R for palaeolimnology

– a manual

Sven Adler[ab] and Thomas Hübener[b]

R is the computer package of choice for many statistical research solutions. The advantages are obviously. R is free available on the internet, it has a public GUI license, all codes are possible to view and check and the number of available functions is incredible high. The disadvantage is obviously too, at the latest if the program was down loaded and opened – R is a console based programming language, all functions has to be typed in. Nevertheless all common used statistical methods used in palaeoecology are available in R, especially due to the work of Jari Oksanen (package vegan), Gavin Simpson (package analogue) and Steve Juggins (package rioja). Several books about R exist, more general and very specific, but also many manuals (e.g., community analysis by Jari Oksanen) can be found in the internet. To our knowledge no basic manual for paleolimnologist exists up to now (but see also the vignettes for the package analogue), so we offer this one online under http://www.biologie.uni-rostock.de/abt/botanik/AG-Phykologie/index-engl.htm.

[a] Swedish University of Agricultural Sciences, 901 83 Umeå, Sweden

[b] Institut of Biosciences, University of Rostock, 18057 Rostock, Germany

1. Introduction
2. The training set
    2.1 descriptive statistics
    2.2 species environmental relation
    2.3 multivariate analysis
3. The core data
    3.1 descriptive statistics
    3.2 stratigraphic zones
    3.3 multivariate analysis
4. Transfer functions
    4.1 WA, WA-PLS, ML, MAT, LWWA (MW)
    4.2 test for significant species response
    4.3 test for significant reconstruction

1. Introduction

As R (R Development Core Team 2012) is, as mentioned, on the first step not easy to use, we develop for our PhD and Master students this manual to use it as a guide and self-learning introduction to analyse their training sets and core data and to run CA, CCA, WA, WA-PLS, and so on with R. Here we describe the way from the data exploration of a given training set and given core data until the reconstruction of a single environmental variable, based on those. The methods themselves like GLM, CCA, WA and WA-PLS are not explained in detailed, as other authors have done that in a very good way, we will mentioned the related literature at the places these methods are used. Of course the manuals of the used packages give valuable information and should be read in a second step / parallel too, to get access to the full functionality of them.

The way we analysed the data here is only a first step to get in, just to produce basic results using a powerful but not easy accessible computer platform. Of course, this is not the only and best way to analyze paleolimnological data, suggestions to improve this manual are welcome (sven.adler@slu.se, thomas.huebener@uni.rostock.de). Please have a look at the mentioned tutorials and links in this document, to get full access to the used functions and packages in this manual and to find different ways of data analysis using R. Especially in chapter 2.2 we just rabble at the top of the iceberg. For getting in the theoretical background of analysing paleolimnological data Birks 1998, Birks and Birks 2006, and most recent Birks et al 2012 should be used in a first step.

How to install R and related packages on the local computer is described in **Appendix A**, additionally some first example codes for data handling are given. How to read data in R is explained in **Appendix B**, examples for loops are given in **Appendix C**.

Is R just installed together with the mentioned packages, all following codes runs under the current version just type the codes in or use 'copy-paste'.

Following Birks et al. 1998, the use of fossil (rests of) organisms (e.g., diatoms, foraminifera, pollen) to reconstruct environmental variables like total phosphorous (TP), pH, or temperature is based on three major components:

- the **training set** enfolded information about the recent distribution of the fossil organisms that the researcher focused on, including a matrix with species (e.g. diatoms) abundance data from different locations (normally upper sediment layer of lakes) and a matrix / or vector enfolding information about the environmental conditions within the sampled locations (normally mean values of at least 3-4 investigations per year).
- The **core data** enfolded abundance / count information about the organisms of a sediment core. These species compositions will be used to reconstruct the development of environmental conditions.
- The **mathematical tools** that are used to combine the information of the training set with the species combination within the core.

Each of these three parts required good knowledge of the scientists about the techniques he/she used, the species, the environmental conditions and mathematical ideas. In the past the scientists had to use different programs to analyze the data, e.g. EXCEL, CANOCO, WA-CALIBRATE, and C2. Now every step of the data analysis can be done in R.

In general it is important for every analysis to be "in" the data, which means to know the data well. Often students want to create as fast as possible p-values (hopefully smaller than 0.05, ignoring that the other case is a good result too) and large r^2 without intensive knowledge of their data, which lead not seldom to false analysis. So here we have set the focus more on data exploration than on hunting for p and r^2 -values.

2. The training set

2.1 Descriptive Statistics

The species data must be organized in the same way as the input data for C2 or CANOCO (see also Appendix B, Figure B.1): Each row is one sample, each column equals on taxa / environmental variable. Normally, the samples and taxa have a code (without any blanks), witch makes a graphical presentation easier, but it is also possible holding the original names, depending on the method you read the data in R (see **Appendix B**, e.g. reading the data from a txt or csv file). Within the packages vegan, analogue, rioja and paltran examples of training sets are given. Here we will use at first the data of the package analogue.

In the following text all codes to type in the R console are formatted *red*, all outputs on the Console are formatted blue. Everything that you write behind a '#' will ignored by the program and is meant to be supportive information for the user.

After you have installed R and the five packages analogue, vegan, rioja, palaeoSig and paltran (Apendix A) open R and type

```
library(analogue)          # this will load the package analogue (Simpson and Oksanen 2011 )
                           # How to cite a package?
                           # type citation(package="analogue") after loading the package
                           # to cite R, type just citation()
```

behind the prompt (the ">" sign at the console) and confirm the entry with the return key at your keyboard. After you have confirmed, nothing will be printed on the console only a new prompt appears. This is a little bit confusing, but that means just, that you have made no error and everything is o.k.! If

```
Error in library(analogue) : there is no package called 'analogue'
```

appears on the console, than the package analogue is not installed on your computer, see Apendix A how to do.

```
data(swapdiat)             # this will make the example data frame accessible for you, how  to
                             read in your own data see Appendix B
?swapdiat                  # opens the help-html side of the data frame (works not with your own
                             data!)
fix(swapdiat)              # opens the R data-editor to look at the data, you can check if everything
                             looks like it should, you have to close it before you can go on working
                             in the console. Notice: decimal numbers are separated with a "."
names(swapdiat)            # will print on the console the column names (taxa code) of the data
                             frame
```

```
[1] "AC001A" "AC002A" "AC004A" "AC013A" "AC014A" "AC014B" "AC014C" "AC017A"
[9] "AC018A" "AC019A" "AC022A" "AC025A" "AC028A" "AC029A" "AC030A" "AC034A"
[17] "AC035A" "AC039A" "AC042A" "AC044A" "AC046A" "AC048A" "AC9964" "AC9965"
…
```

The number in the brackets tells you, that "AC001A" is the name of the first column, "AC017A" the name of the 8th column and "AC042A" the name of the 19th column. The " " means, that the names are interpreted by the program as strings (not as a factor or number, e.g. Delgaard 2011). The function

```
row.names(swapdiat)
```

will display you the sample names.

If you are interested in the size of the data frame typ

```
dim(swapdiat)
[1] 167 277
```

This tells you that the data frame swapdiat enfolds 167 rows (samples) and 277 columns (species). Type *dim(swapdiat)[1]* to get only the number of rows, *dim(swapdiat)[2]* to get only the number of columns (see also Appendix A).

Within different samples not all 277 species will occur. So it is possible to ask, how many species occurs within each sample? The package vegan enfold a function that will count this for you (http://cc.oulu.fi/~jarioksa/softhelp/vegan.html)

```
library(vegan)                    # this will load the package vegan (Oksanen et al. 2012)
specnumber(swapdiat)
1.21   10.21    11  113.21  115.11   12.11    121   15.11 …
 49     42      38    40      46      36       29     46  …
```

In sample “1.21” 49 species occurred and in sample “10.21” 42 species had been found. How many species occurred on average within every sample?

```
mean(specnumber(swapdiat))  # calculates the mean number of species per sample
sd(specnumber(swapdiat))    # calculates the standard deviation of the  species number
min(specnumber(swapdiat))   # calculates the minimum number of species
max(specnumber(swapdiat))   # calculates the maximum number of species
```

If you want to know, in how many samples the single taxa occurred, than you can use the function specnumber, too, but with the modification, that you use the transposed (exchange of column and row denomination) species matrix:

```
specnumber(t(swapdiat))
AC001A AC002A AC004A AC013A AC014A AC014B AC014C AC017A AC018A AC019A AC022A
   11      26      28     118      7      29      88      10      9      17     118
```

So AC001A occurred in 11 samples and AC013A in 118 samples. But which they are?

First we have to reduce the data frame to only those samples, where AC001A has a relative abundance larger than 0. This can be done by the [] arguments (see also **Appendix A**).

```
 dat<-swapdiat[swapdiat$"AC001A">0,]

fix(dat)                    # opens the R data-editor to look at the data, you can check if everything
                             looks like it should, you have to close it before you can go on working in
                             the console. Alternative head(dat) will show you the first 10 rows of the
                             data frame
```

and from this subset we want to know the row.names:

```
row.names(swapdiat[swapdiat$"AC001A">0,])
```

```
[1] "ARR1"  "ARTH1" "CLON1" "DOI1"  "DOON1" "LLDU1" "MUCK1" "S21"   "SKAK1"
[10] "UIS1"  "WOOD2"
```

Other functions of data exploration and analysis within the package vegan are very usefull, interesting and worth full to play with (e.g. specaccum(),beals()) and can be found under http://cran.r-project.org/web/packages/vegan//vegan.pdf.

If you work with relative abundances, the sum of all values within one row should be 100 (respectively 1, depends on the scale you use). To check this type

```
rowSums(swapdiat)
```

The sum of each row must not be exact 100 cause of rounding errors, or leaving out some rare taxa (like here), but values less than 98 or more than 102 should make you nervous if you have not manipulate the data before. If you have missing values within your data frame (coded in R as NA = “not available”), the output of the function rowSums() will be NA, that means nothing was done. You can say the function, that the NA values should be ignored typing

```
rowSums(swapdiat,na.rm=T)
```

na.rm means remove the NA’s and T stand for TRUE. (This works also with other functions, like *mean()*, *sum()*, *sd()*,…).

In general missing values should not appear in your community matrix, as you should know in principle, whether a species was present or not. Missing Values in your environmental data will be cause trouble too, so try to fill the gabs (see for example the powerful R package mice, Buuren and Groothuis-Oudshoorn 2011) or you have to remove the whole sample.

In several publications rare taxa are eliminated. The definition of ‘rare’ differed from author to author but values between a relative abundance under 1% – 3 % might be most common. Let’s have a look at the first sample within the swap training sets:

```
swapdiat[1,]
```

We know from earlier that here 49 Taxa occurred, we can type

```
swapdiat[1,] [swapdiat[1,]>0]
```

to delete all “0” taxa and use the function length() to count the number of the taxa:

```
length(swapdiat[1,] [swapdiat[1,]>0])
```

```
[1] 49
```

We get the same result as with the function *specnumber()*. So now it is easy to find out how many species have a relative abundance larger than 1%, just replace the 0 with an 1:

```
length(swapdiat[1,] [swapdiat[1,]>1])
```

```
[1] 17
```

And for the second sample

```
length(swapdiat[2,] [swapdiat[2,]>1])
```

```
[1] 19
```

It would be nice to do that automatically for all samples. R offers here two options, creating a ‘for’ -loop (see **Appendix C**) or to use the function apply. The apply function was built for matrixes and data frames, to do the same thing for every column or every row. As we want to know the number of species with a relative abundance larger than 1% for every sample (=row), this function might be the best to choose. At least the apply function needs three arguments, the data frame to work with, weather to use the rows (1) or the columns (2) and what to do with them. (type ?apply to get more information about this function). We want to work with the swapdiat data frame, using the rows (1) and want to know for each row x how many species have a relative abundance larger than 1%:

```
apply(swapdiat,1,function(x) length(x[x>1]))
```

To combine this result with the result of the function *specnumber()* within on table and to export this as an txt-file, we need to safe the results as objects in R first:

```
number.of.spec<-specnumber(swapdiat)

number.of.spec.larger.1<- apply(swapdiat,1,function(x) length(x[x>1]))
```

and than we combine both within a data frame:

```
sample.info<-data.frame(number.of.spec, number.of.spec.larger.1)

head(sample.info)     # will show you only the first 10 elements of a given data frame or vector
```

```
        number.of.spec    number.of.spec.larger.1
1.21            49                 17
10.21           42                 19
11              38                 19
113.21          40                 22
115.11          46                 17
12.11           36                 19
```

The following function will export this data frame to an csv file (see also Appendix B)

```
write.csv(sample.info,file="C:/MyDocuments/first_result.csv")  #change path to your favorite
                                                                 one
```

But how to create now a new training set excluding all rare taxa? A combination of *apply* and the function *ifelse()* can be used. If the relative abundance value of a species (column) is less than 1 % replace it with 0 otherwise do not change the value: ifelse(x<1,0,x)

```
swapdiat.1<-apply(swapdiat,2,function(x) ifelse(x<1,0,x))
head(swapdiat.1)
```

Of cause, the rowsum for each sample will not be 100 now. If you want to scale up to 100 percent, again the apply function can be used.

```
apply(swapdiat,1,function(x) x*100/sum(x))
```

As here the focus is on the rows (samples) the result is printed in columns (that is the way this function was designed, sorry), that is not what we want, as we need for all other analysis, that the rows are the samples, so we have to transpose the matrix:

```
swapdiat.1a<-t(apply(swapdiat,1,function(x) x*100/sum(x)))
```

In swapdiat.1a all values <1 are replaced by 0 and the row sum equals now for all samples 100% (type rowSums(swapdiat.1a)).

To calculate the diversity of your samples, you can use the function diversity in the package vegan:

```
library(vegan)                  # just type this if you haven't done this before in this R session!
diversity(swapdiat)
```

```
   1.21         10.21       11         113.21      115.11      12.11    …
3.1554085   2.9278682  2.6357835  3.0075259  3.0276772  3.0143587  …
```

But what kind of diversity was calculated here, and how can this be changed? To get the help page of the function diversity, type

```
?diversity
```

Again, a HTML side will be open and give you much information about the function, literature, links to other useful functions and examples, who to use and modify the function (at the end).

We can combine this again with the species data:

```
div<-diversity(swapdiat)
sample.info<-data.frame(number.of.spec, number.of.spec.larger.1,div)
head(sample.info)
write.csv(sample.info,file="C:/MyDocuments/first_result.csv")
```

Most paleoecologist wants to perform multivariate analysis with the training set. Before we do this, we look at the single species:

2.2 Species Environmental Relations

If you are performing a CA / CCA or a WA, you make indirect assumptions about the species distributions to the related environmental variables of interest. As these methods are very robust mostly the results are stable even if the species distribution follows not perfectly e.g. the normal distribution. Nevertheless, for an interpretation of your results you should have a feeling of the quality of your species data and how sure you can be, that your reconstructions are really good. For example, if a species occurred in your training set with a maximum of relative abundances of 45% but in the core data with 65%, you should recognize the situation, have an idea how the species optimum was estimated by the used transfer function, and think about the quality of the prediction that you make.

```
hist(swapdiat$AS001A)            # AS001A is the code for Asterionella Formosa var. formosa, a
                                   planktonic diatom, see also
                                  library(rioja)
                                  data(SWAP)
```

SWAP$names

The *hist()* function will open the graphic window of R and draw a histogram of the relative abundance data of the column AS001A (*Asterionella formosa*) of the data frame swapdiat (Figure 1.1). It is easily to see, that most values are between 0 and 2.5 %, only rarely this taxon occurred in higher abundances. With the function *boxplot(swapdiat$AS001A)* boxplots of the species distribution can be produce, which makes in the case of data enfolding many zeros not much sense.

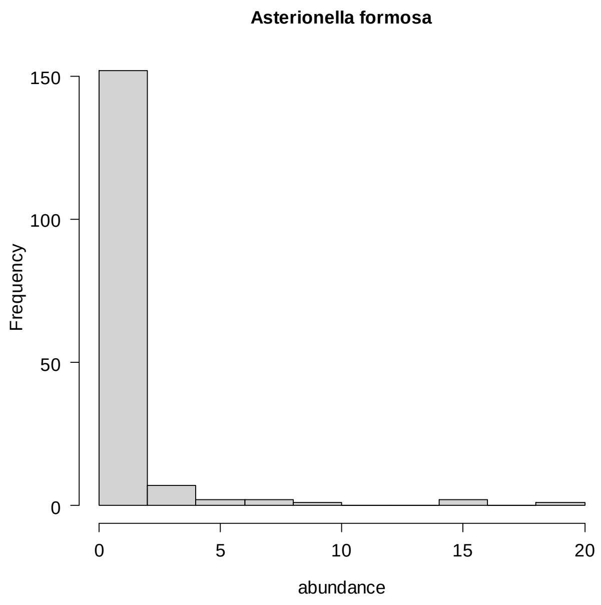


Figure 1.1: Histogram of the taxon *Asterionella formosa*. To improve the graph*: hist(swapdiat$AS001A,main="Asterionella formosa",xlab="abundance",cex.axis =1.2,cex.lab=1.2,col="lightgray",las=1)*

It might be boring always to type swapdiat$AS001A, when plotting or modeling a species out of the swapdiat training set. Alternative you can say R that it should always search within the data frame swapdiat with the command

attach(swapdiat)

and than you only have to type *hist(AS001A)* to get the same plot, as above. But if you want to compare the distribution of *A. formosa* within the swapdiat training set with the distribution of *A. formosa* in the MV training set (package paltran) you will come in trouble, as the code is the same! So working only with one training set, the *attach()* function can safe a lot of typing, working with more training sets the *attach()* function should be used with cautions!

More interesting might be a plot of the species abundance versus the environmental variable of interest. For this we can use the *plot()* function in R (alternatives for *hist()*, *boxplot()* or *plot()* see packages lattice (Sarkar 2008) and ggplot2 (Wickham 2009))

```
data(swappH)                        # makes the pH data within the package analogue available
                                     # for R
plot(swapdiat$AS001A~swappH)
```

The sign "~" called 'tilde' and means that two variables are set in relation. In R stands at the left side of the tilde always the depending variable and at the right on the independent variable. Here we want to display the relative abundance of *A. formosa* in addiction to pH, so *A. formosa* depends on pH and therefore we write *swapdiat$AS001A~swappH*.

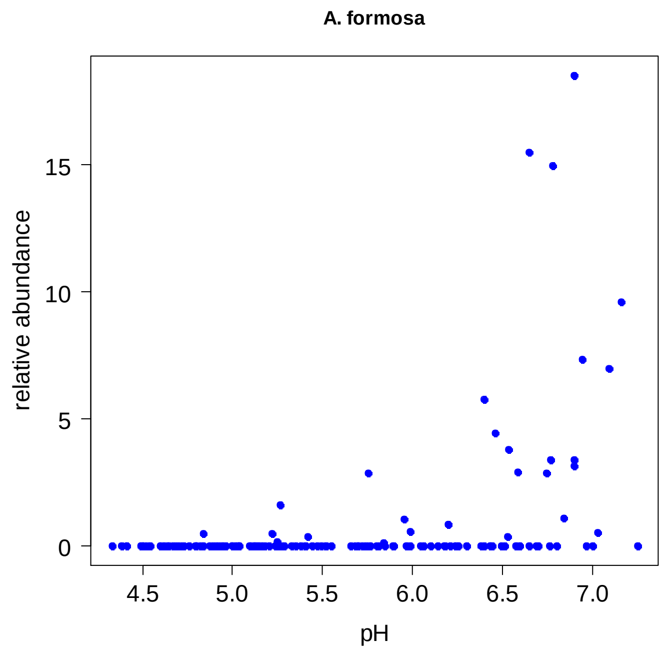


Figure 1.2: Distribution of *Asterionella formosa* against pH. To improve the plot:
plot(swapdiat$AS001A~swappH,cex.lab=1.4,cex.axis=1.4,pch=19,col=4,las=1,xlab="pH",ylab="relative abundance",main="A. formosa")

A. formosa seams to have a uneven distribution to pH (Figure 1.2), but notice, the covered gradient ends here at pH 7.3. It can be interesting to merge this training set with others, to see if this uneven distribution will turn to an unimodal distribution or not. Secondly, in several lakes *A. formosa* did not occurred and the variance of *A. formosa* between pH 6.5 and 7 is remarkable high. At the pH level of ~ 5.75 a relative abundance of ~3% was found but the same relative abundance was found at a pH level of 6.6 and 6.75 (Figure 1.2). At a pH level of ~ 6.75 a relative abundance of 0%, 3% and 15% was found in different sites. So is *A. formosa* a good indicator for pH?

Just to produce such kind of very easy plots for each of the 277 species enfolded in this training set will be boring very fast. To improve this, in **Appendix C** loops are introduced.

For modeling the distribution of *A. formosa* against *pH*, several different methods are available through R. Most common are maybe regression techniques like Linear Models (LM), Generalized Linear Models (GLM, McGullagh and Nelder 1989), and Generalized Additive Models (GAM, Hastie 1990, Wood 2006). A first step in this direction is given by tutorials like http://ecology.msu.montana.edu/labdsv/R/labs/, and books like Crawley 2002, Faraway 2006, Wood 2006, Zuur et al. 2009). Important might be following thoughts: species relative abundance data have two limits - 0% and 100% - that means as underlying assumption of the distribution of the response variable we need one that is restricted too. As the normal distribution allows negative values and is not limited, this might not be appropriate to use, so the use of normal linear regression seams to be wrong here. Instead, generalized linear models (GLM) that are not restricted to a certain distribution of the response can be used. GLM's uses a so called link function, which transforms the data of the response on the basis of a special stochastic distribution. A Binomial distribution is limited to 0 and 1 but allows no value between. In R you can find the so called quasi-Binomial that seams to solve the problem, as it is restricted to the interval [0,1] but allows the values to have every between. (Running

ML regression as transfer function method using rioja (Juggins 2012), that function will exactly use this method, see chapter 4, just type *library(rioja)* and *MLRC.fit* in the Console to see the code.) But the relative abundance values of *A. formosa* are given at the interval [0,100] within the swap training set, how can we solve this?

plot(swapdiat$AS001A/100~swappH)

If you compare this plot with the plot above, you will see, that only the scale of the y axis has changed, not the relation of the points themselves, so we can use this transformation to apply a GLM with a quasi-Binomial link function.

fit1.glm<-glm(swapdiat$AS001A/100~swappH,family="quasibinomial")

What means this code? As we can not give here the complete introduction in GLM and modeling, just briefly:

- glm is the name of the function that computes a generalized linear model (GLM)
- swapdiat$AS001A/100~swappH: We model the relative abundance as a response to pH
- family="quasipoisson" as link function ("transformation of the response data") we use the quasi-Binomial

The result of the function can be viewed by the summary() function

summary(fit1.glm)

```
Call:
glm(formula = swapdiat$AS001A/100 ~ swappH, family = quasibinomial)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.42740  -0.06506  -0.02656  -0.01469   0.62801

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -21.4622     2.4176  -8.878 1.10e-15 ***
swappH        2.6366     0.3596   7.333 9.62e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasibinomial family taken to be 0.02624134)

    Null deviance: 5.5489  on 166  degrees of freedom
Residual deviance: 2.7733  on 165  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 10
```

How to read this? The first row just repeat the formular of the glm, that here the relative abundance of AS001A divided by 100 was modeled against pH using the quasi-Binomial link. In the next block, a summary of the deviance residuals is given (this is different from the residuals knowing from a linear regression, as we have used a data transformation with the link function). In a 'perfect model world' these should be symmetrically distributed around zero with very small minimum and maximum values. In the next block you can find information about the model and test parameter, but be careful again, these values are within the transformed "quasi-Binomial"-world. You can see that the effect of pH is positive and significant, that means the relative abundance of AS001A increase significant with an increasing pH (which we want to prove, fine!). The last block gives you information about the deviance of the Null model and the fitted model. But what is this ominous deviance? In the world of linear

models and the normal distribution we describe the data with the variance. The deviance follows the same concept, the calculation is different, more difficult and not intuitive clear, but we can use the same interpretation as we work with the variance. So following Crawley (2002) we can calculate the “explained variance” for this model as 1-(*2.7733 /5.5489) = 0.50.* **If you want to use GLM’s it is hardly recommended here for details and further information, especially model selection, to read the mentioned literature**!
How can we plot now the model result and how can we get the back transformed predicted model values? Here the function fitted() will help us:

```
fitted(fit1.glm)
points(fitted(fit1.glm)~swappH,col=4)
```

If you want to fit a line you have to introduce an order for the pH values first

```
ord1<-order(swappH)
plot(swapdiat$AS001A/100~swappH,cex.lab=1.4,cex.axis=1.4,pch=19,col=4,las=1,xlab="pH",ylab="re
lative abundance",main="A. formosa")
points(fitted(fit1.glm)[ord1]~swappH[ord1],col=2,type="l",lwd=2)
legend("topleft","fitted glm",col=2,box.lty=0,lty=1,lwd=2)
```

[a very favorite typing error: the "l" is a l as in line not a 1 as the number one! On the console you can see no difference between l and 1]

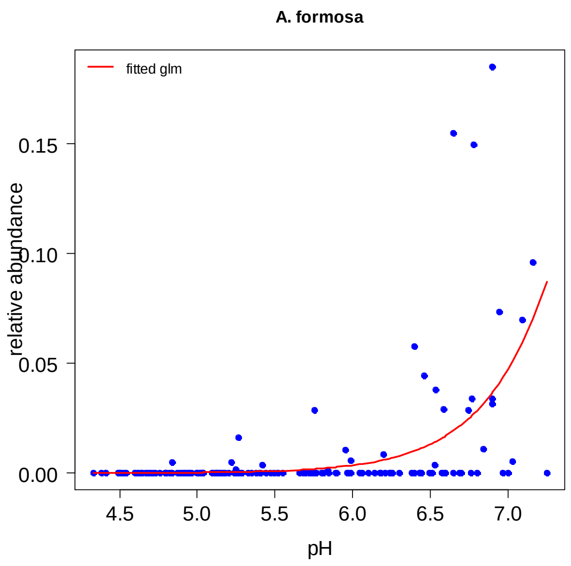


Figure 1.3: Distribution of *Asterionella formosa* against pH: red line – resulting model.

In the model formulation we have written swapdiat$AS001A/100~swappH, that looks like a linear model, so why the resulting plot is a curve? This is due to the link function we have used here, that enfolded the log(x) function.

But is this the best model? We can try a more complex model including a quadratic term:

```
fit2.glm<-glm(swapdiat$AS001A/100~swappH+I(swappH^2),family="quasibinomial")
summary(fit2.glm)
points(fitted(fit2.glm)[ord1]~swappH[ord1],col=3,type="l",lwd=2,lty=2)
legend("topleft",c("fitted glm","fitted 2 order glm"),col=c(2,3),box.lty=0,lty=c(1,2),lwd=2)
```

Looking at the summary, the second order term is not significant and comparing the plots no real improvement can be seen. To test, weather there is a significant difference between both models the *anova()* function can be used:

```
 anova(fit1.glm,fit2.glm,test="F")
```

As the p value is larger than 0.05, there can no difference be found between both models and we can hold the first on.
Focusing on the plot and the “explained variance” of the fit.glm, than it seems, that pH is not the variable, that can explain the species distribution alone, other variables could influence the distribution, what would not be a big surprise. But have a look at the fitted values against the real of the model, respectively at the residuals:

```
spec<- swapdiat$AS001A/100
plot(fitted(fit1.glm)~spec,xlab="real relative abundance",ylab="fitted values", main="A. formosa")
abline(0,1,col=2,lty=2,lwd=2)
```

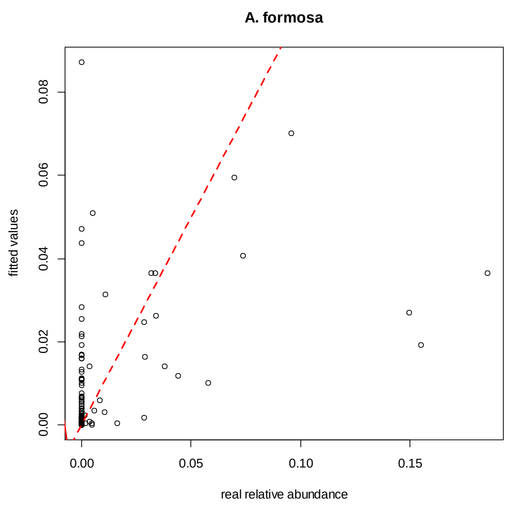


Figure 1.4: fitted relative abundance values against measured values of of *A. Formosa.*

Within a perfect model, all points would lie on the red line in Figure 1.4. But as easily can be seen, especially the higher relative abundances cannot be predict by the model.

```
par(mfrow=c(1,2))                # this split the graphic window in one (1) row and two (2) columns
plot(fitted(fit1.glm)-spec~spec,xlab="real relative abundance",ylab="residuals", main="A. formosa")
abline(0,0,col=2,lty=2,lwd=2)
plot(fitted(fit1.glm)-spec~swappH,xlab="pH",ylab="residuals", main="A. formosa")
abline(0,0,col=2,lty=2,lwd=2)
```

2.3 Multivariate Analysis

Three different types of multivariate analysis can be found, Cluster Analysis , Multidimensional Scaling (MDS), and Correspondence Analysis (CA, and related methods like RDA, CCA) (Cluster Analysis and MDS are both distance based and treated sometimes as the same class of analysis, CA based on vector analysis). Notice, that all of these methods have the aim, to display multivariate data sets in an n-dimensional space, viewing only at the first three dimensions can lead to a lack of information.

```
fit.ca<-cca(swapdiat)                # if within the function cca only species data are available
                                     automatically a CA is performed instead of a CCA
plot(fit.ca,display="sites")         # on some machines the picture is mirror inverted, that is just
                                     due to your computer, the relation of the points to each other
                                     should be the same!
```

This does not look good! Let us try some usual data transformations, like square root and down weighting of rare species:

```
fit.ca<-cca(downweight(sqrt(swapdiat)))
fit.ca                               # will give some basic results
summary(fit.ca)                      # will give detailed results
plot(fit.ca)                         # will plot both, sites and species
plot(fit.ca,display="sites")         # will plot only sites
plot(fit.ca,type="n")
text(fit.ca,"sites")                 # will display site names
```

We can add now information about the sites by different symbols or size:

```
plot(fit.ca,type="n")
points(fit.ca,dis="sites",pch=19,col=4,cex=swappH)        # pch determine the symbol; col the color (1 =
                                                          black, 2 = red, 3 = green, … you can also
                                                          type col="black" or col="lightblue"; cex
                                                          determine the size of the symbols, here we
                                                          use the variable swappH to get more
                                                          information about the sites within this plot:
                                                          there is not much to see, as we have to
                                                          improve the size information:
plot(fit.ca,type="n")
points(fit.ca,dis="sites",pch=19,col=4,cex=swappH/max(swappH)) # the site with the highest pH value
                                                          will have points size 1 , all other
                                                          will be smaller; better but the
                                                          gradient is too short
plot(fit.ca,type="n")
points(fit.ca,dis="sites",pch=19,col=4,cex=3*(swappH-min(swappH))/max(swappH-min(swappH))+.01)
                                                 # (swappH-min(swappH))/max(swappH-
                                                 min(swappH)) scales the date to the intervall
                                                 [0,1], 3*(swappH-min(swappH))/max(swappH-
                                                 min(swappH)) scales the date to the intervall
                                                 [0,3], where the site with the smallest pH will
                                                 get size 0 wich is not meaningful, therefor we
                                                 ad +0.01. Normally 3*swappH/max(swappH)
                                                 will be a good solution, but here the gradient of
                                                 the ph values is just too small.
```

In this latest version, the size of the points is scaled to the interval 0.03 and 3, where the sample with the smallest pH value has the size 0.03, the sample with the largest pH has the size 3 (Figure 2.1).

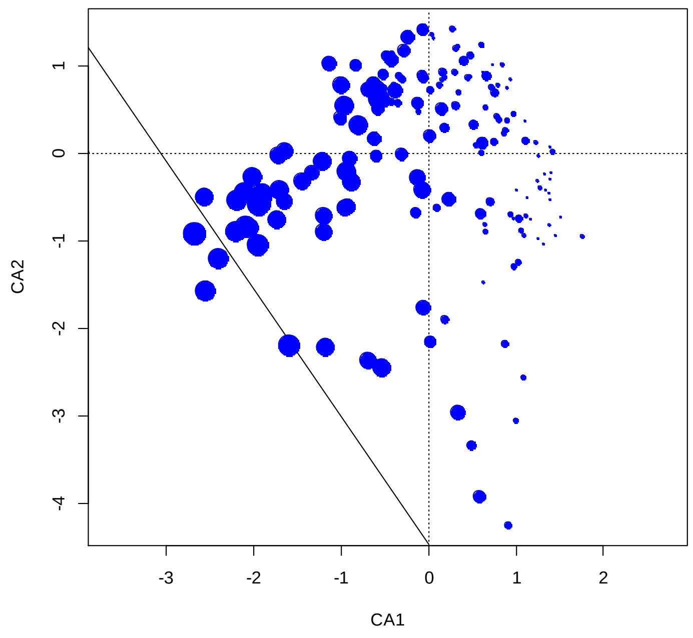


Figure 2.1: CA of the SWAP training set using the given ph values for the size of the symbols

A strong pH gradient along the first CA axis is visible here, but it is also visible, that a large variance occurred along the second axis. It would be nice to know, what kind of environmental variable is related to the second axis, maybe it is the same that caused the large variance of *A. formosa*?

But here we look only at the first two axis. Adding the attribute choices=c(2,3) or choices=c(1,3) will change the axis that will be displayed. Notice, in both functions the plot() and point() you have to add this!

```
plot(fit.ca,type="n",choices=c(2,3))
points(fit.ca,dis="sites",pch=19,col=4,cex=3*(swappH-min(swappH))/max(swappH-min(swappH))+.01,
choices=c(2,3))
```

More strait forward (nice for presentations but bad for publications, is the following code):

```
install.packages("rgl")
library(rgl)
site.scores<-scores(fit.ca,choices=c(1,2,3))$sites
plot3d(site.scores,radius=((swappH-min(swappH))/max(swappH-min(swappH))+.01)/2,type= "s",col=3)
```

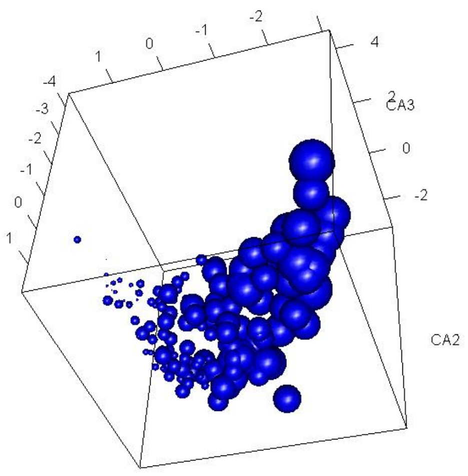


Figure 2.2: 3-dimensional graph for the CA of the SWAP training set

A different method to look multivariate at the date is the MDS. One function that runs a MDS in R is *metaMDS* (type ?metaMDS (library(vegan) to see a detailed description, especially the difference between isoMDS and metaMDS)

```
library(vegan)
fit.mds<-metaMDS(swapdiat)
plot(fit.mds,display= "sites")
```

Is this result different to the result of the CA? Here the vegan package delivers a very nice function to compare different multivariate graphical results:

```
fit.pro<-procrustes(fit.mds,fit.ca)
plot(fit.pro)
protest(fit.mds,fit.ca)
```

The function rotates two different multivariate results to a maximum of similarity and displays the differences between both. The best situation of course is that both methods – CA and MDS – will give the same/similar results, because than it is not important which method you choose. Getting two completely different results (e.g. different graphical groups in CA and MADS) should make you nervous, and you have to start getting deeper insight the theoretical background of these methods.

As the data given in analogue, rioja and paltran enfolded only on environmental variable, running here a CCA makes not too much sense. And, presenting here the use of a CCA in R would be only a reproduction of the very good manual of Jari Oksanen, so we give you here the link to his vegan tutorial. More interesting is, how good the core data are represented in the given training set, this will be analysed in the following chapter.

3. The Core Data

In this chapter we will use the data of the package paltran, as they show a typical problem that is not solved sufficient in the literature. But later more, first we will have a look at the core data of the north German Lake Dudinghausener See (Dressler et al 2006).

```
library(paltran)
```

```
data(dud.df)
dim(dud.df)
names(dud.df)
row.names(dud.df)
?dud.df
```

In this package the information of the age of the core data is given with a separate object, other data organizations are possible, e.g. in rioja (Juggins 2012) the core data and the age data are combined in a list (see chapter 4 and Appendix A)

```
data(age.dud)
?age.dud
```

If we use the standart plot() function in R the information of the age will be given on the x axis:

```
plot(dud.df$XXG974~age.dud$age,type="h")
```

This plot is ‘right’ in a mathematical point of view, but in paleoecology we are used to have the response on the x-axis and the age/depth on the y-axis with the highest depth at the origin of the graph.

All three packages offer functions for plotting such stratigraphic plots (paltran: palplot(), analogue: Stratiplot(), rioja: strat.plot()) and it is hard to say which is the most useful, as they have different advantages and disadvantages. The reason for building paltran was to implement the Moving-Window approach (MW, Hübener et al. 2008). As at that time rioja does not exist and analogue had no Weighted Averaging (WA) and Weightet Averaging– Partial Least Square (WA-PLS) Regression transfer function method implemented, we had to build it by our self, but comparing with the current versions of analogue and rioja the WA and WA-PLS functions (especially the cross validation methods) are much more slower. So the idea is, to include the functions from the rioja or analogue packages in the mw functions in paltran to improve the speed or just to delete the package paltran and including the mw function in analogue or rioja as we can see now advantages of the existence of three different packages doing nearly the same. So we will not focus too much on the functions of paltran but here is one example where it might be useful: In analogue and rioja it is not possible to run strat.plot or Stratiplot with only one species (current version July 2012). This is maybe more an academic problem and not of interest of the ‘normal’ user, but shows a general problem of R: Different colleagues have different ideas how an analysis should be done and will implement different codes. For the user this is sometimes confusing and hard to understand (more extreme example see chapter 4: different types of coding for the transfer functions).

Remember: R is an open source platform, nobody get money for supporting packages, as you do not have to pay for it. If you have questions regarding a function, you can ask the author (as contract details always provided). Every author will be thankful if you have questions, suggestions or if you found errors.

To follow the didactical concept in this manual, getting a deeper understanding of the data before doing any analysis, we can plot the relative abundance of ‘XXG974’ (*Stephanodiscus minutulus*) over time only (currently, July 2012) with the palplot function:

```
palplot(dud.df$ XXG974,age=age.dud$age,ptype="h")
```

If we want to see the development of ‘XXG974’ using Stratiplot or strat.plot, we have to add a second taxa:

```
Stratiplot(dud.df[,c("XXG974","XXG973")],age.dud$age,type="h")            #using rioja
```

```
strat.plot(dud.df[,c("XXG974","XXG973",”AS001A)],age.dud$age)          #using analogue

palplot(dud.df[,c("XXG974","XXG973")],age=age.dud$age,ptype="h")       #and paltran again
```

As usual stratigraphic plots display several, e.g. the most important taxa, so Stratiplot and strat.plot are well done for this porpoise. But how do we choose those most important taxa? If the research interest is focused on changes, than mostly those taxa are presented, that shows that changes, only rarely those are presented that show no changes. If we have a look at the picture of XXG974 (= *Stephanodiscus minutulus*) it is easy to see that the relative abundance of this taxon increases in the younger periods of this lake strongly.

```
palplot(dud.df$ XXG974,age=age.dud$age,ptype="h",p.xlab=c("XXG974 [%]"),trend=T,p.col=4)
data(train_set.MV)data(train_env.MV)
plot(train_set.MV$XXG974~train_env.MV$V1,xlab="log(total Phosphorous)",ylab="relative
abundance",main="XXG974",col=4,pch=19)
```

if you want to see both plots side by side you have to participate the graphic window, as seen obove(at the end of Chapter 2.2)

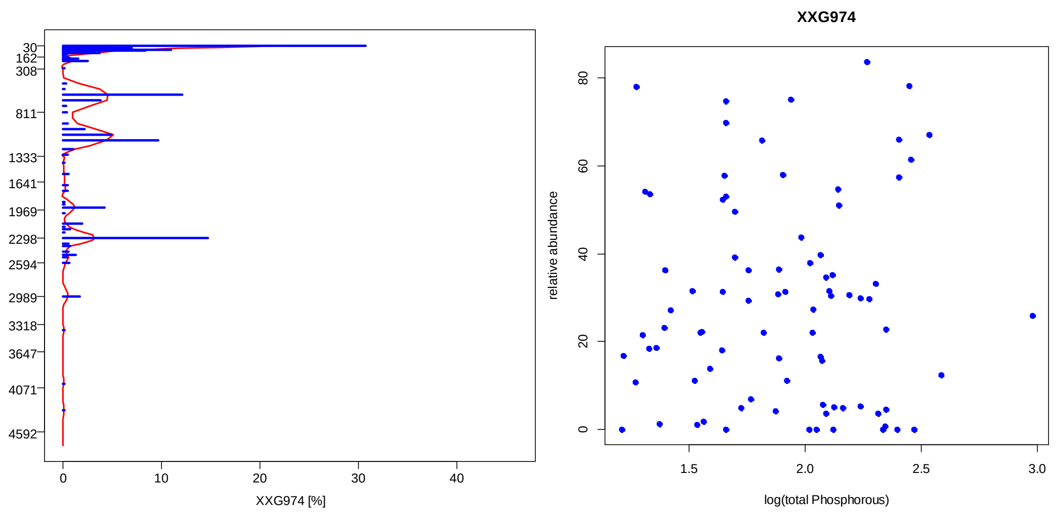


Figure 3.1: left panel: development of S. minutulus through time in the core of L~~l~~ake Dudinghausener See, right panel, distribution of S. minutulus against Total Phosphorous (TP).

But looking at the data in the training set (Figure 3.1, right panel) it is obviously, that in this region no trend of *S. minutulus* in relation to TP is detectable. Nevertheless, as the WA method will calculate an optimum for this taxon, the development of *S. minutulus* will influence the reconstructed TP value, as *S minutulus* occurred in high abundances and these will be multiply with the ‘optimum’. So the calculated TP values will increase cause of the occurrence of *S. minutulus*, but *S. minutulus* itself shows no response to TP. This is what we mean with ‘know your data well’! (for details see chapter 4.3) As different plotting method try also (package rioja):

```
mx <- apply(dud.df, 2, max)
spec <- dud.df[, mx > 1]
inkspot(spec, cex.axis=0.6)
```

3.2. Constrained hierarchical clustering

Focusing on core data, after a brief description of the development of the single taxa, the first question is, are there detectable groups or zones in the core that indicates changes in the environment of the lake. Juggins program Zone is available through the package rioja and the function chclust().

But first, as in every cluster analysis, we have to calculate the difference between the samples, using a distance metric. As the number of core samples is high and the graphical output is not easy to look at, we first reduce the number of samples, to have it a little bit easier:

```
dud.red<-dud.df[c(1:50),]
dud.dist1 <- dist(dud.red)
```

The function dist is implemented within the package stat, which is automatically load when opening R. Typ ?dist, to see what kind of distance measurement is used here. It is hardly recomended to compare different methods and their results with each other. Different measurements of distances are also available through vegan with the function vegdist()

```
dud.clust1 <- chclust(dud.dist1)
```

plot(dud.clust1)The attribute hang=-1 will change the display of the graph in that way, that the labels will hang all at the same line and horiz = T will turn the graph.

```
plot(dud.clust1,hang=-1)
plot(dud.clust1,hang=-1, horiz=TRUE ,x.rev=TRUE)
plot(dud.clust1,hang=-1, horiz=TRUE ,x.rev=TRUE,cex=.6,main="Cluster Analysis of Lake
Dudinghausener See")
```

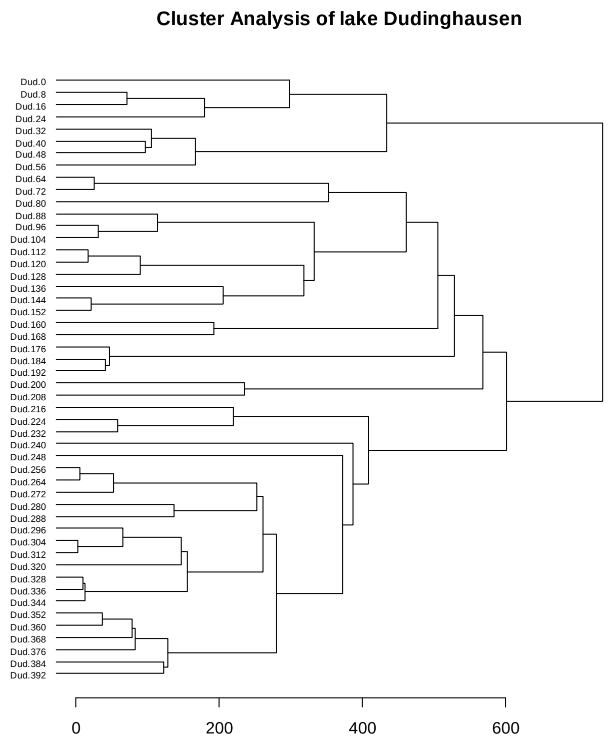


Figure 3.2: Cluster analysis of the first 50 core samples of the lake Dudinghausener See using the function *chclust()*.

Three groups are possible to see (Figure 3.2): sample 1 to 8, sample 9 to 38 and the rest. With the function rect.hclust (vegan) is it possible to visualize them:

```
library(vegan)
plot(dud.clust1,hang=-1,cex=.6,main="Cluster Analysis of Lake Dudinghausener See")
rect.hclust(dud.clust1, 3)
```

Let’s try a different distance metric:

```
dud.dist2 <- vegdist(dud.red)
dud.clust2 <- chclust(dud.dist2)
par(mfrow=c(2,1))
plot(dud.clust1,hang=-1,cex=.6)               #cex specify here the size of the sample labels
plot(dud.clust2,hang=-1,cex=.6)
```

The differences are here small, but nevertheless, what is the best fit? Oksanan (2011) offers here a good tool to deal with: the function *cophenetic*, that extracts the differences of the different samples within the given plot. Using correlation techniques (function *cor*), we can than compare the correlation of the graphical output with the originally distances between the samples:

```
cor(dud.dist1, cophenetic(dud.clust1))
[1] 0.5494308
 cor(dud.dist2, cophenetic(dud.clust2))
[1] 0.5905178
```

For more details see the vegan tutorial. As we have distinguished three groups we can create a vector enfold the information of the membership to this three groups, by numbers 1,2,3:

```
grp <- cutree(dud.clust1, 3)
```

Now we can run a multivariable analysis of the core data:

```
mod<-cca(sqrt(dud.red))
plot(mod,type="n")
text(mod,display="sites",col=grp)
```

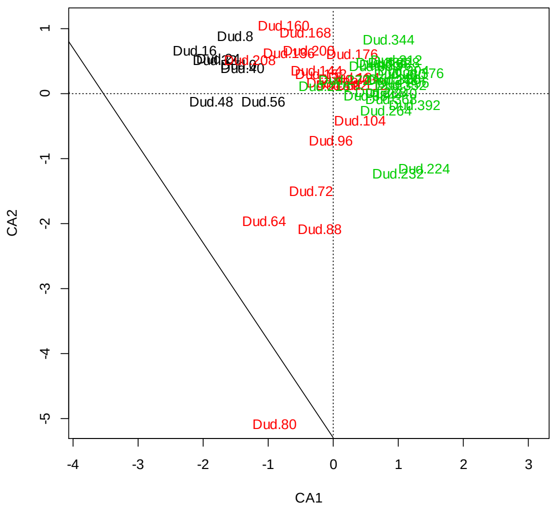


Figure 3.3: CA of the first 50 core samples of the lake Dudinghausener See, the colors of the symbols represent the three groups: black – group 1, red – group 2, green – group 3

In Figure 3.3 it is easy to see, that the three groups found by the distance based cluster analysis can be found with the vector algebra baesd method CA too. If we are interested in the taxa that caused this separating of three groups we can use the following code:

```
par(mfrow=c(1,2))
plot(mod,type="n",main="Asteriomella formosa")
points(mod,display="sites",col=grp,cex=dud.red$AS001A)
plot(mod,type="n",main="Stephanodiscus minutulus")
points(mod,display="sites",col=grp,cex=dud.red$XXG974)
```

As it is possible to see in Figure 3.4, the *taxon Asterionella formosa* is even distributed in the core data, whereas *Stephanodiscus minutulus* occurred abundant in the upper most samples.

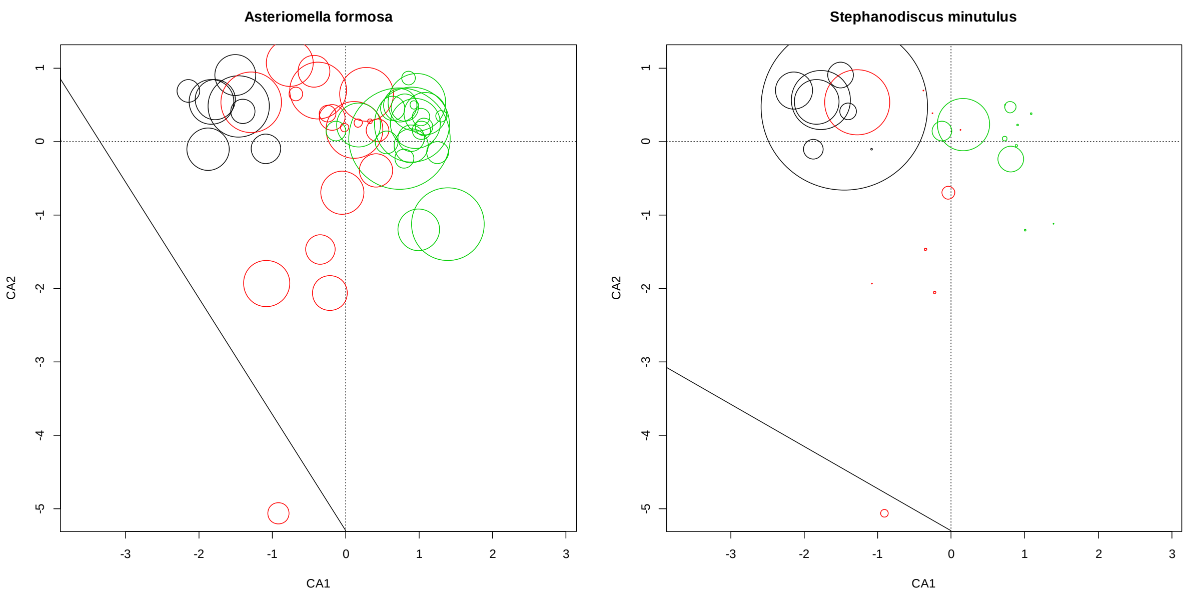


Figure 3.4: distribution of A. formosa and S minutulus within the first 50 core samples of lake Dudinghausener See

Before applying a transfer function to the core data, it has to be checked, whether the core data are well represented by the used training set.

```
data(train_set.MV)
data(train_env.MV)
data(dud.df)
data<-Merge(train_set.MV,dud.df,split=T) # both data sets need to have the same columns, alternative
                                          function: join in package analogue or the function merge().
                                          Using merge, several NA's will be produced that have to
                                          be replaced, e.g. using a combination of the functions
                                          apply(), ifelse and  is.na(). This will be done for you
                                          automatically by the funcitons Merge and join.
names(data)
mod<-cca(downweight(sqrt(data$train_set.MV)))           # normal CA
fit<-predict(mod,newdata=sqrt(data$dud.df),type="wa")  # prediction of the core data using the CA
                                                         model
plot(mod,type="n")
points(mod,dis="sites",pch=19,col=4)
points(fit,col=grp,pch=19)
legend("bottomleft",c("training set MV","core data group 1","core data group 2","core data group
               3"),pch=19,col=c(4,1,2,3))
```

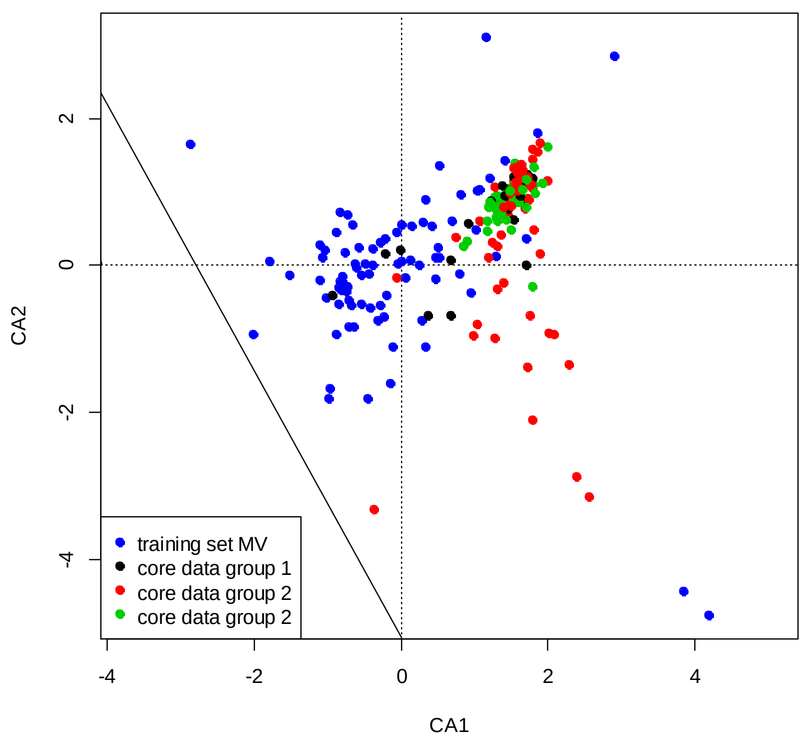


Figure 3.5. Prediction of the first 50 core samples from lake Dudinghausener See in the North German training set.
The core data associated to group 1 and 2 are more or less well represented within the training set, but the samples of group 3 are rarely represented.

We can see in Figure 3.5. that the core samples related to group3 are not well represented by the North German training set. That means for theses samples there are not enough modern analogues available within this training set. This is a problem as a prediction for the core samples based on the training set will be very doubtful. But are group 1 and 2 really good represented or do we have here more than 2 gradients that determine the species distributions (see 2.3). Let's have a look at the third axis:

```
plot(mod,type="n",choices=c(2,3),ylim=c(-5,5))
points(mod,dis="sites",pch=19,col=4, choices=c(2,3))
points(fit[,2],fit[,3],col=grp,pch=19)
legend("bottomleft",c("training set MV","core data group 1","core data group 2","core data group 3"),pch=19,col=c(4,1,2,3))
```

and

```
plot(mod,type="n",choices=c(1,3),ylim=c(-5,5))
points(mod,dis="sites",pch=19,col=4, choices=c(1,3))
points(fit[,1],fit[,3],col=grp,pch=19)
legend("bottomleft",c("training set MV","core data group 1","core data group 2","core data group 3"),pch=19,col=c(4,1,2,3))
```

Of cause we can use again the 3 dimensional view here:

```
library(rgl)
site.scores<-scores(mod,choices=c(1,2,3))$sites
plot3d(site.scores, col=4,radius=0.1,,type= "s")

plot3d(fit[,c(1,2,3)],ad=T,col=grp,type="s",radius=0.1)
```

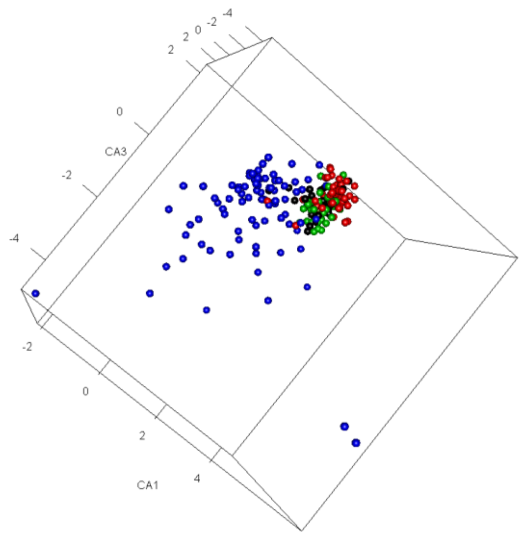


Figure 3.6. 3-dimensional view of the CA of the North German training set with a prediction of the first 50 core samples of Lake Dudinghausener See.

It is obviously, that only a few upper core samples are really represented by the training set and maybe other training sets should be used to reconstruct the development of environmental conditions for this core. For details of this problem see Dressler et al. 2006 and Hübener et al. 2008.

Remark: The type of transformation used (here the square root) can influence the result strongly, but in a sense of reasonability you have to hold the same transformation when using WA, WA-PLS or other methods when reconstructing environmental conditions.

4. Transfer Functions - Environmental reconstructions

When we are reconstructing an environmental variable based on a given species composition we assume, that this variable was the main driver that determine the species composition in the core. The only way to prove this is to analyse the modern training set, for the core exists no prove. That is the work we have to do before the reconstruction and is described in the chapter 2 and the given related literature. Several methods exist to reconstruct the environmental variable of interest (WA, WA-PLS, MAT, ML), and it is hard to choose the right on. Each method has some basic assumptions about the used data and for that reasons some are more appropriate than others.
E.g., it is known, that the ML method is more robust against uneven distributions and predictions outside the covered range of the training set might be more precise that that of the WA. Nevertheless, WA and WA-PLS are the most often used tools, also MAT. Often the results of the different methods

show the same trend but different values, sometimes the reconstructions are nearly parallel with a difference of 10% or more. Only rarely the trends differ. The problem for us is to determine which of the reconstructions is possibly more précises than the other. Remember: We do not trust the weather forecast for the next week, but we do a prediction for an environmental variable for at least the last hundreds or thousands of years. It might be useful to interpret the reconstructed values more as an index for e.g. trophic state or temperature development, rather than in microgram phosphorous or degrees temperature.
The WA is implemented in all three packages analogue, rioja and paltran, but we will use here only the first two, as the programming is more straight forward.

Let us start with the WA (Weighted averaging) transfer function using the classical data from Imbrie and Kipp (1971).

```
data(ImbrieKipp)
data(SumSST)

mod <- wa(SumSST ~., data = ImbrieKipp)
mod
```

```
Weighted Averaging Transfer Function
Call:
wa(formula = SumSST ~ ., data = ImbrieKipp)
Deshrinking  : Inverse
Tolerance DW : No
No. samples  : 61
No. species  : 27
Performance:
    RMSE  R-squared  Avg. Bias  Max. Bias
  2.0188     0.9173     0.0000    -3.8155
```

In the package analogue the function wa will run a Weighted Averaging transfer function for you, here for the training set ImbrieKipp using See Surface Temperatur (SumSST) as environmental variable. The coding is a little bit tricky here, as the SumSST is on the left side of the tilde, so interpreted as the response of the species combination.
In the summary of the result we can see the number of samples and taxa and some information relating to the error statistics. Up to now the RMSE and the R^2 looks very good. To get more information about the result of the transfer function typ:

```
names(mod)
```

```
 [1] "wa.optima"     "tolerances"    "model.tol"     "fitted.values"
 [5] "residuals"     "coefficients"  "rmse"          "r.squared"
 [9] "avg.bias"      "max.bias"      "n.samp"        "n.spp"
[13] "deshrink"      "tol.dw"        "call"          "orig.x"
[17] "orig.env"      "options.tol"
```

And than e.g.

```
mod$wa.optima                    # mod[[1]] will do the same
```

Diagnostic plots for this model can be accessed by typing

```
par(mfrow = c(1,2))
plot(mod)
```

The red smoother in the left panel follows more or less the first directive line but it seems to be a trend in the residuals (right panel). Nevertheless, comparing this plot with different other training sets, it looks good.

To estimate the "real" prediction error of these transfer function, cross validation methods are needed, as long as no real test set is available (as in the usual case, but see Adler and Hübener 2007). This is performed by a different function:

```
cv.boot <- crossval(mod, method = "bootstrap", nboot = 250)
cv.boot
```

```
Model Cross-validation:
crossval(obj = mod, method = "bootstrap", nboot = 250)
Method: bootstrap
No. Bootstraps: 250
       R2       avgBias    maxBias    RMSEP    RMSEP2       s1      s2
1 0.90092   -0.025018    -4.5097    2.2109   2.2602    0.4692  2.2109
```

If you have load both, rioja and analogue, than you will have here a problem, as both packages offer a function called crossval! You have now two possibilities: first type

```
detach("package:rioja")
```

which will hopefully delete all functions from rioja in your current R session and load the library analogue again. But this works not on all R versions, and then you have to use the second way: close R, start it again and load only the package analogue.

The RMSEP is slightly higher than the RMSE and the R^2 slightly lower, but in comparison with the estimated values before these values are good and provide a good model performance. If you want to perform other types of cross validations typ ?crossval to get the information. Running the function crossval of the package analogue two times, the output differ slightly, as the bootstrap method is a random sampling with replacement and due to this the results cannot be the same, especially using a low number of bootstrap cycles. Setting the number higher (e.g.,1000) will give you similar results but it takes more time for calculation. (In C2 the random generator has always the same start point, so here the results are always the same)
But up to now we have done no reconstruction. Using analogue and rioja you will need a second function for that:

```
data(V12.122)                        #core data to reconstruct summer sea surface temperature

v12.pred <- predict(mod, V12.122, CV = "bootstrap", n.boot = 100)
reconPlot(v12.pred, use.labels = TRUE, display = "bars")
```

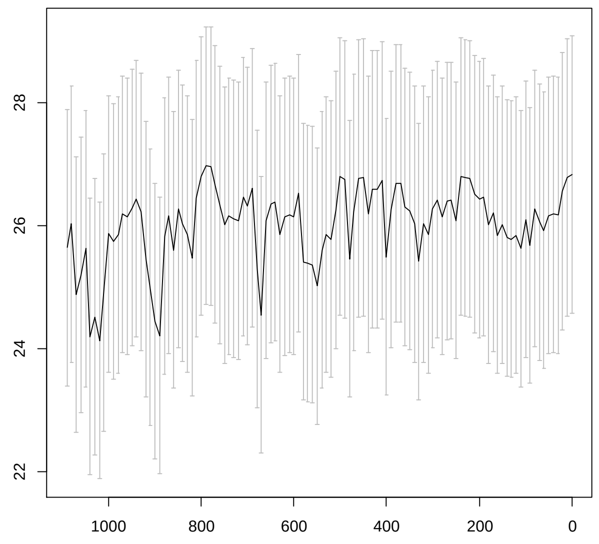


Figure 4.1: Reconstruction of the sea surface summer temperature for the core *V12.122*

The reconstruction of the summer sea surface temperature shows an up and down, but only slightly a trend. Telford and Birks (2011) provide a method to test whether this reconstruction is significant, for details see 4.3.

The function mat for the Modern Analogue Approach (LIT) is provided by the package analogue too. Several new developments are implemented using Monte Carlo methods to optimise the number of modern analogues used. Typ ?mat in the console and follow the examples. As the MAT method makes fewer assumptions for the data used, it might be jusfull to compare the results of the WA with those of the MAT method.

In the package rioja is also a function for Weighted Averaging called *WA*:

```
library(rioja)
?WA
```

As you can see in the example the coding is different to that of the package analogue. But have at first a look at the data:

```
data(SWAP)
names(SWAP)
[1] "spec"  "pH"    "names"
```

The object SWAP is not a data frame, it is a list (see Appendix A, 4.4) enfolding a data frame with the species information, a second data frame enfolding the environmental information and at least a data frame where the species codes are explained. To get in the single elements of the list we can use the [[]] or $ code

```
SWAP$spec                  # equals SWAP[[1]]
SWAP$pH                    # equals SWAP[[2]]
SWAP$names                 # equals SWAP[[3]]
```

We can use them direct for the *WA* function:

```
mod1<-WA(SWAP$spec, SWAP$pH)
```

Or we can create new objects:

```
train<- SWAP$spec
env<- SWAP$pH
mod1<-WA(train,env)
mod1
```

```
Method : Weighted Averaging
Call   : NULL
Tolerance DW : No
No. samples  : 167
No. species  : 277
Cross val.   : none
Deshrinking regression coefficients:
      Inverse d/s Classical d/s
wa.b0     -3.6091       2.6201
wa.b1      1.6497       0.5284
Performance:
         RMSE      R2  Avg.Bias  Max.Bias
WA.inv  0.2756  0.8717         0    0.1933
WA.cla  0.2952  0.8717         0    0.1486
```

```
plot(mod1)
plot(fit, resid=TRUE)
```

You can try to run the same data set with the function wa of the package analogue and you will see, that both results are the same.

Using the resulting model to reconstruct an environmental variable for a given core, again we have to use a prediction function:

```
data(RLGH)
names(RLGH)
[1] "spec"   "depths" "names"
pred <- predict(mod1,RLGH$spec)
```

As the WA calculates a Weighted Averaging transfer function with both, invers and classical deshrinking, the result of the prediction enfolds both too.

```
head(pred)
```

So if we want to plot the result we have to consider for only one part of the reconstruction *pred$fit[, 1]* for inverse deshrinking or *pred$fit[, 1]* for classical deshrinking.

```
plot(RLGH$depths, pred$fit[, 1], type="b")
```

Type *?WA* to read further, especially how to downweight tolerance and choosing other options.
As mentioned before, the function to create a cross validation has the same name as in the package analogue:

```
mod1.xval <- crossval(mod1, cv.method="boot", nboot=1000)
```

```
par(mfrow=c(1,2))
plot(mod1.xval)
plot(mod1, resid=TRUE)
plot(mod1.xval, xval=TRUE)
plot(mod1.xval, xval=TRUE, resid=TRUE)
```

and for the reconstruction of the core:

```
pred <- predict(mod1, RLGH$spec, sse=TRUE, nboot=1000)
```

To run a WA-PLS or ML (Braak & van Dam 1989) transfer function in the package rioja the functions WAPLS and the function MLRC can be used. The coding is in principle similar to that of the WA, notice the rand.t.test function to determine the number of components in the WA-PLS. Using the function MLRC you are using indirect the R function *glm* of the package stats (R Development Core Team 2012). As said before, a quasi-Binomial *glm* is performed here, so all your relative abundance data must have a range between 0 and 1. So before running a MLRC transform your data with

```
train<-train/100
```

and the same with the core data:

```
core<- RLGH$spec/100
```

```
fit <- MLRC(train,env)
```

```
Warning message:
In MLRC(train, env) :
  Trying to fit responses to some taxa with less than 5 occurrences - results may be unreliable
```

This warning message tells you that for some taxa an optimum was calculated with less than 5 data points, which is due to the method used, not a stable result. (The same will be done using the WA or WAPLS function, even that you do not know this!). As we have said before the use of an GLM is not that easy, so try to understand the output of the model, e.g. not all species optima will be an optimum here, it can be minimum too.

If you have found the same problem in your data as we have seen for the North German training set and the core data of Dudinghausener See, that some or most of the core data are not represented in your modern training set, than on possibility to solve this, is to merge different training sets, e.g, the North Germman training set and the Central European training set (see EDDI). Rioja and analogue offers functions to merge training sets (*Merge* and *join*), the basic function in R behind is *merge*. It might not be very useful to create a WA or WAPLS for the hole training set, local solutions might fit here better. The MAT approach seems to be a good solution (see function *mat* in analogue and function MAT in rioja), an alternative is given with the Moving Window Approach (Huebener et al 2008) and the function *mw* in the package paltran. As said, the cross validation is very slow in this package, try first running the function mw with a low number of bootstrap runs. Improvements will come.

4.2. Significant taxa

As mentioned, taxa that have no significant response to one of the environmental variables of interest can influence the reconstruction, we will prove this here.

```
library(rioja)
library(paltran)
data(train_set.MV)
data(train_env.MV)
data(dud.df)
mod<-WA(train_set.MV, train_env.MV)
mod
pre<-predict(mod,dud.df)
```

Notice, the R^2 of this training set is not very satisfying, a method to improve this is discussed by Adler and Huebner (submitted). Even a square root transformation will not improve this:

```
mod_s<-WA(sqrt(train_set.MV), train_env.MV)
mod_s
pre<-predict(mod_s,sqrt(dud.df))
```

so now let us exclude *Stephanodiscus minutulus* from the training set (see 3, Figure 3.1):

```
names(train_set.MV)
train_set.MV1<- train_set.MV[,-152]
mod1<-WA(train_set.MV1, train_env.MV)
mod1
pre1<-predict(mod1,dud.df[,-218])
```

The performance of the transfer function increased slightly and the core date differs slightly at the beginning of the core:

```
reconstruction<-cbind(pre$fit[,1], pre1$fit[,1])
reconstruction
10^reconstruction
```

Using the square root transformation this difference will be smaller. In general we use here rarely square root or log transformed data. But as the within sample variance and the within diatom counter variance is not low (e.g. Adler and Huebener 2012) the use of these transformations might be useful.
The functions wa and wapls in the package paltran offer a check for significant response of the taxa using GAM's and the package mgcv (Wood 2006)

4.3 Significant reconstruction

As a transfer function will always provide data for you, it is hardly required to control the meaning of this data. As said, you will never trust a weather forecast for the next week. Mostly we work with multi proxy studies, so with the information from different proxies we can compare and check them against each other. Telford and Birks (2011) provide a method to test, whether a reconstruction is significant or not:

```
library(palaeoSig)
cor.test<- randomTF(spp = ImbrieKipp, env = data.frame(SumSST), fos = V12.122 ,n=99,fun=WA, col
= 1)
```

In the current version of the ImbrieKipp data set (08-2012) the following error massage will apper:

```
Error in fun(spp, ev, ...) :
  Species data have zero abundances for the following columns: 28,29,30
```

Typing *colSums(ImbrieKipp)* we can see that this is right. As the function randomTF use an RDA where the sum of all rows and columns must be unequal zero, we have to delete them.

```
ImbrieKipp_1<- ImbrieKipp[,-c(28,29,30)]
cor.test<- randomTF(spp = ImbrieKipp_1, env = data.frame(SumSST), fos = V12.122 ,n=99,fun=WA,
col = 1)

cor.test$sig
plot.palaeoSig(cor.test, "SumSST")
```

Of course, a non-significant reconstruction is a result, and a significant reconstruction should be analysed carefully to, as this approach says only, that the reconstructed values are not random. It says nothing about the values themselves, if they are too high or too low, or if they make sense in a ecological manner. For the transect data of Adler and Hübener 2007 the inferred values followed a line were right, nevertheless that reconstruction turned out to be not significant using this test.

APPENDIX A

Information about R can be found at the R homepage (http://www.r-project.org/index.html) under the link Manuals. Beside a very detailed introduction (R-intro) in R you will get information's about how to read data in (R data input) or several additionally tutorials from different fields of statistics (contributed documentation) and a large number of books (books). Here only a fast "Getting started" summary is given:

1. Installation of R

Under http://www.r-project.org/ you find the home page of the R project. To download R got to *Download, Packages*, CRAN (left side). You will be invited to choose a server, where to download the program. Once you have choosen, you will have different possibilities for Windows or Linux. Choose the operating system you are working with, e.g. Windows, and download the *base* version of the actual R version. Once downloaded just double click at the downloaded file (in Windows) and follow the installation instructions.

If you open R a more or less empty Console will appear. A red > (called "prompt") shows you that the program is ready to start. Type e.g.

```
5+4
```

And confirm the entry with the return key at your keyboard, so

```
[1] 9
```

appears.

Variables can be defined very easy in R by the symbol '<-' (type '<' and '-')

```
x<-5
y<-4
x+y
[1] 9
x-y
[1] 1

x+3*y
[1] 17
```

In R different data organisations are given: **vectors, matrixes**, **data frames, and lists**.

2. Data organisations / data types

*2.1 Vectors*A vector is a sequence of numbers like 3,4,5,6,7. In R it would be defined as

```
x<-c(3,4,5,6,7)         # confirm the entry with the return key at your keyboard
```

if you type

```
x
```

(and confirm the entry with the return key at your keyboard)

```
[1] 3 4 5 6 7
```

will be displayed on the console. If you want now to know the length of this vector type

```
length(x)
[1] 5
```

If you want to restrict the vector to values larger than 4, type

```
x[x>4]
[1] 5 6 7
```

And if you want to display single values of this vector, till example the 4. element typ

```
x[4]
[1] 6
```

Try also

```
3*x
x+2
x+x
sqrt(x)
```

If you want to calculate the sum 3+4+5+6+7, installed functions in R can be used, type

```
sum(x)
mean(x)
sd(x)
max(x)
min(x)
```

For every function in R a HTML help page must be produced be the programmers, so that the user gets all informations he need to use the function in a correct way. Typ

```
?c
?sum
?mean
?sd
```

to find the information about these functions.

To produce a sequence of numbers typ

```
x<-c(1:20)
x
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

2.2 Matrices

A matrix is a rectangular mathematical object structured by rows and columns filled with numbers. In a more biological sense: a table filled with numbers:

```
x<-matrix(c(2,3,4, 5,6,7),nrow=2,ncol=3)
```

```
x
     [,1]  [,2] [,3]
[1,]    2    4    6
[2,]    3    5    7
```

or more complex

```
x<-matrix(c(2,3,4, 5,6,7),nrow=2,ncol=3, dimnames = list(c("sample1", "sample2"), c("spec1", "spec2",
                                "spec3")))
x

        spec1 spec2 spec3
sample1     2     4     6
sample2     3     5     7
```

It is on advantage of R that this matrix is now very easy to manipulate! Try:

```
3*x
x-3
x^2
min(x)
max(x)
mean(x)
```

If you want to transform your species data, just typ:

```
log(x)
sqrt(x)
```

If you want to safe your manipulation define a new variable:

```
y<-sqrt(x)
y+x
```

To have access to single values of a matrix you need two information's, the number of the row and the number of the column

```
x[1,1]
[1] 2
x[2,3]
[1] 7
x[2,]
spec1 spec2 spec3
    3     5     7
x[,3]
sample1 sample2
      6       7
```

As matrices are only defined for numbers a third data organisation type is required:

4.3 Data Frames

A data frame can enfold other data types as numbers: factors and strings (both categorical variables). Mostly the read.table function produce such kind of data frames when you read in data to R. As matrices do not need a header, data frames mostly have one and single columns can be assecced by their names unding the name of the data frame and the $ sign.

```
library(analogue)
```

```
data(swapdiat)
swapdiat$AS001A
```

all functions mentioned for matrices are working with data frames, too.

4.4 Lists

Lists combine different types of data mentioned before: data frames, vectors and matrices. The reason to introduce this is, that several outputs in R are given as lists. E.g. if you run a linear model between x and y:

```
y<-c(1,2,3,4,5,6,7)
x<-c(12,15,10,16,22,27,32)
fit<-lm(y~x)      # this will solve a linear model (lm) where y is dependent from x
```

The object fit is a list with different elements. You will get the elements of fit using either

```
names(fit)      # or str(fit), but this is more complicate
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"    "fitted.values" "assign"        "qr"
 [8]  "df.residual"  "xlevels"       "call"          "terms"         "model"
```

So behind the object it, several different types of information were saved:

The coefficients of the linear model, just two values:

```
fit[[1]]                          #alternative type fit$ coefficients
```

```
Intercept)           x
 -0.6306695   0.2419006
```

A vector with the residuals, enfolding 7 values

```
fit[[2]]                          #alternative type fit$ residuals
```

Or the predicted values

```
fit[[5]]
```

It is obviously more clear, to safe all this information within of object as in several on. In the package rioja, the clear names of the coded data are given together with the relative abundance date within the same list, which is very comfortable:

```
library(rioja)
data(SWAP)
names(SWAP)
```

```
[1] "spec"  "pH"    "names"
```

```
head(SWAP$spec)
head(SWAP$names)
```

5. Installation of additionally R packages (vegan, analogue, rioja,…)

Open R and wait until the prompt appears (Figure A1-4). Be sure that you have access to the internet and then type behind the prompt:

install.packages("vegan")

and confirm the entry with the return key at your keyboard. A second window will be open for choosing a mirror (server), where you want to download the package (Figure A1-5). Choose one in the near of you and click ok. In the console a text will appear with information about the CRAN mirror you have chosen, and the place where the downloaded packages were saved. Now the package is saved on your computer, but the functions inside are not yet accessible for you, you have to load the package in R to get its function ability. Type

library(vegan)

It depends on the package, if now a text appears in the console. In some cases information's of the current version of the package is given, which other packages are required to get full functionality or a warning massage, if the version of R is to old in comparison with the package you have loaded.

To follow the whole manual you need the following packages: vegan, analogue, rioja, palaeoSig and paltran. So typ in the console

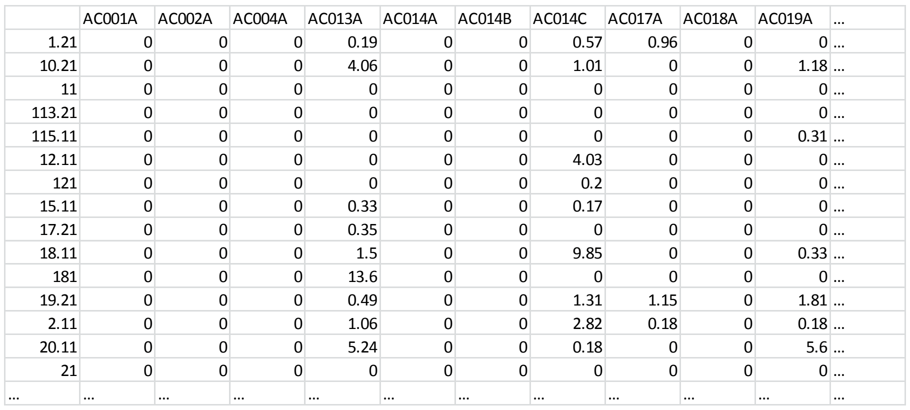
install.packages("analogue")
install.packages("rioja")
install.packages("palaeoSig")
install.packages("paltran")

Once a package is installed (saved on your computer), you do not need to install it again after restarting. But you have to load it in the R console every time. (It is possible to say R that special packages should always be loaded when opening R, for help see the CRAN home page)

APPENDIX B

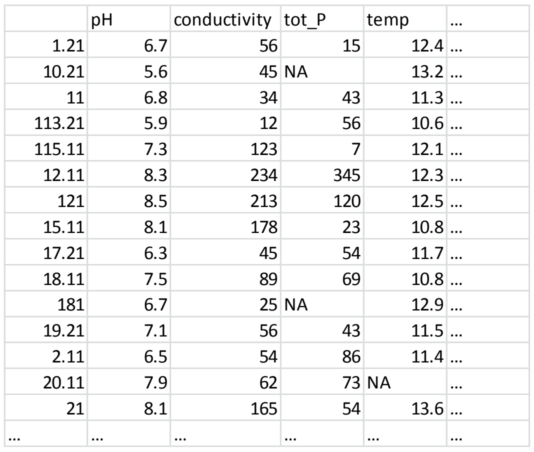
Import and export of data

Mostly scientists safe their data in data bases or table calculation programs like ACCESS, SQL-data bases, EXCEL, or SPSS. It is in many cases no problem to read the data in R, but some small pitfalls exist. For example, in R the sign for the decimal numbers is a '.', which is not automatically in all computer programs in all countries the default option. Different programs use different signs to separate columns and rows, to distinguish between column names and row names. The most usefull and powerfull function to read data in R is the *read.table()* function and its special forms like *read.csv()*, and *read.dbf()*, *read.spss()* (both package foreign). But also a function *read.xls* (package xlsReadWrite) can be found on the R home page (http://www.r-project.org/). In general two different ways of the organization of the species data are usefull, as displaced in figure B.1, B.2 and B.3. In Figure B.1 the maybe most typical data organization of 'biological' data is displayed, like it is used in most table calculation programs. For the packages we will use in this manual (e.g. vegan) it is important that every row displays on sample and every column on species. If a species does not occur in a certain sample, than it should get a 0 as value. Values that are not available (like single values of chemical or physical variables) are coded in R as 'NA' (not available) (Figure B.2)



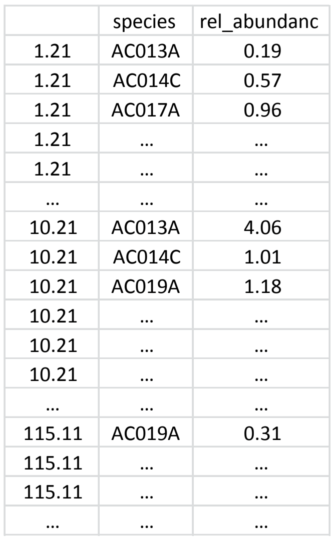
	AC001A	AC002A	AC004A	AC013A	AC014A	AC014B	AC014C	AC017A	AC018A	AC019A	...
1.21	0	0	0	0.19	0	0	0.57	0.96	0	0	...
10.21	0	0	0	4.06	0	0	1.01	0	0	1.18	...
11	0	0	0	0	0	0	0	0	0	0	...
113.21	0	0	0	0	0	0	0	0	0	0	...
115.11	0	0	0	0	0	0	0	0	0	0.31	...
12.11	0	0	0	0	0	0	4.03	0	0	0	...
121	0	0	0	0	0	0	0.2	0	0	0	...
15.11	0	0	0	0.33	0	0	0.17	0	0	0	...
17.21	0	0	0	0.35	0	0	0	0	0	0	...
18.11	0	0	0	1.5	0	0	9.85	0	0	0.33	...
181	0	0	0	13.6	0	0	0	0	0	0	...
19.21	0	0	0	0.49	0	0	1.31	1.15	0	1.81	...
2.11	0	0	0	1.06	0	0	2.82	0.18	0	0.18	...
20.11	0	0	0	5.24	0	0	0.18	0	0	5.6	...
21	0	0	0	0	0	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...

Figure B.1: Organization of the species data within a table calculation program like EXCEL. The first cell (A1) is empty; in the first column the sample names are given. AC001A, AC002A are codes for different species/taxa.



	pH	conductivity	tot_P	temp	...
1.21	6.7	56	15	12.4	...
10.21	5.6	45	NA	13.2	...
11	6.8	34	43	11.3	...
113.21	5.9	12	56	10.6	...
115.11	7.3	123	7	12.1	...
12.11	8.3	234	345	12.3	...
121	8.5	213	120	12.5	...
15.11	8.1	178	23	10.8	...
17.21	6.3	45	54	11.7	...
18.11	7.5	89	69	10.8	...
181	6.7	25	NA	12.9	...
19.21	7.1	56	43	11.5	...
2.11	6.5	54	86	11.4	...
20.11	7.9	62	73	NA	...
21	8.1	165	54	13.6	...
...	...	...	...	...	...

Figure B.2: Organization of the environmental data within a table calculation program like EXCEL. The first cell (A1) is empty; in the first column the sample names are given. Missing values are coded with a NA.



	species	rel_abundanc
1.21	AC013A	0.19
1.21	AC014C	0.57
1.21	AC017A	0.96
1.21	...	...
1.21	...	...
...	...	...
10.21	AC013A	4.06
10.21	AC014C	1.01
10.21	AC019A	1.18
10.21	...	...
10.21	...	...
10.21	...	...
...	...	...
115.11	AC019A	0.31
115.11	...	...
115.11	...	...
...	...	...

Figure B.2: Organization of the species data within a data base like SQL ore ACCESS. The first cell (A1) is empty; that is interpreted by R as in the first column the sample names are given. Missing values are coded with a NA.

If you work with EXCEL (and the data looks like in figure B.1) the fastest way to get the data in R is to copy all data into the clipboard (mark the data and then copy) and write in the R console:

my.data<- read.table("clipboard",header=T)

The first argument of the function read.table() enfoldet the place where the function should search for the data, here the clipboard. The second argument 'header=T' tells the function, that each column has a header. If you work with a computer where the decimal places are marked with a ',' than you have to tell this R:

my.data<- read.table("clipboard",header=T,dec=",")

To check weather everything has worked fine type in the R Console:

head(my.data)

or

fix(my.data)

(Working with EXCEL makes sure that the format of the columns is numeric, and that you have **no empty cells** or **blanks** in the cells. A typical error message is: "Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings, : line 2 did not have 4 elements". That means that in line 2 one cell of the original table is empty)

More strait forward is to produce a txt or csv file out of your EXCEL data. You can do this under "Save as" changing the file type "saving as type", e.g. "Text (Tab delimited)" or "CSV (Comma delimited)" under a path like "C:\MyDocuments\R\my.data.txt" or "C:\MyDocuments\R\my.data.csv". Than you can read the data in R with the following commands:

my.data<- read.table("C:/ MyDocuments/R/my.data.txt "*,header=T)*

or

my.data<- read.csv("C:/ MyDocuments/R/my.data.txt "*,header=T,sep=",")*

Notice: instead of '\' R uses the '/' in the path description (because R was primarily developed on UNIX)! The advantage of a csv file is, that blanks in the column names will be no source of trouble as these will be automatically replaced with a '.' by the function read.csv().

In different countries the separator between the columns is different when creating a csv file. With sep= ", " you can define your country specific sign (e.g. replace with "; ").

But why should we create a txt or csv when it is so easy to work with the clipboard? Mostly data analysis takes more time than one working day, different persons want to work with the same data, and sometimes an analysis should be reproduced after a long time, e.g if reviewers of a publication ask for more information or changes within a plot or analysis. In this cases it is important that you have saved all codes of your analysis and that you have not changed the original data. For saving the codes text files can be used, but more strait forward is to use editors like the script editor delivered with R (under File/New Script) or more comfortable the Tinn-R editor or the RWin-Editor. These editors are connected with R in that way, that just on mouse click will send the marked commands from the script editor to the R console, and analysis of several command lines will be reproduced in a very short time. So in general you need two files to safe and reproduce your analysis and plots, the script file with all successful operating commands and the related data file.

The package foreign offers functions to import data of the formats .dbf, .spss and many more, just typ ?read.dbf so get more information after loading the package in the console. After several years teaching students in basic R courses two rules of the thumb can be given:

- R is not a stupid program, most import errors are caused by errors in your data or typing errors (writing \ instead of /, missing blanks in the path, incomplete path,…)!
- If no error is reported by R, after importing the data, it means not that everything is fine. Take the time and have a look at the imported data! (use fix() or head())

If you open the data editor in R you can click with the mouse at the column names, a small window will be openend and you can see if the data are interpreted by R as numeric or as character. If you have for example a German computer (defining decimal places with a ‘,’) all numbers will interpreted as characters if you have forgotten to set the attribute dec= ", ".

A nice function is file.choose() that opens a window to choose the file that you like to read in:

```
my.data<- read.csv(file.choose(),header=T,sep=",")
```

To safe the data, the function write.table is the most use full (also available as write.csv, write.dbf,…).

The first attribute you have to give the function is the object you want to safe, the second is the place.

```
write.table(my.result,file= "C:/MyDocuments/result.txt ")
```

Plots can be saved by clicking the plot window with the right mouse button and choosing on of the offered options. But it is also possible to save a plot with a command in the Console:

```
savePlot(filename = " C:/MyDocuments/nice.picture", type="emf")
```

To see what kind of types are possible type ?savePlot.

Appendix C

The big advantages of a computer program is, that we do not have to repeat the same thing several times, we can teach the computer to do this for us.

Instead writing

```
x<-c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
```

we can write

```
x<-rep(1,19)
```

where rep is a function that do n times the same thing. Several small functions like this exist, but sometimes we have to create our own ones.

As said in chapter 2 it might be of interest to look for all taxa, how they are related to pH. As the SWAP training set enfolds more than 200 taxa, this will be a very boring work to do this for each one.

So we need a loop, a small program code, that will do this for us.

What do we want? We want to plot the species response to pH, so

```
library(rioja)

data(SWAP)

plot(SWAP$spec[,1]~SWAP$pH,main=SWAP$names[1,2])

# SWAP$spec[,1] = first taxon of this data set
# SWAP$names[1,2] = name of the first taxon of this data set
```

But we want to do this for a taxon i:

```
i=1

plot(SWAP$spec[,i]~SWAP$pH,main=SWAP$names[i,2])
```

where i runs from 1 to 277 (=277 taxa are in this training set, *dim(SWAP$spec)*). In R this will do the loop *for…* for us.

```
for (i in 1:277) plot(SWAP$spec[,i]~SWAP$pH,main=SWAP$names[i,2])
```

the 277 pictures will run so fast through the graph window, that we cannot see anything. So it might be god to safe the plots. The function savePlot will do that for us, but before we have to generate automatically for each picture a single file name, e.g. “SWAP_nameofthetaxa”

```
nam<-paste("C:/MyDocuments/SWAP_", SWAP$names[i,2])

nam                    #MyDocuments should be replaced by the path you prefere, i equals here 277,
                         from the last run of the loop.
```

Now we can combine both, the plot and the savePlot command. Notice, are there more than 1 commands in the for loop, they have to be put in {}:

```
for (i in 1:277)
        {
        nam<-paste("C:/MyDocuments/SWAP_", SWAP$names[i,1])
        plot(SWAP$spec[,i]~SWAP$pH,main=SWAP$names[i,2])
        savePlot(nam,type="emf")
        }
```

Some of the taxa will make problems here, as the taxa name enfold signs like [,],\ that are no signs allowed, when saving files. Replace *SWAP$names[i,2]) with SWAP$names[i,1]) and now problem will occur, nevertheless you will have now the taxa codes as file names.*

Literatur

Adler and Huebener (2012) Rare diatoms – setting a sense, Poster, IPS 2012, Glasgow.

Birks, H.J.B., 1998. Numerical tools in palaeolimnology e progress, potentialities, and problems. Journal of Paleolimnology 20, 307e332.

Birks, HH & Birks, HJB (2006) Multi-proxy studies in palaeolimnology. *Vegetation History and Archaeobotany*, 15, 235-51.

Birks, H.J.B. 2012. Overview of numerical methods in palaeolimnology. In: Birks, H.J.B., Lotter, A.F., Juggins, S. & Smol, J.P. (eds). Tracking Environmental Change Using Lake Sediments, Volume 5: Data Handling and Numerical Techniques. Springer, Dordrecht, pp. 19-92. 10.1007/978-94-007-2745-8_2.

Crawley, M.J., 2002. Statistical Computing e an Introduction to Data Analysis Using S-plus. Wiley & Sons, London.Faraway, J.J.(2006) Extending the Linear Model with R. Chapman & Hall, London.

Dalgaard P. (2011) Introductory Statistics with R, Springer, New York

Dreßler, M., Selig, U., Dörfler, W., Adler, S., Schubert, H., Hübener, T., 2006. Environmental changes and the migration period in Europe by the example of Lake Dudinghausen northern Germany. Quaternery Research 66, 25e37.

Telford and Birks (2011)

Hastie, T., Tibshirani, R., (1990). Generalized Additive Models. Chapman and Hall, London.

Hübener, T., Dreßler, M., Schwarz, A., Langner, K., Adler, S., 2008. Dynamic adjustment of training sets ('moving-window' reconstruction) by using transfer functions in paleolimnology e a new approach. Journal of Paleolimnology 40, 79e95.

Juggins, S., (2012). rioja: Analysis of Quaternary Science Data, R package version 0.7-3. (http://cran.r-project.org/package=rioja).

McCullagh, P., Nelder, J.A., (1989). Generalized Linear Models. Chapman and Hall, London. pp. 511.

Oksanan J (2011)

Oksanen J, Blanchet F.G., Kindt R. , Legendre P., Minchin P.R., O'Hara R.B., Simpson G.L., Solymos P, Stevens M.H.H., Wagner H.(2012). vegan: Community Ecology Package. R package version 2.0-3. http://CRAN.R-project.org/package=vegan

R Development Core Team (2012). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Sarkar D. (2008) Lattice: Multivariate Data Visualization with R. Springer, New York. ISBN 978-0-387-75968-5

Simpson, G.L. and Oksanen, J. (2011). analogue: Analogue matching and Modern Analogue Technique transfer function models. (R package version 0.8-0). (http://cran.r-project.org/package=analogue).

van Buuren S. , Groothuis-Oudshoorn K. (2011). mice: Multivariate Imputation by Chained Equations in R. Journal of Statistical Software, 45(3), 1-67. URL http://www.jstatsoft.org/v45/i03/.

Wickham H. (2009). ggplot2: elegant graphics for data analysis. Springer New York.
Zuur, A.F., Ieno, E.N., Walker, N.J., Saveliev, A.A., Smith, G.M.,(2009). Mixed Effects Models and Extensions in Ecology with R. Springer, New York. pp. 574.

Wood, S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC.