fac3+Fac4))
```
- select most relevant variables and plot them into ordination
- test variables by permutation test
- run reduced model

Structure of data

- no missing values accepted
- sums of each single row and column must be >0 (species)
- factors can be used (environ. variables)

Use short identifiers (e.g. TP instead of Total Phosphorus (µg/L))
>> easier to read on plot

<table>
<thead>
<tr>
<th>rowname</th>
<th>Cal.val</th>
<th>Emp.nig</th>
<th>Led.pal</th>
<th>Vac.myr</th>
<th>Vac.vft</th>
<th>Pln.yi</th>
<th>Des.fle</th>
<th>Bet.pub</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.1</td>
<td>0.25</td>
<td>11.13</td>
<td>0</td>
<td>0</td>
<td>17.0</td>
<td>0.07</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.2</td>
<td>0.67</td>
<td>0.17</td>
<td>0</td>
<td>0.36</td>
<td>12.13</td>
<td>0.12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.3</td>
<td>0.1</td>
<td>1.55</td>
<td>0</td>
<td>0</td>
<td>13.47</td>
<td>0.26</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.42</td>
<td>7.92</td>
<td>15.97</td>
<td>0</td>
<td>3.7</td>
</tr>
<tr>
<td>S.5</td>
<td>0</td>
<td>12.68</td>
<td>0</td>
<td>0</td>
<td>22.73</td>
<td>0.03</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.6</td>
<td>0</td>
<td>0.92</td>
<td>0</td>
<td>2.42</td>
<td>10.28</td>
<td>0.12</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td>S.7</td>
<td>4.73</td>
<td>5.12</td>
<td>1.55</td>
<td>6.05</td>
<td>12.4</td>
<td>0.1</td>
<td>0.78</td>
<td>0.02</td>
</tr>
<tr>
<td>S.8</td>
<td>4.47</td>
<td>5.33</td>
<td>0</td>
<td>2.15</td>
<td>4.33</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.9</td>
<td>0</td>
<td>1.53</td>
<td>0.36</td>
<td>19.27</td>
<td>7.13</td>
<td>0.05</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>S.10</td>
<td>0.21</td>
<td>1.9</td>
<td>0.07</td>
<td>0.22</td>
<td>6.3</td>
<td>0.12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.11</td>
<td>3.76</td>
<td>6.85</td>
<td>0</td>
<td>0.08</td>
<td>5.3</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.12</td>
<td>0.02</td>
<td>6.45</td>
<td>0</td>
<td>0</td>
<td>14.13</td>
<td>0.07</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.13</td>
<td>0</td>
<td>6.93</td>
<td>0</td>
<td>0</td>
<td>10.5</td>
<td>0.02</td>
<td>0.1</td>
<td>0.02</td>
</tr>
<tr>
<td>S.14</td>
<td>0</td>
<td>0.83</td>
<td>0</td>
<td>0</td>
<td>8.2</td>
<td>0</td>
<td>0.05</td>
<td>0</td>
</tr>
<tr>
<td>S.15</td>
<td>0</td>
<td>0.13</td>
<td>0</td>
<td>0</td>
<td>2.75</td>
<td>0.03</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.16</td>
<td>0.5</td>
<td>5.75</td>
<td>0</td>
<td>0</td>
<td>10.5</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.17</td>
<td>0.03</td>
<td>3.05</td>
<td>0</td>
<td>0</td>
<td>4.43</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S.18</td>
<td>3.4</td>
<td>0.63</td>
<td>0</td>
<td>0</td>
<td>1.99</td>
<td>0.05</td>
<td>0.05</td>
<td>0</td>
</tr>
<tr>
<td>S.19</td>
<td>0.05</td>
<td>8.3</td>
<td>0</td>
<td>0</td>
<td>8.5</td>
<td>0.03</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Structure of data: usually 2 matrices with matching rows

<table>
<thead>
<tr>
<th>dependent matrix (community)</th>
<th>independent matrix (env. parameters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rows must match</td>
<td></td>
</tr>
</tbody>
</table>

Structure of data

what to do with missing values?
- delete column (loose species/predictor)
- delete row (delete observation)
- fill gap with predicted value (requires some consideration)
Comparing ordinations - Procrustes rotation

Procrustes analysis determines a **linear transformation** (translation, reflection, orthogonal rotation, and scaling) of the points in matrix Y to best conform them to the points in matrix X.

(...) Theseus “**fitted**” Procrustes to his own bed and cut off his head and feet. (...) Killing Procrustes was the last adventure of Theseus on his journey from Troezen to Athens. (from Wikipedia)
4.1 Multivariate analyses: Examples

4.1.1 Example 4: Basics in vegan

> rm(list=ls())  # clean workspace
> require(vegan)  # load vegan
Loading required package: vegan
[1] TRUE

Let's start with an example dataset from Jari Oksanen.
> data(varespec)  # data on distribution of 44 understorey plant species on 24 sites
> data(varechem)  # environmental data for the sampling sites

First we use Nonmetric Multidimensional Scaling (NMDS)
> x11(10,5)  # set size of window: x11(width, height)
> par(mfrow=c(1, 2))  # split into two panels
> nml <- metaMDS(varespec)
Square root transformation
Wisconsin double standardization
Loading required package: MASS
Run 0 stress 18.44915
Run 1 stress 23.60978
Run 2 stress 21.43612
Run 3 stress 22.97361
Run 4 stress 19.82376
Run 5 stress 19.48413
Run 6 stress 22.81606
Run 7 stress 21.37383
Run 8 stress 19.5049
Run 9 stress 18.25658
... New best solution
... rmse 0.04516871 max residual 0.1694442
Run 10 stress 20.4831
Run 11 stress 26.25915
Run 12 stress 19.69805
Run 13 stress 18.25658
... New best solution
... rmse 4.832084e-05 max residual 0.0001559065
*** Solution reached

> nml # inspect result

Call:
metaMDS(comm = varespec)

Nonmetric Multidimensional Scaling using isoMDS (MASS package)

Data:  wisconsin(sqrt(varespec))
Distance: bray

Dimensions: 2
Stress:  18.25658
Two convergent solutions found after 13 tries
Score scaling: centring, PC rotation, halfchange scaling

> plot(nml)
R doesn't want to show us the species. OK, plot in two steps:
> plot(nml, type="n")  # empty plot
> text(nml, "species", col=2)  # the species
> text(nml, "sites", col=1)
Fitting environmental data to the NMDS:

envfit() fits environmental variables into existing ordination.

If 'perm=xxx' is given, the fit is tested with xxx permutations and statistics are returned.

```r
> (ef.nm1 <- envfit(nm1, varechem, perm = 1000))

***VECTORS
NMDS1   NMDS2   r2  Pr(>|r|)
N     -0.057241 -0.998360 0.2536  0.043 *
P      0.619606  0.784913 0.1938  0.100 .
K      0.766361  0.642411 0.1809  0.135
Ca     0.685118  0.728432 0.1119  0.180
Mg     0.632459  0.774594 0.1507  0.161
S      0.191286  0.981534 0.1752  0.139
Al    -0.871651  0.490127 <0.001 ***
Fe    -0.936054  0.351857 0.0450  0.100.
Mn     0.798774 -0.601632 0.1879  0.122
Zn     0.617446  0.786613 0.1752  0.139
Mo    -0.903091  0.429449 0.1223  0.100 .
Baresoil  0.924936 -0.380124 0.2508  0.100 .
Humdepth 0.932874 -0.360203 0.5200  0.001 ***
pH    -0.648097  0.761558 0.2308  0.077 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that when commands are written in parantheses, the results are written in the console.

Add environmental gradients:
```r
> plot(ef.nm1)  # note: plot.envfit draws into existing ordination
> plot(ef.nm1, p.max = 0.05, col = "red")  # highlight the significant variables
```

Arrows and names look a bit messy together (not shown) - why not split into two plots (Figure 45):
```r
> par(mfrow=c(1, 2))
> plot(nm1) #
```

Note that this plot may look slightly different on your pc, because nmds() starts with arbitrary initial values and converges to an optimal solution from these initial values.
```r
> plot(ef.nm1)  # add environmental gradients
> plot(ef.nm1, p.max = 0.05, col = "red")  # highlight the significant ones
> plot(nm1, type="n")  # empty plot
> text(nm1, "species", col=2)  # the species
> text(nm1, "sites", col=1)
```
**Figure 45.** Non-metric multidimensional scaling (NMDS) for the *varespec*-data.

Compare with PCA (Figure 46):

```r
> x11(6, 6)
> ca1 <- cca(varespec)
> plot(ca1)
```

Note that `pca()` shows the species names. There are different settings for default plot function.

```r
> (ef.ca1 <- envfit(ca1, varechem, perm = 1000))
```

### VECTORS

|       | CA1    | CA2    | r2   | Pr(>|r|) |
|-------|--------|--------|------|----------|
| N     | 0.474695 | -0.880150 | 0.2196 | 0.086 .  |
| P     | 0.448265 | 0.893901  | 0.3054 | 0.020 *  |
| K     | 0.736164 | 0.716838  | 0.3064 | 0.025 *  |
| Ca    | 0.697240 | 0.676803  | 0.2466 | 0.057 .  |
| Mg    | 0.773175 | 0.998680  | 0.4995 | 0.001 ***|
| S     | 0.051368 | 0.631921  | 0.0902 | 0.389    |
| Al    | -0.974910 | -0.496790 | 0.4995 | 0.001 ***|
| Fe    | -0.963899 | -0.476270 | 0.3682 | 0.011 *  |
| Mn    | 0.914377 | 0.995470  | 0.1766 | 0.133    |
| Zn    | 0.770385 | 0.637578  | 0.1766 | 0.133    |
| Mo    | -0.638085 | -0.769966 | 0.0539 | 0.588    |
| Baresoil | 0.979466 | -0.201608 | 0.2533 | 0.058 .  |
| Humdepth | 0.916024 | 0.401123  | 0.4524 | 0.002 ** |
| pH    | -0.998306 | 0.058180  | 0.2187 | 0.106    |

---

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

P values based on 1000 permutations.

```r
> plot(ef.ca1) # note: this plotting method requires existing plot
> plot(ef.ca1, p.max = 0.05, col = "red") # as above
```
Figure 46. PCA of same data as in Figure 45.

Compare how well environmental variables could be fit into the two different ordinations. Discuss differences; which method works better with this dataset?

Illustrate major gradients (Figure 47):
> x11()
> plot(nm1); plot(ef.nm1)
> ordisurf(nm1, varechem$Al, add=T, col=2) # fit surface (GAM) over ordination

Loading required package: mgcv
This is mgcv 1.3-13
Loading required package: akima

Family: gaussian
Link function: identity

Formula:
y ~ s(x1, x2, k = knots)

Estimated degrees of freedom:
5.087141  total = 6.08714

GCV score: 8798.657
> ordisurf(nml, varechem$Humdepth, add=T, col=3)

Family: gaussian  
Link function: identity  

Formula:  
y ~ s(x1, x2, k = knots)  

Estimated degrees of freedom:  
5.481066  total =  6.481066  

GCV score:  0.2213074  

---

**Figure 47.** NMDS with surface fits for the environmental variables ‘Al’ (red) and ‘Humdepth’ (green).
4.1.2 Example 5: Applications of non-metric multidimensional scaling (NMDS) & canonical correspondence analysis (CCA) to benthos data

Unless we restarted R we don't have to reload vegan.

```R
> rm(list=ls()) # clean workspace
```

Load data from Hilde Trannum: open EXCEL-sheet, highlight the data in sheet 'speci' and copy table to the clipboard (= press 'ctr1'+C). Then return to R and run

```R
> traspe <- read.delim(file="clipboard", row.names=1) # '..., header=T' redundant, is the default
```

Now go again to EXCEL and copy the environmental data in the same way, then run

```R
> traenv <- read.delim(file="clipboard", row.names=1)
```

If this doesn't work, load these files instead

```R
> load("traspe.bin")
> load("traenv.bin")
```

We can save these tables for further use in R (open with `load("file.bin")`)

```R
> save(traspe, file="traspe.bin")
> save(traenv, file="traenv.bin")
```

It is useful to have separate identifier for binary files.

Function `file.bin()` stores data in binary format. This can only be opened within R. Advantage: easier to open: `load("file.bin")`, instead of `read.table("file.txt", header=T, row.names=T)`.

I want to check if the rows match; the function `match(x,y)` finds matching objects among two vectors.

```R
> match(rownames(traspe), rownames(traenv))
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
> match(rownames(traspe), rownames(traenv))/1:20
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The species list is very long. Problem: many taxa occur only at one site. Should we exclude taxa occurring below a minimum number? This depends on the focus of the analysis. If the focus is on the sites, these taxa give still info on species richness, but if focus is on the species and their distribution, we should exclude rare taxa (stochastic occurence). First let's see how the data are distributed.

Visualize number of sites per taxon. The `apply()` function is useful for aggregation of arrays/matrices:

```R
apply(array, margin, function): margin=1 -> by row; margin=2 -> by column
```

An example for usage of `apply()`:

```R
> mat <- traspe[1:3, 1:3]
> mat
     Abra.sp Abys.sp Agla.mal
GOL1-1   0     28     5
GOL1-10  0     41     7
GOL1-11  0     20     9
> apply(mat, 1, sum)
GOL1-1  GOL1-10  GOL1-11
  33      48      29
> apply(mat, 2, sum)
     Abra.sp Abys.sp Agla.mal
  0     89     21
> apply(mat>0, 1, sum) # row-sums = sums per site
GOL1-1  GOL1-10  GOL1-11
  2      2      2
> apply(mat>0, 2, sum) # column-sums = sums per taxon
     Abra.sp Abys.sp Agla.mal
  0     3     3
> x11(6, 6)
```

85
> hist(apply(traspe>0, 2, sum), xlab="Nr. sites per taxon")
This gives too few breaks. We can increase the number of breaks (Figure 48):
> hist(apply(traspe>0, 2, sum), breaks=20, xlab="Nr. sites per taxon")

<table>
<thead>
<tr>
<th>Histogram of apply(traspe &gt; 0, 2, sum)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>20</td>
</tr>
</tbody>
</table>

**Figure 48.** Histogram of no. of sites per taxon.

You can abbreviate command within function: 'bre=.' instead of 'breaks=' (or any non-ambiguous abbreviation).
So let's exclude taxa with less than 5 occurrences:
> sel <- which(apply(traspe>0, 2, sum)>4)
Check if this works (**Figure 49**):
> hist(apply(traspe[sel]>0, 2, sum), breaks=20, xlab="Nr. sites per taxon")

<table>
<thead>
<tr>
<th>Histogram of apply(traspe[sel] &gt; 0, 2, sum)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>20</td>
</tr>
</tbody>
</table>

**Figure 49.** Histogram of no. of sites per taxon.

Should these taxa be excluded? If yes:
# > traspe<-traspe[sel] # below, we continue with the full dataset

Check environmental variable for their distribution
> x11()
> pairs(traenv) # not shown
There are too many at once; focus on metals:
> pairs(traenv[1:11])
Let's see log-transformed data now (Figure 50):

---

86
Figure 50. Command `pairs`: pair wise scatterplots of environmental variables.

Log-transformed values are not better. Log-transformation of THC and Cd would yield negative values. This can avoid by multiplying by 1000.

```r
> traenv[, 1:2] <- log10(1000*traenv[, 1:2])
```

Check correlations among environmental variables. The function `cor(x)` gives a correlation matrix.

```r
> round(cor(traenv),2) # not so clear to read, try:
            THC  Cd  Pb  Al  Ba  Cr  Cu  Fe  Li  Ti  Zn
THC  1.00 -0.33 -0.04 -0.05 0.33 0.34 0.33 0.35 0.04 0.31 0.29
Cd -0.33  1.00  0.21 -0.32 -0.09 -0.05 -0.07 0.00 -0.08 -0.07  0.02
Pb -0.04  0.21  1.00  0.32 -0.13 -0.02 -0.22 -0.01  0.55 -0.16  0.27
Al -0.05 -0.32  0.32  1.00 -0.15  0.19  0.24  0.06  0.69  0.10  0.35
Ba  0.33 -0.09 -0.13 -0.15  1.00  0.89  0.81  0.94  0.31  0.99  0.76
Cr  0.34 -0.05 -0.02  0.19  0.89  1.00  0.91  0.98  0.60  0.93  0.93
Cu  0.33 -0.07 -0.22  0.24  0.81  0.91  1.00  0.88  0.45  0.85  0.81
Fe  0.35  0.00 -0.01  0.06  0.94  0.98  0.88  1.00  0.53  0.96  0.91
Li  0.04  0.08  0.55  0.69  0.31  0.60  0.45  0.53  1.00  0.34  0.80
```
The function `symnum(x)` gives a symbolic representation of numbers:

```r
> symnum(cor(traenv))  # 'symnum' symbolizes the matrix produced by 'cor'
```
```
TH  Cd  Pb  A  B  Cr  Cu  Fe  L  Ti  Z  THC  Depth  Fine.sand  Pelite
THC  .    .    .    .    .    .    .    .    .    .    1
Cd    .    1    .    .    .    .    .    .    .    .    .    .    .    .
Pb    .    .    1    .    .    .    .    .    .    .    .    .    .    .    .
Al    .    .    .    1    .    .    .    .    .    .    .    .    .    .    .
Ba    .    .    .    .    1    .    .    .    .    .    .    .    .    .    .
Cr    .    .    .    .    .    1    .    .    .    .    .    .    .    .    .
Cu    .    .    .    .    .    .    1    .    .    .    .    .    .    .    .
Fe    .    .    .    .    .    .    .    1    .    .    .    .    .    .    .
Li    .    .    .    .    .    .    .    .    1    .    .    .    .    .    .
Ti    .    .    .    .    .    .    .    .    .    1    .    .    .    .    .
Zn    .    .    .    .    .    .    .    .    .    .    1    .    .    .    .
TOM   .    .    .    .    .    .    .    .    .    .    .    1    .    .    .
Depth .    .    .    .    .    .    .    .    .    .    .    .    1    .    .
Fine.sand .    .    .    .    .    .    .    .    .    .    .    .    .    1    .
Pelite .    .    .    .    .    .    .    .    .    .    .    .    .    .    1
```

attr("legend")
```
[1]  0  ' ' 0.3  '.' 0.6  ',' 0.8  '+' 0.9  '*' 0.95  'B' 1
```

Let's see how the distribution of metals is related to the morphology of the sites:

```r
> nm.env <- metaMDS(traenv[,1:11])
```

Wisconsin double standardization
```
Run 0 stress 6.960043
Run 0 stress 6.954604
... New best solution
... rmse 0.001560382 max residual 0.002971707
*** Solution reached
```

```r
> x11()
> plot(nm.env, type="n", main="NMDS of contaminants")  # Gives Figure 51
```
Figure 51. NMDS of contaminants.

```r
> text(nm.env, "species", col=2) # the species
> text(nm.env, "sites", col=1)
> (ef.nm.env<-envfit(nm.env, traenv[12:15], perm=1000))

***VECTORS

|        | NMDS1  | NMDS2  | r2    | Pr(>|r|) |
|--------|--------|--------|-------|----------|
| TOM    | 0.60443| 0.79666| 0.1239| 0.326    |
| Depth  | 0.19037| 0.98171| 0.0857| 0.382    |
| Fine.sand | -0.37688| -0.92626| 0.2647| 0.106    |
| Pelite | 0.41919| 0.90790| 0.2409| 0.135    |

P values based on 1000 permutations.

> # only weak effects

> # NMDS of sqrt-transformed species data
> nm.sp<-metaMDS(sqrt(traspe))
Run 0 stress 13.99375
Run 1 stress 13.97234
... New best solution
... rmse 0.006179893 max residual 0.01612660
Run 2 stress 14.42431
Run 3 stress 13.97234
... New best solution
... rmse 0.0004250777 max residual 0.001250656
*** Solution reached

> x11()
> plot(nm.sp, main="all species") # Figure 52
**Figure 52.** NMDS for all species.

Are the ordinations for contaminants and species similar? We can test by procrustes rotation

```
> (pro<-protest(nm.env, nm.sp))
```

**Call:**

protest(X = nm.env, Y = nm.sp)

**Correlation in a symmetric Procrustes rotation:** 0.6625  
**Significance:** < 0.001  
**Based on 1000 permutations.**

The two ordinations are correlated (Figure 53), which is not a surprise if the contaminants have an effect.

```
> x11()  
> plot(pro)
```

**Figure 53.** Procrustes rotation comparing the NMDS of species and contaminants.
We assume that both site morphology and heavy metals influence species distribution, so we first test all parameters in the ordination.

```r
> (ef.nm.sp<-envfit(nm.sp, traenv, perm=1000))
***VECTORS

<table>
<thead>
<tr>
<th></th>
<th>NMDS1</th>
<th>NMDS2</th>
<th>r2 Pr(&gt;r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>THC</td>
<td>0.943970</td>
<td>0.330030</td>
<td>0.3357</td>
</tr>
<tr>
<td>Cd</td>
<td>-0.894964</td>
<td>0.446138</td>
<td>0.0469</td>
</tr>
<tr>
<td>Pb</td>
<td>-0.347455</td>
<td>0.937697</td>
<td>0.0094</td>
</tr>
<tr>
<td>Al</td>
<td>-0.547694</td>
<td>0.836679</td>
<td>0.0630</td>
</tr>
<tr>
<td>Ba</td>
<td>0.992854</td>
<td>-0.119331</td>
<td>0.7945</td>
</tr>
<tr>
<td>Cr</td>
<td>0.989163</td>
<td>0.146824</td>
<td>0.6686</td>
</tr>
<tr>
<td>Cu</td>
<td>0.999901</td>
<td>0.014096</td>
<td>0.4370</td>
</tr>
<tr>
<td>Fe</td>
<td>0.998195</td>
<td>0.060055</td>
<td>0.7562</td>
</tr>
<tr>
<td>Li</td>
<td>0.443578</td>
<td>0.896236</td>
<td>0.2674</td>
</tr>
<tr>
<td>Ti</td>
<td>0.994916</td>
<td>-0.100706</td>
<td>0.7655</td>
</tr>
<tr>
<td>Zn</td>
<td>0.958421</td>
<td>0.285358</td>
<td>0.5642</td>
</tr>
<tr>
<td>TOM</td>
<td>0.614459</td>
<td>0.788948</td>
<td>0.1607</td>
</tr>
<tr>
<td>Depth</td>
<td>0.169195</td>
<td>0.985583</td>
<td>0.2600</td>
</tr>
<tr>
<td>Fine.sand</td>
<td>-0.344008</td>
<td>-0.938783</td>
<td>0.2034</td>
</tr>
<tr>
<td>Pelite</td>
<td>0.340739</td>
<td>0.940158</td>
<td>0.2668</td>
</tr>
</tbody>
</table>
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
P values based on 1000 permutations.
```

Well, many relationships, but hard to interpret (overfitting! – remember we have only 19 sites!)

A better way with constrained ordination?

```r
> x11(); par(mfrow=c(2, 2))
> plot(nm.sp, main="metaMDS traspe")  # Figure 54, upper left
> plot(ef.nm.sp)
> plot(ef.nm.sp, p.max = 0.05, col = "red")
> (cca1<-cca(sqrt(traspe)~., traenv)) # Figure 54, upper right
```

Call:
`cca(formula = sqrt(traspe) ~ THC + Cd + Pb + Al + Ba + Cr + Cu + Fe + Li + Ti + Zn + TOM + Depth + Fine.sand + Pelite, data = traenv)`

Inertia
Total 1.0304
Constrained 0.8433 15
Unconstrained 0.1871 4

Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
```
CCA1  CCA2  CCA3  CCA4  CCA5  CCA6  CCA7  CCA8  CCA9  CCA10
0.11174 0.08121 0.07868 0.07037 0.06634 0.06042 0.05543 0.05044 0.04818 0.04759
CCA11 CCA12 CCA13 CCA14 CCA15
0.03975 0.03756 0.03396 0.03195 0.02966
```

Eigenvalues for unconstrained axes:
```
CA1  CA2  CA3  CA4
0.05270 0.05043 0.04377 0.04020
```

A high amount of variability is explained; this is expected with high number of explanatory variables. However, does this help?

```r
> ca1 <- cca(sqrt(traspe))
> plot(ca1, main="CA traspe")  # Figure 54, lower left
> (pro <- protest(ca1, cca1))
```

Call:
protest(X = cal, Y = cca1)

Correlation in a symmetric Procrustes rotation:  0.9455
Significance:  < 0.001
Based on 1000 permutations.

> plot(pro)  # Figure 54, lower right

Figure 54. Various ordinations and procrustes errors between overfitted CCA and unconstrained PCA (lower left).

'Unconstrained' CCA (practically identical to CA, see Figure 54) has too many variables, and is hard to interpret. We should therefore build a better (restricted) model.

  > mod0 <- cca(sqrt(traspe) ~ 1, traenv)
  > mod1 <- cca(sqrt(traspe) ~ ., traenv)
  > mod <- step(mod0, scope=list(lower = ~1, upper=formula(mod1)))
Start:  AIC= 101.6
sqrt(traspe) ~ 1

<table>
<thead>
<tr>
<th>Df</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Pelite</td>
<td>1 101.52</td>
</tr>
<tr>
<td>+ Depth</td>
<td>1 101.60</td>
</tr>
<tr>
<td>&lt;none&gt;</td>
<td>101.61</td>
</tr>
<tr>
<td>+ Fine.sand</td>
<td>1 101.64</td>
</tr>
<tr>
<td>+ Li</td>
<td>1 101.64</td>
</tr>
<tr>
<td>+ Zn</td>
<td>1 101.78</td>
</tr>
<tr>
<td>+ TOM</td>
<td>1 101.81</td>
</tr>
<tr>
<td>+ Cr</td>
<td>1 102.00</td>
</tr>
<tr>
<td>+ Cu</td>
<td>1 102.01</td>
</tr>
<tr>
<td>+ Fe</td>
<td>1 102.01</td>
</tr>
<tr>
<td>+ Al</td>
<td>1 102.11</td>
</tr>
<tr>
<td>+ Ti</td>
<td>1 102.13</td>
</tr>
<tr>
<td>+ Ba</td>
<td>1 102.15</td>
</tr>
<tr>
<td>+ Cd</td>
<td>1 102.23</td>
</tr>
<tr>
<td>+ Pb</td>
<td>1 102.32</td>
</tr>
<tr>
<td>+ THC</td>
<td>1 102.48</td>
</tr>
</tbody>
</table>

Step:  AIC= 101.52
sqrt(traspe) ~ Pelite

<table>
<thead>
<tr>
<th>Df</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none&gt;</td>
<td>101.52</td>
</tr>
<tr>
<td>- Pelite</td>
<td>1 101.61</td>
</tr>
<tr>
<td>+ Cu</td>
<td>1 101.91</td>
</tr>
<tr>
<td>+ Fe</td>
<td>1 102.00</td>
</tr>
<tr>
<td>+ Zn</td>
<td>1 102.00</td>
</tr>
<tr>
<td>+ Ti</td>
<td>1 102.01</td>
</tr>
<tr>
<td>+ Ba</td>
<td>1 102.02</td>
</tr>
<tr>
<td>+ Cr</td>
<td>1 102.03</td>
</tr>
<tr>
<td>+ Cd</td>
<td>1 102.05</td>
</tr>
<tr>
<td>+ Pb</td>
<td>1 102.25</td>
</tr>
<tr>
<td>+ Depth</td>
<td>1 102.27</td>
</tr>
<tr>
<td>+ THC</td>
<td>1 102.28</td>
</tr>
<tr>
<td>+ Al</td>
<td>1 102.35</td>
</tr>
<tr>
<td>+ Li</td>
<td>1 102.39</td>
</tr>
<tr>
<td>+ Fine.sand</td>
<td>1 102.41</td>
</tr>
<tr>
<td>+ TOM</td>
<td>1 102.45</td>
</tr>
</tbody>
</table>

> mod
call:
cca(formula = sqrt(traspe) ~ Pelite, data = traenv)

Inertia Rank
Total  1.0304
Constrained  0.1019   1
Unconstrained  0.9285   18
Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
CCA1
0.1019

Eigenvalues for unconstrained axes:
CA1  CA2  CA3  CA4  CA5  CA6  CA7  CA8
0.08413 0.08018 0.07580 0.07199 0.06573 0.05875 0.05716 0.05520
(Showed only 8 of all 18 unconstrained eigenvalues)
Stepwise selection does not work well (all exc. Pelite removed). Pelite seems to be a major, overriding factor. We can ‘partial out’ the effect of Pelite in order to make a potential effect of the heavy metals more visible.

```r
> (cca.Ba <- cca(sqrt(traspe) ~ Ba + Condition(Pelite), traenv))
Inertia Rank
Total         1.03038
Conditional   0.10190    1
Constrained   0.06715    1
Unconstrained 0.86133    17
Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
   CCA1
0.06715

Eigenvalues for unconstrained axes:
   CA1   CA2   CA3   CA4   CA5   CA6   CA7   CA8
0.08046 0.07678 0.07357 0.06791 0.06356 0.05731 0.05522 0.05381
(Showed only 8 of all 17 unconstrained eigenvalues)

> anova(cca.Ba) # permutation test on effect of Ba
Permutation test for cca under direct model
Model: cca(formula = sqrt(traspe) ~ Ba + Condition(Pelite), data = traenv)
   Df Chisq      F N.Perm  Pr(>F)
Model     1 0.0672 1.3254   1200 0.0675 .
Residual 17 0.8613
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of ‘Ba’ after removing the effect of ‘Pelte’ is marginally significant. We can perform this test with all potential candidates

```r
> (cca.Zn <- cca(sqrt(traspe) ~ Zn + Condition(Pelite), traenv)) # same for Zn
Inertia Rank
Total         1.03038
Conditional   0.10190    1
Constrained   0.06792    1
Unconstrained 0.86056    17
Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
   CCA1
0.06792

Eigenvalues for unconstrained axes:
   CA1   CA2   CA3   CA4   CA5   CA6   CA7   CA8
0.08193 0.07850 0.07254 0.06574 0.06157 0.05723 0.05582 0.05399
(Showed only 8 of all 17 unconstrained eigenvalues)

> anova(cca.Zn)
Permutation test for cca under direct model
Model: cca(formula = sqrt(traspe) ~ Zn + Condition(Pelite), data = traenv)
   Df Chisq      F N.Perm  Pr(>F)
Model     1 0.0679 1.3417   2300 0.06391 .
Residual 17 0.8606
---
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Is species richness related to the environmental variables? We need a non-parametric test as data are clearly non-normally distributed.

```r
> require(Hmisc)
[1] TRUE
> nspe <- as.numeric(apply(traspe > 0, 1, sum))
> x11()

> par(mfrow=c(2, 2)) # Figure 55
> plot(nspe ~ Depth, data=traenv)
> plot(nspe ~ Pelite, data=traenv)
> plot(nspe ~ Ba, data=traenv)
> plot(nspe ~ Zn, data=traenv)
```

**Figure 55.** Number of species per site plotted against different environmental factors.

```r
> spearman.test(nspe, traenv$Zn) # and same way for other variables
Rsquare           F         df1         df2      pvalue           n
0.23140882  5.41947241  1.00000000 18.00000000  0.03177332 20.00000000
Alternatively:
> cor.test(nspe, traenv$Zn, method="spearman")
```

For more than 1 parameter, we can use 2-dimensional GAM (Section 5).

```r
> #
> # L U N C H
> # B R E A K >>{(} (*>
> #
```
4.1.3 Example 6: constrained ordination. Extracting indicator values from an ordination.

```r
> rm(list=ls())
> load("NOgen.bin") # phytoplankton composition data from Norwegian lakes (genus level)
> load("NOenv.bin") # corresponding chemistry and lake morphometry
```

NMDS cannot be used with such large dataset (R will get stuck). The genus data are very skewed, with many zeros. We will therefore apply a square-root-transformation when performing ordinations (`sqrt(data)`, see below).

Some species may be very rare or absent.

```r
> sel.gen <- which(apply(NOgen>0, 2, sum)>10)
> NOgen <- NOgen[sel.gen]
```

Environmental variables - should we apply log-transformation?

```r
> pairs(NOenv[3:9])
> pairs(log10(NOenv[3:9]))
> lNOenv <- log10(NOenv)
> symnum(cor(lNOenv))
```

```
sy sm M A S SC C TN TP
syear         1
smonth           1
Mean_depth          1
Altitude              1
Surface_area        .   1
SCa                 .      1
Chlorophyll.a       .      .  1
TotN                       .  . 1
TotP                .      .  , ,  1
attr(,"legend")
[1] 0 ' ' 0.3 '.' 0.6 ',' 0.8 '+' 0.9 '*' 0.95 'B' 1
```

It doesn't look too bad (Figure 56).

```r
> (ca1 <- cca(sqrt(NOgen)))
```

```
Call:
cca(X = sqrt(NOgen))

Inertia Rank
Total           5.405
Unconstrained   5.405   82
Inertia is mean squared contingency coefficient

Eigenvalues for unconstrained axes:
  CA1    CA2    CA3    CA4    CA5    CA6    CA7    CA8
0.4318  0.3338  0.2828  0.2740  0.2440  0.2327  0.2060  0.2040
(Showed only 8 of all 82 unconstrained eigenvalues)
```

```r
> x11()
> plot(cal) # Figure 57.
```
Figure 56. Pair-wise scatterplots of environmental variables.
Figure 57. PCA of the phytoplankton matrix.

> (ef.ca1 <- envfit(ca1, lNOenv, perm=1000))

***VECTORS

<table>
<thead>
<tr>
<th></th>
<th>CA1</th>
<th>CA2</th>
<th>r2</th>
<th>Pr(&gt;r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>syear</td>
<td>0.584072</td>
<td>-0.811702</td>
<td>0.1367</td>
<td>0.003 **</td>
</tr>
<tr>
<td>smonth</td>
<td>0.765518</td>
<td>-0.643414</td>
<td>0.0007</td>
<td>0.948</td>
</tr>
<tr>
<td>Mean_depth</td>
<td>-0.831706</td>
<td>0.555216</td>
<td>0.2036 &lt;0.001 ***</td>
<td></td>
</tr>
<tr>
<td>Altitude</td>
<td>-0.999551</td>
<td>0.029968</td>
<td>0.1096 &lt;0.001 ***</td>
<td></td>
</tr>
<tr>
<td>Surface_area</td>
<td>-0.888171</td>
<td>0.459514</td>
<td>0.0184 0.140</td>
<td></td>
</tr>
<tr>
<td>SCa</td>
<td>0.899901</td>
<td>-0.436093</td>
<td>0.3687 &lt;0.001 ***</td>
<td></td>
</tr>
<tr>
<td>Chlorophyll.a</td>
<td>0.887499</td>
<td>-0.460809</td>
<td>0.6504 &lt;0.001 ***</td>
<td></td>
</tr>
<tr>
<td>TotN</td>
<td>0.882269</td>
<td>-0.470746</td>
<td>0.3658 &lt;0.001 ***</td>
<td></td>
</tr>
<tr>
<td>TotP</td>
<td>0.923900</td>
<td>-0.382634</td>
<td>0.5109 &lt;0.001 ***</td>
<td></td>
</tr>
</tbody>
</table>

---

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

P values based on 1000 permutations.

We may suspect that the main gradient is eutrophication. Thus it would be nice to illustrate role of Chlorophyll (i.e. eutrophication).

The function `cut(x, nbreaks,...)` breaks vector into pieces of even intervals.

> chl.code <- cut(lNOenv$Chlorophyll.a, 30, labels=F) # (see part 6. for details)
> x11(); par(bg="darkgrey")
> plot(scores(cal)$sites, pch=20, col=heat.colors(30)[chl.code])
Figure 58. PCA of the phytoplankton matrix. The colorcode illustrates the Chlorophyll-a concentration (dark=low, light=high).

Figure 58 illustrates that communities become more different as Chl-a increases. From previous analyses we know that in addition to eutrophication, lake depth and alkalinity (Ca, here SCa) are important variables.

```r
> (cca1<-cca(sqrt(NOgen)~Chlorophyll.a+Mean_depth+SCa, data=lNOenv))

Call: cca(formula = sqrt(NOgen) ~ Chlorophyll.a + Mean_depth + SCa, data = lNOenv)

Inertia Rank
Total           5.405
Constrained     0.401    3
Unconstrained   5.004   82
Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
  CCA1  CCA2  CCA3
0.31334 0.06839 0.01925

Eigenvalues for unconstrained axes:
  CA1  CA2  CA3  CA4  CA5  CA6  CA7  CA8
0.3551 0.2723 0.2665 0.2452 0.2335 0.2111 0.2007 0.1780
(Showed only 8 of all 82 unconstrained eigenvalues)

Note the difference between inertia of first and second constrained axes.
```

> x11()
> par(bg="grey")
> plot(cca1)
> points(cca1, pch=20, col=heat.colors(30)[chl.code])

The first axis is mainly eutrophication (Figure 59).
Figure 59. CCA of the phytoplankton data with three predictors.

For extracting indicator values, we need to extract the optima of the genera on the eutrophication gradient. We could directly use the species scores from the first axis, but there is obviously correlation with the other variables. Let's partial out SCa first.

```r
> (cca2<-cca(sqrt(NOgen)~Chlorophyll.a+Condition(Mean_depth+SCa), data=lNOenv))
```

Call:
`cca(formula = sqrt(NOgen) ~ Chlorophyll.a + Condition(Mean_depth + SCa), data = lNOenv)`

Inertia Rank

<table>
<thead>
<tr>
<th>Total</th>
<th>Conditional</th>
<th>Constrained</th>
<th>Unconstrained</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4054</td>
<td>0.2346</td>
<td>0.1664</td>
<td>5.0044</td>
</tr>
</tbody>
</table>

Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:

- CCA1: 0.1664

Eigenvalues for unconstrained axes:

- CA1: 0.3551, CA2: 0.2723, CA3: 0.2665, CA4: 0.2452, CA5: 0.2335, CA6: 0.2111, CA7: 0.2007, CA8: 0.1780

(Showed only 8 of all 82 unconstrained eigenvalues)

Now the first axis explains much less than before.

> x11()
> plot(cca2) # Figure 60
> points(cca2, pch=20, col=heat.colors(30)[chl.code])
Re-infering the lake trophic status from the optima of the species: we can use the optima derived from the CCA for predicting lake trophic status. This is a way to test if the species are useful indicators. We want to avoid circularity and will therefore split the data into training and predicting dataset. We divide into two random groups using the `sample()` function.

```r
> set1 <- sort(sample(1:501, 250, replace=F))
> set2 <- (1:501)[-set1]
> (cca.set1 <- cca(sqrt(NOgen[set1, ]) ~ Chlorophyll.a + Condition(Mean_depth+SCa),
+ data=lNOenv[set1, ]))
```

Call:  
cca(formula = sqrt(NOgen[set1, ]) ~ Chlorophyll.a + Condition(Mean_depth + SCa), data = lNOenv[set1, ])

Inertia Rank
Total  4.9851
Conditional  0.2683  2
Constrained  0.1945  1
Unconstrained  4.5223  82
Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
CCA1
0.1945

Eigenvalues for unconstrained axes:
  CA1  CA2  CA3  CA4  CA5  CA6  CA7  CA8
0.3373 0.3126 0.2525 0.2300 0.2209 0.2023 0.1945 0.1807
(Showed only 8 of all 82 unconstrained eigenvalues)

The output is comparable to the output of total data set (above). Species optima for Chl-a correspond to the first ordination axis.

```r
> specopt <- scores(cca.set1)$species[, 1]
```
Plot species-optima with function `dotchart()`

```r
> par(mfrow=c(1, 2))
> dotchart(sort(specopt)[1:41], xlim=range(specopt)) (Figure 61).
> dotchart(sort(specopt)[42:83], xlim=range(specopt))
```

Both plots should have same x-scale, though showing different optima. The function `range()` extracts min and max of the optima, and can be used to set the scale for the x-axis.

![Figure 61. Dotchart illustrating the species optima.](image)

Using weighted averaging, we calculate a trophic score as predicted from the species. The function `wascores(x, y)` calculates weighted averages for sites from species optima(x) and the transposed composition matrix (\(t(y)\)). See Figure 62.

```r
> wascr <- wascores(specopt, t(sqrt(NOgen[set2,])))
> x11(6, 6)
> plot(lNOenv$Chlorophyll.a[set2], wascr, xlab="log(Chlrophyll-a)",
+ ylab="trophic score")
> plot(NOenv$Chlorophyll.a[set2], wascr, log="x", xlab="Chlrophyll-a",
+ ylab="trophic score", main="all species")
```
Figure 62. The calculated trophic scores plotted against the chlorophyll-a concentration.

```r
> require(mgcv)
[1] TRUE
> gmod<-gam(wascr~s(LNOenv$Chlorophyll.a[set2]))
> gpred<-predict(gmod, se.fit=T)
> lines(NOenv$Chlorophyll.a[set2], gpred$fit, col=3)
> lines(NOenv$Chlorophyll.a[set2], gpred$fit+gpred$se.fit, col=3, lty=2)
> lines(NOenv$Chlorophyll.a[set2], gpred$fit-gpred$se.fit, col=3, lty=2)
```

Now we used all genera, but we can also select those with best fit.

```r
> x11(6, 4)
The function `goodness()` extracts value for how well a site or species fits in the ordination (Figure 63). The larger the better, low goodness implies that species are randomly distributed.
```r
> hist(goodness(cca.set1, "species"), bre=30)
> abline(v=0.05, col=2)
```

The red line in the histogram indicates the cut-off level for weak fit.

Figure 63. Histogram of the species’ ‘goodness’ in the CCA.
> sel.ind <- which(goodness(cca2, "species")>0.05)
> wascr <- wascores(specopt[sel.ind], t(sqrt(NOgen[set2, sel.ind])))
> x11(6, 6) # Figure 64
> plot(NOenv$Chlorophyll.a[set2], wascr, log="x", xlab="Chlorophyll-a",
+ ylab="trophic score", main="selected species")

Figure 64. As Figure above, but for selected species only.

> gfit<-gam(wascr~s(lNOenv$Chlorophyll.a[set2]))
> gpred<-predict(gfit, se.fit=T)
> lines(NOenv$Chlorophyll.a[set2], gpred$fit, col=3)
> lines(NOenv$Chlorophyll.a[set2], gpred$fit+gpred$se.fit, col=3, lty=2)
> lines(NOenv$Chlorophyll.a[set2], gpred$fit-gpred$se.fit, col=3, lty=2)

> # - optional for ### P A R T T H R E E ###
Is there another way to obtain species optima? Can we use median of all the sites where a species
occurs? How do we get the Chl-values of the sites where species 1 occurs?
> NOgen[, 1]>0
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
(…)
This function returns TRUE for all sites, so that we can get the corresponding Chl-values:
> lNOenv$Chlorophyll.a[NOgen[, 1]>0]
  [1]  0.09691001  0.04575749  0.00000000  0.11394335  0.11394335  0.20411998
  [7]  0.23044892  0.23044892  0.23044892  0.25527251  0.30103000  0.30103000
[13]  0.38021124  0.43136376  0.44715803  0.58433122  0.59106461  0.65321251
(…)
and the median
> median(lNOenv$Chlorophyll.a[NOgen[, 1]>0])
[1] 0.2428607

OK, now let's produce a vector with all genus optima
> meds <- NA
> for(i in 1:length(NOgen)) {
+ meds[i] <- median(lNOenv$Chlorophyll.a[NOgen[, i]>0])
+ } wascr <- wascores(meds, t(sqrt(NOgen)))
> x11()
> plot(NOenv$Chlorophyll.a, wascr, log="x", main="median optima - all species")
> gmod <- gam(wascr ~ s(lNOenv$Chlorophyll.a))
> gpred <- predict(gmod, se.fit=T)
> lines(NOenv$Chlorophyll.a, gpred$fit, col=3)
> lines(NOenv$Chlorophyll.a, gpred$fit+gpred$se.fit, col=3, lty=2)
> lines(NOenv$Chlorophyll.a, gpred$fit-gpred$se.fit, col=3, lty=2)
Figure 26. Predicted trophic scores from the species’ medians using weighted averaging. The figure is very similar compared to the figure produced from the CCA scores (Fig. 26). The reason is that also the CCA uses weighted averaging for finding species scores, and because we had only one constraining variable (chlorophyll-a) in the CCA.

> # FINI :-}
5. DATA VISUALIZATION AND LATTICE

(Robert Ptacnik)

TOPIC: create color-gradients; 3-dim GAM; Lattice

5.1 Colour gradients and 3-D GAM plots

> rm(list=ls())

Defining colorcodes

> load("Nstati7-9.bin") # North-European lake data, samples jul-sept (REBECCA DB)
> attach(Nstati)

The following object(s) are masked from Nstati (position 3):

- Alkalinity
- Alkalinity_type
- Altitude
- Altitude_type
- Ca
- Chlorophyll.a
- Colour
- country_code
- entry_code
- entry_no
- GIG
- GIG_type
- Humic_type
- lake_code
- Mean_depth
- Mean_depth_Type
- Nsamp
- pH
- Reference_lake
- Scока
- SColour
- sday
- site_code
- site_nr
- smonth
- Surface_area
- Surface_area_type
- syear
- Temperature
- tot.biov
- TotN
- TotP
- Transparency

> chl.code<-cut(log(Chlorophyll.a), 30, labels=F)
> x11(8,4)
> par(mfrow=c(1, 2)) # 'par(mfrow=c(X,Y))': split plotting window by X x Y rows x columns
> plot(Chlorophyll.a, chl.code, log="x")

See Figure 65, left panel.

> plot(Chlorophyll.a, chl.code, log="x", col=cm.colors(30)[chl.code], pch=19)

See Figure 65, right panel.

![Figure 65](image-url)

**Figure 65.** Example for how to code a color gradient. The y-values range from 0-30 and code the corresponding cyan-magenta colors: `cm.colors(30)`.
Colors are hard to see; change background (Figure 66).

```r
> x11(12,4)
> par(bg="dark grey")
> par(mfrow=c(1, 3))
> plot(Chlorophyll.a, chl.code, log="x", col=cm.colors(30)[chl.code], pch=19,
> main="cm.colors")
> plot(Chlorophyll.a, chl.code, log="x", col=heat.colors(30)[chl.code], pch=19,
> main="heat.colors")
> plot(Chlorophyll.a, chl.code, log="x", col=rainbow(30)[chl.code], pch=19,
> main="rainbow")
```

**Figure 66.** Different default color gradients on grey background. (cm.colors(), heat.colors(), rainbow()).

function 'identify':

\[ 'identify(x, y, labels=...)’ \]

identify dots in scatterplot (works also e.g. within vegan-objects)

```r
> x11(5,5)
> plot(TotP, Mean_depth, log="xy", col=heat.colors(30)[chl.code], pch=19)
> identify(TotP, Mean_depth)
[1] 498 798
> identify(TotP, Mean_depth, labels=rownames(Nstati)) #specify labels
[1] 708 997
```

Right-mouse click 'stop' to interrupt
Figure 67. Mean depth plotted against total phosphorus, and showing chlorophyll-a as color gradient. The labels were produced using the ‘identify’ function.

Interpolate and visualize areas

```r
> require(mgcv)
[1] TRUE
> require(akima) # interpolation applications
[1] TRUE

Nstati contains missing values; for GAM analysis need datamatrix without missing values

```r
> mat <- data.frame(TP=TotP, MD=Mean_depth, Chl.a=Chlorophyll.a)
> detach(Nstati)
> wh <- which(apply(mat, 1, sum)>0) #which rowsums are positive (i.e. have no NA)
> mat <- mat[wh, ]
> gmod <- gam(log(Chl.a)~s(log(TP), k=4)+s(log(MD),k=4), data=mat)
> summary(gmod)

Family: gaussian
Link function: identity

Formula: 
log(Chl.a) ~ s(log(TP), k = 4) + s(log(MD), k = 4)

Parametric coefficients:

| Estimate | Std. Error | t value | Pr(>|t|) |
|----------|------------|---------|----------|
| (Intercept) | 1.28410 | 0.01689 | 76.02 | <2e-16 *** |

---

Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:

<table>
<thead>
<tr>
<th></th>
<th>edf</th>
<th>Est.rank</th>
<th>F</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s(log(TP))</td>
<td>2.928</td>
<td>3.000</td>
<td>460.651</td>
<td>&lt; 2e-16 ***</td>
</tr>
<tr>
<td>s(log(MD))</td>
<td>2.149</td>
<td>3.000</td>
<td>6.812</td>
<td>0.000152 ***</td>
</tr>
</tbody>
</table>

---

Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) = 0.646  Deviance explained = 64.8%
GCV score = 0.2742  Scale est. = 0.27246  n = 955

> gpred <- predict(gmod)

interp(x, y, z) interpolates between three vectors of equal length such that they can e.g. be plotted by surface plot contour() (Figure 68)

```r
> ip <- interp(log(mat$TP), log(mat$MD), gpred, duplicate="mean")
> x11(5,5)
> contour(ip, xlab="TP", ylab="Mean Depth", nlevels=15)
```
Figure 68. 3-dimentional interpolation of the data shown in Figure 67, showing the third variable (here chl-a) as contour lines (function contour()).

```r
> x11(5,5)
> filled.contour(ip, xlab="TP", ylab="Mean Depth",
+ plot.axes={axis(1);axis(2);points(log(mat$TP), log(mat$MD), col="grey")}) ## [not shown]
Filled contour doesn't accept the log() function; we have to define desired axis-ticks manually
> xtick<-log(c(0.5, 1:10, seq(10, 100, 10)))
> ytick<-log(c(3:10, seq(20, 100, 20), 150, 200))
> x11(5,5)
> filled.contour(ip, xlab="TP", ylab="Mean Depth", color.palette=rainbow,
+ plot.axes={axis(1, at=xtick, labels=exp(xtick));
+ axis(2, at=ytick, labels=exp(ytick));points(log(mat$TP), log(mat$MD),
+ col="grey")})
```

Figure 69. Example for the filled.contour() function.

Wonderful colors (Figure 69)... Overlay of datapoints:

```r
> require(MASS)
 [1] TRUE
> x11(5,5)
> plot(Chlorophyll.a~TotP, log="xy", data=Nstati)
```
Figure 70. A scatterplot can be difficult to read when too many dots are plotted together.

Overlay of datapoints - high density blurs trends in data (Figure 70) -> 2-dim density estimation (Figure 71).

```r
ip <- kde2d(log(Nstati$TotP), log(Nstati$Chlorophyll.a))
x11(5,5)
filled.contour(ip)  # [not shown]
xtick <- log(c(0.5, 1:10, seq(10, 100, 10)))
ytick <- log(c(0.5, 1:10, seq(20, 100, 20), 150, 200))
filled.contour(ip, plot.axes={axis(1, at=xtick, labels=exp(xtick));
+ axis(2, at=ytick, labels=exp(ytick))})
```

Figure 71. 2-dim density estimation of the data shown in Figure 70.

Compare Figure 71 with scatterplot (Figure 69) – density plot shows better where data centers.
5.2 Lattice

What mean 'lattice' and 'trellis' in R...

Trellis Graphics is a framework for data visualization developed at the Bell Labs by Rick Becker, Bill Cleveland et al, extending ideas presented in Bill Cleveland's 1993 book _Visualizing Data_.

Lattice is best thought of as an implementation of Trellis Graphics for R.

(from the R help menu - '?lattice')

.. and elsewhere?

"Lattice is suitable for trellis or as decorative sections in fences, handrails, arbors, pergolas, garden or deck privacy screens, windbreaks etc. ..."

So it is some kind of geometric arrangement of data? read further...

> require(lattice)
[1] TRUE

Simple density estimation (Figure 72):
> x11(5,5)
> plot(density(log10(Nstati$Chlorophyll.a)))

density.default(x = log10(Nstati$Chlorophyll.a])

Figure 72. A density plot, indicating the number of observations along a gradient
In lattice, we can split by any factor, e.g. country (Figure 73):

```r
> x11(5,5)
> densityplot(log10(Nstati$Chlorophyll.a), groups=Nstati$country_code, auto.key=T)
```

![Density plot with country groups](image1)

**Figure 73.** Implementation of the density function in the lattice package. The data are split by countries.

```r
> x11(5,5)
> boxplot(log10(Chlorophyll.a)~country_code, data=Nstati) # standard boxplot, not shown
> x11(5,5)
> bwplot(log10(Chlorophyll.a)~country_code, data=Nstati) # lattice version...
> x11(5,5)
> bwplot(log10(Chlorophyll.a)~country_code|GIG_type, data=Nstati) # ...allows split by factors (Figure 74).
```

![Boxplot with country and GIG_type groups](image2)

**Figure 74.** A boxplot produced with the lattice command `bwplot`. 
Ordinary scatterplot (Figure 75):
> x11(5,5)
> plot(Chlorophyll.a~TotP, log="xy", data=Nstati, pch=as.numeric(country_code))

![Ordinary scatterplot](image)

**Figure 75.** Standard scatterplot

Countries are not really separated; try lattice scatterplot `xyplot()` (Figure 76):
> x11(5,5)
> xyplot(log10(Chlorophyll.a)~log(TotP), groups=country_code, data=Nstati, auto.key=T)

![Lattice scatterplot](image)

**Figure 76.** `xyplot` produces scatterplots (and others, see `?xyplot`). Countries are grouped by a color-code.

Split by country (Figure 77):
> x11(5,5)
> xyplot(log10(Chlorophyll.a)~log(TotP)|country_code, data=Nstati)
Figure 77. *xyplot*. Data split by countries into sub-panels.

Split by lake types (Figure 78):

```r
> xyplot(log10(Chlorophyll.a)~log(TotP)|GIG_type, groups=country_code, data=Nstati, auto.key=T)
```

Figure 78. *xyplot*. Combining split-and group function.

Illustrate y against two or more x variables (makes sense only if they have same unit; Figure 79):

```r
> xyplot(log10(Chlorophyll.a)~log10(TotN)+log10(TotP)|country_code, data=Nstati, auto.key=T)
```
Visualize aggregated information of the Nstati table (Figure 80). Make an aggregated table first. 'x' in aggregate(x, list(y), func) will be aggregated by function func().

```r
> agg.Nstati<-aggregate(Nstati$TotP>0, list(country=Nstati$country_code, +  GIG_type=Nstati$GIG_type), sum)
> x11(5,5)
> barchart(x~GIG_type|country, data=agg.Nstati, ylab="nr. of lakes")
```

See Figure 81.
Figure 81. The dotplot function.

> #################
> # T H E   E N D #
> #################
NIVA: Norway’s leading centre of competence in aquatic environments

NIVA provides government, business and the public with a basis for preferred water management through its contracted research, reports and development work. A characteristic of NIVA is its broad scope of professional disciplines and extensive contact network in Norway and abroad. Our solid professionalism, interdisciplinary working methods and holistic approach are key elements that make us an excellent advisor for government and society.