Multivariate data analysis in R

A collection of R functions for multivariate data analysis

Michail Tsagris

Department of Economics, University of Crete, Rethymnon

mtsagris@uoc.gr

Version 9.8

Nottingham, Abu Halifa, Athens, Herakleion and Rethymnon

9 June 2022

Contents

Prologue

The motivation for the writing of these functions was to offer some form of an alternative R-package with simple (and easy to modify) functions. The resume of the theory is also provided, so that somebody does not have to read a whole chapter or a paper to understand what is the key message there. Most of the functions are not available in any R package, but R is a very popular statistical language and packages are uploaded very frequently. So maybe some of the functions exist already in other packages, or even are built-in functions or they have been added in packages after I wrote them here. Some functions have been tested using example data sets found at the references. The others were tested numerically, for example, the hypothesis testing procedures, do they estimate correctly the type I error?

As I update the versions, by adding new stuff, I also check for mistakes and correct them. So I would suggest you keep the newest versions. However, mistakes will still be around I am afraid, and they are, so corrections or any comments are most welcome and of course required. Note also, that I have added a log of changes so that anybody can track any changes from version to version. Also within one single version sometimes I upload updates with corrections. The log can be found at the end of the document, just before the references. I know that even this version needs a bit polishing and in some cases more explanation of the algorithms. These are being done, even slowly.

This document has somehow changed direction and now more functions for univariate data are included. For example, ridge regression, kernel regression and principal components regression. The reason for which they are included here is their use of multivariate techniques, such as PCA.

I decided to start using R packages inside my functions. The reason for this is that they give me more flexibility and ability to add more. For instance, the package *doParallel* allows me to do parallel calculations and thus speed up many codes which is also beneficiary for the readers.

Another slight change of direction is towards education. What I mean, is that I have added functions which make no use for the practitioners or the researchers, but they help in understanding some stuff. Students will find them very useful in gaining insight in some things. For example, the relationship between Hotelling's T^2 and James test or the bootstrap correlation coefficient. In the second case, there are two functions, one non vectorised and one vectorised. This serves two goals. The first one is to get an idea of how you can vectorise (if possible) your codes (educational reason and also a helpful tip) and the second is time. In small to moderate sample sizes, the non vectorised is slower, but in big samples, the non vectorised is faster(?!). For the tuning of the bandwidth in the kernel density estimate via maximization of the pseudo log-likelihood I present two functions. One using parallel computation and another one without. If you have small datasets the second version is faster. But, if you have big data and many cores, then maybe the first function is faster (I

have not tried it and do not know).

Feel free to contribute your own functions and you will be credited of course. If you want the functions in a .txt or .R format please send me an e-mail. If you cannot download this document for some reason, send me an e-mail as well.

I would like to express my gratitude to Andrew Rattray (postgraduate student at the university of Nottingham during 2012-2013) for pointing out a mistake in the Box's M test code.

A very good (in my opinion) manual with R functions is written by Paul Hewson. Don't forget to see R Inferno.

Professor Andy Wood and Dr Simon Preston from the university of Nottingham are highly appreciated for being my supervisors during my PhD (compositional data analysis) and post-doc (directional data analysis).

Georgios Pappas from the university of Nottingham helped me construct the contour plots of the von Mises-Fisher and the Kent distribution.

Christopher Fallaize and Theo Kypraios from the university of Nottingham have provided a function for simulating from the Bingham distribution using rejection sampling. So any questions regarding this function should be addressed to them.

Kwang-Rae Kim from the university of Nottingham helped me create a front end with Matlab.

Marco Maier from the Institute for Statistics and Mathematics in Vienna suggested me to change the format of my R functions and told me about the R package *formatR*. He even sent me the command with an example, so he is greatly appreciated.

Giorgos Borboudakis from the Foundation of Research and Technology (FORTH) and member of the MXM group pointed out to me a not so clear message in the algorithm of generating random values from the von Mises-Fisher distribution.

Panagiotis (pronounced Panayiotis) Tzirakis (master student at the computer science department in Herakleion) showed me how to perform parallel computing in R and he is greatly acknowledged and appreciated not only from me but from all the readers of this document. He also helped me with the vectorization of some contour plot functions.

Giorgos Athineou (master student at the computer science department in Herakleion) has taught me a few tips in R and he is greatly appreciated.

Professor John Kent from the university of Leeds is acknowledged for clarifying one thing with the β (ovalness) parameter in his distribution.

Professor Changliang Zou from the Nankai University is greatly acknowledged for sending me his function for outlier identification for high-dimensional data (function *rmdp*).

Manos Papadakis (undergraduate in the computer science department in Herakleion) pointed out the need to avoid matrix multiplications. That takes time. Indeed, you can check the functions *spat.med* and *spat.med_old* to see for yourselves. He also gave inspiration for many more functions to be implemented efficiently.

Kleio Lakiotaki (post-doc at the department of computer science in Herakleion) showed me the potentials of the function *outer* and the amazing speed of *prcomp*. The second command should be used for principal component analysis when you have matrices with more than 100 variables. The more variables the bigger the difference from doing *eigen(cova(x))*.

1 Some things about R

1.1 A few tips for faster implementations

I will show a few tips for faster computations. If you want more tips have a look at this unpublished paper (Tsagris and Papadakis, 2018). In small sample and or small dimensionalities you may see small differences, but in bigger datasets the differences arise. You might observe a time difference of only 5 seconds in the whole process. I saw a difference from 40 down to 12 seconds for example. That was very successful. In another case from 40 seconds to 22. Still successful. But not always this kind of differences. Some times, one tip gives you 1 second and then another tip 1 second and so on until you save 5 seconds. If you have 1000 simulations, then you save 5000 seconds. Even small decreases matter. Perhaps for someone who needs a simple car, a very expensive car or a jeep type might not be of such use, especially if he or she does not go to the village. But for the user who needs a jeep, every computational power, every second he/she can gain matters.

The *nlm* is much faster than *optim* for optimization purposes but *optim* is more reliable and robust. Try in your examples or cases, if they give the same results and choose. Or use first *nlm* followed by *optim*. The exponential term in the multivariate normal can be either calculated using matrices or simply with the command *mahalanobis*. If you have many observations and many dimensions and or many groups, this can save you a looot of time (I have seen this).

```
x <- matrix( rnorm(1000 * 20), ncol = 20 )
m <- colMeans(x)
n <- dim(x)[1]
p <- dim(x)[2]
s <- cov(x)
## slow version
a1 = diag( (x - rep(m, rep(n, p)) ) %*% solve(s) %*% t(x - rep(m, rep(n, p)) ) )
a2 = diag( t( t(x)- m ) %*% solve(s) %*% t(x)- m )  ### slow
a3 = mahalanobis(x, m, s) ## much much faster
```

Suppose you want to calculate the product of an $n \times p$ matrix $\mathbf{X}^T\mathbf{X}$ for example. The command *crossprod(X)* will do the job faster than if you do the matrix multiplication.

Next, you want to center some data, you can try with *apply* for example

```
cent <- function(x) x - mean(x)
apply(data, 2, cent)
```

or using this

```
m <- colMeans(data)
```

```
n <- dim(data)[1]  ;  p <- dim(data)[2]
y <- t( t(data) - m )
data <- scale(data, center = TRUE, scale = FALSE)
data <- sweep(data, 2L, m)
data <- data - rep( m, rep(n, p) ) ## looks like the fastest
```

See also Gaston Sanchez's webpage for a comparison of these. Or you can compare the times yourself.

If you want to extract the mean vector of each group you can use a loop (*for* function) or

```
aggregate(x, by = list(ina), mean)
```

where *ina* is a numerical variable indicating the group. A faster alternative is the built-in command *rowsum*

```
x1 <- rowsum(x, ina)
id <- as.vector( table(ina) )
x1 / id
```

I found this suggestion here suggested by Gabor Grothendieck.

For the covariances the command by could be used but the matrices are stored in a list and then you need *simplify2array* to convert the list to an array in order to calculate for example the determinant of each matrix. The for loop is faster, at least that's what I saw in my trials.

Vectorization is a big thing. It can save tremendous amount of time even in the small datasets. Try to avoid *for* loops by using matrix multiplications. For example, instead of

```
for (i in 1:n) y[i] <- x[i]^2
```

you can use

```
y <- x^2
```

Of course, this is a very easy example, but you see my point. This one requires a lot of thinking and is not always applicable. But, if it can be done, things can be super faster. See the bootstrap correlation coefficient for example, where I have two functions, *boot.correl* with a *for* loop and *bootcor*, which is vectorised.

Use *apply* or *aggregate* we saw before whenever possible. But, use *colMeans* or *colSums* instead of *apply(x, 2, mean)* to get the mean vector of a sample because it's faster. For the median though, you have to use *apply(x, 2, median)* instead of a *for* going to every column of the matrix. Imagine you have a dependent variable y and many independent variables x_is and you want to perform regression of y on every x_i and obtain the betas for every simple linear regression. You can do a *for* loop or you can do this

```
funa <- function(x) coef( lm(y~x) )
apply(x, 2, funa)
```

What if you have an array with matrices and want to calculate the sum or the mean of all the matrices? The obvious answer is to use *apply(x, 1:2, mean)*. R works rather in a cloumn-wise fashion than in a row-wise fashion. Instead of the apply you can try *t(colSums(aperm(x)))* and *t(colMeans(aperm(x)))* for the *sum* and *mean* operations respectively.

```
x <- array( dim = c(1000,10,10) )
for (i in 1:10) x[, , i] = matrix( rnorm(1000* 10), ncol = 10 )
a1 <- t( colMeans( aperm(x) ) )
a2 <- apply(x, 1:2, mean)
```

If you want the matrix of distances, with the zeros in the diagonal and the upper triangular do not use the command *as.matrix(dist(x))* but use *dist(x, diag = TRUE, upper = TRUE)*. Also, the package *fields* (Nychka et al., 2015) has a function called *rdist* which is faster than the built-in *dist* in R. Suppose you want the Euclidean distance of a single vector from many others (say thousands for example). The inefficient way is to calculate the distance matrix of all points and take the row which corresponds to your vector. The efficient way is to use the Mahalanobis distance with the identity ad the covariance matrix

```
x <- MASS::mvrnorm(1, numeric(50), diag( rexp(50,0.4)) ) ##  vector in R^50
y <- MASS::mvrnorm(1000, numeric(50), diag( rexp(50,0.4)) ) ##  vector in R^50.
a <- dist( rbind(x, y) )
a[1, ]  ## inefficient way
Ip <- diag(50)
a <- mahalanobis( y, center = x, cov = Ip, inverted = TRUE ) ## efficient
```

Can we make the above faster? The answer is yes, by avoiding the matrix multiplications. You see the matrix multiplications are performed in C++ using a *for* loop. Even though it's fast, it's not as fast as you think, FORTRAN for example is much much faster.

```
z <- y - x
a <- sqrt( colSums(z^2) )
```

Try both ways and see. Check the spatial median Section where I have kept two functions, one with the Mahalanobis and one with the above trick. Put large data and check the time required by either function; you will be amazed.

Speaking of Mahalanobis distance, check my function *mahala* which is twice as fast as the built-in function *mahalanobis*.

As for the covariance and correlation matrices I have found a nice way described by Andrey A. Shabalin and is presented in Section 6.1. I have tested its time with other packages

and functions, but it still does better. The difference with the standard R functions becomes more apparent as you move to higher than 1,000 dimensions.

The function *mean* is slower than *sum(x)/length(x)*. If you type *sum* you will see it is a *.Primitive* function, whereas *crossprod* and *colMeans* are both *.Internal* ones. By the way the *colMeans* is a really really fast function. My 2 points are a) create your own functions, you will be surprised to see that you may do faster than R's built-in functions (it doesn't always work that way) and b) use *.Internal* functions whenever possible. An example of the point is the *var* function. Create your own and you will see it is faster. An example of the second point is the function *colVars* which uses *colMeans*.

Search for functions that take less time. For example, the command *lm.fit(x,y)* is a wrapper for *lm(y x)*, which means that the first one is used by the second one to give you the nice output. But, if you need only the coefficients, for example, then use the first one. The syntax is a bit different, the x must be the design matrix, but the speed is very different especially in the big cases. Finally, the multinomial regression is offered in the package VGAM but it also offered in the package nnet. The implementation in the second package is much faster. The same is true for the implementation of the ordinal logistic regression in the VGAM and in the ordinal. The latter package does it much faster. Many fast functions can also be found in the package Rfast (Papadakis et al., 2019).

If you have a function for which some parameters have to be positive, do not use constrained optimization, but instead put an exponential inside the function. The parameter can take any values in the whole of R but inside the function its exponentiated form is used. In the end, simply take the exponential of the returned value. As for its variance use the Δ method (Casella and Berger, 2002). If you did not understand this check the MLE of the inverted Dirichlet distribution and the Dirichlet regression (ϕ parameter).

Speaking of Dirichlet distribution, I have two functions for estimating the parameters of this distributions. One which uses *nlm* and *optim* and another one which uses the Newton-Raphson algorithm. I did some simulations and I saw the Newton-Raphson can be 10 times faster. The same is true for the circular regression (*spml.reg*) where I use the E-M algorithm. Switching to E-M or the Newton-Raphson and not relying on the *nlm* command can save you a looot of time. If you want to write a code and you have the description of the E-M or the Newton-Raphson algorithm available, because somebody did it in a paper for example, or you can derive it yourself, then do it.

I found this article (pages 18-20) by Douglas Bates very useful and in fact I have taken some tips from there.

```
solve(X) %*% Y     #### classical
solve(X, Y)       #### much more efficient
t(X) %*% Y         #### classical
crossprod(X, Y)   ### more efficient
X %*% t(Y)   #### classical
```

```
tcrossprod(X, Y)   #### more efficient
t(X) %*% X    #### classical
crossprod(X)    #### more efficient
```

Sticking with *solve(X)*, if you want to only invert a matrix then you should use *chol2inv(chol(X))* as it is faster.

Douglas Bates mentions in the same article, that calculating $\mathbf{X}^T\mathbf{Y}$ in R as $t(X)\% * \%Y$ instead of *crossprod(X,Y)* causes X to be transposed twice; once in the calculation of $t(X)$ and a second time in the inner loop of the matrix product. The *crossprod* function does not do any transposition of matrices.

The trace of the square of a matrix tr $\left(\mathbf{A}^2\right)$ can be evaluated either via

```
sum( diag( crossprod(A) ) )
```

or faster via

```
sum(A * A)
sum(A^2)
```

If you want to calculate the following trace involving a matrix multiplication tr $\left(\mathbf{X}^T\mathbf{Y}\right)$ you can do either

```
sum( diag( crossprod(X, Y) ) )  ## just like before
```

or faster

```
sum(X * Y)  ## faster, like before
```

Moving in the same spirit, you want the diagonal of the crossproduct of two matrices, such as

```
diag( tcrossprod(X, Y) )  ## for example
rowSums(X * Y)  ## this is faster
```

Suppose you have two matrices **A**, **B** and a vector **x** and want to find **ABx** (the dimensions must match of course).

```
A %*% B %*% x  ## inefficient way
A %*% (B %*% x) ## efficient way
```

In the first case you have a matrix by matrix by vector calculations. In the second case you have a matrix by vector which is a vector and then a matrix by a vector. You do less calculations. The final tip is to avoid unnecessary and/or extra calculations and try to avoid doing calculations more than once.

As for the eigen-value decomposition, there are two ways to do the multiplication

```
s = cov(iris[, 1:4])
eig = eigen(s, symmetric = TRUE)
vec = eig$vectors
lam= eig$values
vec %*% diag(lam) %*%t(vec)
vec %*% ( t(vec) * lam )  ## faster way
```

If you have an iterative algorithm, such as Newton-Raphson, E-M or fixed points and you stop when the vector of parameters does not change any further, do not use *rbind*, *cbind* or *c()*. Store only two values, *vec.old* and *vec.new*. What I mean, is do not do for example

```
u[i, ] <- u[i - 1, ] + W%*%B  ## not efficient
u.new <- u.old + W%*%B
```

So, every time keep two vectors only, not the whole sequence of vectors. The same is true for the log-likelihood or whatever you have. Unless you want a trace of how things change, then ok, keep everything. Otherwise, apart from begin faster it also helps the computer run faster since less memory is used. See the functions *spat.med* and *spat.med_old* to get an idea. This tip is due to Manos Papadakis (undergraduate student of the computer science department in Herakleion).

Avoid unnecessary calculations. In a discriminant analysis setting for example there is no need to calculate constant parts, such as $\log(2\pi)$, every time for each group and every iteration. This only adds time and takes memory and does not affect the algorithm or the result.

When working with arrays it is more efficient to have them transposed. For example, if you have K covariance matrices of dimension $p \times p$, you would create an array of dimensions $c(p, p, K)$. Make its dimensions $c(K, p, p)$. If you want for example to divide each matrix with a different scalar (number) in the first case you will have to use a for loop, whereas in the transposed case you just divide the array by the vector of the numbers you have.

1.2 Parallel computing

Before I begin with the functions, I would like to say a few words about the parallel computing in R. If you have a machine that has more than 1 cores, then you can put them all to work simultaneously and speed up the process a lot. If you have tricks to speed up your code that is also beneficiary. I have started taking into account tricks to speed up my code as I have mentioned before.

Panagiotis Tzirakis (master student at the department of computer science of the university of Crete in Herakleion) has showed me how to perform parallel computing in R. He is

greatly acknowledged not only by me, but also by the readers of these notes, since they will save time as well.

The idea behind is to use a library that allows parallel computing. Panayiotis suggested me the doParallel package (which uses the *foreach* package) and that is what I will use from now on. Below are some instructions on how to use the package in order to perform parallel computing. In addition, I have included the parallel computing as an option in some functions and in some others I have created another function for this purpose. So, if you do not understand the notes below, you can always see the functions throughout this text.

```
 ## requires(doParallel)
 Create a set of copies of R running in parallel and communicating
 ## over sockets.
 cl <- makePSOCKcluster(nc) ## nc is the number of cluster you
 ## want to use
 registerDoParallel(cl) ## register the parallel backend with the
 ## foreach package.
 ## Now suppose you want to run R simulations, could be
 ## R=1000 for example
 ## Divide the number of simulations to smaller equally
 ## divided chunks.
 ## Each chunk for a core.
 ba <- round( rep(R/nc, nc) )
 ## Then each core will receive a chunk of simulations
 ww <- foreach(j = 1:nc,.combine = rbind) %dopar% {
 ## see the .combine = rbind. This will put the results in a matrix.
 ## Every results will be saved in a row.
 ## So if you have matrices, make them vectors. If you have lists
 ## you want to return,
 ## you have to think about it.
 a <- test(arguments, R = ba[j], arguments)$results
 ## Instead of running your function "test" with R simulations
 ## you run it with R/nc simulations.
 ## So a stores the result of every chunk of simulations.
 return(a)
  }
stopCluster(cl) ## stop the cluster of the connections.
```

To see your outcome all you have to press is *ww* and you will see something like this

```
result.1 .....
```

```
result.2 .....
result.3 .....
result.4 .....
```

So, the object *ww* contains the results you want to see in a matrix form. If every time you want a number, the *ww* will be a matrix with 1 column. We will see more cases later on. Note that f you choose to use parallel computing for something simple, multicore analysis might take the same or a bit more time than single core analysis only because it requires a couple of seconds to set up the cluster of the cores. In addition, you might use 4 cores, yet the time is half than with 1 core. This could be because not all 4 cores work at 100% of their abilities. Of course you can always experiment with these things and see.

1.3 Duration of a processes

If you want to see how much time your process or computation or simulation needs, you can do the following in R

```
ti <- proc.time()
## put your function here
ti <- proc.time() - ti
## ti gives you 3 numbers (all in seconds) like the ones below
user  system elapsed
0.18    0.07    3.35
```

The elapsed is what you want. Alternatively you can download the package microbenchmark which allows you to compare two or more functions measuring the time even in nanoseconds.

1.4 Libraries required

The packages required for some functions are listed below

- library(doParallel) for parallel computing.
- library(MASS) which is already in R.
- library(fields) for graphics.
- library(quantreg) for the spatial median regression.
- library(sn) for the skew normal distribution.
- library(R.matlab) connection between R and Matlab for the normalizing constant of the Fisher-Bingham distribution.

- library(nnet) for the multinomial regression.
- library(Rfast) for many functions.
- library(robust) for the forward search and in general for a robust covariance matrix using MCD.
- library(RcppZiggurat) for generating random values from a standard normal distribution.

To install a package using internet just type in R *install.packages("package name")*. A window tab will appear asking you to choose country (cran mirror). You select one and then wait a few seconds. If the package fails to be installed, try another country. Don't forget to load it then before trying to use it.

2 Hypothesis testing for mean vectors

In this section we shall see many approaches for hypotheses regarding one sample and two sample mean vectors.

2.1 Hotelling's one-sample T^2 test

We begin with the hypothesis test that a mean vector is equal to some specified vector $H_0 : \boldsymbol{\mu} = \boldsymbol{\mu}_0$. We assume that $\boldsymbol{\Sigma}$ is unknown. The first approach to this hypothesis test is parametrically, using the Hotelling's T^2 test Mardia et al., 1979, pg. 125-126. The test statistic is given by

$$T^2 = \frac{(n-p)\, n}{(n-1)\, p} \left(\bar{\mathbf{X}} - \boldsymbol{\mu}\right)^T \mathbf{S}^{-1} \left(\bar{\mathbf{X}} - \boldsymbol{\mu}\right) \tag{2.1}$$

Under the null hypothesis, the above test statistic follows the $F_{p,n-p}$ distribution. The bootstrap version of the one-sample multivariate generalization of the simple t-test is also included in the function. An extra argument (R) indicates whether bootstrap calibration should be used or not. If $R = 1$, then the asymptotic theory applies, if $R > 1$, then the bootstrap p-value will be applied and the number of re-samples is equal to (B).

```
hotel1T2 <- function(x, M, a = 0.05, R = 999, graph = FALSE) {
  ## x is the data set
  ## a is the level of significance set by default to 0.05
  ## M is the hypothesised mean
  ## R is the number of bootstrap replicates set by default to 999
  ## if R=1 no bootstrap will be implemented
  ## Bootstrap is used for the p-value
  m <- Rfast::colmeans(x)  ## sample mean vector
  s <- cov(x)  ## sample covariance matrix
  n <- dim(x)[1]  ## sample size
  p <- dim(x)[2]  ## dimensionality of the data
  dm <- m - M
  test <- as.vector( n * (n - p) / (n - 1) / p * dm %*% solve(s, dm) )
  ## test is the test statistic
  if (R == 1) {
    pvalue <- pf(test, p, n - p, lower.tail = FALSE)  ## p-value
    crit <- qf(1 - a, p, n - p)  ## critival value of the F distribution
    info <- c(test, pvalue, crit, p, n - p)
    names(info) <- c("test", "p-value", "critical", "numer df", "denom df")
```

```
    result <- list(m = m, info = info)
  }
  if (R > 1) {
    ## bootstrap calibration
    tb <- numeric(R)
    mm <-  - m + M
    y <- x + rep( mm, rep(n, p) ) ## brings the data
    ## under the null hypothesis, i.e. mean vector equal to M
    for (i in 1:R) {
      b <- sample(n, n, replace = TRUE, prob = NULL)
      yb <- y[b, ]
      sb <- cov(yb)
      mb <- Rfast::colmeans(yb)
      dmb <- mb - M
      tb[i] <- dmb %*% solve(sb, dmb)
    }
    tb <- n * (n - p) / (n - 1) / p * tb
    pvalue <- ( sum(tb > test) + 1 )/(R + 1)  ## bootstrap p-value
    if ( graph ) {
      hist(tb, xlab = "Bootstrapped test statistic", main = " ")
      abline(v = test, lty = 2, lwd = 2)  ## The dotted vertical line
      ## is the test statistic value
    }
    result <- list(m = m, pvalue = pvalue)
  }
  result
}
```

2.2 Hotelling's two-sample T^2 test

The fist case scenario is when we assume equality of the two covariance matrices. This is called the two-sample Hotelling's T^2 test (Mardia et al., 1979, pg. 1391-40 and Everitt, 2005, pg. 139). The test statistic is defined as

$$T^2 = \frac{n_1 n_2}{n_1 + n_2} \left(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2 \right)^T \mathbf{S}^{-1} \left(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2 \right),$$

where $\mathbf{S}$ is the pooled covariance matrix calculated under the assumption of equal covariance matrices

$$\mathbf{S} = \frac{(n_1 - 1)\,\mathbf{S}_1 + (n_2 - 1)\,\mathbf{S}_2}{n_1 + n_2 - 2}.$$

Under H_0 the statistic F given by

$$F = \frac{(n_1 + n_2 - p - 1)\,T^2}{(n_1 + n_2 - 2)\,p}$$

follows the F distribution with p and $n_1 + n_2 - p - 1$ degrees of freedom. Similar to the one-sample test, an extra argument (B) indicates whether bootstrap calibration should be used or not. If $B = 1$, then the asymptotic theory applies, if $B > 1$, then the bootstrap p-value will be applied and the number of re-samples is equal to (B). The estimate of the common mean used in the bootstrap to transform the data under the null hypothesis the mean vector of the combined sample, of all the observations.

The built-in command *manova* does the same thing exactly. Try it, the asymptotic F test is what you have to see. In addition, this command allows for more mean vector hypothesis testing for more than two groups. I noticed this command after I had written my function and nevertheless as I mention in the introduction this document has an educational character as well.

```
hotel2T2 <- function(x1, x2, a = 0.05, R = 999, graph = FALSE) {
  ## x1 and x2 are the two multivariate samples a is the level
  ## of significance, which by default is set to to 0.05
  ## R is the number of bootstrap replicates
  ## set by default to 999
  ## if R=1 no bootstrap will be implemented
  ## Bootstrap is used for the p-value
  p <- dim(x1)[2]  ## dimensionality of the data
  n1 <- dim(x1)[1]  ## size of the first sample
  n2 <- dim(x2)[1]  ## size of the second sample
  n <- n1 + n2  ## total sample size
  xbar1 <- Rfast::colmeans(x1)  ## sample mean vector of the first sample
  xbar2 <- Rfast::colmeans(x2)  ## sample mean vector of the second sample
  dbar <- xbar2 - xbar1  ## difference of the two mean vectors
  mesoi <- rbind(xbar1, xbar2)
  rownames(mesoi) <- c("Sample 1", "Sample 2")
  if ( is.null(colnames(x1)) ) {
    colnames(mesoi) <- colnames(mesoi) <- paste("X", 1:p, sep = "")
  } else  colnames(mesoi) <- colnames(x1)
```

```
v <- ( (n1 - 1) * cov(x1) + (n2 - 1) * cov(x2) )/(n - 2)
## v is the pooled covariance matrix
t2 <- n1 * n2 * (dbar %*% solve(v, dbar) ) /n
test <- as.vector( (n - p - 1) * t2 / (n - 2) / p )  ## test statistic

if (R <= 1) {
  crit <- qf(1 - a, p, n - p - 1)  ## critical value of the F distribution
  pvalue <- pf(test, p, n - p - 1, lower.tail = FALSE)  ## p-value
  info <- c(test, pvalue, crit, p, n - p - 1)
  names(info) <- c("test", "p-value", "critical", "numer df", "denom df")
  result <- list(mesoi = mesoi, info = info)
}
if (R > 1) {
  ## bootstrap calibration
  mc <- Rfast::colmeans( rbind(x1, x2) )  ## combined sample mean vector
  ## the next two rows bring the mean vectors of the two sample equal
  ## to the combined mean and thus equal under the null hypothesis
  mc1 <-  - xbar1 + mc
  mc2 <-  - xbar2 + mc
  y1 <- x1 + rep( mc1, rep(n1, p) )
  y2 <- x2 + rep( mc2, rep(n2, p) )
  tb <- numeric(R)
  for (i in 1:R) {
    b1 <- sample(1:n1, n1, replace = TRUE)
    b2 <- sample(1:n2, n2, replace = TRUE)
    yb1 <- y1[b1, ]    ;   yb2 <- y2[b2, ]
    db <- Rfast::colmeans(yb1) - Rfast::colmeans(yb2)  ## means difference
    vb <- ( (n1 - 1) * cov(yb1) + (n2 - 1) * cov(y2) ) / (n - 2)
    ## vb is the pooled covariance matrix
    tb[i] <- n1 * n2 * ( db %*% solve(vb, db) ) / n
  }
  tb <- (n - p - 1) * tb / (n - 2) / p
  pvalue <- ( sum(tb > test) + 1 )/(R + 1)
  if ( graph ) {
    hist(tb, xlab = "Bootstrapped test statistic", main = " ")
    abline(v = test, lty = 2, lwd = 2)  ## The line is the test statistic
  }
  result <- list(mesoi = mesoi, pvalue = pvalue)
}
```

```
  result
}
```

2.3 MANOVA assuming equal covariance matrices

The extension of the Hotelling's T^2 test for two samples to three or more samples is called MANOVA (Multivariate Analysis of Variance). There is a built-in function as I mentioned before, but it's not very easy to use. Therefore I programmed my own function for this purpose.

Suppose we have n observations in total from g distinct p-variate normal populations with means $\boldsymbol{\mu}_i$ and covariance matrices $\boldsymbol{\Sigma}_i$ and we want to test whether $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = \ldots = \boldsymbol{\mu}_g$, when assuming that $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \ldots = \boldsymbol{\Sigma}_g$ holds true. This is the multivariate generalization of the (univariate) analysis of variance, and thus the M letter in the front. We need the within and the between sum of squares only

$$\mathbf{W} = \sum_{i=1}^{g} (n_i - 1)\, \boldsymbol{\Sigma}_i \text{ and } \mathbf{B} = \sum_{i=1}^{g} n_i \left(\bar{\mathbf{X}}_i - \bar{\mathbf{X}}\right) \left(\bar{\mathbf{X}}_i - \bar{\mathbf{X}}\right)^T$$

respectively, where $\bar{\mathbf{X}}$ is the mean vector of all samples together.

Wilks's Λ is defined as (Johnson and Wichern, 2007)

$$\Lambda = \frac{|\mathbf{W}|}{|\mathbf{W} + \mathbf{B}|} = \frac{1}{|\mathbf{I}_p + \mathbf{W}^{-1}\mathbf{B}|}.$$

Johnson and Wichern, 2007, pg. 303 mention that

$$\begin{array}{ll} \frac{n-p-1}{p} \frac{1-\Lambda}{\Lambda} \sim F_{p,n-p-1} & \text{if } g = 2 \\ \frac{n-p-2}{p} \frac{1-\sqrt{\Lambda}}{\sqrt{\Lambda}} \sim F_{2p,2(n-p-2)} & \text{if } g = 3. \end{array}$$

As for the other cases I found a paper by Todorov and Filzmoser (2010) who mention that a popular approximation to Wilk's Λ is Bartlett's χ^2 approximation given by

$$-\left[n - 1 - (p + g)/2\right] \log \Lambda \sim \chi^2_{p(g-1)}.$$

In the function below a message will appear mentioning which approximation has been used, the F or the χ^2.

```
maov <- function(x, ina) {
  ## x is a matrix with the data
  ## ina is a grouping  variable indicating the groups
  ina <- as.numeric(ina)
```

```
  ni <- tabulate(ina)  ## group sample sizes
  n <- dim(x)[1]  ## total sample size
  g <- max(ina)  ## number of groups
  p <- dim(x)[2]  ## dimensionality of the data
  m <- rowsum(x, ina) / ni
  me <- Rfast::colmeans(x)  ## total mean vector
  y <- sqrt(ni) * (m - rep(me, rep(g, p)) )
  B <- crossprod(y)
  Tot <- cov(x) * (n - 1)
  lam <- det(Tot - B) / det(Tot)
  if (g == 2 ) {
    stat <- (n - p - 1 ) / p * (1 - lam)/lam
    pvalue <- pf( stat, p, n - p - 1, lower.tail = FALSE )
    note <- paste("F approximation has been used")
  } else if (g == 3) {
    stat <- (n - p - 2 )/p * (1 - sqrt(lam)) / sqrt(lam)
    pvalue <- pf( stat, 2 * p, 2 * (n - p - 2), lower.tail = FALSE )
    note <- paste("F approximation has been used")
  } else {
    stat <-  -( n - 1 - (p + g)/2 ) * log(lam)
    pvalue <- pchisq( stat, p * (g - 1), lower.tail = FALSE )
    note <- paste("Chi-square approximation has been used")
  }
  result <- c(stat, pvalue)
  names(result) <- c('stat', 'p-value')
  list(note = note, result = result)
}
```

2.4 Two two-sample tests without assuming equality of the covariance matrices

In his section we will show the modified version of the two-sample T^2 test in the case where the two covariances matrices cannot be assumed to be equal.

James (1954) proposed a test for linear hypotheses of the population means when the variances (or the covariance matrices) are not known. Its form for two p-dimensional samples is:

$$T_u^2 = \left(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2\right)^T \tilde{\mathbf{S}}^{-1} \left(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2\right), \text{ with } \tilde{\mathbf{S}} = \tilde{\mathbf{S}_1} + \tilde{\mathbf{S}_2} = \frac{\mathbf{S}_1}{n_1} + \frac{\mathbf{S}_2}{n_2}. \tag{2.2}$$

James (1954) suggested that the test statistic is compared with $2h(\alpha)$, a corrected χ^2 distribution whose form is

$$2h(\alpha) = \chi^2\left(A + B\chi^2\right),$$

where

$$\begin{aligned} A &= 1 + \frac{1}{2p}\sum_{i=1}^{2}\frac{\left(tr\tilde{\mathbf{S}}^{-1}\tilde{\mathbf{S}}_i\right)^2}{n_i - 1} \text{ and} \\ B &= \frac{1}{p(p+2)}\left[\sum_{i=1}^{2}\frac{tr\left(\tilde{\mathbf{S}}^{-1}\tilde{\mathbf{S}}_i\right)^2}{n_i - 1} + \frac{1}{2}\sum_{i=1}^{2}\frac{\left(\text{tr}\tilde{\mathbf{S}}^{-1}\tilde{\mathbf{S}}_i\right)^2}{n_i - 1}\right]. \end{aligned}$$

If you want to do bootstrap to get the p-value, then you must transform the data under the null hypothesis. The estimate of the common mean is given by (Aitchison, 2003)

$$\hat{\boldsymbol{\mu}}_c = \left(n_1\mathbf{S}_1^{-1} + n_2\mathbf{S}_2^{-1}\right)^{-1}\left(n_1\mathbf{S}_1^{-1}\bar{\mathbf{X}}_1 + n_2\mathbf{S}_2^{-1}\bar{\mathbf{X}}_2\right) = \left(\tilde{\mathbf{S}}_1^{-1} + \tilde{\mathbf{S}}_2^{-1}\right)^{-1}\left(\tilde{\mathbf{S}}_1^{-1}\bar{\mathbf{X}}_1 + \tilde{\mathbf{S}}_2^{-1}\bar{\mathbf{X}}_2\right) \quad (2.3)$$

The modified Nel and van der Merwe (1986) test is based on the same quadratic form as that of James but the distribution used to compare the value of the test statistic is different. It is shown in Krishnamoorthy and Yu (2004) that $T_u^2 \sim \frac{\nu p}{\nu - p + 1}F_{p,\nu-p+1}$ approximately, where

$$\nu = \frac{p + p^2}{\frac{1}{n_1}\left\{\text{tr}\left[\left(\mathbf{S}_1\tilde{\mathbf{S}}\right)^2\right] + \text{tr}\left[\left(\mathbf{S}_1\tilde{\mathbf{S}}\right)\right]^2\right\} + \frac{1}{n_2}\left\{\text{tr}\left[\left(\mathbf{S}_2\tilde{\mathbf{S}}\right)^2\right] + \text{tr}\left[\left(\mathbf{S}_2\tilde{\mathbf{S}}\right)\right]^2\right\}}.$$

The algorithm is taken by Krishnamoorthy and Xia (2006). The R-code for both versions (with the option for a bootstrap p-value) is the following

```
james <- function(y1, y2, a = 0.05, R = 999, graph = FALSE) {
  ## y1 and y2 are the two samples
  ## a is the significance level and
  ## if R==1 the James test is performed
  ## if R==2 the Nel and van der Merwe test is performed
  ## if R>2 bootstrap calculation of the p-value is performed
  ## 999 bootstrap resamples are set by default
  ## Bootstrap is used for the p-value
  ## if graph is TRUE, the bootstrap statics are plotted
  p <- dim(y1)[2]  ## dimensionality of the data
  n1 <- dim(y1)[1]   ;   n2 <- dim(y2)[1]  ## sample sizes
  ybar1 <- Rfast::colmeans(y1)  ## sample mean vector of the first sample
  ybar2 <- Rfast::colmeans(y2)  ## sample mean vector of the second sample
  dbar <- ybar2 - ybar1  ## difference of the two mean vectors
```

```
mesoi <- rbind(ybar1, ybar2)
rownames(mesoi) <- c("Sample 1", "Sample 2")

if ( is.null(colnames(y1)) ) {
  colnames(mesoi) <- paste("X", 1:p, sep = "")
} else  colnames(mesoi) <- colnames(y1)
A1 <- Rfast::cova(y1)/n1
A2 <- Rfast::cova(y2)/n2
V <- A1 + A2  ## covariance matrix of the difference
Vinv <- Rfast::spdinv(V)    ## same as solve(V), but faster
test <- sum( dbar %*% Vinv * dbar )
b1 <- Vinv %*% A1
b2 <- Vinv %*% A2
trb1 <- sum( diag(b1) )
trb2 <- sum( diag(b2) )

if ( R <= 1 ) {
  ## James test
  A <- 1 + ( trb1^2/(n1 - 1) + trb2^2/(n2 - 1) ) / (2 * p)
  B <- ( sum(b1^2) / (n1 - 1) + sum(b2^2)/(n2 - 1) + 0.5 * trb1 ^ 2/ (n1 - 1) + 0.5 * t
  x2 <- qchisq(1 - a, p)
  delta <- (A + B * x2)
  twoha <- x2 * delta  ## corrected critical value of the chi-square
  pvalue <- pchisq(test/delta, p, lower.tail = FALSE)  ## p-value of the test statistic
  info <- c(test, pvalue, delta, twoha)
  names(info) <- c("test", "p-value", "correction", "corrected.critical")
  note <- paste("James test")
  result <- list(note = note, mesoi = mesoi, info = info)

} else if ( R == 2 ) {
  ## MNV test
  low <- ( sum( b1 %*% b1 ) + trb1^2 ) / n1 + ( sum( b2 %*% b2 ) + trb2^2 ) / n2
  v <- (p + p^2) / low
  test <- as.numeric( (v - p + 1) / (v * p) * test )  ## test statistic
  crit <- qf(1 - a, p, v - p + 1)  ## critical value of the F distribution
  pvalue <- pf(test, p, v - p + 1, lower.tail = FALSE)  ## p-value of the test statisti
  info <- c(test, pvalue, crit, p, v - p + 1)
  names(info) <- c("test", "p-value", "critical", "numer df", "denom df")
  note <- paste("MNV variant of James test")
```

```
    result <- list(note = note, mesoi = mesoi, info = info)

  } else  if ( R > 2 ) {
    ## bootstrap calibration
    runtime <- proc.time()
    a1inv <- Rfast::spdinv(A1)
    a2inv <- Rfast::spdinv(A2)
    mc <- solve( a1inv + a2inv ) %*% ( a1inv %*% ybar1 + a2inv %*% ybar2 )
    ## mc is the combined sample mean vector
    ## the next two rows bring the mean vectors of the two sample equal
    ## to the combined mean and thus equal under the null hypothesis
    mc1 <-  - ybar1 + mc
    mc2 <-  - ybar2 + mc
    x1 <- Rfast::eachrow(y1, mc1, oper = "+")
    x2 <- Rfast::eachrow(y2, mc2, oper = "+" )
    tb <- numeric(R)
    for (i in 1:R) {
  b1 <- Rfast2::Sample.int(n1, n1, replace = TRUE)
  b2 <- Rfast2::Sample.int(n2, n2, replace = TRUE)
      xb1 <- x1[b1, ]     ;      xb2 <- x2[b2, ]
      db <- Rfast::colmeans(xb1) - Rfast::colmeans(xb2)  ## difference of the two mean ve
      Vb <- Rfast::cova(xb1) / n1 + Rfast::cova(xb2) / n2  ## covariance matrix of the di
      tb[i] <- sum( db %*% solve(Vb, db ) )
    }
    pvalue <- ( sum(tb > test) + 1 ) / (R + 1)

    if ( graph ) {
      hist(tb, xlab = "Bootstrapped test statistic", main = " ")
      abline(v = test, lty = 2, lwd = 2)  ## The line is the test statistic
    }
    note <- paste("Bootstrap calibration")
    runtime <- proc.time() - runtime
    result <- list(note = note, mesoi = mesoi, pvalue = pvalue, runtime = runtime)
  }

  result
}
```

2.5 Relationship between the Hotelling's T^2 and James test

Emerson (2009, pg. 76-81) mentioned a very nice result between the Hotelling's one sample T^2 and James test for two mean vectors

$$J(\boldsymbol{\mu}) = T_1^2(\boldsymbol{\mu}) + T_2^2(\boldsymbol{\mu}), \tag{2.4}$$

where $J(\boldsymbol{\mu})$ is the James test statistic (2.2) and $T_1^2(\boldsymbol{\mu})$ and $T_1^2(\boldsymbol{\mu})$ are the two one sample Hotelling's T^2 test statistic values (2.1) for each sample from their common mean (2.3). In fact, James test statistic is found from *minimizing* the right hand side of (2.4) with respect to $\boldsymbol{\mu}$. The sum is *minimized* when $\boldsymbol{\mu}$ takes the form of (2.3). The same is true for the t-test in the univariate case.

I have created a small code illustrating this result, so this one is for educational purposes. It calculates the James test statistic, the sum of the two T^2 test statistics, the common mean vector and the one found via numerical optimization. In the univariate case, the common mean vector is a weighted linear combination of the two sample means. So, if we take a segment connecting the two means, the common mean is somewhere on that segment. This is not true however for the multivariate case.

```
james.hotel <- function(x, y) {
  ## x and y are the multivariate samples
  n1 <- dim(x)[1]  ;   n2 <- dim(y)[1]  ## sample sizes
  m1 <- Rfast::colmeans(x)  ## sample mean vector of the first sample
  m2 <- Rfast::colmeans(y)  ## sample mean vector of the second sample
  dbar <- m2 - m1  ## difference of the two mean vectors
  s1 <- cov(x)      ;    s2 <- cov(y)
  A1 <- s1/n1   ;    A2 <- s2/n2
  V <- A1 + A2  ## covariance matrix of the difference
  test <- as.numeric(   dbar %*% solve(V, dbar) )
  a1inv <- chol2inv( chol(A1) )
  a2inv <- chol2inv( chol(A2) )
  mc <- solve( a1inv + a2inv ) %*% ( a1inv %*% m1 + a2inv %*% m2 )
  a1 <- a1inv / n1   ;   a2 <- a2inv / n2
  funa <- function(m) {
    n1 * (m1 - m) %*% a1 %*% ( m1 - m ) +  ## Hotelling's test statistic
    n2 * (m2 - m) %*% a2 %*% ( m2 - m )    ## Hotelling's test statistic
 }
  bar <- optim( (m1 + m2)/2, funa )
  bar <- optim( bar$par, funa )
  tests <- c(test, bar$value )
```

```
  names(tests) <- c("James test", "T1(mu) + T2(mu)")
  list(tests = tests, mathematics.mean = t(mc), optimised.mean = bar$par )
}
```

2.6 MANOVA without assuming equality of the covariance matrices

James (1954) also proposed an alternative to MANOVA when the covariance matrices are not assumed equal. The test statistic for k samples is

$$J = \sum_{i=1}^{k} \left(\bar{\mathbf{x}}_i - \bar{\mathbf{X}}\right)^T \mathbf{W}_i \left(\bar{\mathbf{x}}_i - \bar{\mathbf{X}}\right), \tag{2.5}$$

where $\bar{\mathbf{x}}_i$ and n_i are the sample mean vector and sample size of the i-th sample respectively and $\mathbf{W}_i = \left(\frac{\mathbf{S}_i}{n_i}\right)^{-1}$, where $\mathbf{S}_i$ is the covariance matrix of the i-sample mean vector and $\bar{\mathbf{X}}$ is the estimate of the common mean $\bar{\mathbf{X}} = \left(\sum_{i=1}^{k} \mathbf{W}_i\right)^{-1} \sum_{i=1}^{k} \mathbf{W}_i \bar{\mathbf{x}}_i$. We used the corrected χ^2 distribution James (1954) proposed and no bootstrap calibration.

In case you do not have access to James's paper see page 11 of this document (or send me an e-mail). Normally one would compare the test statistic (2.5) with a $\chi^2_{r,1-\alpha}$, where $r = p\left(k-1\right)$ are the degrees of freedom with k denoting the number of groups and p the dimensionality of the data. There are r constraints (how many univariate means must be equal, so that the null hypothesis, that all the mean vectors are equal, holds true), that is where these degrees of freedom come from. James compared the test statistic (2.5) with a corrected χ^2 distribution instead. Let A and B be

$$\begin{aligned} A &= 1 + \frac{1}{2r} \sum_{i=1}^{k} \frac{\left[\text{tr}\left(\mathbf{I}_p - \mathbf{W}^{-1}\mathbf{W}_i\right)\right]^2}{n_i - 1} \\ B &= \frac{1}{r\left(r+2\right)} \sum_{i=1}^{k} \left\{ \frac{\text{tr}\left[\left(\mathbf{I}_p - \mathbf{W}^{-1}\mathbf{W}_i\right)^2\right]}{n_i - 1} + \frac{\left[\text{tr}\left(\mathbf{I}_p - \mathbf{W}^{-1}\mathbf{W}_i\right)\right]^2}{2\left(n_i - 1\right)} \right\}. \end{aligned}$$

The corrected quantile of the χ^2 distribution is given as before by

$$2h\left(\alpha\right) = \chi^2\left(A + B\chi^2\right).$$

```
maovjames <- function(x, ina, a = 0.05) {
  ## x contains all the groups together
  ## a is the significance level
  x <- as.matrix(x)  ## makes sure x is a matrix
  ina <- as.numeric(ina)  ## the group indicator variable
  ni <- tabulate(ina)  ## the group sample sizes
```

```
  k <- max(ina)  ## the number of groups
  p <- dim(x)[2]  ## the dimensionality
  n <- dim(x)[1]  ## the total sample size
  ## the objects below will be used later
  me <- mi <- W <- matrix(nrow = k, ncol = p)
  ta <- numeric(k)
  wi <- array( dim = c(p, p, k) )
  ## the next for function calculates the
  ## mean vector and covariance matrix of each group
  for (i in 1:k) {
    zi <- x[ina == i, ]
    mi[i, ] <- Rfast::colmeans( zi )
    wi[, , i] <- ni[i] * chol2inv( chol( var( zi ) ) )
    me[i, ] <- mi[i, ] %*% wi[, , i]
  }
  W <- t( colSums( aperm(wi) ) )
  Ws <- solve(W)
  ma <- Rfast::colsums(me)
  mesi <- Ws %*% ma  ## common mean vector
  t1 <- t2 <- numeric(k)
  Ip <- diag(p)
  for (i in 1:k) {
  ta[i] <- sum( (mi[i,] - mesi) * ( wi[, , i] %*% (mi[i, ] - mesi) ) )
    exa1 <- Ip - Ws %*% wi[, , i]
    t1[i] <- sum( diag(exa1) )
    t2[i] <- sum( exa1^2 )
   }
  test <- sum(ta)  ## the test statistic
  r <- p * (k - 1)
  A <- 1 + sum( t1^2/(ni - 1) ) / 2 / r
  B <- sum( t2 / (ni - 1) + t1^2 / 2 / (ni - 1) ) / r / (r + 2)
  x2 <- qchisq(1 - a, r)
  delta <- (A + B * x2)
  twoha <- x2 * delta  ## corrected critical value of chi-square distribution
  pvalue <- pchisq(test/delta, r, lower.tail = FALSE)  ## p-value
  result <- c(test, delta, twoha, pvalue)
  names(result) <- c("test", "correction", "corr.critical", "p-value")
  result
}
```

2.7 Relationship between James's MANOVA and Hotelling's T^2

The relationship we saw for the James two sample test (2.4) is true for the case of the MANOVA. The estimate of the common mean (2.3) is in general, for g groups, each of sample size n_i, written as

$$\hat{\boldsymbol{\mu}}_c = \left(\sum_{i=1}^{g} n_i \mathbf{S}_i^{-1} \right)^{-1} \sum_{i=1}^{g} n_i \mathbf{S}_i^{-1} \bar{\mathbf{X}}_i.$$

The next R code is just a proof of the mathematics you will find in Emerson (2009, pg. 76-81) and is again intended for educational purposes.

```
maovjames.hotel <- function(x, ina) {
  ## contains the data
  ## ina is a grouping  variable indicating the groups
  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there
  nu <- tabulate(ina)
  mi <- rowsum(x, ina) / nu  ## group mean vectors
  p <- dim(x)[2]
  si <- array( dim = c(p, p, g) )
  for (i in 1:g)  si[, , i] <- solve( cov( x[ina == i, ] ) )
  f <- numeric(g)
  funa <- function(m) {
    for (i in 1:g)  f[i] <- nu[i] * ( mi[i, ] - m ) %*% si[, , i] %*% (mi[i, ] - m)
    sum(f)
  }
  bar <- optim(Rfast::colmeans(x), funa, control = list(maxit=2000) )
  bar <- optim(bar$par, funa, control = list(maxit=2000) )
  bar <- optim(bar$par, funa, control = list(maxit=2000) )
  list(test = bar$value, mc = bar$par)
}
```

2.8 A two sample mean test for high dimensional data

High dimensional data are the multivariate data which have many variables (p) and usually a small number of observations (n). It also happens that $p > n$ and this is the case here in this Section. We will see a simple test for the case of $p > n$. In this case, the covariance matrix is not invertible and in addition it can have a lot of zero eigenvalues.

The test we will see was proposed by Bai and Saranadasa (1996). Ever since, there have been some more suggestions but I chose this one for its simplicity. There are two data, $\mathbf{X}$

and $\mathbf{Y}$ of sample size n_1 and n_2 respectively. Their corresponding sample mean vectors and covariance matrices are $\bar{\mathbf{X}}$, $\bar{\mathbf{Y}}$ and $\mathbf{S}_1$, $\mathbf{S}_2$ respectively. The assumption here is the same as that of the Hotelling's test we saw before.

Let us define the pooled covariance matrix at first, calculated under the assumption of equal covariance matrices

$$\mathbf{S}_n = \frac{(n_1 - 1)\,\mathbf{S}_1 + (n_2 - 1)\,\mathbf{S}_2}{n},$$

where $n = n_1 + n_2$. Then define

$$B_n = \sqrt{\frac{n^2}{(n+2)\,(n-1)}\left\{\mathrm{tr}\,(\mathbf{S}_n^2) - \frac{1}{n}\left[\mathrm{tr}\,(\mathbf{S}_n)\right]^2\right\}}.$$

The test statistic is

$$Z = \frac{\frac{n_1 n_2}{n_1+n_2}\,(\bar{\mathbf{X}} - \bar{\mathbf{Y}})^T\,(\bar{\mathbf{X}} - \bar{\mathbf{Y}}) - \mathrm{tr}\,(\mathbf{S}_n)}{\sqrt{\frac{2(n+1)}{n}}\,B_n}. \tag{2.6}$$

Under the null hypothesis (equality of the two mean vectors) the test statistic (2.6) follows the standard normal distribution. Chen et al. (2010) mentions that Bai and Saranadasa (1996) established the asymptotic normality of the test statistics and showed that it has attractive power property when $p/n \to c < \infty$ and under some restriction on the maximum eigenvalue of the common population covariance matrix. However, the requirement of p and n being of the same order is too restrictive to be used in the "large p small n" situation.

For this reason Chen et al. (2010) proposed a modification of the test statistic we showed. Their test statistic is more general and allows for unequal covariance matrices and is applicable in the "large p small n" situation. This procedure along with some others can be found in the R package highD2pop created by Gregory (2014).

The code for the test proposed by Bai and Saranadasa (1996) is available below. Note, that both x and y must be matrices.

```
sarabai <- function(x, y) {
  ## x and y are high dimensional datasets
  n1 <- dim(x)[1]    ;    n2 <- dim(y)[1]  ## sample sizes
  m1 <- Rfast::colmeans(x)   ;   m2 <- Rfast::colmeans(y)  ## sample means
  n <- n1 + n2 - 2
  z1 <- t(x) - m1
  z2 <- t(y) - m2
  Sn <- tcrossprod( z1 ) + tcrossprod( z2 )
  ## Sn is the pooled covariance matrix
```

```
  trSn <- sum( z1^2 )/n + sum( z2^2 ) /n
  trSn2 <- sum(Sn^2)/n^2
  Bn <- sqrt( n^2/( (n + 2) * (n - 1) ) * (trSn2 - trSn^2/n) )
  up <- n1 * n2/(n1 + n2) * sum( (m1 - m2)^2 ) - trSn
  down <- sqrt( 2 * (n + 1)/n ) * Bn
  Z <- up / down  ## test statistic
  pvalue <- pnorm(Z, lower.tail = FALSE)
  res <- c(Z, pvalue)
  names(res) <- c('Z', 'p-value')
  res
}
```

2.9 Repeated measures ANOVA (univariate data) using Hotelling's T^2 test

We will show how can one use Hotelling's T^2 test to analyse univariate repeated measures. Univariate analysis of variance for repeated measures is the classical way, but we can use this multivariate test as well. In the repeated measures ANOVA case, we have many repeated observations from the same n subjects, usually at different time points and the interest is to see whether the means of the samples are equal or not $\mu_1 = \mu_2 = \ldots = \mu_k$ assuming k repeated measurements. We can of course change this null hypothesis and test many combinations of means. The idea in any case is to construct a matrix of contrasts. I will focus here in the first case only and in particular the null hypothesis and the matrix of contrasts $\mathbf{C}$ are

$$\begin{pmatrix} \mu_1 = \mu_2 \\ \mu_2 = \mu_3 \\ \vdots \\ \mu_{k-1} = \mu_k \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 & \ldots & 0 \\ 1 & 0 & -1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \ldots & -1 \end{pmatrix} \boldsymbol{\mu} = \mathbf{C}\boldsymbol{\mu}.$$

The contrast matrix $\mathbf{C}$ has $k-1$ independent rows and if there is no treatment effect, $\mathbf{C}\boldsymbol{\mu} = \mathbf{0}$ See for more information see Ranjan Maitra's teaching slides Paired Comparisons and Repeated Measures.

The test statistic is

$$T_r^2 = \frac{(n-k+1)}{(n-1)(k-1)} n \left(\mathbf{C}\bar{\mathbf{x}}\right)^T \left(\mathbf{C}\mathbf{S}\mathbf{C}^T\right)^{-1} \left(\mathbf{C}\bar{\mathbf{x}}\right) \sim F_{k-1,n-k+1}.$$

Below is the relevant R function.

```
rm.hotel <- function(x, a = 0.05) {
```

```
  ## x is the data set
  ## a is the level of significance set by default to 0.05

  m <- Rfast::colmeans(x)
  s <- cov(x)  ## sample mean vector and covariance matrix
  n <- dim(x)[1]  ## sample size
  p <- dim(x)[2]  ##  dimensionality of the data
  C <-  - diag(p)
  C[, 1] <- 1
  A <- C %*% m
  B <- solve(C %*% s %*% C, A)

  T2 <- n * sum(A * B)
  test <- (n - p + 1) / (n - 1) / (p - 1) * T2  ## test statistic

  pvalue <- pf(test, p - 1, n - p + 1, lower.tail = FALSE)  ## p-value
  crit <- qf(1 - a, p - 1, n - p + 1)  ## critical value of the F distribution
  result <- c(test, pvalue, crit, p - 1, n - p + 1)
  names(result) <- c("test", "p-value", "critical", "numer df", "denom df")
  list(m = m, result = result)

}
```

3 Hypothesis testing for covariance matrices

The first section comprises of tests regarding one or more covariance matrices.

3.1 One sample covariance test

Let's begin with the hypothesis test that the the sample covariance is equal to some specified covariance matrix: $H_0 : \boldsymbol{\Sigma} = \boldsymbol{\Sigma}_0$, with $\boldsymbol{\mu}$ unknown. The algorithm for this test is taken from Mardia et al., 1979, pg. 126-127. The test is based upon the log-likelihood ratio test. The form of the test is

$$-2\log\lambda = n\text{tr}\left\{\boldsymbol{\Sigma}_0^{-1}\mathbf{S}\right\} - n\log\left|\boldsymbol{\Sigma}_0^{-1}\mathbf{S}\right| - np, \tag{3.1}$$

where n is the sample size, $\boldsymbol{\Sigma}_0$ is the specified covariance matrix under the null hypothesis, $\mathbf{S}$ is the sample covariance matrix and p is the dimensionality of the data (or the number of variables). Let α and g denote the arithmetic mean and the geometric mean respectively of the eigenvalues of $\boldsymbol{\Sigma}_0^{-1}\mathbf{S}$, so that $tr\left\{\boldsymbol{\Sigma}_0^{-1}\mathbf{S}\right\} = p\alpha$ and $\left|\boldsymbol{\Sigma}_0^{-1}\mathbf{S}\right| = g^p$, then (3.1) becomes

$$-2\log\lambda = np\left(\alpha - log(g) - 1\right)$$

The degrees of freedom of the X^2 distribution are $\frac{1}{2}p\left(p+1\right)$.

```
cov.equal <- function(x, Sigma, a = 0.05) {
  ## x is the data set
  ## Sigma is the assumed covariance matrix
  ## a is the level of significance set by default to 0.05
  Sigma <- as.matrix(Sigma)
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size
  S <- cova(x)  ## sample covariance matrix
  mesa <- solve(Sigma, S)
  test <- n * sum( diag(mesa) ) - n * log( det(mesa) ) - n * p ## test statistic
  dof <- 0.5 * p * (p + 1)  ## the degrees of freedom of  chi-square distribution
  pvalue <- pchisq(test, dof, lower.tail = FALSE)  ## p-value
  crit <- qchisq(1 - a, dof)  ## critical value of chi-square distribution
  res <- c(test, dof, pvalue, crit)
  names(res) <- c("test", "df", "p-value", "critical")
  res
}
```

3.2 Multi-sample covariance matrices

We will show the two versions of Box's test for the hypothesis test of the equality of at least two covariance matrices: $H_0 : \mathbf{\Sigma}_1 = \ldots = \mathbf{\Sigma}_k$. The algorithms are taken from Aitchison, 2003, pg. 155 and Mardia et al., 1979, pg. 140.

3.2.1 Log-likelihood ratio test

At first we will see the likelihood-ratio test. This is the multivariate generalization of Bartlett's test of homogeneity of variances. The test has the form

$$-2log\lambda = n \log |\mathbf{S}| - \sum_{i=1}^{k} n_i \log |\mathbf{S_i}| = \sum_{i=1}^{k} n_i \log \left| \mathbf{S}_i^{-1} \mathbf{S} \right|, \tag{3.2}$$

where $\mathbf{S}_i$ is the ith sample biased covariance matrix and $\mathbf{S} = n^{-1} \sum_{i=1}^{k} n_i \mathbf{S}_i$ is the m.l.e. of the common covariance matrix (under the null hypothesis) with $n = \sum_{i=1}^{k} n_i$. The degrees of freedom of the asymptotic chi-square distribution are $\frac{1}{2}(p+1)(k-1)$.

```
cov.likel <- function(x, ina, a = 0.05) {
  ## x is the data set
  ## ina is a numeric vector indicating the groups of the data set
  ## a is the level of significance, set to 0.05 by default
  ina <- as.numeric(ina)
  p <- dim(x)[2]  ## dimension of the data set
  n <- dim(x)[1]  ## total sample size
  k <- max(ina)  ## number of groups
  nu <- tabulate(ina) ## the sample size of each group
  t1 <- rep( (nu - 1)/nu, each = p^2 )
  t2 <- rep(nu - 1, each = p^2 )
  s <- array( dim = c(p, p, k) )
  ## the next 3 lines create the pooled covariance matrix
  ## and calculate the covariance matrix of each group
  for (i in 1:k)  s[, , i] <- cov( x[ina == i, ] )
  mat <- t1 * s
  mat1 <- t2 * s
  Sp <- colSums( aperm(mat1) ) / n
  deta <- apply(mat, 3, det)
  pame <- det(Sp) / deta
  test <- sum(nu * log(pame))  ## test statistic
  dof <- 0.5 * p * (p + 1) * (k - 1) ## degrees of freedom of chi-square
  pvalue <- pchisq(test, dof, lower.tail = FALSE)  ## p-value of test statistic
```

```
  crit <- qchisq(1 - a, dof)  ## critical value of chi-square distribution
  res <- c(test, pvalue, dof, crit)
  names(res) <- c('test', 'p-value', 'df', 'critical')
  res
}
```

3.2.2 Box's M test

According to Mardia et al., 1979, pg. 140, it may be argued that if n_i is small, then (3.2) gives too much weight to the contribution of $\mathbf{S}$. This consideration led Box (1949) to propose the test statistic in place of that given in (3.2). Box's $\mathbf{M}$ is given by

$$\mathbf{M} = \gamma \sum_{i=1}^{k} (n_i - 1) \log \left| \mathbf{S}_i^{-1} \mathbf{S}_p \right|,$$

where

$$\gamma = 1 - \frac{2p^2 + 3p - 1}{6\,(p+1)\,(k-1)} \left(\sum_{i=1}^{k} \frac{1}{n_i - 1} - \frac{1}{n-k} \right)$$

and $\mathbf{S}_i$ and $\mathbf{S}_p$ are the i-th unbiased covariance estimator and the pooled covariance matrix respectively with

$$\mathbf{S}_p = \frac{\sum_{i=1}^{k} (n_i - 1)\, \mathbf{S}_i}{n-k}$$

Box's M also has an asymptotic chi-square distribution with $\frac{1}{2}(p+1)(k-1)$ degrees of freedom. Box's approximation seems to be good if each n_i exceeds 20 and if k and p do not exceed 5 (Mardia et al., 1979, pg. 140).

```
cov.Mtest <- function(x, ina, a = 0.05) {
  ## x is the data set
  ## ina is a numeric vector indicating the groups of the data set
  ## a is the level of significance, set to 0.05 by default
  p <- dim(x)[2]  ## dimension of the data set
  n <- dim(x)[1]  ## total sample size
  ina <- as.numeric(ina)
  k <- max(ina)  ## number of groups
  nu <- tabulate(ina)  ## the sample size of each group
  ni <- rep(nu - 1, each = p^2)
  mat <- array(dim = c(p, p, k))
  ## next is the covariance of each group
```

```
  for (i in 1:k)  mat[, , i] <- cov(x[ina == i, ])
  mat1 <- ni * mat
  pame <- apply(mat, 3, det)  ## the detemirnant of each covariance matrix
  ## the next 2 lines calculate the pooled covariance matrix
  Sp <- colSums( aperm(mat1) ) / ( n - k )
  pamela <- det(Sp)  ## determinant of the pooled covariance matrix
  test1 <- sum( (nu - 1) * log(pamela/pame) )
  gama1 <- ( 2 * (p^2) + 3 * p - 1 ) / ( 6 * (p + 1) * (k - 1) )
  gama2 <- sum( 1/(nu - 1) ) - 1/(n - k)
  gama <- 1 - gama1 * gama2
  test <- gama * test1  ## this is the M (test statistic)
  dof <- 0.5 * p * (p + 1) * (k - 1)  ## degrees of freedom of
  ## the chi-square distribution
  pvalue <- pchisq(test, dof, lower.tail = FALSE)  ## p-value
  crit <- qchisq(1 - a, dof)  ## critical value of chi-square distribution
  result <- c(test, pvalue, dof, crit)
  names(result) <- c('M.test', 'p-value', 'df', 'critical')
  result
}
```

4 Correlation, regression and discriminant analysis

In this section we will present functions for correlation, multivariate regression and discriminant analysis.

4.1 Correlation

4.1.1 Correlation coefficient confidence intervals and hypothesis testing using Fisher's transformation

Fisher's transformation for the correlation coefficient is defined as

$$\hat{z} = \frac{1}{2} \log \frac{1+r}{1-r} \tag{4.1}$$

with inverse equal to

$$\frac{\exp(2\hat{z}) - 1}{\exp(2\hat{z}) + 1}$$

The estimated standard error of (4.1) is $\frac{1}{\sqrt{n-3}}$ (Efron and Tibshirani, 1993, pg. 54). If on the other hand, you choose to calculate Spearman's correlation coefficients, the estimated standard error is slightly different $\simeq \frac{1.029563}{\sqrt{n-3}}$ (Fieller et al., 1957, Fieller and Pearson, 1957). R calculates confidence intervals based in a different way and does hypothesis testing for zero values only. The following function calculates asymptotic confidence intervals based upon (4.1), assuming asymptotic normality of (4.1) and performs hypothesis testing for the true (any, non only zero) value of the correlation. The sample distribution though is a t_{n-3}.

```
correl <- function(y, x, type = "pearson", a = 0.05, rho = 0, plot = F) {
  ## y and x are the two variables
  ## type supported is either "pearson" or "spearman"
  ## a is the significance level
  ## rho is the hypothesised correlation
  n <- length(y)
  if (type == "pearson") {
    r <- cor(y, x)  ## the correlation value
    zh0 <- 0.5 * log( (1 + rho) / (1 - rho) ) ## Fisher transformation for Ho
    zh1 <- 0.5 * log( (1 + r) / (1 - r) )  ## Fisher transformation for H1
    se <- 1/sqrt(n - 3)  ## standard error for Fisher transformation of Ho
  } else if (type == "spearman") {
    r <- cor(y, x, method = "spearman")  ## the correlation value
    zh0 <- 0.5 * log( (1 + rho) / (1 - rho) )  ## Fisher transformation for Ho
```

```
    zh1 <- 0.5 * log( (1 + r) / (1 - r) )  ## Fisher transformation for H1
    se <-  1.029563 / sqrt(n - 3)  ## standard error for Fisher transformation of Ho
  }
  test <- (zh1 - zh0)/se  ## test statistic
  pvalue <- 2 * ( 1 - pt( abs(test), n - 3 ) )  ## p-value
  zL <- zh1 - qt(1 - a/2, n - 3) * se
  zH <- zh1 + qt(1 - a/2, n - 3) * se
  fishL <- (exp(2 * zL) - 1)/(exp(2 * zL) + 1)  ## lower confidence limit
  fishH <- (exp(2 * zH) - 1)/(exp(2 * zH) + 1)  ## upper confidence limit
  ci <- c(fishL, fishH)
  names(ci) <- paste(c( a/2 * 100, (1 - a/2) * 100 ), "%", sep = "")
  r0 <- seq( max(-0.99, r - 0.2), min(0.99, r + 0.2), by=0.001 )
  z0 <- 0.5 * log( (1 + r0) / (1 - r0) )  ## Fisher's transformation
  ## for many Hos
  stat <- abs(zh1 - z0)/se   ## test statistics
  pval <- 2 * pt( -abs(stat), n - 3 )   ## p-values

  if ( plot ) {
    par( mfrow = c(1,2) )
    plot(r0, stat, type = "l", xlab = "Correlation values",
    ylab = "Test statistic")
    abline(h = qnorm(0.975), col = 2)
    abline( v = min( r0[stat < qt(0.975, n - 3)] ), col = 3, lty = 3 )
    abline( v = max( r0[stat < qt(0.975, n - 3)] ), col = 3, lty = 3 )
    plot(r0, pval, type = "l", xlab = "Correlation values",
    ylab = "P-values")
    abline(h = a, col = 2)
    abline(v = min(r0[pval > a]), col = 3, lty = 3)
    abline(v = max(r0[pval > a]), col = 3, lty = 3)
  }

  result <- c(r, pvalue)
  names(result) <- c('correlation', 'p-value')
  list(result = result, ci = ci)
}
```

4.1.2 Non-parametric (bootstrap and permutation) hypothesis testing for a zero correlation coefficient

We show how to perform a non-parametric bootstrap hypothesis testing that the correlation coefficient is zero. A good pivotal statistic is the Fisher's transformation (4.1). Then the data have to be transformed under the null hypothesis ($\rho = 0$). This is doable via the eigen-analysis of the covariance matrix. We transform the bivariate data such that the covariance (and thus the correlation) matrix equals the identity matrix (see the function of standardization for more information about this). We remind that the correlation matrix is independent of measurements and is location free. The next step is easy, we draw bootstrap samples (from the transformed data) and every time we calculate the Fisher's transformation. The bootstrap p-value is calculated in the usual way (Davison and Hinkley, 1997).

```
boot.correl <- function(x, B = 999) {
  ## x is a 2 column matrix containing the data
  ## B is the number of bootstrap replications
  s <- cov(x)
  n <- dim(x)[1]
  eig <- eigen(s, symmetric = TRUE)
  lam <- eig$values
  vec <- eig$vectors
  A <- vec %*% ( t(vec) / sqrt(lam) )
  z <- x %*% A  ## This makes the correlation matrix
  ## equal to the identity matrix, thus rho = 0
  rb <- numeric(B)
  r <- cor(x)[2]
  test <- 0.5 * log( (1 + r)/(1 - r) )  ## the test statistic
  for (i in 1:B) {
    nu <- sample(1:n, replace = TRUE)
    rb[i] <- cor( z[nu, ] )[2]
  }
  tb <- 0.5 * log( (1 + rb)/(1 - rb) )
  pvalue <- (sum( abs(tb) >  abs(test) ) + 1)/(B + 1)  ## bootstrap p-value
  hist(tb, xlab = "Bootstrapped test statistic", main = " ")
  abline(v = test, lty = 2, lwd = 2)
  ## The dotted vertical line is the test statistic value
  result <- c(r, pvalue)
  names(result) <- c('correlation', 'p-value')
  result
```

```
}
```

If you want to perform a non-parametric bootstrap hypothesis for a value of the correlation other than zero the procedure is similar. The data have already been transformed such that their correlation is zero. Now instead of the zeroes in the off-diagonal values of the identity matrix you will have the value of the correlation matrix you want to test. Eigen analysis of the matrix is performed and the square root of the matrix is used to multiply the transformed data. I could write a more general function to include all case, but I will leave this task to you. If you do write it please send it to me and I will put it with your name of course.

The next function is a vectorised version of the above function. Instead of using a *for* loop you can do things vectorised. This idea cam when I found the vectorised bootstrap correlation by Neto (2015). I cannot say I understood fully what he did, so I decided to write my own code based on the direction he pointed.

Pearson's correlation coefficient of x and y for a sample size n is given by

$$r = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sqrt{\left(\sum_{i=1}^{n} x_i^2 - n\bar{x}^2\right)\left(\sum_{i=1}^{n} y_i^2 - n\bar{y}^2\right)}}. \tag{4.2}$$

So, we can see that need 5 terms to calculate, $\sum_{i=1}^{n} x_i y_i, \bar{x}, \bar{y}, \sum_{i=1}^{n} x_i^2$ and $\sum_{i=1}^{n} y_i^2$. After transforming the data under the null hypothesis using the spectral decomposition we proceed as follows with B number of resamples.

Algorithm for vectorised bootstrap

1. Set a seed number in R, such as 123456. This is to make sure that the pairs of (x_i, y_i) are still the same.

2. Sample with replacement $B \times n$ values of x and put them in a matrix with n rows and B columns, named XB.

3. Sample with replacement $B \times n$ values of y and put them in a matrix with n rows and B columns, names YB.

4. Calculate the mean vector of XB and YB. These are the means of the bootstrap samples of x and y respectively ().

5. Calculate the sum vector of XB^2 and YB^2. These are the sums of the squares of the bootstrap samples of x and y respectively.

6. Finally calculate the sum vector of $XB * YB$. This is the term $\sum_{i=1}^{n} x_i y_i$ for all resamples.

So we now have 5 vectors containing the 5 terms we want. We calculate the correlation coefficient (4.2) and then the Fisher's transformation (4.1) and so we have B bootstrap test statistics. In order to see the time gain I tested both of these functions with $B = 9999$ resamples and 1000 repetitions. The function *boot.correl* required 538 seconds, whereas the function *bootcor* required 140. The time is reduced to 1/4 of its initial. The gain is not super wow, I would like it if it was 1/10, but even saw, it is still good. Parallelised versions reduce time to 1/3, so from this perspective, I did better. If we now put parallel inside this vectorised version, computations will be even faster. I leave this with you.

But, I noticed one thing, the same thing Neto (2015) mentions. For big sample sizes, for example 1000 pairs, the time difference is not that big and perhaps *for* is faster. The big difference is in the small to moderate sample sizes. At least for this example. What I mean by this is that you should not be afraid and say, then why? If I have big sample, I do not need vectorization. Maybe yes, but even then I still recommend it. Maybe someone else will have a better alternative for vectorization which is better even in the big samples, for the correlation of course. In the contour plots though, vectorised versions are always faster no matter what.

```
bootcor <- function(x, R = 999) {
  ## x is a 2 column matrix containing the data
  ## B is the number of bootstrap replications
  s <- cov(x)
  n <- dim(x)[1]
  eig <- eigen(s, symmetric = TRUE)
  lam <- eig$values
  vec <- eig$vectors
  A <- vec %*% ( t(vec) / sqrt(lam) )
  z <- x %*% A  ## This makes the correlation matrix equal to
  ## the identity matrix, thus rho = 0
  r <- s[2] / sqrt( s[1] * s[3] )
  test <- 0.5 * log( (1 + r)/(1 - r) )  ## the test statistic
  set.seed(12345)  ## this is so that the pairs are the same
  x1 <- matrix(sample(z[, 1], R * n, replace = TRUE), nrow = n)
  set.seed(12345)  ## this is so that the pairs are the same
  x2 <- matrix(sample(z[, 2], R * n, replace = TRUE), nrow = n)
  yb1 <- Rfast::colmeans(x1)   ;    yb2 <- Rfast::colmeans(x2)
  y1 <- Rfast::colsums(x1^2)   ;    y2 <- Rfast::colsums(x2^2)
  sxy <- Rfast::colsums(x1 * x2)
  rb <- (sxy - n * yb1 * yb2) / sqrt( (y1 - n * yb1^2) * (y2 - n * yb2^2) )
  tb <- 0.5 * log( (1 + rb)/(1 - rb) )  ## the test statistic
  pvalue <- (sum( abs(tb) > abs(test) ) + 1)/(R + 1)  ## bootstrap p-value
```

```
  hist(tb, xlab = "Bootstrapped test statistic", main = " ")
  abline(v = test, lty = 2, lwd = 2)
  ## The dotted vertical line is the test statistic value
  result <- c(r, pvalue)
  names(result) <- c('correlation', 'p-value')
  result
}
```

Next is a permutation based p-value for the same test. The idea is this: instead of transforming the data under the null hypothesis and re-sampling with replacement we can permute the observations. Te basic difference is that the data are assumed to be under the null hypothesis already. Secondly, what we have to do, is to destroy the pairs. For example, the pairs (a, b), (c, d) and (e, f) in one permutation they can be (c, b), (a, f) and (e, d). And this thing will happen many times, say $R = 999$. Then we have R pseudo-samples again. Everything else is the same as in the bootstrap case. A trick is that we need not change the order of both variables, just the one is enough. This will sped up the process. And guess what, it is faster than bootstrap. It does not require the data to be transformed under the null hypothesis and you only need to permute one variable, in contrast to the bootstrap case, where you must resample from both variables.

```
permcor2 <- function(x, R = 999) {
  ## x is a 2 column matrix containing the data
  ## R is the number of permutations
  n <- dim(x)[1]
  r <- cor(x)[2]
  test <- 0.5 * log((1 + r)/(1 - r))  ## the test statistic
  x1 <- replicate( R, sample(x[ , 1], n) )
  x2 <- x[, 2]
  m1 <- mean(x[, 1])      ;   m12 <- sum(x[, 1]^2)
  yb1 <- numeric(R) + m1  ;  y1 <- numeric(R) + m12
  m2 <- mean(x2)          ;   m22 <- sum(x2^2)
  yb2 <- numeric(R) + m2  ;  y2 <- numeric(R) + m22
  sxy <- Rfast::colsums(x1 * x2)
  rb <- (sxy - n * yb1 * yb2) / sqrt( (y1 - n * yb1^2) * (y2 - n * yb2^2) )
  tb <- 0.5 * log( (1 + rb)/(1 - rb) )  ## the test statistic
  pvalue <- ( sum( abs(tb) > abs(test) ) + 1 ) / (R + 1)  ## bootstrap p-value
  res <- c(r, pvalue)
  names(res) <- c('correlation', 'p-value')
  res
}
```

I believe though this version with a *for* is faster than the vectorised version. A possible reason for this is that R cannot handle big matrices easily.

```
permcor <- function(x, R = 999) {
  ## x is a 2 column matrix containing the data
  ## type can be either "pearson" or "spearman"
  ## R is the number of permutations
   x1 <- x[, 1]      ;     x2 <- x[, 2]
  n <- length(x1)
  m1 <- sum(x1)     ;     m12 <- sum(x1^2)
  m2 <- sum(x2)     ;     m22 <- sum(x2^2)
  up <-  m1 * m2 / n
  down <- sqrt( (m12 - m1^2 / n) * (m22 - m2^2 / n) )
  r <- ( sum(x1 * x2) - up) / down
  test <- log( (1 + r) / (1 - r) )  ## the test statistic
  sxy <- numeric(R)
  for (i in 1:R) {
    y1 <- sample(x1, n, replace = FALSE)
    sxy[i] <- sum(y1 * x2)
  }
  rb <- (sxy - up) / down
  tb <- log( (1 + rb) / (1 - rb) )  ## the test statistic
  pvalue <- ( sum( abs(tb) > abs(test) ) + 1 ) / (R + 1)  ## bootstrap p-value
  res <- c( r, pvalue )
  names(res) <- c('correlation', 'p-value')
  res
}
```

4.1.3 Correlation coefficient between a variable and many others

Suppose you have a (dependent) variable Y and a a matrix of many variables $\mathbf{X}$ and you want to get all the correlations between Y and X_i for all i. if you type $cor(y, x)$ in you will get a vector of the correlations. What I offer here is confidence interval for each of the correlations, the test statistic and the p-values for the hypothesis that each of them is equal to some value ρ. The p-values and test statistics are useful for meta-analysis for example, combination of the p-values in one or even to see the false discovery rate (see the package fdrtool by Korbinian Strimmer).

```
correls <- function(y, x, type = "pearson", a = 0.05, rho = 0) {
  ## y is a numerical vector
```

```
  ## x is a matrix containing at least two variables
  ## type supported is either "pearson" or "spearman"
  ## a is the significance level
  ## rho is the hypothesised correlation
  n <- length(y)
  if (type == "pearson") {
    r <- as.vector( cor(y, x) ) ## the correlation value between y and all the xs
    zh0 <- 0.5 * log( (1 + rho) / (1 - rho) )  ## Fisher's transformation for Ho
    zh1 <- 0.5 * log( (1 + r) / (1 - r) )  ## Fisher's transformation for H1
    se <- 1/sqrt(n - 3)  ## standard error for Fisher's transformation of Ho
  } else if (type == "spearman") {
    r <- as.vector( cor(y, x, method = "spearman") ) ## the correlation value
    ##  between y and all the xs
    zh0 <- 0.5 * log( (1 + rho) / (1 - rho) )  ## Fisher transformation for Ho
    zh1 <- 0.5 * log( (1 + r) / (1 - r) )  ## Fisher transformation for H1
    se <- 1.029563 / sqrt(n - 3)  ## standard error under Ho
  }
  test <- as.vector( (zh1 - zh0) / se ) ## test statistic
  pvalue <-  2 * ( pt( -abs(test), n - 3 ) )  ## p-value
  b1 <- zh1 - qt(1 - a/2, n - 3) * se
  b2 <- zh1 + qt(1 - a/2, n - 3) * se
  ca <- cbind(b1 ,b2)
  ela <- exp( 2 * ca )
  ci <- ( ela - 1 ) / ( ela + 1 )  ## confidence intervals
  res <- cbind(r, pvalue, test, ci)
  colnames(res) <- c( 'correlation', 'p-value', 'z-stat',
                      paste( c( a/2 * 100, (1 - a/2) * 100 ), "%", sep = "") )
  if ( is.null(colnames(x)) ) {
    rownames(res) <- paste("X", 1:dim(x)[2], sep = "")
  } else  rownames(res) <- colnames(x)

  res
}
```

Below is the sam function as above, only this time the p-value is produced via permutations and no confidence intervals are produced.

```
perm.correls <- function(y, x, R = 999) {
  ## x is a 2 column matrix containing the data
  ## type can be either "pearson" or "spearman"
```

```
  ## R is the number of permutations
  p <- dim(x)[2]
  n <- length(y)
  r <- as.vector( cor(y, x) )
  test <- 0.5 * log( (1 + r) / (1 - r) )  ## the test statistic
  m1 <- sum(y)          ;     m12 <- sum(y^2)
  m2 <- Rfast::colSums(x)     ;     m22 <- Rfast::colsums(x^2)
  up <-  m1 * m2 / n
  down <- sqrt( (m12 - m1^2 / n) * (m22 - m2^2 / n) )
  sxy <- matrix(0, p, R)
  for (i in 1:R) {
    y1 <- sample(y, n)
    sxy[, i] <- Rfast::colsums(y1 * x)
  }
  rb <- (sxy - up) / down
  tb <- 0.5 * log( (1 + rb)/(1 - rb) )  ## the test statistic
  pvalue <- ( Rfast::rowsums( abs(tb) > abs(test) ) + 1 ) / (R + 1)
  res <- cbind( r, pvalue )
  colnames(res) <- c('correlation', 'p-value')
  if ( is.null(colnames(x)) ) {
    rownames(res) <- paste("X", 1:dim(x)[2], sep = "")
  } else  rownames(res) <- colnames(x)
  res
}
```

4.1.4 Partial correlation coefficient

Suppose you want to calculate the correlation coefficient between two variables controlling for the effect of (or conditioning on) one or more other variables. So you cant to calculate $\hat{\rho}(X, Y|\mathbf{Z})$, where $\mathbf{Z}$ is a matrix, since it does not have to be just one variable. This idea was captures by Ronald Fisher some years ago. To calculate it, one can use linear regression as follows.

1. Calculate the residuals $\hat{e}_x$ from the linear regression $X = a + bZ$.
2. Calculate the residuals $\hat{e}_y$ from the linear regression $Y = c + dZ$.
3. Calculate the correlation between $\hat{e}_x$and $\hat{e}_y$. This is the partial correlation coefficient between X and Y controlling for $\mathbf{Z}$.

The standard error of the Fisher's transformation of the sample partial correlation is

(Anderson, 2003)

$$\text{SE}\left(\frac{1}{2}\log\frac{1+\hat{\rho}\left(X,Y|\mathbf{Z}\right)}{1-\hat{\rho}\left(X,Y|\mathbf{Z}\right)}\right)=\frac{1}{n-d-3},$$

where n is the sample size and d is the number of variables upon which we control. The standard error is very similar to the one of the classical correlation coefficient. In fact, the latter one is a special case of the first when $d = 0$ and thus there is no variable whose effect is to be controlled. The R code below calculates the partial correlation coefficient, performs hypothesis testing and calculates confidence intervals.

```
partial.corr <- function(y, x, z, type = "pearson", rho = 0, a = 0.05, plot = F) {
  ## y and x are the two variables whose correlation is of interest
  ## z is a set of variable(s), one or more variables
  ## It accepts two types only, either "pearson" or "spearman"
  ## over which the condition takes place
  ## rho is the hypothesised correlation
  ## a is the significance level, set to 0.05 by default
  n <- length(y)  ## sample size
  d <- dim(z)[2]  ## dimensionality of z
  res <- resid( lm( cbind(y, x) ~ z ) )  ## residuals of y and x on z
  r <- cor(res, method = type)[1, 2]  ## partial correlation of y and
  ##   x conditioning on z
  zh0 <- 0.5 * log( (1 + rho) / (1 - rho) )  ## Fisher's transform for Ho
  zh1 <- 0.5 * log( (1 + r) / (1 - r) )  ## Fisher's transform for H1
  if (type == "pearson") {
    se <- 1/sqrt(n - d - 3)  ## standard error under Ho
  } else if ( type == "spearman" ){
      se <-  1.029563 / sqrt(n - d - 3)  ## standard error under Ho
  }
  test <- (zh1 - zh0)/se  ## test statistic
  pvalue <- 2 * ( 1 - pt( abs(test), n - d - 3 ) )  ## p-value
  zL <- zh1 - qt(1 - a/2, n - d - 3) * se
  zH <- zh1 + qt(1 - a/2, n -d - 3) * se
  fishL <- (exp(2 * zL) - 1)/(exp(2 * zL) + 1)  ## lower confidence limit
  fishH <- (exp(2 * zH) - 1)/(exp(2 * zH) + 1)  ## upper confidence limit
  ci <- c(fishL, fishH)
  names(ci) <- paste( c( a/2 * 100, (1 - a/2) * 100 ), "%", sep = "" )
  r0 <- seq( max(-0.99, r - 0.2), min(0.99, r + 0.2), by = 0.001 )
  z0 <- 0.5 * log( (1 + r0)/(1 - r0) )  ## Fisher's transformation
  ## for many Hos
```

```
  stat <- abs(zh1 - z0)/se  ## test statistics
  pval <- 2 * ( 1 - pt( abs(stat), n - d - 3 ) )  ## p-values
  if (plot) {
    par(mfrow = c(1, 2))
    plot(r0, stat, type = "l", xlab = "Correlation values",
    ylab = "Test statistic")
    abline( h = qt(0.975, n - d - 3), col = 2 )
    abline( v = min( r0[stat < qt(0.975, n - d - 3)] ), col = 3, lty = 3 )
    abline( v = max( r0[stat < qt(0.975, n - d - 3)] ), col = 3, lty = 3 )
    plot(r0, pval, type = "l", xlab = "Correlation values",
    ylab = "P-values")
    abline(h = a, col = 2)
    abline( v = min( r0[pval > a] ), col = 3, lty = 3 )
    abline( v = max( r0[pval > a] ), col = 3, lty = 3 )
  }
  result <- c(r, pvalue)
  names(result) <- c('correlation', 'p-value')
  list(result = result, ci = ci)
}
```

4.1.5 Matrix of partial correlation coefficients

Suppose you want to calculate the partial correlation matrix, where each coefficient has been conditioned on all the other variables. One way would be to use a *for* loop or a similar function and fill the matrix. Opgen-Rhein and Strimmer (2006) uses a much more convenient and faster way. This way is implemented in the corpcor package written by Schaefer et al. (2007). The option for a Spearman based partial correlation is now available. The steps of the calculation are described here

1. Calculate the correlation coefficient and then change the sign of all correlations.
2. Calculate the inverse of the previous matrix. (Schaefer et al. (2007) use the Moore-Penrose inverse for this purpose, but I don't. They have other things in mind, more general than these).
3. Turn the diagonal elements of the new matrix into positive.
4. Use the *cov2cor* function of R to make this matrix a correlation matrix.

```
pcor.mat <- function(x, type = "pearson") {
  ## x is a matrix with data
  ## type can be either "pearson" or "spearman"
```

```
  ## it can of course be "kendall" but I have not examined it
  ## in other functions
  r <- cor(x, method = type) ## correlation matrix of x
  r2 <-  - chol2inv( chol(r) )
  diag(r2) <-  -diag(r2)
  cov2cor(r2)
}
```

4.1.6 Hypothesis testing for two correlation coefficients

The test statistic for the hypothesis of equality of two correlation coefficients is the following:

$$Z = \frac{\hat{z}_1 - \hat{z}_2}{\sqrt{1/\left(n1 - 3\right) + 1/\left(n2 - 3\right)}},$$

where $\hat{z}_1$ and $\hat{z}_2$ denote the Fisher's transformation (4.1) applied to the two correlation coefficients and n_1 and n_2 denote the sample sizes of the two correlation coefficients. The denominator is the sum of the variances of the two coefficients and as you can see we used a different variance estimator than the one we used before. This function performs hypothesis testing for the equality of two correlation coefficients. The result is the calculated p-value from the standard normal distribution.

```
correl2 <- function(r1, r2, n1, n2, type = "pearson") {
  ## r1 and r2 are the two correlation coefficients
  ## n1 and n2 are the two sample sizes
  ## type can be either "pearson" or "spearman"
  z1 <- 0.5 * log( (1 + r1) / (1 - r1) )  ## Fisher's transformation
  z2 <- 0.5 * log( (1 + r2) / (1 - r2) )  ## Fisher's transformation

  if (type == "pearson") {
    test <- (z1 - z2) / sqrt( 1/(n1 - 3) + 1 / (n2 - 3) )  ## test statistic
  } else if (type == "spearman") {
    test <- (z1 - z2) / sqrt( 1.029563/(n1 - 3) + 1.029563 / (n2 - 3) )
  }

  pvalue <- 2 * pnorm( abs(test), lower.tail = FALSE )  ## p-value
  result <- c(test, pvalue)
  result <- c("test", "p-value")
  result

}
```

4.1.7 Squared multivariate correlation between two sets of variables

Mardia et al., 1979, pg. 171 defined two squared multiple correlation coefficient between the dependent variable $\mathbf{Y}$ and the independent variable $\mathbf{X}$. They mention that these are a similar measure of the coefficient determination in the univariate regression. Assume that the multivariate regression model (more in Section 4.2) is written as

$$\mathbf{Y} = \mathbf{XB} + \mathbf{U},$$

where $\mathbf{U}$ is the matrix of residuals. Then, they write $\mathbf{D} = \left(\mathbf{Y}^T\mathbf{Y}\right)^{-1}\hat{\mathbf{U}}^T\hat{\mathbf{U}}$, with $\hat{\mathbf{U}}^T\hat{\mathbf{U}} = \mathbf{Y}^T\mathbf{P}\mathbf{Y}$ and $\mathbf{P}$ is defined in (4.3). The matrix $\mathbf{D}$ is a generalization of $1 - R^2$ in the univariate case. Mardia et al., 1979, pg. 171 mentioned that the dependent variable $\mathbf{Y}$ has to be centred.

The squared multivariate correlation should lie between 0 and 1 and this property is satisfied by the trace correlation r_T and the determinant correlation r_D, defined as

$$r_T^2 = d^{-1}\text{tr}\left(\mathbf{I} - \mathbf{D}\right) \text{ and } r_D^2 = \det\left(\mathbf{I} - \mathbf{D}\right)$$

respectively, where d denotes the dimensionality of $\mathbf{Y}$. So, high values indicate high proportion of variance of the dependent variables explained. Alternatively, one can calculate the trace and the determinant of the matrix $\mathbf{E} = \left(\mathbf{Y}^T\mathbf{Y}\right)^{-1}\hat{\mathbf{Y}}^T\hat{\mathbf{Y}}$. Try something else also, use the *sq.correl* function in a univariate regression example and then calculate the R^2 for the same dataset. Try this example again but without centering the dependent variable in the *sq.correl* function. In addition, take two variables and calculate their squared correlation coefficient (*cor* and then square it) and using the function below.

```
sq.correl <- function(y, x) {
  ## y is the dependent variable
  ## x is the independent variable
  n <- dim(y)[1]  ## sample size
  d <- dim(y)[2]  ## dimensions
  y <- y - rep( Rfast::colmeans(y), rep(n, d) )  ## centering of Y
  YY <- crossprod(y)
  X <- cbind(1, x)
  U <- Rfast::lmfit(X, y)$residuals
  if ( !is.matrix(U) ) U <- matrix(U)
  UU <- crossprod(U)
  D <- solve(YY, UU)
  r2T <- mean( 1 - diag( D ) )
  r2D <- det( diag(d) - D )
  result <- c(r2T, r2D)
  names(result) <- c("Trace R-squared", "Determinant R-squared")
```

```
  result
}
```

4.2 Regression

4.2.1 Classical multivariate regression

In this function we assume that both the dependent and independent variables can either be vectors or matrices. The parameters of the independent variables are estimated through maximum likelihood estimation procedures and the final formula is the following

$$\hat{\mathbf{B}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}\mathbf{Y},$$

where $\mathbf{X}$ is the set of independent variables, or the design matrix, with the first column being the vector of 1_s and $\mathbf{Y}$ is the multivariate (or univariate) dependent variable. The covariance matrix of the estimated parameters is given by this formula

$$\hat{V}\left(\hat{\mathbf{B}}\right) = \hat{\mathbf{\Sigma}}_{\mathbf{e}} \otimes \left(\mathbf{X}^T\mathbf{X}\right)^{-1},$$

where $\hat{\mathbf{\Sigma}}_{\mathbf{e}} = \frac{1}{n-p-1}\mathbf{Y}^T\mathbf{P}\mathbf{Y}$ with

$$\mathbf{P} = \mathbf{I}_n - \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T \tag{4.3}$$

is the error covariance matrix. The sample size is denoted by n, p indicates the number of independent variables and $\otimes$ is the Kronecker product of two matrices.

In order to see if an observation is an outlier or leverage (influential) point several techniques have been suggested in the literature. We will use a simple graphical procedure. We will calculate the Mahalanobis distances of the residuals and of the observations in the X space

$$DE_i = \sqrt{\hat{\mathbf{e}}_i^T\hat{\mathbf{\Sigma}}_{\mathbf{e}}^{-1}\hat{\mathbf{e}}_i^T} \text{ and } DX_i = \sqrt{\left(\mathbf{X}_i - \hat{\boldsymbol{\mu}}_{\mathbf{X}}\right)^T\hat{\mathbf{\Sigma}}_{\mathbf{XX}}^{-1}\left(\mathbf{X}_i - \hat{\boldsymbol{\mu}}_{\mathbf{X}}\right)} \tag{4.4}$$

respectively, where $\hat{\mathbf{\Sigma}}_{\mathbf{e}}$ is the error covariance matrix as before and $\hat{\boldsymbol{\mu}}_{\mathbf{X}}$ and $\hat{\mathbf{\Sigma}}_{\mathbf{XX}}$ are the mean vector and covariance matrix of the independent variables respectively (without the constant). Let us denote by d the dimensionality of the dependent variables $\mathbf{Y}$ and by p the dimensionality of the independent variables $\mathbf{X}$. If DE_i is larger than $\sqrt{\chi^2_{d,0.975}}$ we will say the i-th dependent variable observation has a possible residual outlier. If DX_i is larger than $\sqrt{\chi^2_{p,0.975}}$ we will say that the i-th observation of the independent variables is a potential leverage point. This is to help us see graphically which observations seem to influence the regression parameters.

```
multivreg <- function(y, x, plot = TRUE, xnew = NULL) {
  ## y is the dependent variable and must be a matrix
  ## with at least two columns
  ## x contains the independent variable(s) which have to be
  ## in a matrix format or a vector if you have just one

  n <- dim(y)[1]  ## sample size
  d <- dim(y)[2]  ## dimensionality of y
  x <- as.matrix(x)
  p <- dim(x)[2]  ## dimensionality of x
  mod <- lm(y ~ x)   ## linear regression
  res <- resid(mod)  ## residuals
  s <- cov(res) * (n - 1) / (n - p - 1)
  sxx <- cov(x)  ## covariance of the independent variables
  dres <- sqrt( Rfast::mahala( res, numeric(d), s ) ) ## Mahalanobis distances
  ## of the residuals

  mx <- Rfast::colmeans(x)  ## mean vector of the independent variales
  dx <- sqrt( Rfast::mahala(x, mx, sxx) )  ## Mahalanobis distances
  ## of the independent variables
  crit.res <- sqrt( qchisq(0.975, d) )
  crit.x <- sqrt( qchisq(0.975, p) )

  if ( plot ) {
    plot(dx, dres, xlim = c(0, max(dx) + 0.5), ylim = c(0, max(dres) + 0.5),
         xlab = "Mahalanobis distance of x", ylab = "Mahalanobis distance
         of residuals")
    abline(h = crit.res)
    abline(v = crit.x)
  }

  resid.out <- as.vector( which(dres > crit.res) )
  x.leverage <- which(dx > crit.x)
  out.and.lever <- which(dx > crit.x & dres > crit.res)

  if ( is.null(xnew) ) {
    est <- fitted(mod)
  } else {
    xnew <- cbind(1, xnew)
```

```
    est <- xnew %*% coef(mod)
  }

  moda <- summary(mod)
  suma <- array( dim = c(1 + p, 6, d) )
  r.squared <- numeric(d)
  mse <- deviance(mod)/( n - p - 1 )

  for (i in 1:d) {
    wa <- as.matrix( coef(moda[[i]]) )
    wa <- cbind( wa, wa[, 1] - qt(0.975, n - p - 1) * mse[i] ,
                 wa[, 1] + qt(0.975, n - p - 1) * mse[i] )
    colnames(wa)[5:6] <- paste(c(2.5, 97.5), "%", sep = "")
    suma[, , i] <- wa
    r.squared[i] <- as.numeric( moda[[i]]$r.squared )
  }

  if ( is.null(colnames(y)) ) {
    dimnames(suma) <- list( rownames(wa), colnames(wa),
                            paste("Y", 1:d, sep = "") )
    names(r.squared) <- paste("Y", 1:d, sep = "")
    colnames(est) <- paste("Y", 1:d, sep = "")
  } else {
    dimnames(suma) <- list( rownames(wa), colnames(wa), colnames(y) )
    names(r.squared) <- colnames(y)
    colnames(est) <- colnames(y)
  }

  list(suma = suma, r.squared = r.squared, resid.out  = resid.out,
       x.leverage = x.leverage, out = out.and.lever, est = est)

}
```

Unfortunately, the function accepts only a design matrix (without the first line of ones). So, if you have categorical and continuous or categorical only independent variables then the next function would be useful. Suppose x1 consists of one or more continuous variables, i.e. it is either a vector or matrix and x2 is a categorical variable. Then you have to do the following thing in R to obtain the design matrix mat.

```
ff <- y ~ x1+ x2
```

```
m <- model.frame(ff)
mat <- model.matrix(ff, m)[, -1]
```

Then, you go to the next function and put mat in the place of x.

```
multivreg(y, mat)
```

Alternatively, if you have many independent variables and you cannot use the previous step you can use the next function. But, bear in mind that it does not calculate outliers in the independent variables space. The next function, being more general than *multivreg* offers bagging (bootstrap aggregation). Suppose you want to predict the values ($\mathbf{Y}$) of some new data $\mathbf{X}_{new}$ (which could be your observed data and in this case you obtain bagged fitted values). Note that in this case I do no standardization, so if you wanted to do standardization you would have to be careful (see for example k-NN regression, presented later).

The idea is simple, you bootstrap your observed data ($\mathbf{X}$, $\mathbf{Y}$) and fit the linear model on the bootstrapped sample. Use this model to obtain estimates $\hat{\mathbf{Y}}_b$ for the $\mathbf{X}_{new}$. Repeat this $B = 100$ or $B = 200$ times and take the average of the estimates $\hat{\mathbf{Y}} = \frac{\sum_{b=1}^{B} \hat{\mathbf{Y}}_b}{B}$. This idea is attributed to Breiman (1996). Note also, that the univariate R^2 values, the coefficients and their standard errors are calculated from the classical regression. Bootstrap aggregation is applied only for the predictions.

```
mvreg.cat <- function(y, x, xnew = x, B = 1) {
  ## y is the dependent variable and must be a matrix
  ## with at least two columns
  ## x contains the independent variable(s) which have to be
  ## in a matrix format or a vector if you have just one
  ## are there any new data whose y you want to predict?
  ## if xnew = x the fitted vlaues wil be returned
  ## If B = 1 no bagging is performed
  ## If you want bagging, B must be greater than 1, say 100 or 200

  n <- dim(y)[1]  ## sample size
  d <- dim(y)[2]  ## dimensionality of y
  p <- dim(x)[2]  ## dimensionality of
  mod <- lm(y ~ ., data = data.frame(x) )  ## linear regression
  if ( is.null(colnames(x)) )  colnames(x) <- paste("X", 1:p, sep = "")
  xnew <- as.data.frame(xnew)
  colnames(xnew) <- colnames(x)
    if (B == 1) {
      est <- predict(mod, xnew)  ## predict the xnew
    } else {
```

```
      esa <- array( dim = c(n, d, B) )
      for (i in 1:B) {
        ina <- sample(1:n, n, replace = TRUE)
        mod <- lm(y[ina, ] ~ ., data = x[ina, ])  ## linear regression
        esa[, , i] <- predict(mod, xnew)  ## predict the xnew
      }
     est <- apply(esa, 1:2, mean)
   }
  moda <- summary(mod)
  p <- nrow(coef(moda)[[1]])
  suma <- array(dim = c(p, 6, d))
  r.squared <- numeric(d)
  mse <- deviance(mod)/( n - p - 1 )
  for (i in 1:d) {
    wa <- as.matrix( coef(moda[[i]]) )
    wa <- cbind( wa, wa[, 1] - qt(0.975, n - p - 1) * mse[i],
    wa[, 1] + qt(0.975, n - p - 1) * mse[i] )
    colnames(wa)[5:6] <- paste(c(2.5, 97.5), "%", sep = "")
    suma[, , i] <- wa
    r.squared[i] <- as.numeric( moda[[i]]$r.squared )
  }
  if ( is.null(colnames(y)) ) {
    dimnames(suma) <- list( rownames(wa), colnames(wa),
    paste("Y", 1:d, sep = "") )
    names(r.squared) <- paste("Y", 1:d, sep = "")
    colnames(est) <- paste("Y", 1:d, sep = "")
  } else {
    dimnames(suma) <- list( rownames(wa), colnames(wa), colnames(y) )
    names(r.squared) <- colnames(y)
    colnames(est) <- colnames(y)
  }
  list(suma = suma, r.squared = r.squared, est = est)
}
```

4.2.2 *k*-NN regression

This is a non-parametric regression which depends only upon the distances among the independent variables. It involves a tuning, choice of a free parameter, whatever you want to call it. That is *k*, the number of nearest neighbours. Hence, *k*-NN stands for *k* nearest neigh-

bours. The dependent variable can be either univariate or multivariate, but the independent variables must be numerical, continuous.

The next code performs k-NN multivariate, or univariate if you have a univariate dependent variable, regression for a given value of k. At first it standardises the independent variables and then uses the same mean and standard deviation to scale the new (independent variables) observations as well. If the *xnew* argument in the function is the same as x, the fitted values will be returned.

```
knn.reg <- function(xnew, y, x, k = 5, type = "euclidean", estim = "arithmetic") {
  ## xnew is the new observation
  ## y is the multivariate or univariate dependent variable
  ## x contains the independent variable(s)
  ## k is the number of nearest neighbours to use
  ## type is for the distance, Euclidean or Manhattan distance.
  ## The function dist()  allows for more distance types
  ## which can of course use here.
  ## Type ?dist so see more
  ## estim is either 'arithmetic', 'harmonic'. How to calculate the
  ## estimated value of the Y using the arithmetic mean or the
  ## harmonic mean of the closest observations

  y <- as.matrix(y)
  d <- dim(y)[2]  ## dimensions of y
  p <- dim(x)[2]  ## dimensions of x
  n <- dim(y)[1]
  xnew <- as.matrix(xnew)
  xnew <- matrix(xnew, ncol = p)
  nu <- dim(xnew)[1]
  m <- Rfast::colmeans(x)
  s <- Rfast::colVars(x, std = TRUE)
  x <- t( ( t(X) - m ) / s ) ## standardize the independent variables
  ina <- 1:n

  if (p == 1) {
    xnew <- (xnew - m) / s
  } else {
    s <- diag(1/s)
    xnew <- ( xnew - rep(m, rep(nu, p)) ) %*% s  ## standardize the xnew values
  }
```

```
if (p == 1) {
  x <- as.matrix(x)
  x <- matrix(x, ncol = p)
  xnew <- as.matrix(xnew)
  xnew <- matrix(xnew, ncol = p)
}


apostasi <- dist(rbind(xnew, x), method = type, diag = TRUE, upper = TRUE)
apostasi <- as.matrix(apostasi)
est <- matrix(nrow = nu, ncol = dim(y)[2])
dis <- apostasi[1:nu, -c(1:nu)]
nam <- 1:n
est <- matrix(nrow = nu, ncol = d)

if (estim == "arithmetic") {
  for (i in 1:nu) {
    xa <- cbind(ina, disa[i, ])
    qan <- xa[order(xa[, 2]), ]
    a <- qan[1:k, 1]
    yb <- as.matrix( y[a, ] )
    est[i, ] <- Rfast::colmeans( yb )
  }

} else if (estim == "harmonic") {
  for (i in 1:nu) {
    xa <- cbind(ina, disa[i, ])
    qan <- xa[order(xa[, 2]), ]
    a <- qan[1:k, 1]
    yb <- as.matrix( y[a, ] )
    est[i, ] <- k / Rfast::colsums( yb )
  }
}

if ( is.null(colnames(y)) ) {
   colnames(est) <- paste("yhat", 1:d, sep = "" )
 } else  colnames(est) <- colnames(y)
if (d == 1)  est <- as.vector(est)
est
```

```
}
```

A cross validation algorithm to choose the value of k is described below and after that the relevant code is given below.

Since I am interested in prediction analysis I will use a K-fold cross-validation to choose the value of α. I split the data into K sets (fold). Every time I leave a set out and fit the model in the remaining sample (chose the best value of k and so on). Then, I scale the test set, using the mean and standard deviation of the training set, and calculate the MSPE in order to measure the performance. This is repeated for all K sets (folds) of data and the average MSPE is computed.

But, since many models are being tested at every time (each value of k gives a different model) the resulting performance is a bit biased, a bit overestimated. To overcome this, nested cross-validation (Aliferis et al., 2010, Statnikov et al., 2005, Tsamardinos et al., 2014) could be used, but since this is a computationally heavier design we rely on the method suggested by Tibshirani and Tibshirani (2009), termed hereafter as TT.

Calculate the best performance as the minimum of the average (over all folds) performance and keep the corresponding value of k which minimizes the performance. Call this k^*. For each fold extract the best performance and subtract from it the performance when using the best k^*. The estimated bias is the average of these differences. Finally, add this bias to the overall performance. The chosen, best value of k does not change, the estimated performance changes.

The function *knnreg.tune* has the two following two features. At first, for all different values of k, the training and test samples are always the same. Secondly, there is the option of *seed*. If it is true, then no matter how many times we repeat the analysis, the split of the folds is always the same and thus the results will be the same.

```
knnreg.tune <- function(y, x, M = 10, A = 10, ncores = 2,
  type = "euclidean", estim = "arithmetic", mat = NULL) {
  ## y is the multivariate (or univariate) dependent variable
  ## x contains the independent variables(s)
  ## M is the number of folds, set to 10 by default
  ## it is assumed that the training set contains at least 11 observations
  ## A is the highest number of nearest neighbours
  ## ncores specifies how many cores to use
  ## type is for the distance, Euclidean or Manhattan distance.
  ## The function dist()  allows for more distance types
  ## which can of course use here.
  ## Type ?dist so see more
  ## estim is either 'arithmetic', 'harmonic'. How to calculate the
  ## estimated value of the Y using the arithmetic mean or the
```

```
## harmonic mean of the closest observations.

y <- as.matrix(y)
n <- dim(y)[1]
d <- dim(y)[2]

if ( is.null(mat) ) {
  nu <- sample(1:n, min( n, round(n / M) * M ) )
  ## It may be the case this new nu is not exactly the same
  ## as the one specified by the user
  ## to a matrix a warning message should appear
  suppressWarnings()
  mat <- matrix( nu, ncol = M )
} else  mat <- mat

M <- dim(mat)[2]
rmat <- dim(mat)[1]
per <- matrix(nrow = M, ncol = A - 1)

if (ncores == 1) {
  for (vim in 1:M) {
    ytest <- as.matrix( y[mat[, vim], ] )  ## test set dependent vars
    ytrain <- as.matrix( y[-mat[, vim], ] )  ## train set dependent vars
    xtrain <- as.matrix( x[-mat[, vim], ] )  ## train set independent vars
    xtest <- as.matrix( x[mat[, vim], ] )  ## test set independent vars
    for ( l in 1:c(A - 1) ) {
      knn <- l + 1
      est <- knn.reg(xtest, ytrain, xtrain, knn, type = type, estim = estim)
      per[vim, l] <- sum( (ytest - est)^2 ) / rmat
    }
  }
} else {
  require(doParallel, quiet = TRUE, warn.conflicts = FALSE)
  cl <- makePSOCKcluster(ncores)
  registerDoParallel(cl)
  pe <- numeric(A - 1)
  per <- foreach(i = 1:M, .combine = rbind, .export = "knn.reg") %dopar% {
    ## will always be the same
    ytest <- as.matrix( y[mat[, i], ] )  ## test set dependent vars
```

```
      ytrain <- as.matrix( y[-mat[, i], ] )  ## train set dependent vars
      xtrain <- as.matrix( x[-mat[, i], ] )  ## train set independent vars
      xtest <- as.matrix( x[mat[, i], ] )  ## test set independent vars
      for ( l in 1:c(A - 1) ) {
        knn <- l + 1
        est <- knn.reg(xtest, ytrain, xtrain, knn, type = type, estim = estim)
        pe[l] <- sum( (ytest - est)^2 ) / rmat
      }
      return(pe)
    }
    stopCluster(cl)
  }

  mspe <- Rfast::colmeans(per)
  bias <- per[ ,which.min(mspe)] - apply(per, 1, min)  ## TT estimate of bias
  estb <- mean( bias )  ## TT estimate of bias
  names(mspe) <- paste("k=", 2:A, sep = "")
  plot(2:c(length(mspe) + 1), mspe, xlab = "Nearest neighbours",
  ylab = "MSPE", type = "b")
  names(mspe) <- 2:c(length(mspe) + 1)
  performance <- c( min(mspe) + estb, estb)
  names(performance) <- c("Estimated percentage", "Estimated bias")
  list(mspe = mspe, k = which.min(mspe) + 1, performance = performance)
}
```

If you take a look at the above code you will see that each time, for every training and test set splits I calculate the distance matrix. In the classification task using the k-NN, for compositional data (Section for compositional data) you will see that I calculate the distance matrix once and then I simply remove rows and columns corresponding to the test set. The reason for this here is the scaling. I scale (standardize each variable) the training set and then use those means and standard deviations to scale the test set. I am pretending that the test set contains new data for which I know nothing. If I scaled the whole dataset from the beginning that would induce positive bias, it would overestimate the performance of the regression. I do not want that, I want to test and train my regression algorithm as fairly and unbiasedly as possible.

4.2.3 Kernel regression

Kernel regression is another form of non parametric regression. But let us see what is the kernel. at first we will say that a good book for kernel density estimation is the Wand and

Jones (1995) one. The book might seem difficult for introduction but once you take the hand of it, then you appreciate its value. Another very good book is written by Tsybakov (2009).

The kernel function estimating the (univariate) density of a value has this form

$$f\left(\hat{x};h\right)=\frac{1}{nh}\sum_{i=1}^{n}K\left(\frac{X_i-x}{h}\right). \tag{4.5}$$

An example of a kernel function is the standard normal. Thus, (4.5) can be written as

$$f\left(\hat{x};h\right)=\frac{1}{nh\sqrt{2\pi}}\sum_{i=1}^{n}e^{-\frac{-\left(X_i-x\right)^2}{2h^2}}. \tag{4.6}$$

There are many kernel functions in the literature. For this reason we also use another one, which is based on the L_1 metric denoted as Laplacian kernel by Kim and Scott (2012)

$$f\left(\hat{x};h\right)=\frac{c}{nh}\sum_{i=1}^{n}e^{-\frac{|X_i-x|}{h}}, \tag{4.7}$$

where c is the normalizing constant of the Kernel function.

So if we want an estimate of the density at a point x we use all the sample points X_i ($i=1,\ldots,n$) and a smoothing parameter or bandwidth h. The h determines the smoothness of the final estimated density. k-NN is a case of kernel regression, where the kernel is a very simple one. If we have one independent variable, then we have only one h. If we have more than one independent variables (say p), the we have a $p\times p$ matrix bandwidth $\mathbf{H}$. Here for simplicity we will assume $\mathbf{H}=h\mathbf{I}_p$, where $\mathbf{I}_p$ is the $p\times p$ identity matrix.

We want to do this kernel density estimation in the multivariate case when covariates are present. So, we want to estimate the dependent variable values with out using any regression coefficients. The formula to estimate the i-th dependent variable value is

$$\hat{m}\left(x,p,h\right)=\mathbf{e}_1^T\left[\mathbf{X}^T\left(\mathbf{x},p\right)\mathbf{W}_{\mathbf{x}}\mathbf{X}\left(\mathbf{x},p\right)\right]^{-1}\mathbf{X}^T\left(\mathbf{x},p\right)\mathbf{W}_{\mathbf{x}}\mathbf{Y}. \tag{4.8}$$

Let us now see what are all these matrices. The $\mathbf{Y}$ is the $n\times q$ dependent variables matrix, where q denotes the dimensionality of $\mathbf{Y}$. The $\mathbf{W}_{\mathbf{x}}$ is an $n\times n$ diagonal matrix containing the kernel functions for all the observations

$$\mathbf{W}_{\mathbf{x}}=\text{diag}\left\{K\left(\frac{\mathbf{X}_1-\mathbf{x}}{h}\right),\ldots K\left(\frac{\mathbf{X}_n-\mathbf{x}}{h}\right)\right\}.$$

$\mathbf{X}(\mathbf{x}, p)$ is a $n \times (p+1)$ matrix of the independent variables defined as

$$\mathbf{X}(\mathbf{x}, p) = \begin{bmatrix} 1 & \mathbf{X}_1 - \mathbf{x} & (\mathbf{X}_1 - \mathbf{x})^2 & \dots & (\mathbf{X}_1 - \mathbf{x})^p \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & \mathbf{X}_n - \mathbf{x} & (\mathbf{X}_n - \mathbf{x})^2 & \dots & (\mathbf{X}_n - \mathbf{x})^p \end{bmatrix}.$$

We subtract the value $\mathbf{x}$ from every independent variable and all the sample values. Then we decide on the degree p of the local polynomial. For this reason kernel regression is also called local polynomial regression. The polynomial is applied locally to each point whose dependent variable we want to estimate. In my R function I allow only for $p = 0$ and $p = 1$, because I think it gets too complicated afterwards, especially as the number of variables increases.

If $p = 0$ then we end up with the Nadaraya-Watson estimator (Nadaraya, 1964, Watson, 1964) and in this case (4.8) can also be written as (Tsybakov, 2009)

$$\hat{m}(x, 0, h) = \frac{\sum_{i=1}^{n} K\left(\frac{\mathbf{X}_i - \mathbf{x}}{h}\right) \mathbf{Y}_i}{\sum_{i=1}^{n} K\left(\frac{\mathbf{X}_i - \mathbf{x}}{h}\right)} \text{ if } \sum_{i=1}^{n} K\left(\frac{\mathbf{X}_i - \mathbf{x}}{h}\right) \neq 0$$

and $\hat{m}(x, 0, h) = 0$ if $\sum_{i=1}^{n} K\left(\frac{\mathbf{X}_i - \mathbf{x}}{h}\right) = 0$.

Another key thing we have to note is the choice of the bandwidth h. Since we are in the multivariate case the bandwidth is a $q \times q$ matrix $\mathbf{H}$ having many smoothing parameters if we think that even for $q = 2$ we need 4 smoothing parameters. To keep it simple I made it $\mathbf{H} = h^2 \mathbf{I}_q$, where $\mathbf{I}_q$ is the identity matrix. Thus the kernel functions (4.6) and (4.7) are written as

$$f(\hat{\mathbf{x}}; h) = \frac{1}{nh^d (2\pi)^{d/2}} \sum_{i=1}^{n} e^{-\frac{-\|\mathbf{X}_i - \mathbf{x}\|^2}{2h^2}} \text{ and } f(\hat{x}; h) = \frac{c}{nh^d} \sum_{i=1}^{n} e^{-\frac{\|\mathbf{X}_i - \mathbf{x}\|}{h}}$$

respectively, where $\|.\|$ stands for the Euclidean metric and $\|\mathbf{x} - \mathbf{y}\|_1 = \sum_{i=1}^{d} |x_i - y_i|$. Since we are doing regression, note that the part which is outside the two sums cancels out.

Standardization of the independent variables is a must I would say, and so I did here. This means, that this functions allows only for continuous variables. The next code performs local polynomial regression for a given polynomial which I restrict it to be at most 1. It estimates the value of the dependent variable (univariate or multivariate) based on measurements from the continuous (only) independent variable(s). At first it standardises the independent variables and then uses the same mean and standard deviation to scale the new (independent variables) observations as well. This was also done by Lagani et al. (2013).

```
kern.reg <- function(xnew, y, x, h = seq(0.1, 1, length = 10), type = "gauss") {
  ## y is the multivariate (or univariate) dependent variable
```

```
## X contains the independent variable(s)
## x are the new independent variable(s) values
## h is the bandwidth
## r is the degree of the local polynomial.
## r is set by default to 0. This corresponds to Nadaraya-Watson estimator
## type denotes the type of kernel to be used, 'gauss' or 'laplace'

y <- as.matrix(y)
x <- as.matrix(x)
xnew <- as.matrix(xnew)
d <- dim(y)[2]
p <- dim(x)[2]
n <- dim(y)[1]
nu <- dim(xnew)[1]
m <- Rfast::colmeans(x)
s <- Rfast::colVars(x, std = TRUE)
x <- t( ( t(x) - m ) / s )  ## standardize the independent variables
xnew <- t( ( t(xnew) - m ) / s )  ## standardize the x values

if ( length(h) == 1 ) {
  if (type == "gauss") {
    a1 <- 0.5 * Rfast::dista(xnew, x)^2/h^2
  } else  a1 <- Rfast::dista(xnew, x, type = "manhattan" )/h
  z <- exp(-a1)

  ta <- Rfast::rowsums(z)
  mhx <- ( z %*% y) / ta
  mhx[ is.na(mhx) ] <- 0

  if ( is.null(colnames(y)) ) {
    colnames(mhx) <- paste("yhat", 1:d, sep = "" )
  } else  colnames(mhx) <- colnames(y)
  if  ( d == 1 )  mhx <- as.vector(mhx)

} else {
  len <- length(h)

  if (type == "gauss") {
    a1 <- 0.5 * Rfast::dista(xnew, x, square = TRUE)
```

```
      h <- h^2
    } else  a1 <- Rfast::dista(xnew, x, type = "manhattan" )

    if ( d == 1 ) {
      mhx <- matrix(nrow = nu, ncol = len)
      for (i in 1:len) {
        z <- exp( -a1 / h[i] )
        ta <- Rfast::rowsums(z)
        mhx[, i] <- ( z %*% y) / ta
        z <- NULL
      }
      mhx[ is.na(mhx) ] <- 0
      colnames(mhx) <- paste("h=", h, sep = "")

    } else {
      names <- paste("h=", h, sep = "")
      mhx <- sapply(names, function(x) NULL)
      for (i in 1:len) {
        z <- exp( -a1 / h[i] )
        ta <- Rfast::rowsums(z)
        mhx[[ i ]] <- ( z %*% y) / ta
        z <- NULL
      }
    }

  }

  mhx
}
```

My way to choose h is rather simple but it works. I use 1-fold cross validation in almost the same manner that was described in the k-NN multivariate regression before. Instead of choosing a value of k I choose a value of h and the algorithm contains more repetitions. But apart from this, all the other steps are the same. The next code chooses the value of h for a given local polynomial. This means, that one can change the order of the polynomial and see if the MSPE is reduced.

If the option *seed* is true, then no matter how many times we repeat the analysis, the spit between training and test samples is always the same and thus the results will be the same. The same seed number is used in the functions *knn.tune* and *pcr.tune*. Thus, the MSPE for all three methods is directly comparable.

```
kernreg.tune <- function(y, x,  h = seq(0.1, 1, length = 10), type = "gauss",
                nfolds = 10, folds = NULL, seed = NULL, graph = FALSE, ncores = 1) {

  if ( is.matrix(y) )  {
    n <- dim(y)[1]
    D <- dim(y)[2]
  } else {
    n <- length(y)
    D <- 1
  }

  runtime <- proc.time()
  if ( !is.matrix(x) )  x <- as.matrix(x)
  ina <- 1:n
  if ( is.null(folds) )  folds <- Compositional::makefolds(ina, nfolds = nfolds,
                                  stratified = FALSE, seed = seed)
  nfolds <- length(folds)
  msp <- matrix( nrow = length(folds), ncol = length(h) )

  if (ncores == 1) {
    for (i in 1:nfolds) {
      nu <- folds[[ i ]]
      if (D > 1)  {
        ytrain <- y[-nu, , drop = FALSE]
        ytest <- y[nu, , drop = FALSE]
      } else {
        ytest <- y[nu]
        ytrain <- y[-nu]
      }
      xtest <- x[nu, , drop = FALSE]
      xtrain <- x[-nu, , drop = FALSE]
      est <- Compositional::kern.reg(xtest, ytrain, xtrain, h, type = type)
      if (D > 1) {
        for ( i in 1:length(h) )  mspi[i, ] <- mean( est[[ i ]] - ytest)^2
      } else  msp[i, ] <- Rfast::colmeans( (ytest - est)^2 )
    }

  } else {
```

```
    requireNamespace("doParallel", quietly = TRUE, warn.conflicts = FALSE)
    cl <- parallel::makePSOCKcluster(ncores)
    doParallel::registerDoParallel(cl)
    pe <- numeric( length(h) )
    msp <- foreach(i = 1:nfolds, .combine = rbind, .export = "kern.reg",
                   .packages = "Rfast") %dopar% {
      nu <- folds[[ i ]]
      if (D > 1)  {
        ytrain <- y[-nu, , drop = FALSE]
        ytest <- y[nu, , drop = FALSE]
      } else {
        ytest <- y[nu]
        ytrain <- y[-nu]
      }
      xtest <- x[nu, , drop = FALSE]
      xtrain <- x[-nu, , drop = FALSE]
      est <- kern.reg(xtest, ytrain, xtrain, h, type = type)
      if (D > 1) {
        for ( i in 1:length(h) )  pe[i] <- mean( est[[ i ]] - ytest)^2
      } else  msp[i, ] <- Rfast::colmeans( (ytest - est)^2 )
      return(pe)
    }
    parallel::stopCluster(cl)
  }

  runtime <- proc.time() - runtime

  mspe <- Rfast::colmeans(msp)
  names(mspe) <- paste("h=", h, sep = "")
  if (graph) {
    plot(h, mspe, xlab = "Bandwidth parameter (h)", ylab = "MSPE", cex.lab = 1.2, cex.axi
    abline(v = h, col = "lightgrey", lty = 2)
    abline(h = seq(min(mspe), max(mspe), length = 10), col = "lightgrey", lty = 2)
    points(h, mspe, pch = 19)
    lines(h, mspe, lwd = 2)
  }

  list(mspe = mspe, h = h[ which.min(mspe) ], performance = min(mspe), runtime = runtime)
}
```

4.2.4 Principal components regression

I decided to put this technique here (and not in a subsequent Section), in the regression context since principal components analysis is used as a tool for regression. In some, the idea is that one can use principal component analysis on the independent variables in a unidimensional (dependent variable is univariate) regression setting. A good reason to do so is either because there is a high number of independent variables and or because there are collinearity problems. One or more of the continuous independent variables are highly correlated other variables. This method has however some limitations (see for example Hadi and Ling, 1998).

The algorithm to perform principal components regression can be described as follows

1. At first standardize the independent variables. This way, $\mathbf{X}^T\mathbf{X}$ (the $n \times p$ design matrix, which includes the p independent variables but not the intercept term) is proportional to the the correlation matrix for the predictor variables. This is what **?** does. The n stands for the sample size.

2. Perform eigen analysis on $\mathbf{X}^T\mathbf{X}$ and calculate the matrix of the eigenvectors $\mathbf{V}$ and the scores $\mathbf{Z} = \mathbf{XV}$.

3. Estimate the regression coefficients by

$$\hat{\mathbf{B}} = \mathbf{V}\left(\mathbf{Z}^T\mathbf{Z}\right)^{-1}\mathbf{Z}^T\mathbf{y},$$

   where $\mathbf{y}$ is the vector containing the values of the dependent variable.

4. Estimate the covariance matrix of the estimated regression coefficients by

$$Var\left(\hat{\mathbf{B}}\right) = \sigma^2\mathbf{V}\left(\mathbf{Z}^T\mathbf{Z}\right)^{-1}\mathbf{V}^T,$$

   where σ^2 is the conditional variance of the dependent variable calculated from the classical multiple regression analysis based upon the given number of principal components. It is the error variance, whose estimate is the (unbiased) mean squared error.

The key point is that we can have p different sets of estimated regression coefficients, since we can use the first eigenvector (or principal component), the first two eigenvectors or all of them. If we use all of them, then we end up with the same regression coefficients as if we performed a classical multiple regression analysis. Below we provide a code to perform principal component regression using from one to all the principal components and each time the following objects are calculated: estimated regression coefficients, their corresponding standard errors, mean squared error (also plotted), adjusted R^2 (also plotted). Note, that the fitted values are calculated in the usual way, multiplying the independent variables (and

not the principal component scores) by their corresponding coefficients adding the mean of the values of the dependent variable.

In addition, I have an option of estimation of new X values. If the new X values are the same as the observed ones, then the classical fitted values will be returned. Note, that the new X are scaled using the mean and standard deviation of the observed X values just like I did in the kernel regression (function *kern.reg*).

```
pcr <- function (y, x, k, xnew = NULL) {
  ## xnew is the new independent variables values
  ## whose values of y you want to estimate
  ## by default xnew is the x, so you will get the fitted values
  ## y is the univariate dependent variable
  ## x contains the independent variables
  ## k shows the number of components to keep
    dm <- dim(x)
    n <- dm[1]   ;   p <- dm[2]
    if  ( length(k) == 1 )  {
      k1 <- 1:k
    } else k1 <- k
    m <- Rfast::colmeans(x)
    s <- Rfast::colVars(x, suma = n * m, std = TRUE)
    x <- t( ( t(x) - m ) / s )

    #eig <- prcomp(x, center = FALSE, scale = FALSE)
    #values <- eig$sdev^2
    #vec <- eig$rotation[, k1, drop = FALSE]
    #z <- eig$x[, k1, drop = FALSE]

    eig <- Rfast2::pca(x, center = FALSE, scale = FALSE,
                       k = max(k1), vectors = TRUE)
    values <- eig$values
    per <- cumsum( values ) / p  ## cumulative eigenvalue proportion
    vec <- eig$vectors
    z <- x %*% vec  ## PCA scores
    zzk <- 1 / Rfast::colsums(z^2)
    com <- ( zzk * crossprod(z, y) )
    a <- t(vec) * as.vector(com)
    be <- t( Rfast::colCumSums(a) )  ## PCA based coefficients
    est <- NULL
    if ( length(k) == 1 )   be <- be[, k, drop = FALSE]
```

```
    if (!is.null(xnew)) {
      xnew <- matrix(xnew, ncol = p)
      xnew <- t( ( t(xnew) - m ) / s )
      est <- xnew %*% be  ## predicted values for PCA model
    }
    nam <- colnames(x)
    if ( is.null(nam) )  nam <- paste("X", 1:p, sep = "")
    rownames(be) <- nam
    colnames(be) <- paste("PC", k1, sep = "")
    if ( !is.null(est) ) colnames(est) <- paste("PC", k1, sep = "")
    list(be = be, per = per[k1], vec = vec, est = est)
}
```

The next function is more for a visualization and exploration, rather than inference. It shows the adjusted R^2 values and the cumulative proportion of the eigenvalues as a function of the number of principal components.

```
pcr.plot <- function(y, x) {
  ## y is the UNIVARIATE dependent variable
  ## x contains the independent variables
  ## k shows the number of components to keep

  y <- as.vector(y)
  m <- mean(y)
  y <- y - m  ## standardize the dependent variable
  n <- dim(x)[1]
  p <- dim(x)[2]

  x <- Rfast::standardise(x)  ## standardize the independent variables
  eig <- eigen( crossprod(x), symmetric = TRUE )  ## eigen analysis of the design matrix
  values <- eig$values  ## eigenvalues
  r2 <- per <- cumsum( values/sum(values) )  ## cumulative prop of eigenvalues
  vec <- eig$vectors  ## eigenvectors, or principal components
  z <- x %*% vec  ## PCA scores
  yhat <- matrix(nrow = n, ncol = p)

  for (i in 1:p){
    zzk <- crossprod( z[, 1:i] )
    b <- vec[, 1:i] %*% solve( zzk, crossprod( z[, 1:i], y ) )
    yhat[, i] <- as.vector( m + x %*% b )
```

```
    r2[i] <- 1 - (n - 1)/(n - i - 1) * (1 - cor(y, yhat[, i])^2)
  }

  plot( 1:p, r2, ylim = c(0, 1), type = 'b', xlab = 'Number
  of principal components', ylab = 'Adjusted R-squared and
  proportion of eigenvalues')
  points( 1:p, per, col = 2 )
  lines( 1:p, per, col = 2, lty = 2 )
  legend(p-2, 0.4, c(expression(paste("Adjusted", R^2, sep = " ")),
  expression( lambda[i]/sum(lambda[i]) ) ), col = c(1, 2),
  lty = c(1, 2), pch = c(1, 1))

  result <- rbind(r2, per)
  rownames(result) <- c('Adjusted R-squared', 'Cum prop')
  colnames(result) <- paste('PC', 1:p, sep = ' ')
  result

}
```

We saw how to perform principal component regression. We can choose the number of principal components based on the maximum adjusted R^2 value or the minimized mean squared error. If no maximum or minimum is met, we can keep the number of components after which these quantities do not change significantly. Alternatively we can use an m-fold cross validation with the TT estimate of bias.

If the option *seed* is true, then no matter how many times we repeat the analysis, the spit between training and test samples is always the same and thus the results will be the same. The same seed number is used in the functions *knn.tune* and *kern.tune*. Thus, the MSPE for all three methods is directly comparable.

```
pcr.tune <- function(y, x, nfolds = 10, maxk = 50, folds = NULL,
  ncores = 1, seed = FALSE, graph = TRUE) {
  ## y is the univariate dependent variable
  ## x contains the independent variables(s)
  ## M is the number of folds, set to 10 by default
  ## maxk is the maximum number of eigenvectors to conside
  ## ncores specifies how many cores to use
  n <- length(y)  ## sample size
  p <- dim(x)[2]  ## number of independent variables
  if ( maxk > p )  maxk <- p  ## just a check
  if ( is.null(folds) )
```

```
folds <- Compositional::makefolds(y, nfolds = nfolds,
         stratified = FALSE, seed = seed)
nfolds <- length(folds)

if (ncores <= 1) {

  runtime <- proc.time()
  msp <- matrix( nrow = nfolds, ncol = maxk )
  for (vim in 1:nfolds) {
    ytest <- y[ folds[[ vim ]] ]  ## test set dep vars
    ytrain <- y[ -folds[[ vim ]] ]   ## train set dep vars
    xtrain <- x[ -folds[[ vim ]], , drop = FALSE] ## train set indep vars
    xtest <- x[ folds[[ vim ]], , drop = FALSE]  ## test set indep vars
    est <- pcr(ytrain, xtrain, k = 1:maxk, xnew = xtest)$est
    msp[vim, ] <- Rfast::colmeans( (est - ytest)^2 )
  }

  runtime <- proc.time() - runtime

} else {

  runtime <- proc.time()

  cl <- parallel::makePSOCKcluster(ncores)
  doParallel::registerDoParallel(cl)
  er <- numeric(maxk)
  if ( is.null(folds) )
  folds <- Compositional::makefolds(y, nfolds = nfolds,
           stratified = FALSE, seed = seed)
  msp <- foreach::foreach(vim = 1:nfolds, .combine = rbind,
  .packages = c("Rfast", "Compositional") ) %dopar% {
    ytest <-  y[ folds[[ vim ]] ]  ## test set dependent vars
    ytrain <- y[ -folds[[ vim ]] ]   ## train set dependent vars
    xtrain <- x[ -folds[[ vim ]], , drop = FALSE]   ## train set indep vars
    xtest <- x[ folds[[ vim ]], , drop = FALSE]  ## test set indep vars
    est <- pcr(ytrain, xtrain, k = 1:maxk, xnew = xtest)$est
    er <- Rfast::colmeans( (est - ytest)^2 )
    return(er)
  }
```

```
    parallel::stopCluster(cl)

    runtime <- proc.time() - runtime
  }

  mspe <- Rfast::colmeans(msp)
  if ( graph )  plot(1:maxk, mspe, xlab = "Number of principal components",
  ylab = "MSPE", type = "b", cex.lab = 1.3)
  names(mspe) <- paste("PC", 1:maxk, sep = " ")
  performance <- min(mspe)
  names(performance) <- "MSPE"
  list(msp = msp, mspe = mspe, k = which.min(mspe),
  performance = performance, runtime = runtime)
}
```

4.2.5 Principal components regression for binary and count data

I decided to include the binary logistic and poisson regression in the principal components regression. The function below does the same as *pcr* but with binary or count data (dependent variable). In the case of count data, there is the option for the poisson distribution. In any case the independent variables must be continuous. For the case of logistic regression the reader is addressed to (Aguilera et al., 2006).

```
glm.pcr <- function(y, x, k = 1, xnew = NULL) {
  ## y is either a binary variable 0, 1 (binomial) or
  ## a numerical variable with counts (poisson)
  ## x contains the independent variables
  ## k shows the number of components to keep
  ## oiko can be "binomial" or "poisson"
  y <- as.vector(y)
  n <- dim(x)[1]
  p <- dim(x)[2]
  m <- Rfast::colmeans(x)

  x <- Rfas::standardise(x)   ## standardize the independent variables
  eig <- eigen( crossprod(x), symmetric = TRUE )  ## eigen analysis of the design matrix
  values <- eig$values  ## eigenvalues
  per <- cumsum( values / sum(values) )  ## cumulative proportion of eigenvalues
  vec <- eig$vectors  ## eigenvectors, or principal components
  z <- x %*% vec  ## PCA scores
```

```
    if ( length(unique(y)) == 2 ) {
    oiko <- "binomial"
  } else oiko <- "poisson"

  mod <- glm(y ~ z[, 1:k], family = oiko )
  b <- coef(mod)
  be <- vec[, 1:k] %*% as.matrix( b[-1] )

  if ( !is.null(xnew) ) {
    xnew <- matrix(xnew, ncol = p)
    s <- Rfast::colVars(x, std = TRUE)
    xnew <- t( ( t(xnew) - m ) / s ) ## standardize the xnew values
    es <- as.vector( xnew %*% be ) + b[1]
  } else  es <- as.vector( x %*% be ) + b[1]

  if (oiko == "binomial") {
    est <- as.vector(  exp(es) / (1 + exp(es))  )
  } else est <- as.vector( exp(es) )    ## fitted values for PCA model

  list(model = summary(mod), per = per[k], est = est)
}
```

Again a plot to visualize the principal components regression. It shows the deviance, the percentage of the drop in the deviance and the cumulative proprtion of the eigenvalues as a function of the number of principal components.

```
glmpcr.plot <- function(y, x) {
  ## y is either a binary variable 0, 1 (binomial) or
  ## a numerical variable with counts (poisson)
  ## x contains the independent variables
  ## k shows the number of components to keep
  ## oiko can be 'binomial' or 'poisson'
  y <- as.vector(y)
  n <- dim(x)[1]
  p <- dim(x)[2]
  x <- Rfas::standardise(x)   ## standardize the independent variables
  eig <- eigen( crossprod(x), symmetric = TRUE )  ## eigen analysis of the design matrix
  values <- eig$values  ## eigenvalues
  per <- cumsum( values / sum(values) )  ## cumulative proportion of eigenvalues
```

```
  vec <- eig$vectors  ## eigenvectors, or principal components
  z <- x %*% vec  ## PCA scores
  devi <- numeric(p)

  if ( length(unique(y)) == 2 ) {
    oiko <- "binomial"
  } else oiko <- "poisson"

  for (i in 1:p){
    mod <- glm(y ~ z[, 1:i], family = oiko )
    b <- coef(mod)
    be <- vec[, 1:i] %*% as.matrix( b[-1] )
    es <- as.vector( x %*% be ) + b[1]
    if (oiko == "binomial") {
      est <- as.vector(  exp(es) / (1 + exp(es))  )
      devi[i] <-  -2 * sum( y * log(est) + (1 - y) * log(1 - est) )

    } else {
      est <- as.vector( exp(es) )    ## fitted values for PCA model
      devi[i] <- 2 * sum( y * log( y / est ), na.rm = TRUE )
      ## fitted values for the PCA model
    }
  }
  dev <- (mod$null.deviance - devi)/mod$null.deviance

  plot(1:p, dev, ylim = c(0, 1), type = 'b', xlab =
  'Number of principal components', ylab =
  '% of deviance drop and proportion of eigenvalues')
  points( 1:p, per, col = 2 )
  lines( 1:p, per, col = 2, lty = 2 )
  legend(p-3, 0.4, c('% of deviance drop',
  expression(lambda[i]/sum(lambda[i])) ),
  col = c(1, 2), lty = c(1, 2), pch = c(1, 1))

  result <- rbind(devi, dev, per)
  rownames(result) <- c('Deviance', '% of deviance drop', 'Cumul prop')
  colnames(result) <- paste('PC', 1:p, sep = ' ')
  result
}
```

In order to tune the the number of principal components we want, we will use cross validation. It is important now to define the error function we use. For example, in the univariate case we saw before that is the mean squared error of prediction (MSPE). That is, the sum of squares of the residuals divided by the test sample size. We will do the same thing here, but, instead of the classical residuals we will calculate the deviance residuals whose form depends upon the distribution used.

- For the binomial distribution the deviance residuals are defined as

$$r_i = s_i \sqrt{-2 \left[y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i) \right]},$$

  where $s_i = 1$ if $y_i = 1$ and $s_i = -1$ if $y_i = 0$.

- For the Poisson distribution the deviance residuals are defined as

$$r_i = \text{sign} \left(y_i - \hat{y}_i \right) \sqrt{2 \left[y_i \log \frac{y_i}{\hat{y}_i} - (y_i - \hat{y}_i) \right]},$$

  where the *sign* operation indicates the sign of a number.

```
glmpcr.tune <- function(y, x, nfolds = 10, maxk = 10, folds = NULL, ncores = 1,
                        seed = FALSE, graph = TRUE) {
  ## y is the UNIVARIATE dependent variable
  ## y is either a binary variable (binary logistic regression)
  ## or a discrete variable (Poisson regression)
  ## x contains the independent variables
  ## fraction denotes the percentage of observations
  ## to be used as the test set
  ## the 1-fraction proportion of the data will be the training set
  ## R is the number of cross validations
  ## if ncores==1, then 1 processor is used, otherwise more are
  ## used (parallel computing)
  n <- dim(x)[1]
  p <- dim(x)[2]
  if ( maxk > p ) maxk <- p  ## just a check
  if ( is.null(folds) ) {
   folds <- Compositional::makefolds(y, nfolds = nfolds,
                                     stratified = FALSE, seed = seed)
  }
  nfolds <- length(folds)
  msp <- matrix( nrow = nfolds, ncol = maxk )
  ## deigma will contain the positions of the test set
```

```
## this is stored but not showed in the end
## the user can access it though by running
## the commands outside this function
if ( length( Rfast::sort_unique(y) ) == 2 ) {
  oiko <- "binomial"
} else oiko <- "poisson"

if (ncores <= 1) {
  runtime <- proc.time()
  for (vim in 1:nfolds) {
    ytest <- y[ folds[[ vim ]] ]   ## test set dependent vars
    ytrain <- y[ -folds[[ vim ]] ]   ## train set dependent vars
    xtrain <- x[ -folds[[ vim ]], , drop = FALSE]   ## train set independent vars
    xtest <- x[ folds[[ vim ]], , drop = FALSE ]  ## test set independent vars
  vec <- prcomp(xtrain, center = FALSE)$rotation
    z <- xtrain %*% vec  ## PCA scores

    for ( j in 1:maxk) {
      if (oiko == "binomial") {
        be <- Rfast::glm_logistic(z[, 1:j], ytrain)$be
      } else {
        be <- Rfast::glm_poisson(z[, 1:j], ytrain)$be
      }
      ztest <- xtest %*% vec[, 1:j, drop = FALSE]  ## PCA scores
      es <- as.vector( ztest %*% be[-1] ) + be[1]

      if (oiko == "binomial") {
        est <- as.vector(  exp(es) / ( 1 + exp(es) )  )
        ri <-  -2 *( ytest * log(est) + (1 - ytest) * log(1 - est) )
      } else {
        est <- as.vector( exp(es) )
        ri <- 2 * ytest * log(ytest / est)
      }
      msp[vim, j] <- sum( ri, na.rm = TRUE )
    }
  }
  runtime <- proc.time() - runtime

} else {
```

```
runtime <- proc.time()
cl <- parallel::makePSOCKcluster(ncores)
doParallel::registerDoParallel(cl)
er <- numeric(maxk)
if ( is.null(folds) ) {
  folds <- Compositional::makefolds(y, nfolds = nfolds,
                                    stratified = FALSE, seed = seed)
}
msp <- foreach::foreach(vim = 1:nfolds, .combine = rbind,
  .packages = "Rfast", .export = c("glm_logistic", "glm_poisson") ) %dopar% {
  ytest <- y[ folds[[ vim ]] ]  ## test set dependent vars
  ytrain <-  y[ -folds[[ vim ]] ]   ## train set dependent vars
  xtrain <- x[ -folds[[ vim ]], , drop = FALSE]   ## train set independent vars
  xtest <- x[ folds[[ vim ]], , drop = FALSE]  ## test set independent vars
vec <- prcomp(xtrain, center = FALSE)$rotation
  z <- xtrain %*% vec  ## PCA scores

  for ( j in 1:maxk) {
    if (oiko == "binomial") {
      be <- Rfast::glm_logistic(z[, 1:j], ytrain)$be
    } else {
      be <- Rfast::glm_poisson(z[, 1:j], ytrain)$be
    }
    ztest <- xtest %*% vec[, 1:j, drop = FALSE]  ## PCA scores
    es <- as.vector( ztest %*% be[-1] ) + be[1]

    if (oiko == "binomial") {
      est <- exp(es) / ( 1 + exp(es) )
      ri <-  -2 *( ytest * log(est) + (1 - ytest) * log(1 - est) )
    } else {
      est <- exp(es)
      ri <- 2 * ytest * log(ytest / est)
    }
    er[j] <- sum( ri, na.rm = TRUE )
  }
  return(er)
}
stopCluster(cl)
runtime <- proc.time() - runtime
```

```
  }

  mpd <- Rfast::colmeans(msp)
  if ( graph )  plot(1:maxk, mpd, xlab = "Number of principal components",
  ylab = "Mean predicted deviance", type = "b" ,cex.alb = 1.3)

  names(mpd) <- paste("PC", 1:maxk, sep = " ")
  performance <- min(mpd)
  names(performance) <- "MPD"
  list(msp = msp, mpd = mpd, k = which.min(mpd), performance = performance,
  runtime = runtime)
}
```

4.2.6 Ridge regression

Ridge regression in the univariate case can be described as follows: minimize the sum of the squared residuals subject to the sum of the squared beta coefficients is less than a constant

$$min \left\{ \sum_{i=1}^{n} y_i - \alpha - \sum_{j=1}^{p} \beta_j x_j \right\} \text{ subject to } \lambda \sum_{j=1}^{p} \beta_j^2 \leq s,$$

where n and p denote the sample size and the number of independent variables respectively. If we do the derivatives by hand the formula for the beta coefficients is

$$\hat{\boldsymbol{\beta}}^{ridge} = \left(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_p \right)^{-1} \mathbf{X}^T \mathbf{y},$$

where $\mathbf{X}$ contains the independent variables **only**, the first column is **not** the column of 1s. It becomes clear that if $\lambda = 0$ we end up with the ordinary least squares (OLS) estimates.

The reason for ridge regression is multicollinearity. When multicollinearity among the covariates ($\mathbf{X}$), the term $\left(\mathbf{X}^T\mathbf{X}\right)$ will not be invertible and thus no OLS betas will be estimated. Ridge regression is a regularised regression method because it regularises this matrix so that it becomes invertible. Alternatively, one can use principal component regression we saw before. The estimated betas will be biased, but at least we obtain an answer. If there is no multicollinearity, ridge regression can still be used because ridge regression can lead to better predicted values than the classical regression. In any case, the choice of the value of λ is the key question.

In multivariate regression, the parameter λ becomes a matrix, but I saw that Brown and Zidek (1980) use a scalar, so I will use a scalar also. The corresponding formula is the same,

but instead of the vectors **fi** and **y** we have matrices **B** and **Y**

$$\hat{\mathbf{B}}^{ridge} = \left(\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_p\right)^{-1} \mathbf{X}^T\mathbf{Y}.$$

When $\lambda = 0$ we end up with the OLS beta coefficients.

Just to inform you that R has a built-in function for ridge regression inside the *MASS* library called *lm.ridge*. It does not give any predictions, so you have to do it yourselves. The next R function performs ridge regression for a given value of λ. Not that the independent variables have to be continuous or if they are categorical you must turn them into dummy variables. In addition, note that the dependent variable(s) is centered and the predictor variables are standardised. If you want to predict the values of the dependent variable(s) for some new values of the independent variables, they must be scaled first using the mean and standard deviation calculated from the observed independent variables. This is the same procedure we did in the kernel and principal components regression.

```
ridge.reg <- function(y, x, lambda, B = 1, xnew = NULL) {
  ## xnew is the new independent variables values
  ## whose values of y you want to estimate
  ## by default xnew is the x, so you will get the fitted values
  ## y is a real valued vector
  ## x contains the independent variable(s)
  ## lambda is the ridge regularization parameter
  ## if lambda=0, the classical multivariate regression is implemented
  ## B is for bootstrap estimation of the standard errors of the betas
  ## if pred is TRUE it means that you want to predict new y
  ## but if xnew is x (by default), the pred is not important
  ## the pred is important if xnew is not x

  y <- as.vector(y)
  if ( all( y > 0 & y < 1 ) ){
    y <- log(y / ( 1 - y) ) ## logistic normal
  }

  n <- length(y)  ## sample size
  p <- dim(x)[2]  ## dimensionality of x
  my <- sum(y) / n
  yy <- y - my   ## center the dependent variables
  mx <- Rfast::colmeans(x)
  s <- Rfast::colVars(x, std = TRUE)
  xx <- ( x - rep(mx, rep(n, p) ) ) %*% diag(1 / s) ## standardize the independent variab
```

```
  lamip <- lambda * diag(p)
  xtx <- crossprod(xx)
  W <- solve( xtx + lamip )
  beta <- W %*% crossprod(xx, yy)
  est <- as.vector( xx %*% beta + my )

  va <- var(y - est) * (n - 1) / (n - p - 1)
  # vab <- kronecker(va, W %*% xtx %*% W  )
  # seb <- matrix( sqrt( diag(vab) ), nrow = p )
  vab <- va * mahalanobis(W, numeric(p), xtx, inverted = TRUE)
  seb <- sqrt( vab )

  if (B > 1) { ## bootstrap estimation of the standard errors
    be <- matrix(nrow = B, ncol = p )
    for ( i in 1:B) {
      id <- sample(1:n, n, replace = TRUE)
      yb <- yy[id, ]     ;     xb <- xx[id, ]
      be[i, ] <- solve( crossprod(xb) + lamip, crossprod(xb, yb) )
    }
    seb <- Rfast::colVars(be, std = TRUE)  ## bootstrap standard errors of betas
  }

  ## seb contains the standard errors of the coefficients
  if ( is.null( colnames(x) ) ) {
    names(seb) <- paste("X", 1:p, sep = "")
    names(beta) <- paste("X", 1:p, sep = "")
  } else  names(seb) <- names(beta) <- colnames(x)

  if ( !is.null(xnew) ) {
    xnew <- matrix(xnew, ncol = p)
    ## scale the xnew values
    xnew <- ( t(xnew) - rep(mx, rep(dim(xnew)[1], p) ) %*% diag( 1 / s)
    est <- as.vector( xnew %*% beta + my )

  } else est <- est

  list(beta = beta, seb = seb, est = est)
}
```

An alternative formula is given via Singular Value Decomposition (SVD) and this is what I use here. We can write $\mathbf{X}$, the matrix of the standardised independent variables as

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T,$$

$\mathbf{D}$ is a diagonal matrix containing the singular values (square root of the eigenvalues). For more information on SVD see Sections 6.5 and 6.10. The beta coefficients can be written as

$$\beta_\lambda = \mathbf{V}\frac{d_j}{d_j^2 + \lambda}\mathbf{U}^T\mathbf{Y}.$$

The next function calculates the ridge coefficients for a range of values of λ and plots them. This will work only when the dependent variable is univariate.

```
ridge.plot <- function(y, x, lambda = seq(0, 5, by = 0.1) ) {
  ## if y is a vector only
  ## x contains the independent, continuous only, variables
  ## lambda contains a grid of values of the ridge regularization parameter

  y <- as.vector(y)

  if ( all( y > 0 & y < 1 ) ){
    y <- log(y / ( 1 - y) ) ## logistic normal
  }

  n <- length(y)  ## sample size
  p <- dim(x)[2]  ## dimensionality of x
  R <- length(lambda)
  be <- matrix(nrow = p, ncol = R)
  yy <- y - sum(y) / n  ## center the dependent variables
  xx <- Rfast::standardise(x)  ## standardize the independent variables
  sa <- svd(xx)
  d <- sa$d    ;    v <- t(sa$v)    ;     tu <- t(sa$u)
  d2 <- d^2    ;    A <- d * tu %*% yy

  for (i in 1:R)  be[, i] <-  crossprod( v / ( d2 + lambda[i] ), A )

  plot(lambda, be[1,], type = "l", col = 1, lty = 1,
       ylim = c( min(be), max(be) ), xlab = expression(paste(lambda, " values") ),
       ylab = "Beta coefficients")
  for (i in 2:p)  lines(lambda, be[i, ], col = i, lty = i)
```

```
}
```

The next R function uses cross validation to choose the value of λ that minimizes the mean squared error of prediction, in the same way we did for the principal component, the k-NN and the kernel regression implemented before. My suggestion for saving time is to search for λ at big steps, for example. $\lambda = 0, 0.5, 1, 1.5, 2....$ Then see the plot where the minimum is obtained (for example between 0.5 and 1) and redo the search at that region with a smaller step, for example $\lambda = 0.5, 0.51, 0.52, ..., 1$.

```
ridge.tune <- function(y, x, M = 10, lambda = seq(0, 2, by = 0.1),
                       mat = NULL, ncores = 1, graph = FALSE) {

  ## y is the univariate or multivariate dependent variable
  ## x contains the independent variables(s)
  ## M is the number of folds, set to 10 by default
  ## lambda is a vector with a grid of values of lambda
  ## ncores is the number of cores to use

  y <- as.matrix(y)  ## makes sure y is a matrix
  n <- dim(y)[1]  ## sample size
  k <- length(lambda)

  di <- dim(y)[2]  ## dimensionality of y
  p <- dim(x)[2]  ## dimensionality of x
  if ( is.null(mat) ) {
    nu <- sample(1:n, min( n, round(n / M) * M ) )
    ## It may be the case this new nu is not exactly the same
    ## as the one specified by the user
    ## to a matrix a warning message should appear
    oop <- options(warn = -1)
    on.exit(options(oop))
    mat <- matrix( nu, ncol = M ) # if the length of nu does not fit
  } else  mat <- mat

  M <- dim(mat)[2]
  rmat <- dim(mat)[1]
  msp <- matrix( nrow = M, ncol = k)

  ## deigma will contain the positions of the test set
```

```
## this is stored but not showed in the end
## the user can access it though by running
## the commands outside this function

if (ncores == 1) {
  runtime <- proc.time()
  for (vim in 1:M) {
    ytest <- as.matrix( y[mat[, vim], ] )  ## test set dependent vars
    ytrain <- as.matrix( y[-mat[, vim], ] )  ## train set dependent vars
    my <- Rfast::colmeans(ytrain)
    yy <- t( t(ytrain) - my ) ## center the dependent variables
    xtrain <- as.matrix( x[ -mat[, vim], ] )  ## train set independent vars
    mx <- Rfast::colmeans(xtrain)
    xtest <- as.matrix( x[ mat[, vim], ] )  ## test set independent vars
    s <- Rfast::colVars(xtrain, std = TRUE)
    xtest <- t( ( t(xtest) - mx ) / s ) ## standardize the xtest
    xx <- t( ( t(xtrain) - mx ) / s )  ## standardize the independent variables

    sa <- svd(xx)
    d <- sa$d    ;    v <- t(sa$v)    ;     tu <- t(sa$u)
    d2 <- d^2    ;    A <- d * tu %*% yy

    for ( i in 1:k ) {
      ## beta <- ( v %*% (tu *  d / ( d^2 + lambda[i] ) ) ) %*% yy
      beta <- crossprod( v / ( d2 + lambda[i] ), A )
      est <- xtest %*% beta + my
      msp[vim, i] <- sum( (ytest - est)^2 ) / rmat
    }

  }
  runtime <- proc.time() - runtime

} else {
  runtime <- proc.time()
  cl <- makePSOCKcluster(ncores)
  registerDoParallel(cl)
  pe <- numeric(k)

  msp <- foreach(vim = 1:M, .combine = rbind, .packages = "Rfast",
```

```
                .export = c("colmeans", "colVars") ) %dopar% {
    ytest <- as.matrix( y[mat[, vim], ] )  ## test set dependent vars
    ytrain <- as.matrix( y[-mat[, vim], ] )  ## train set dependent vars
    my <- Rfast::colmeans(ytrain)
    yy <- t(ytrain) - my  ## center the dependent variables
yy <- t(yy)
    xtrain <- as.matrix( x[ -mat[, vim], ] )  ## train set independent vars
    mx <- Rfast::coleans(xtrain)
    xtest <- as.matrix( x[ mat[, vim], ] )  ## test set independent vars
    s <- Rfast::colVars(xtrain, std = TRUE)
    xtest <- t( ( t(xtest) - mx ) / s )  ## standardize the xtest
    xx <- t( ( t(xtrain) - mx ) / s ) ## standardize the independent variables

    sa <- svd(xx)
    d <- sa$d    ;    v <- t(sa$v)    ;     tu <- t(sa$u)
    d2 <- d^2    ;    A <- d * tu %*% yy

    for ( i in 1:k ) {
      ## beta <- ( v %*% (tu *  d / ( d^2 + lambda[i] ) ) ) %*% yy
      beta <-  crossprod( v / ( d2 + lambda[i] ), A )
      est <- xtest %*% beta + my
      pe[i] <- sum( (ytest - est)^2 ) / rmat
    }

    return(pe)
  }

  runtime <- proc.time() - runtime
  stopCluster(cl)
}

mspe <- Rfast::colmeans(msp)
bias <- msp[ , which.min(mspe)] - apply(msp, 1, min)  ## TT estimate of bias
estb <- mean( bias )  ## TT estimate of bias

if ( graph ) {
  plot(lambda, mspe, type = 'b', ylim = c(min(mspe), max(mspe)),
       ylab = "Mean squared error of prediction",
       xlab = expression(paste(lambda, " values")) )
```

```
  }

  names(mspe) <- lambda
  performance <- c( min(mspe) + estb, estb)
  names(performance) <- c("MSPE", "Estimated bias")
  list(msp = msp, mspe = mspe, lambda = which.min(mspe), performance = performance,
       runtime = runtime)

}
```

4.3 Discriminant analysis

We will now show some ways of parametric discriminant analysis, namely Fisher's method, linear, quadratic and regularised discriminant analysis.

4.3.1 Fisher's linear discriminant function

Fisher's discriminant rule is a non parametric linear function. We need to find the first unit eigenvector (usually called λ) (the eigenvector corresponding to the largest eigenvalue) of the matrix $\mathbf{W}^{-1}\mathbf{B}$, where $\mathbf{W}$ and $\mathbf{B}$ are the within and between sum of squares matrices respectively (Mardia et al., 1979, pg. 318-320). Then we use the mean of each group and the λ to allocate a new observation using the decision algorithm below.
Allocate an observation z to group i iff

$$\left|\boldsymbol{\lambda}^T\mathbf{z} - \boldsymbol{\lambda}^T\bar{\mathbf{x}}_i\right| = \min_{1\leq j\leq g}\left|\boldsymbol{\lambda}^T\mathbf{z} - \boldsymbol{\lambda}^T\bar{\mathbf{x}}_j\right|$$

where $i, j = 1, ..., g$, with g indicating the number of groups.

The code below requires the data, their group and the new data whose group is to be predicted.

```
fisher.da <- function(znew, z, ina) {
  ## znew is the new data whose group is to predicted
  ## z contains the data
  ## ina denotes the groups
  n <- dim(z)[1]  ## sample size
  d <- dim(z)[2]  ## dimensionality
  znew <- as.matrix(znew)
  znew <- matrix(znew, ncol = d)
  nu <- dim(znew)[1]
  ni <- tabulate(ina)
```

```
  k <- length(ni)  ## how many groups are there
  zbar <- Rfast::colmeans(z)
  mi <- rowsum(z, ina) / ni
  B <- ni[1] * tcrossprod( mi[1, ] - zbar )
  for (i in 2:k)  B <- B +  ni[i] * tcrossprod( mi[i, ] - zbar )
  B <- B / (k - 1)  ## the between sum of squares
  m <- sqrt(n) * zbar
  W <- crossprod(z) - tcrossprod(m) - B
  M <- solve(W, B)
  lambda <- as.vector( eigen(M)$vectors[, 1] )  ## Fisher's discriminant
  A <-  as.vector( tcrossprod( lambda, znew ) )
  A <- matrix(rep(A, each = k), nrow = nu, byrow = TRUE)
  ma <- tcrossprod( lambda, mi)
  crit <- abs( eachrow(A, ma, oper = "-") )
  pred <- Rfast::rowMins(crit)  ## the predicted group
  list(lambda = lambda, pred = pred)
}
```

We have to note that in all cases the robust estimation of the covariance and or of the location are available in within the MASS library. For the linear and quadratic discriminant analysis that can happen automatically, by choosing the robust option. In the regularised case, you will have to modify the estimates such that the robust estimates are obtained. Another option is to use the estimates obtained from the t distribution. Bear in mind that even though the univariate t distribution has some robustness properties, the multivariate t distribution is not as robust as many people think. We show how to estimate the parameters under this model later on. In all the other cases, we leave these changes to the interested reader.

The function below estimates the performance of the Fisher classifier using k-leave-out cross validation with either stratified or simple random sampling for the test set.

```
fisher.cv <- function(z, ina, fraction = 0.1, R = 500, strata = TRUE,
  seed = FALSE) {
  ## z contains the data
  ## group denotes the groups
  ## fraction denotes the percentage of the sample to
  ## be used as the test sample
  ## R is the number of cross validations
  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there
  n <- dim(z)[1]  ## sample size
```

```
k <- round(fraction * n)  ## test set sample size
ni <- tabulate(ina)
num <- 1:n
p <- ni/n
esa <- as.vector( round(p * k) )
k <- sum(esa)  ## test set sample size
p <- numeric(R)
deigma <- matrix(nrow = R, ncol = k)
## if seed==TRUE then the results will always be the same
if ( seed )  set.seed(123456)

runtime <- proc.time()
if ( strata ) {  ## stratified random sampling
  deigmata <- matrix(nrow = g, ncol = max(esa))

  for (i in 1:R) {
    for (j in 1:g) {
      ta <- sample( num[ina == j], esa[j] )
      deigmata[j, ] <- c( ta, numeric( max(esa) - length(ta) ) )
    }
    deigma[i, ] <- as.vector( t(deigmata) )
  }

} else {
  for (i in 1:R)  deigma[i, ] <- sample(1:n, k)
}

  for ( i in 1:R ) {
    xtest <- z[ deigma[i, ], ]
    xtrain <- z[ -deigma[i, ], ]
    gtrain <- ina[ -deigma[i, ] ]
    gtest <- ina[ deigma[i, ] ]
    est <- fisher.da(xtest, xtrain, gtrain)$pred
    p[i] <- sum(est == gtest) / k
  }

per <- sum(p) / R
s1 <- sd(p)  ;   s2 <- sqrt( per * (1 - per)/R )
conf1 <- c(per - 1.96 * s1, per + 1.96 * s1)  ## 1st type of conf. interval
```

```
  conf2 <- c(per - 1.96 * s2, per + 1.96 * s2)  ## 2nd type of conf. interval
  ## next we check if the confidence limits exceeds the allowed limits
  if (conf1[2] > 1)  conf1[2] <- 1
  if (conf1[1] < 0)  conf1[1] <- 0
  if (conf2[2] > 1)  conf2[2] <- 1
  if (conf2[1] < 0)  conf2[1] <- 0
  conf3 <- quantile(p, probs = c(0.025, 0.975))  ## 3rd type of conf. interval
  runtime <- proc.time() - runtime

  ci <- rbind(conf1, conf2, conf3)
  colnames(ci) <- c("2.5%", "97.5%")
  rownames(ci) <- c("standard", "binomial", "empirical")
  list(percentage = per, ci = ci, runtime = runtime)

}
```

4.3.2 Repeated cross validation for linear and quadratic discriminant analysis

The built in functions in R for linear and quadratic discriminant analysis offer 1-fold cross validation. This function uses these built in functions to extent to the repeated cross validation. The user specifies the value of the percentage for the splits and then the function removes this percentage (test sample) at random. It performs discriminant analysis for the remaining values (training sample) and then classifies the test sample. This is performed by default $R = 1000$ and in the end an estimate of the distribution of the error is available. Thus, we can construct 3 types of confidence intervals. The first two use the standard approach where the standard deviation is calculated from the $R = 1000$ repetitions and via the binomial distribution. The third one is an empirical (or percentile) one, since it uses the 2.5% upper and lower quantiles of the distribution of the error. This function is more to train the two methods (linear and quadratic discriminant analysis) and see how well each of them performs. The bottom line is to select one over the other.

```
da.cv <- function(x, ina, fraction = 0.2, R = 1000,
  method = "lda", seed = FALSE) {
  ## x is the data
  ## ina is the group indicator variable
  ## fraction denotes the percentage of the sample to
  ## be used as the test sample
  ## R is the number of cross validations
  ## method denotes whether "lda" or "qda" is to be used
  p <- numeric(R)
```

```
  n <- dim(x)[1]
  ina <- as.factor(ina)
  k <- round(fraction * n)  ## test sample size
  ## if seed==TRUE then the results will always be the same
  if ( seed )  set.seed(1234567)

  runtime <- proc.time()
  for (i in 1:R) {
    nu <- sample(1:n, k)
    id <- ina[-nu]
    train <- x[-nu, ]
    test <- x[nu, ]
    if (method == "lda") {
       dok <- lda(train, id)
    }  else   dok <- qda(train, id)
    g <- predict(dok, test)$class
    p[i] <- sum( g == ina[nu] ) /k
 }

  per <- mean(p)
  s1 <- sd(p)
  s2 <- sqrt(per * (1 - per)/R)
  conf1 <- c(per - 1.96 * s1, per + 1.96 * s1)  ## 1st type of conf. interval
  conf2 <- c(per - 1.96 * s2, per + 1.96 * s2)  ## 2nd type of conf. interval

  ## next we check if the confidence limits exceeds the allowed limits
  if (conf1[2] > 1)  conf1[2] <- 1
  if (conf1[1] < 0)  conf1[1] <- 0
  if (conf2[2] > 1)  conf2[2] <- 1
  if (conf2[1] < 0)  conf2[1] <- 0
  conf3 <- quantile(p, probs = c(0.025, 0.975))  ## 3rd type of conf. interval
  runtime <- proc.time() - runtime

  ci <- rbind(conf1, conf2, conf3)
  colnames(ci) <- c("2.5%", "97.5%")
  rownames(ci) <- c("standard", "binomial", "empirical")
  list(percentage = per, ci = ci)

}
```

4.3.3 A simple model selection procedure in discriminant analysis

We will show a simple procedure for model selection in quadratic discriminant analysis. the R code given below is made for quadratic discriminant analysis but with a simple modification it can be applied to linear discriminant analysis as well.

It utilizes the function *kfold.da* where the split is 80% and 20% for the training and the test set respectively. The number of cross validations is set 500 and always the splits are the same. But as I mentioned before, this input parameters can change easily within the function.

The idea is simple and similar to the stepwise variable selection in multiple regression analysis. Below is the algorithm explained.

Algorithm for model selection in discriminant analysis

1. Perform discriminant analysis bases on one variable only. The first chosen variable is the one with the highest estimated rate of correct classification.

2. Next, we look for the second best variable. We try all of them (now we have two variables included) and keep the variable, which combined with the first one, leads to the highest estimated rate of correct classification.

3. We repeat step 2, adding one variable at the time.

4. We stop when the difference between two successive rates is less than or equal to a tolerance level (taken to be 0.001 or 0.1%).

There can be two cases, a) the rate keeps increasing by adding more variables. The tolerance level will prevent from adding more variables than necessary. And b) the rate at some point will decrease. The tolerance level will see the change and will terminate the process. For this reason I use a *while* function.

This is a simple model selection procedure and a faster one would be via the BIC. I am just giving a method here and my purpose is to motivate the interested reader in learning more about it. Also to make the reader aware of the model selection process in discriminant analysis.

```
select.da <- function(x, ina, tol = 0.001) {
  ## x contains the data
  ## ina is the group indicator variable
  ## tol is the stopping difference between two successive rates
  p <- dim(x)[2]
  per <- numeric(p)
```

```
## STEP 1
est <- numeric(p)
z <- NULL
for (j in 1:length(est)) {
  z1 <- x[, j]
  est[j] <- kfold.da(z1, ina, fraction = 0.2, R = 100,
  method = "qda", seed = TRUE)$percentage
}
per[1] <- max(est)
id <- which.max(est)
z <- cbind(z, x[, id])
z1 <- x[, -id]
## STEP 2
est <- numeric(p - 1)
for (j in 1:length(est)) {
  z2 <- z1[, j]
  est[j] <- kfold.da(cbind(z, z2), ina, fraction = 0.2, R = 100,
  method = "qda", seed = TRUE)$percentage
}
per[2] <- max(est)
id <- which.max(est)
z <- cbind(z, z1[, id])
z1 <- z1[, -id]
## STEP 3 AND BEYOND
i <- 2
while (per[i] - per[i - 1] > tol) {
  i <- i + 1
  est <- numeric(p - i + 1)
  for (j in 1:length(est)) {
    z2 <- as.matrix(z1[, j])
    est[j] <- kfold.da(cbind(z, z2), ina, fraction = 0.2, R = 100,
    method = "qda", seed = TRUE)$percentage
  }
  per[i] <- max(est)
  id <- which.max(est)
  z <- cbind(z, z1[, id])
  z1 <- as.matrix(z1[, -id])
}
per <- per[per > 0]
```

```
  plot(per, type = "b", xlab = "Number of variables",
  ylab = "Estimated correct rate")
  list(percentage = per, vars = z)
}
```

4.3.4 Box-Cox transformation in discriminant analysis

We will use the Box-Cox transformation as an additional feature which can lead to better classification results. This power transformation is defined as

$$y(\lambda) = \left\{ \begin{array}{ll} \frac{x^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0 \\ \log x & \text{if } \lambda = 0 \end{array} \right\}$$

Note that the x has to have strictly positive values if one uses the logarithm. When $\lambda \neq 0$ this is not an issue, but if there are zero values, then λ has to be strictly positive. The R code presented below is a simple one. The first step is to apply the Box-Cox transformation for a value of λ and then use the function *kfold.da* we saw before. This is repeated for a range of values of λ and every time the estimated percentage of correct classification is saved. A plot is also created for graphical visualization of the estimated percentage of correct classification as a function of λ.

```
bckfold.da = function(x, ina, fraction = 0.2, R = 1000,
  method = 'lda', lambda = seq(-1, 1, by = 0.1) ) {
  ## x is the matrix with the data
  ## ina is the group indicator variable
  ## fraction denotes the percentage of the sample to be used as test sample
  ## R is the number of cross validations
  ## quad denotes whether lda or qda is to be used
  ## lambda is the range of values for the Box-Cox transformation
  B <- length(lambda)
  percent <- numeric(B)
  conf1 <- conf2 <- conf3 <- matrix(nrow = B, ncol = 2)
  ## for every lambda the same test samples are used
  for (i in 1:B) {
  ## Next is the Box-Cox transformation depending on the value of lambda
  if (lambda[i] != 0) y <- (x ^ lambda[i] - 1) / lambda[i]
  if (lambda[i] == 0) y <- log(x)
  mod <- kfold.da(x = y, ina = ina, fraction = fraction, R = R,
  method = method, seed = TRUE)
  percent[i] <- mod$percentage
```

```
  conf1[i, ] <- mod$ci[1, ]
  conf2[i, ] <- mod$ci[2, ]
  conf3[i, ] <- mod$ci[3, ]
  names(percent) <- lambda
  plot(lambda, percent, ylim = c( min(conf3[, 1]), max(conf3[, 2]) ),
  type = 'b', col = 3, xlab = expression( paste(lambda," values") ),
  ylab = 'Estimated percentage of correct classification')
  lines(lambda, conf3[, 1], lty = 2, lwd = 2, col = 2)
  lines(lambda, conf3[, 2], lty = 2, lwd = 2, col = 2)
  ## the plot contains the 3rd type confidence limits also
  rownames(ci)<- lambda
  colnames(ci) <- c('2.5%', '97.5%')
  ## I show only the the third type of confindence intervals
  list(percentage = percent, ci = conf3)
}
```

4.3.5 Regularised discriminant analysis

Linear and quadratic discriminant analyses can be thought of as special cases of what is called regularised discriminant analysis denoted by RDA(δ, γ) (Hastie et al., 2001). The discriminant analysis in general has a rule. Every vector $\mathbf{z}$ is allocated to the group for which the density of the vector calculated using the multivariate normal is the highest. The algorithm is as follows

- Calculate $\pi_i f_i(\mathbf{z})$ for $i = 1, ..., g$, where g indicates the number of groups.
- Allocate $\mathbf{z}$ to the group for which the above quantity takes the highest value.

The $f_i(x)$ is assumed a multivariate normal and $\pi_i = n_i/n$, where n_i is the sample size of the i-th group and $n = n_1 + ... + n_g$ is the total sample size. The π_i plays the role of the prior, thus making the rule a naive Bayes classifier. Alternatively the first step of the algorithm can be substituted by the logarithm of the density

$$\xi_i(\mathbf{z}) = -\frac{1}{2}\log|\mathbf{S}_i| - \frac{1}{2}(\mathbf{z} - \hat{\boldsymbol{\mu}}_i)^T \mathbf{S}_i^{-1}(\mathbf{z} - \hat{\boldsymbol{\mu}}_i) + \log \pi_i,$$

The vector $\mathbf{z}$ is allocated to the group with the highest value $\xi_i(\mathbf{z})$. The idea of RDA(δ, γ) is to substitute the covariance matrix for each group ($\mathbf{S}_i$) by a weighted average

$$\mathbf{S}_i(\delta, \gamma) = \delta \mathbf{S}_i + (1 - \delta)\mathbf{S}(\gamma),$$

$$\text{where } \mathbf{S}(\gamma) = \gamma \mathbf{S}_p + (1 - \gamma) s^2 \mathbf{I}_d$$

and $\mathbf{S}_p$ is the pooled covariance matrix

$$\mathbf{S}_p = \frac{\sum_{i=1}^{g} (n_i - 1)\, \mathbf{S}_i}{n - g}$$

The regularization of the pooled covariance matrix ($\mathbf{S}_p$) is the one mentioned in Hastie et al. (2001). They used ($s^2\mathbf{I}$), where $s^2 = \frac{tr\mathbf{S}_p}{d}$ and d is the number of dimensions. Thus we end up with a general family of covariance matrices which is regularised by two parameters δ and γ each of which takes values between 0 and 1. When $\delta = 1$ then we end up with QDA, and if $\delta = 0$ and $\gamma = 1$ we end up with LDA. The posterior probabilities of group allocation are calculated as follows

$$P\left(\mathbf{z}_i \in group_j \middle| \xi_j\left(\mathbf{z}_i\right)\right) = \frac{\pi_j f_j\left(\mathbf{z}_i\right)}{\sum_{l=1}^{g} \pi_l f_l\left(\mathbf{z}_i\right)},$$

The code presented below accepts new observations and predicts their groups, for a given value of γ and λ.

```
rda <- function(xnew, x, ina, gam = 1, del = 0) {
  ## xnew is the new observation
  ## x contains the data
  ## ina is the grouping variable
  ## gam is between pooled covariance and diagonal
  ## gam*Spooled+(1-gam)*diagonal
  ## del is between QDA and LDA
  ## del*QDa+(1-del)*LDA
  ## mesi is NULL by default or it can be a matrix with the group means
  ## info can be NULL or it can be a list containing the
  ## covariance matrix of each group, the pooled covariance matrix
  ## and the spherical covariance matrix (this order must be followed)
  ## the mesi and info are particularly useful for the tuning of the rda, as
  ## they can speed the computations a lot.
  n <- dim(x)[1]
  D <- dim(x)[2]
  xnew <- as.matrix(xnew)
  xnew <- matrix(xnew, ncol = D)
  nu <- dim(xnew)[1]  ## number of the new observations
  ina <- as.numeric(ina)
  ng <- tabulate(ina)
  nc <- length(ng)
  ta <- matrix(nrow = nu, ncol = nc)
```

```
  ci <- 2 * log(ng / n)
  sk <- vector("list", nc)
  mesos <- rowsum(x, ina) / ng
  sa <- 0
  for (i in 1:nc) {
    xi <- x[ina == i, ]
    m <- sqrt(ng[i]) * mesos[i, ]
    sk[[ i ]] <- ( crossprod(xi) - tcrossprod(m) )
    sa <- sa + sk[[ i ]]
    sk[[ i ]] <- sk[[ i ]] / (ng[i] - 1)
  }
  Sp <- sa/(n - nc)
  sp <- diag( sum( diag( Sp ) ) / D, D ) ## spherical covariance matrix
  Sa <- gam * Sp + (1 - gam) * sp  ## regularised covariance matrix

  for (j in 1:nc) {
    Ska <- del * sk[[ j ]] + (1 - del) * Sa
    ta[, j] <- ci[j] - log( det( Ska ) ) - Rfast::mahala( xnew, mesos[j, ], Ska )
    ## the scores are doubled for efficiency, i did not multiply with 0.5
  }

  est <- Rfast::rowMaxs(ta)
  expta <- exp(ta)
  prob <- expta / Rfast::rowsums( expta ) ## the probability of classification
  list(prob = prob, scores = ta, est = est)
}
```

We now how how to tune the parameters of the regularised discriminant analysis. The idea is similar to all the techniques we have seen in this Section. The bias correction estimate of Tibshirani and Tibshirani (2009) is again applied.

```
rda.tune <- function(x, ina, nfolds = 10, gam = seq(0, 1, by = 0.1),
            del = seq(0, 1, by = 0.1), ncores = 1, folds = NULL,
            stratified = TRUE, seed = FALSE) {
  ## x contains the data
  ## gam is between pooled covariance and diagonal
  ## gam*Spooled+(1-gam)*diagonal
  ## del is between QDA and LDA
  ## del*QDa+(1-del)*LDA
  ## if ncores==1, then 1 processor is used, otherwise more are
```

```
## used (parallel computing)
## if a matrix with folds is supplied in mat the results will
## always be the same. Leave it NULL otherwise
ina <- as.numeric(ina)
n <- dim(x)[1]  ## total sample size
nc <- max(ina) ## number of groups
D <- dim(x)[2]  ## number of variables
sk <- array( dim = c(D, D, nc) )
lg <- length(gam)    ;    ld <- length(del)
if ( is.null(folds) )  folds <- Compositional::makefolds(ina,
 nfolds = nfolds, stratified = stratified, seed = seed)
nfolds <- length(folds)

if (ncores > 1) {
  runtime <- proc.time()
  group <- matrix(nrow = length(gam), ncol = length(del) )
  cl <- parallel::makePSOCKcluster(ncores)
  doParallel::registerDoParallel(cl)
  if ( is.null(folds) )  folds <- Compositional::makefolds(ina,
   nfolds = nfolds, stratified = stratified, seed = seed)
  ww <- foreach(vim = 1:nfolds, .combine = cbind,
        .export = c("mahala", "rowMaxs"), .packages = "Rfast") %dopar% {
    test <- x[ folds[[ vim ]], , drop = FALSE]  ## test sample
    id <- ina[ folds[[ vim ]] ] ## groups of test sample
    train <- x[ -folds[[ vim ]], ]   ## training sample
    ida <- ina[ -folds[[ vim ]] ]  ## groups of training sample
    na <- tabulate(ida)
    ci <- 2 * log(na / sum(na) )
    mesi <- rowsum(train, ida) / na
    na <- rep(na - 1, each = D^2)
    ## the covariance matrix of each group is now calculated
    for (m in 1:nc)  sk[ , , m] <- Rfast::cova( train[ida == m, ] )
    s <- na * sk
    Sp <- colSums( aperm(s) ) / (sum(na) - nc)  ## pooled covariance
    sp <- diag( sum( diag( Sp ) ) / D, D )
    gr <- matrix(nrow = length( folds[[ vim ]] ), ncol = nc)

    for ( k1 in 1:length(gam) ) {
      Sa <- gam[k1] * Sp + (1 - gam[k1]) * sp  ## regularised covariance
```

```
      for ( k2 in 1:length(del) ) {
        for (j in 1:nc) {
          Ska <- del[k2] * sk[, , j] + (1 - del[k2]) * Sa
          gr[, j] <- ci[j] - log( det( Ska ) ) -
          Rfast::mahala( test, mesi[j, ], Ska )
          ## I did not multiply with 0.5 for efficiency
        }
        g <- Rfast::rowMaxs(gr)
        group[k1, k2] <- mean( g == id )
      }
    }
    return( as.vector( group ) )
  }
  stopCluster(cl)

  per <- array( dim = c( lg, ld, nfolds ) )
  for ( i in 1:nfolds )  per[, , i] <- matrix( ww[, i], nrow = lg )
  runtime <- proc.time() - runtime

} else {
  runtime <- proc.time()
  per <- array( dim = c( lg, ld, nfolds ) )

  for (vim in 1:nfolds) {

    test <- x[ folds[[ vim ]], , drop = FALSE ]   ## test sample
    id <- ina[ folds[[ vim ]] ] ## groups of test sample
    train <- x[ -folds[[ vim ]], ]  ## training sample
    ida <- ina[ -folds[[ vim ]] ]   ## groups of training sample
    na <- tabulate(ida)
    ci <- 2 * log(na / sum(na) )
    mesi <- rowsum(train, ida) / na
    na <- rep(na - 1, each = D^2)
    ## the covariance matrix of each group is now calculated
    for (m in 1:nc)  sk[ , , m] <- Rfast::cova( train[ida == m, ] )
    s <- na * sk
    Sp <- colSums( aperm(s) ) / (sum(na) - nc)  ## pooled covariance
    sp <- diag( sum( diag( Sp ) ) / D, D )
    gr <- matrix(nrow = length( folds[[ vim ]] ), ncol = nc)
```

```
      for ( k1 in 1:length(gam) ) {
        Sa <- gam[k1] * Sp + (1 - gam[k1]) * sp  ## regularised covariance
        for ( k2 in 1:length(del) ) {
          for (j in 1:nc) {
            Ska <- del[k2] * sk[, , j] + (1 - del[k2]) * Sa
            gr[, j] <- ci[j] - log( det( Ska ) ) -
            Rfast::mahala( test, mesi[j, ], Ska )
            ## I did not multiply with 0.5 for efficiency
          }
          g <- Rfast::rowMaxs(gr)
          per[k1, k2, vim] <- mean( g == id )
        }
      }
    }
    runtime <- proc.time() - runtime
  }

  percent <- t( colMeans( aperm(per) ) )
  su <- apply(per, 1:2, sd)
  dimnames(percent) <- dimnames(su) <- list(gamma = gam, delta = del)
  confa <- as.vector( which(percent == max( percent ),
  arr.ind = TRUE )[1, ] )
  result <- cbind( max(percent), gam[ confa[1] ], del[ confa[2] ] )
  colnames(result) <- c('optimal', 'best gamma', 'best delta')
  list(per = per, percent = percent, se = su, result = result,
  runtime = runtime)
}
```

4.3.6 Discriminant analysis with mixed data

In all the previous cases we saw how to discriminate between groups containing continuous data but we did not mention the case of mixed (categorical and continuous) or only categorical data. This problem is solved, obviously, by employing the multinomial regression (Agresti, 2002). In fact, the multinomial regression is used for the case of multinomial responses, not just 0 or 1, not binary, but with more outcomes. It is the generalization of the binomial distribution. From *bin*(omial) we go to *multin*(omial). The mass function of this

discrete distribution is

$$P\left(Y_1 = y1, \ldots, Y_k = y_k\right) = \frac{n!}{y_1! \ldots y_k!} y_1^{p_1} \ldots y_k^{p_k},$$

where $\sum_{i=1}^{k} y_i = n$ and $\sum_{i=1}^{k} p_i = 1$. So, in our case, each Y_i can take one value at the time, one outcome. Just like in the binomial distribution we can link the probabilities p_i to some independent variables

$$p_i = \left\{ \begin{array}{ll} \frac{1}{1+\sum_{j=2}^{k} e^{\mathbf{X}\boldsymbol{\beta}_j}} & \text{if } i = 1 \\ \frac{e^{\mathbf{X}\boldsymbol{\beta}_i}}{1+\sum_{j=2}^{k} e^{\mathbf{X}\boldsymbol{\beta}_j}} & \text{if } i = 2, \ldots, k \end{array} \right\}.$$

Again, just like in binary regression we do multinomial regression. But, it happens, as usually in statistics, that this is exactly what we want to do here also. For every observation, the fitted values is a vector of k elements, probabilities. We assign that observation to an outcome (or to a group) according to the highest probability. If a fitted value for example has this form $\hat{\mathbf{y}}_i = (0.1, 0.4, 0.5)$, then we assign to $\hat{\mathbf{y}}_i$ the value of 3, (or the third outcome).

Bear in mind and note one important thing. The outcomes are nominal, i.e. the ordering is not important. We do not say, for example *good*, *better*, *best*. But we say, *red*, *yellow*, *orange*. The ordering of the outcomes is totally arbitrary and has no effect on the data. One more thing is that this method can be used when you have continuous data only, as well. It is the most flexible of all with respect to the data used. In order to implement this type of regression (for discrimination purposes) you could use of the VGAM package. Alternatively, you can use (and this is what I use) the nnet package.

```
mrda <- function(y, x, xnew = NULL) {
  ## requires the library nnet
  ## xnew is the new data whose groups is to be predicted
  ## if xnew is NULL, then the fitted values are returned
  ## y is the grouping variable and is expected to be a factor
  ## levels 1, 2, 3 and so on. Do not have zeros.
  ## x is a data frame with all data. Therefore, you cna have
  ## categorical variables as well. Even if you have only continuous
  ## data it will still be turned into a data frame
  y <- as.factor(y)
  p <- dim(x)[2]
  if ( is.null(colnames(x)) )  colnames(x) <- paste("X", 1:p, sep = "")
  mod <- nnet::multinom(y ~ . , data = as.data.frame(x), trace = FALSE)
  if ( !is.null(xnew) ) {
    xnew <- as.data.frame(xnew)
```

```
    colnames(xnew) <- colnames(x)
    est <- predict(mod, xnew)
  } else {
    probs <- fitted(mod)
    est <- Rfast::rowMaxs(probs)
  }
  list(suma = summary(mod), est = est)
}
```

In order to estimate its performance I have written another code which uses parallel computing to speed up the cross validation procedure.

```
mrda.cv <- function(y, x, fraction = 0.2, strata = TRUE, seed = FALSE,
  R = 500, ncores = 4) {

  ## y is a factor variable with the categories
  ## levels 1, 2, 3 and so on. Do not have zeros.
  ## x is a matrix containing all the data
  ## fraction is the percentage of data to be used for testing purposes
  ## the remaining data belong to the training set
  ## R is the number of cross validations to be performed
  ## ncores is the number of cores to use

  n <- dim(x)[1]  ## total sample size
  g <- length( Rfast::sort_unique(y) ) ## how many groups are there
  nu <- round(fraction * n)  ## test set sample size
  k <- round(fraction * n)
  esa <- round( tabulate(y) * nu / n )  ## group sample sizes
  num <- 1:n
  ina <- as.numeric(y)

  if (ncores > 1) {
    runtime <- proc.time()
    if ( seed ) set.seed( 12345 )
    require(doParallel, quiet = TRUE, warn.conflicts = FALSE)
    cl <- makePSOCKcluster(ncores)
    registerDoParallel(cl)
    ba <- rep(R/ncores, ncores)
    p <- numeric(ba[1])
    ww <- foreach(j = 1:ncores, .combine = cbind, .packages = "nnet",
```

```
    .export= "mrda") %dopar% {
    for (vim in 1:ba[j]) {
      if ( strata ) {
        deigma <- matrix(nrow = g, ncol = max(esa))
        for (i in 1:g) {
          ta <- sample(num[ina == i], esa[i])
          deigma[i,] <- c( ta, numeric( max(esa) - length(ta) ) )
        }
        deigma <- as.vector(deigma)
      } else  deigma <- sample(1:n, k)
      xtest <- x[deigma, ]  ## test sample
      id <- ina[deigma]  ## groups of test sample
      xtrain <- x[-deigma, ]  ## training sample
      ytrain <- ina[-deigma]  ## groups of training sample
      est <- mrda(ytrain, xtrain, xtest)$est
      p[vim] <- sum(est == id)/k
    }
   return(p)
  }
  stopCluster(cl)
  p <- as.vector(ww)
  runtime <- proc.time() - runtime

} else {
  runtime <- proc.time()
  if ( seed ) set.seed( 12345 )
  for ( vim in 1:R ) {
    if ( strata ) {
      deigma <- matrix(nrow = g, ncol = max(esa))
      for (i in 1:g) {
        ta <- sample(num[ina == i], esa[i])
        deigma[i,] <- c( ta, numeric( max(esa) - length(ta) ) )
      }
      deigma <- as.vector(deigma)
    } else  deigma <- sample(1:n, k)
    xtest <- x[deigma, ]  ## test sample
    id <- ina[deigma]  ## groups of test sample
    xtrain <- x[-deigma, ]  ## training sample
    ytrain <- ina[-deigma]  ## groups of training sample
```

```
      est <- mrda(ytrain, xtrain, xtest)$est
      p[vim] <- sum(est == id)/k
    }
    runtime <- proc.time() - runtime
  }

  per <- sum(p) / R
  su <- sd(p)
  conf <- quantile(p, probs = c(0.025, 0.975))
  list(per = per, su = su, conf = conf, runtime = runtime)
}
```

4.3.7 Discriminant analysis for multinomial data

Suppose we have multinomial data and we know the different populations from which they come. An example to give is the text analysis. You have some documents of some authors. For each of the document you know how many sentences he/she has written and consequently you know the number of words with $1, 2, 3, 4, 5$ and 6 or more syllables. So for example, a document of the author X has $100, 50, 150, 300, 100, 250$ words of $1, 2, 3, 4, 5$ and 6 or more syllables, with a total of 950 words. Imagine now you have this kind of information for all his documents and for all the documents of the other authors. The target is to discriminate between the authors. The difference now is that instead of continuous data, we have discrete data.

Another example to think of this distribution is via the binomial. In the binomial distribution you have two possible scenarios, in the multinomial you have three or more. In football for example, you may loose, win or draw. Another example is the voters. There are 20% of the voters who would vote for party A, 30% for party B, 25% for party C and 25% for party D. We pick at random 80 people. What is the probability that 30 will vote for party A, 20 for party B, 15 for party C and 15 for party D?

I will use the multinomial distribution for this purpose

$$f(x_1, \ldots, x_K; p_1, \ldots, p_K) = \frac{n!}{x_1! \ldots x_K!} p_1^{x_1} \ldots p_K^{x_K},$$

where $\sum_{i=1}^{K} x_i = n$ and $\sum_{i=1}^{K} p_i = 1$ and K denotes the number of categories. If $K = 2$ we obtain the binomial distribution. Given that we have an $n \times K$ matrix $\mathbf{X}$ of data. The estimated probabilities p_i are given by

$$\hat{p}_i = \frac{\sum_{j=1}^{n} x_{ij}}{n}, \quad \text{for} \quad i = 1, \ldots, K.$$

Suppose now that we have a new vector $\mathbf{x} = (x_1, \ldots, x_K)$ and we want to know where to allocate it, did it come from a multinomial population with parameters (probabilities) $(\alpha_1, \ldots, \alpha_K)$ or to a population with parameters $(\beta_1, \ldots, \beta_K)$?

After estimating the parameters of each of the two populations, we calculate the score function

$$\delta = \sum_{i=1}^{K} x_i \log \frac{\hat{\alpha}_i}{\hat{\beta}_i}.$$

If $\lambda > 0$, we allocate $\mathbf{x}$ e to the first population and to the second otherwise. When we have more than two populations, then we calculate $\lambda_j = \sum_{i=1}^{K} x_i \log \hat{\alpha}_{ij}$ for each of the g groups and we allocate $\mathbf{x}$ to the group with the highest score function λ_j.

Alternatively, you may assume that each variable (or category) is independent from each other and fit a product of Poisson distributions. That is, for each category fit a separate Poisson distribution. Hence, assume that $\mathbf{x} \sim PP(\lambda_1, \ldots, \lambda_K)$

$$f(x_1, \ldots, x_K; \lambda_1, \ldots, \lambda_K) = \prod_{i=1}^{K} e^{-\lambda_i} \frac{\lambda_i^{x_i}}{x_i!}$$

The PP and the multinomial will not differ much if the data are not overdispersed, i.e. if they do come from a multinomial distribution. If there is overdispersion though, then a better alternative is the Dirichlet-multinomial, which is presented in §5.1.9, but included as an option for discrimination. The function *multinom.da* requires the group samples as they are, the grouping information, the new data to be classified and the type of the distribution used.

```
multinom.da <- function(xnew = x, x, ina, type = "multinom") {
  ## x is a matrix containing all the data
  ## set to x by default
  ## ina is the group indicator variable
  ## xnew is the new data to be classified
  ## type is either "multinom" (default), "dirmult" or "poisson"

  p <- dim(x)[2]
  xnew <- matrix( xnew, ncol = p )
  x <- as.matrix(x)  ## makes sure x is a matrix
  ina <- as.numeric(ina)
  nu <- tabulate(ina)
  g <- length( nu )

  if (type == "multinom") {
```

```
    ## normalize the data, so that each observation sums to 1
    y <- x / Rfast::rowsums(x)
    m <- rowsum(y, ina) / nu
    score <- tcrossprod( xnew, log(m) )

  } else if (type == "poisson") {
    m <- rowsum(x, ina) / nu
    score <-  - Rfast::rowsums(m) + tcrossprod( xnew, log(m) )
    ## the next line is not necessary, as the score does not change
##  - Rfast::rowsums( lgamma(xnew + 1) )

  } else {
    m <- matrix(nrow = g, ncol = p )
    for (i in 1:g) {
      A <- x[ina == i, ]
      m[i, ] = dirimultinom(A)$para
      score[, i] <- lgamma( sum(m[i, ]) ) - lgamma( rowSums(A) + sum(m[i, ]) ) +
      lgamma( A + rep( m[i, ], rep(nu[i], p) ) ) - sum( lgamma( m[i, ] ) )
    }
  }

  Rfast::rowMaxs(score)

}
```

The function *multinom.da* requires the group samples as they are and the grouping information. The other arguments are relevant to the replications. It will give an estimate of the rate of correct classification. Of course, different test set sizes will give different percentages.

```
multinomda.cv <- function(x, ina, fraction = 0.2, R = 500,
  seed = FALSE, type = "multinom") {
  ## x is a matrix containing all the data
  ## ina is the group indicator variable
  ## fraction is the percentage of data to be used for testing purposes
  ## the remaining data belong to the training set
  ## R is the number of cross validations to be performed
  ## type is either  "multinom", "poisson" or "dirmult"

  n <- dim(x)[1]  ## total sample size
  ina <- as.numeric(ina)
```

```
g <- max(ina)  ## how many groups are there
nu <- round(fraction * n)  ## test set sample size
esa <- round( tabulate(ina) * nu/n )  ## group sample sizes
sam <- 1:n

p <- numeric(R)  ## the percentages of correct classification will be stored
## deigmata contains the test sets
## each test set is sampled via stratified random sampling
## this ensures that all groups are always represented
## in the test sets
## if seed==TRUE then the results will always be the same
if ( seed ) set.seed(1234567)

 runtime <- proc.time()
 deigma <- matrix(0, R, nu)
 deigmata <- matrix(0, g, nu)

 for (jim in 1:R) {
   for (i in 1:g) {
     ta <- sample( sam[ ina == i ], esa[i] )
     deigmata[i, ] <- c(ta, numeric( nu - esa[i]) )
   }
  deigma[jim, ] <- as.vector(deigmata[deigmata > 0])
}

for (i in 1:R) {
  test <- x[deigma[i, ], ]
  id <- ina[deigma[i, ] ]
  train <- x[-deigma[i, ], ]
  ina2 <- ina[-deigma[i, ] ]
  est <- multinom.da(test, train, ina2, type = type)
  p[i] <- sum(est == id)/nu
}

per <- sum(p) / R
su <- sd(p)
conf <- quantile(p, probs = c(0.025, 0.975))
runtime <- proc.time() - runtime
list(per = per, su = su, conf = conf, runtime = runtime)
```

```
}
```

5 Distributions

5.1 Maximum likelihood estimation

5.1.1 Kullback-Leibler divergence between two multivariate normal populations

The Kullbacvk-Libler divergence (Kullback, 1997) between two multivariate normal populations in $\mathbb{R}^d$ is equal to

$$KL\left(MN_1||MN_2\right) = \frac{1}{2}\left[tr\left(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1\right) + \left(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1\right)^T \boldsymbol{\Sigma}_2^{-1}\left(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1\right) - \log\frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_2|} - d\right],$$

```
kl.norm <- function(m1, s1, m2, s2) {
  ## m1 and s1 are the parameters of the first normal
  ## m2 and s2 are the parameters of the second normal
  ## this measures the distance from a MVN(m1,s1) to MVN(m2,s2)
  sinv <- chol2inv( chol(s2) )
  a <- sinv %*% s1
  0.5 * ( sum( diag( a ) ) +  (m2 - m1) %*% sinv %*% ( m2 - m1 ) +
  log( det(a) ) - length(m1) )
}
```

5.1.2 Estimation of the parameters of a multivariate log-normal distribution

When a random variable $\mathbf{X}$ follows a multivariate normal, then the exponential of it $\mathbf{Y} = \exp(\mathbf{X})$ follows a multivariate log-normal distribution. In reverse, if some positive random variable follow the log-normal distribution, then its logarithm follows the normal distribution. So the support of $\mathbf{X}$ is $\mathbb{R}^d$, but the support of $\mathbf{Y}$ is only the positive side of $\mathbb{R}^d$.

The density of the multivariate log-normal distribution in d dimensions is (Tarmast, 2001)

$$f(y) = \frac{1}{|2\pi\boldsymbol{\Gamma}|^{1/2}} e^{-\frac{1}{2}(\log\mathbf{y}-\boldsymbol{\nu})^T\boldsymbol{\Gamma}^{-1}(\log\mathbf{y}-\boldsymbol{\nu})} \prod_{i=1}^{d}\frac{1}{y_i},$$

where final bit is the Jacobian of the transformation and $\boldsymbol{\nu}$ and $\boldsymbol{\Gamma}$ are the mean vector and covariance matrix of the logarithmically transformed data, of the multivariate normal distribution. The logarithm of the vector is taken component-wise $\log\mathbf{y} = (\log y_1, \ldots, \log y_d)$ and $0 < y_i < \infty$ for $i = 1, \ldots, d$.

The maximum likelihood estimators of the mean and the covariance of a multivariate

log-normal distribution are given by

$$\begin{aligned} \mathrm{E}(\mathbf{Y}) = \boldsymbol{\mu} &= (\mu_1, \ldots, \mu_d) = \left(e^{\nu_1 + 0.5\boldsymbol{\Gamma}_{11}}, \ldots, e^{\nu_d + 0.5\boldsymbol{\Gamma}_{dd}}\right) \text{ and} \\ \mathrm{Var}(\mathbf{Y}) &= \boldsymbol{\Sigma}, \end{aligned}$$

where $\boldsymbol{\Sigma}_{ij} = e^{\nu_i + \nu_j + 0.5\left(\boldsymbol{\Gamma}_{ii} + \boldsymbol{\Gamma}_{jj}\right)} \left(e^{\boldsymbol{\Gamma}_{ij}} - 1\right)$.

```
mvlnorm.mle <- function(x) {
  y <- Log(x)  ## transform the data to the whole of R^d
  m1 <- colmeans(y)  ## mean vector of y
  s <- cova(y)  ## covariance matrix of y
  d <- dim(x)[2]
  s1 <- diag(s)
  m <- exp( m1 + 0.5 * s1 )  ## mean vector of x

  m2 <- outer(m1, m1, "+")
  s2 <- outer(s1, s1, "+")
  s <- exp( m2 + 0.5 * s2 ) * ( exp(s) - 1 )

  if ( is.null(colnames(x)) ) {
    names(m) = colnames(s) = rownames(s) = paste("X", 1:d, sep = "")
  } else names(m) = colnames(s) = rownames(s) = colnames(x)

  list(m = m, s = s)
}
```

5.1.3 Estimation of the parameters of a multivariate t distribution

The density of the multivariate t distribution is

$$f_d(\mathbf{y}) = \frac{\Gamma\left(\frac{\nu+d}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right) |\pi\nu\Sigma|^{1/2} \left[1 + \frac{1}{\nu}(\mathbf{y} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{y} - \boldsymbol{\mu})\right]^{\frac{\nu+d}{2}}}, \tag{5.1}$$

where the parameter ν is called degrees of freedom and the the mean vector and variance matrix are defined as follows

$$\begin{aligned} \mathrm{E}(\mathbf{y}) &= \boldsymbol{\mu} \text{ if } \nu > 1, \text{ otherwise undefined and} \\ \mathrm{Var}(\mathbf{y}) &= \frac{\nu}{\nu - 2}\boldsymbol{\Sigma} \text{ if } \nu > 2 \text{ otherwise undefined.} \end{aligned}$$

Numerical optimization is again required to estimate the parameters and we have to say

that in the special case of $\nu = 1$, the distribution is called multivariate Cauchy. The *MASS* library in R offers estimation of the mean vector and covariance matrix of this distribution for specific degrees of freedom. We have extended the *cov.trob* command to incorporate the degrees of freedom and end up with the maximum likelihood estimates for all the parameters.

The function will return the location and scatter matrix of the multivariate t distribution along with the degrees of freedom (ν) and also the classical mean vector and covariance matrix, which essentially are calculated assuming a multivariate normal. There is an option to construct asymptotic 95% confidence intervals for the degrees of freedom. If the argument *plot* is *TRUE*, the confidence intervals are presented graphically.

```
multivt <- function(y, plot = FALSE) {
  ## the next mvt function is for the appropriate
  ## degrees of freedom
  ## y contains the data
   mvt <- function(v, y, n, p) {
    ## the next function 'a' estimates the mean and covariance for given
    ## degeees of freedom. It's a built-in function
    a <- MASS::cov.trob(y, nu = v)
    se <- a$cov
    me <- as.vector(a$center)
    n * lgamma( (v + p)/2 ) - n * lgamma(v/2) - 0.5 * n * p *
    log(pi * v) - 0.5 * n * log( det(se) ) - 0.5 * (v + p) *
        sum( log1p( mahalanobis(y, me, se)/v ) )
   }
  dm <- dim(y)
  mod <- optimize(mvt, c(0.9, 20000), y = y, n = dm[1], p = dm[2],
       maximum = TRUE)
  dof <- mod$maximum
  loglik <- mod$objective
  ## df is the optimal degrees of freedom
  ## if the df is a big number, then a multivariate normal is fine as well
  result <- MASSS::cov.trob(y, nu = dof)  ## the center and covariance matrix

  ## will be calculated based on the optimal degrees of freedom
  ## the classical mean and covariance are given in the results
  ## for comparison pruposes
  apotelesma <- list(center = result$center, scatter = result$cov,
  df = dof, loglik = loglik, mesos = Rfast::colmeans(y), covariance = cov(y))
  if ( plot ) {
```

```
    lik <- deg <- seq(max(1, df - 20), df + 20, by = 0.1)
    for (i in 1:length(deg)) lik[i] <- mvt(y, deg[i])
    plot(deg, lik, type = "l", xlab = "Degrees of freedom",
    ylab = "Log likelihood")
    b <- max(lik) - 1.92
    abline(h = b, col = 2)
    a1 <- min(deg[lik >= b])
    a2 <- max(deg[lik >= b])
    abline(v = a1, col = 3, lty = 2)
    abline(v = a2, col = 3, lty = 2)
    conf <- c(a1, a2)
    names(conf) <- c("2.5%", "97.5%")
    apotelesma <- list(center = result$center, scatter = result$cov,
    df = dof, conf = conf, loglik = loglik, mesos = Rfast::colmeans(y),
    covariance = cov(y))
  }
  apotelesma
}
```

Nadarajah and Kotz (2008) stated a few methods for estimating the scatter and location parameters of the multivariate t distribution. One of them is the maximum likelihood estimation via the EM algorithm. The log-likelihood, ignoring constant terms is written as

$$\ell\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu\right) = -\frac{n}{2}\log|\boldsymbol{\Sigma}| - \frac{\nu + p}{2}\sum_{i=1}^{n}\log\left(\nu + s_i\right),$$

where n and p denote the sample size and number of dimensions respectively, ν are the degrees of freedom and $s_i = \left(\mathbf{y}_i - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_i - \boldsymbol{\mu}\right)$. Differentiating with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ leads to the following estimating equations

$$\begin{aligned} \boldsymbol{\mu} &= \frac{\sum_{i=1}^{n} w_i \mathbf{y}_i}{\sum_{i=1}^{n} w_i} \text{ and} \\ \boldsymbol{\Sigma} &= \frac{\sum_{i=1}^{n} w_i \left(\mathbf{y}_i - \boldsymbol{\mu}\right)\left(\mathbf{y}_i - \boldsymbol{\mu}\right)^T}{\sum_{i=1}^{n} w_i}, \\ \text{where } w_i &= \frac{\nu + p}{\nu + \left(\mathbf{y}_i - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_i - \boldsymbol{\mu}\right)}. \end{aligned}$$

Repeating the three equations above iteratively until the full log-likelihood does not change, is what the function below does.

```
multivt2 <- function(x, v = 5, tol = 1e-07){
```

```
## x contains the data
## v is the degrees of freedom, set to 5 by default
dm <- dim(x)
p <- dm[2]   ;    n <- dm[1]  ## dimensions
if ( v == 1 )  {
  R <- cov(x)
} else R <- abs( v - 1 ) / v  * cova(x)
m <- Rfast::colmeans(x)  ## initial parameters
con <- n * lgamma( (v + p)/2 ) - n * lgamma(v/2) - 0.5 * n * p * log(pi * v)
### step 1
wi <- (v + p) / ( v + Rfast::mahala(x, m, R) )  ## weights
y <- sqrt(wi) * ( Rfast::eachrow(x, m, oper = "-" ) )
sumwi <- sum(wi)
R <- crossprod(y) / sumwi   ## scatter estimate
m <- Rfast::colsums(wi * x) / sumwi  ## location estimate
dis <- Rfast::mahala(x, m, R)
el1 <-  - n * log( det(R) ) - (v + p) * sum( log1p(dis/v) )
### step 2
wi <- (v + p) / ( v + dis )  ## weights
y <- sqrt(wi) * Rfast::eachrow(x, m, oper = "-" )
sumwi <- sum(wi)
R <- crossprod(y) / sumwi  ## scatter estimate
m <- Rfast::colsums(wi * x) / sumwi  ## location estimate
dis <- Rfast::mahala(x, m, R)
el2 <-  - n * log( det(R) ) - (v + p) * sum( log1p(dis/v) )
## Step 3 and above
i <- 2
while ( el2 - el1 > tol ) { ## 1e-06 is the tolerance level
  ## between two successive values of the log-likelihood
  i <- i + 1
  el1 <- el2
  wi <- (v + p) / ( v + dis) ## updated weights
  y <- sqrt(wi) * Rfast::eachrow(x, m, oper = "-" )
  sumwi <- sum(wi)
  R <- crossprod(y) / sumwi  ## updated scatter estimate
  m <- Rfast::colsums(wi * x) / sumwi  ## updated location estimate
  dis <- Rfast::mahala(x, m, R)
  el2 <-  - n * log( det(R) )- (v + p) * sum( log1p(dis/v) )
}  ## updated log-likelihood
```

```
  list(iters = i, loglik = 0.5 * el2 + con, location = m, scatter = R)
}
```

5.1.4 Estimation of the parameters of a multivariate Laplace distribution

I wrote "a", instead of "the" multivariate Laplace distribution because I am bit confused with its many representations. The one I will use here can be found in Eltoft et al. (2006)

$$f_p(\mathbf{y}) = \frac{2}{\lambda (2\pi)^{p/2}} \frac{K_{p/2-1}\left(\sqrt{\frac{2}{\lambda} q(\mathbf{y})}\right)}{\left(\sqrt{\frac{2}{\lambda} q(\mathbf{y})}\right)^{(p/2)-1}}, \tag{5.2}$$

where p stands for the number of variables and $q(\mathbf{y}) = (\mathbf{y}-\boldsymbol{\mu})^T \boldsymbol{\Gamma}^{-1} (\mathbf{y}-\boldsymbol{\mu})$. The matrix $\boldsymbol{\Gamma}$ is a covariance structure matrix, but it has the constraint that is determinant is 1, $\det(\boldsymbol{\Gamma}) = 1$. $K_m(x)$ denotes the modified Bessel function of the second kind and order m, evaluated at x. The support of this distribution is the whole of $\mathbb{R}^p$.

I will only mention the first way of obtaining (moment) estimates for the parameters $\boldsymbol{\mu}$, $\boldsymbol{\Gamma}$ and λ of (5.2). The sample mean is the estimate of the location parameter, $\hat{\boldsymbol{\mu}} = \bar{\mathbf{y}}$. Let $\hat{\mathbf{R}} = \frac{1}{n-1}\sum_{i=1}^{n} (\mathbf{y}-\boldsymbol{\mu})(\mathbf{y}-\boldsymbol{\mu})^T$ be the unbiased sample covariance matrix of the n observations $\mathbf{y}_i$. Eltoft et al. (2006) divide by n and not $n-1$. I did some simulations and saw that the estimate of λ changes slightly, but the MSE and bias are slightly better with the unbiased covariance matrix. Since $\det(\boldsymbol{\Gamma}) = 1$, $\hat{\lambda} = [\det(\mathbf{R})]^{1/p}$ and thus $\hat{\boldsymbol{\Gamma}} = \frac{1}{\hat{\lambda}}\hat{\mathbf{R}}$.

```
mom.mlaplace <- function(x) {
  ## x contains the data
  n <- dim(x)[1]  ## sample size of  the data
  d <- dim(x)[2]  ## dimensionality of  the data
  mu <- Rfast::colmeans(x)
  R <- cov(x)
  lam <- det(R)^(1/d)  ## estimate of the mean of the exponential distribution
  G <- R/lam  ## covariance matrix with determinant 1
  list(mu = mu, G = G, lambda = lam)
}
```

5.1.5 Estimation of the parameters of an inverted Dirichlet distribution

A vector $\mathbf{x} \in \mathbb{R}_+^p$ follows the inverted Dirichlet distribution if its density is

$$f(\mathbf{x}; \boldsymbol{\alpha}) = \frac{1}{\mathbf{B}(\boldsymbol{\alpha})} \frac{\prod_{i=1}^{p} x_i^{\alpha_i - 1}}{\left(1 + \sum_{i=1}^{p} x_i\right)^{\sum_{j=1}^{p+1} \alpha_j}},$$

where

$$\mathbf{B}(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{p+1} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{p+1} \alpha_i\right)} \text{ and } \boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_p, \alpha_{p+1}).$$

The expected value, variances and covariances are given by

$$\begin{aligned} \mathrm{E}(x_i) &= \frac{\alpha_i}{\alpha_{d+1} - 1} \text{ if } \alpha_{d+1} > 1 \\ \mathrm{Var}(x_i) &= \frac{\alpha_i (\alpha_i + \alpha_{d+1} - 1)}{(\alpha_{d+1} - 1)^2 (\alpha_{d+1} - 2)} \text{ if } \alpha_{d+1} > 2 \\ \mathrm{Cov}(x_i, x_j) &= \frac{\alpha_i \alpha_j}{(\alpha_{d+1} - 1)^2 (\alpha_{d+1} - 2)} \text{ if } \alpha_{d+1} > 2 \end{aligned}$$

All we do in the next R function is to write down the log-likelihood of the inverted Dirichlet density and given a sample to maximise it with respect to the vector of $\boldsymbol{\alpha}$ using *optim*.

```
invdir.est <- function(x) {
  ## x is the positive data
  x <- as.matrix(x)  ## makes sure x is a matrix
  n <- dim(x)[1]  ## sample size
  d <- dim(x)[2]  ## dimensionality of x
  zx <- t( log(x) )
  rsx <- sum( log1p( Rfast::rowsums(x) ) )

   ## loglik is for the mle of the alphas
   loglik <- function(b, zx = zx, rsx = rsx, n, d) {
     a <- exp(b)
     sa <- sum(a)
     - n * lgamma( sa ) + n * sum( lgamma(a) ) -
          sum( zx * (a[1:d] - 1) ) + sa * rsx
   }
  oop <- options(warn = -1)
  on.exit(options(oop))
```

```
  da <- nlm(loglik, c( log( Rfast::colmeans(x) ), log( max(min(x), 3) ) ),
        zx = zx, rsx = rsx, n = n, d = d, iterlim = 2000)
  da <- nlm( loglik, da$estimate, zx = zx, rsx = rsx, n = n, d = d,
        iterlim = 2000 )
  da <- optim( da$estimate, loglik, zx = zx, rsx = rsx, n = n, d = d,
        control = list(maxit = 20000), hessian = TRUE )
  lik = -da$value     ;     a <- exp(da$par)
  mu <- NULL
  s <- NULL

  if (a[d + 1] > 1)  {
    mu <- a[1:d]/(a[d + 1] - 1)
  } else {
    mu <- paste("Mean vector does not exist")
    s <- paste("Covariance matrix does not exist")
  }

  if (a[d + 1] > 2) {
    mu <- a[1:d]/(a[d + 1] - 1)
    s <- matrix(nrow = d, ncol = d)
    for (i in 1:d) {
      for (j in 1:d) {
        down <- (a[d + 1] - 1)^2 * (a[d + 1] - 2)
        if (i == j)
          s[i, j] <- a[i] * (a[i] + a[d + 1] - 1)/down
        if (i != j)
          s[i, j] <- a[i] * a[j]/down
      }
    }

  }  else  s <- paste("Covariance matrix does not exist")

  list(loglik = lik, alpha = a, mu = mu, s = s)
}
```

5.1.6 Multivariate kernel density estimation

We saw kernel regression before but now we will see multivariate kernel density estimation. The kernel to be used is again that of a multivariate standard normal distribution

$$K(\mathbf{x}) = \frac{e^{-\frac{1}{2}\mathbf{x}^T\mathbf{x}}}{(2\pi)^{p/2}},$$

where p is the dimensionality of the data respectively. This leads to the following kernel estimator Wand and Jones (1995)

$$\begin{aligned} \hat{f}(\mathbf{x};\mathbf{H}) &= \frac{1}{n\prod_{j=1}^{p}\mathbf{H}_{jj}^{1/2}}\sum_{i=1}^{n}K\left[\mathbf{H}^{-1/2}(\mathbf{x}-\mathbf{X}_i)\right] \\ &= \frac{1}{n\prod_{j=1}^{p}\mathbf{H}_{jj}^{1/2}}\sum_{i=1}^{n}\frac{e^{-\frac{1}{2}(\mathbf{x}-\mathbf{X}_i)^T\mathbf{H}^{-1}(\mathbf{x}-\mathbf{X}_i)}}{(2\pi)^{p/2}}, \end{aligned} \tag{5.3}$$

where n is the sample size and h is the bandwidth parameter to be chosen. I have assumed that $\mathbf{H}$ is a $p \times p$ diagonal matrix, where $\mathbf{H} = h^2\mathbf{I}_q$, where $\mathbf{I}_q$ is the identity matrix for all $i = 1, \ldots, p$. In the next R function the user can either specify his own diagonal bandwidth matrix $\mathbf{H}$, a matrix with the same elements in the diagonal, so give only a scalar or use a rule of thumb, either Silverman's or Scott's. The Silverman and the Scott rules are

$$\begin{aligned} \mathbf{H}_{ii}^{1/2} &= \left(\frac{4}{p+2}\right)^{\frac{1}{p+4}} n^{\frac{-1}{p+4}}\sigma_i \text{ and} \\ \mathbf{H}_{ii}^{1/2} &= n^{\frac{-1}{p+4}}\sigma_i \text{ respectively,} \end{aligned}$$

where σ_i is the standard deviation of the i-th variable. The idea is that for every $\mathbf{x}$ one must calculate (5.3) n times, for all the observations in the sample. The following function does that

```
mkde <- function(x, h, thumb = "silverman") {
  ## h is the h you want, which is either a vector or a single number
  ## thumb can be either "none" so the specified h is used, or
  ## "scott", or "silverman"
  n <- dim(x)[1]
  d <- dim(x)[2]  ## sample and dimensionality of x

  if ( thumb == "silverman" ) {
    s <- Rfast::colVars(x, std = TRUE)
    h <- ( 4/(d + 2) )^( 1/(d + 4) ) * s * n^( -1/(d + 4) )
```

```
  } else  if ( thumb == "scott" ) {
    s <- Rfast::colVars(x, std = TRUE)
    h <- s * n^( -1/(d + 4) )
  } else if ( thumb == "estim" ) {
    h <- mkde.tune(x)$hopt
  } else  h <- h

  if ( length(h) == 1 ) {
    h <- diag( 1 / h, d )
  } else h <- diag( 1 / h)

  con <- prod( diag( h ) )
  y <- x %*%  h
  a1 <- Rfast::Dist(y, method = "euclidean", square = TRUE)
  (0.5 / pi)^(d/2) * con * Rfast::rowmeans( exp( - 0.5 * a1 ) )
}
```

How does one choose h? What I have done here is maximum likelihood cross validation (Guidoum, 2015). Actually this way was proposed by Habbema et al. (1974) and Robert (1976). The cross validation idea is to replace $\hat{f}(x;h)$ in (5.3) by the leave-one-out estimator

$$\hat{f}_{-i}(\mathbf{x}_i;h) = \frac{1}{(n-1)h^p}\sum_{j\neq i}^{n} K\left(\frac{\mathbf{x}_i - \mathbf{X}_j}{h}\right). \tag{5.4}$$

Then choose the value of h which maximizes

$$MLCV(h) = \frac{1}{n}\sum_{i=1}^{n}\log\left[\hat{f}_{-i}(\mathbf{x}_i;h)\right]. \tag{5.5}$$

Wand and Jones (1995) mention that since we have one value of h for all observations, and not a different one for every variable (in fact it could be a bandwidth matrix $\mathbf{H}$) we should transform the data so that they have unit covariance matrix first and then try to choose the value of h. The same is noted in Silverman (1986) who also mentions that instead of transforming the data, choose h and then transform them back to calculate the kernel density he says that this is equivalent to using a kernel density estimation where the covariance matrix is already there

$$\hat{f}(x;h) = \frac{1}{|\mathbf{S}|^{1/2}nh^p}\sum_{i=1}^{n}\frac{e^{-\frac{1}{2}h^{-2}(\mathbf{x}-\mathbf{X}_i)^T\mathbf{S}^{-1}(\mathbf{x}-\mathbf{X}_i)}}{(2\pi)^{p/2}}, \tag{5.6}$$

where $\mathbf{S}$ is the sample covariance matrix. So we use 5.6 in the calculations of 5.4 and 5.5 in

order to choose h and then use 5.3 in order to calculate the kernel density. Silverman (1986) advises us to use a robust version **S**. We have implemented the next code using the classical covariance matrix, but there is the option to change it.

```
mkde.tune_2 <- function(x, h = seq(0.1, 1, by = 0.01), plot = TRUE, ncores = 4) {
  x <- as.matrix(x)  ## makes sure x is a matrix
  ## h is the bandwidth
  ## s can be replaced by another covariance type matrix
  ## ncores is the number of cores you want to use
  ## requires(doParallel)

  n <- dim(x)[1]
  d <- dim(x)[2]  ## sample and dimensionality of x
  s <- cov(x) ## can put a robust covariance matrix here if you want
  cv <- numeric( length(h) )

  eig <- eigen(s, symmetric = TRUE)
  lam <- eig$values  ## eigenvalues of the covariance matrix
  vec <- eig$vectors  ## eigenvectors of the covariance matrix
  B <- vec %*% ( t(vec)* ( 1/ sqrt(lam) ) )
  z <- x %*% B
  a2a <- Rfast::Dist( z, square = TRUE )
  a2a <- exp(-0.5 * a2a)
  ds <- 1 / prod(lam)^0.5

  if (ncores > 1) {
    require(doParallel, quiet = TRUE, warn.conflicts = FALSE)
    oop <- options(warn = -1)
    on.exit(options(oop))
    val <- matrix(h, ncol = ncores) ## if the length of h is not equal to the
    ## dimensions of the matrix val a warning message should appear
    ## but you will not see it
    cl <- makePSOCKcluster(ncores)
    registerDoParallel(cl) ## make the cluster
    ww <- foreach(j = 1:ncores, .combine = cbind) %dopar% {
      ba <- val[, j]
      for ( l in 1:length( val[, j] ) ) {
        a <- a2a^( 1 / val[l, j]^2 )

        f <- ds / (2 * pi) ^ (d/2) * ( 1 / val[l, j]^d ) *
```

```
          ( rowSums( a ) - 1 ) / (n - 1)
        ba[l] <- mean( log(f) )
      }
      return(ba)
    }
    stopCluster(cl) ## stop the cluster
    cv <- as.vector(ww)[ 1:length(h) ]
  } else{
    for ( j in 1:length( h ) ) {
      a <- a2a^( 1 / val[l, j]^2 )
      f <- ds / (2 * pi) ^ (d/2) * ( 1 / h[j]^d ) *
        ( rowSums( a ) - 1 ) / (n - 1)
      cv[j] <- mean( log(f) )
    }
  }

  if ( plot ) {
    plot(h, cv, type = "l")
  }
  hopt <- h[ which.max(cv) ]
  list(hopt = hopt, cv = cv)
}
```

I could have incorporated the *mkde.function* inside *mkde*, but I wanted to leave them as two separate functions so that the user has a better control of what is happening. An alternative and faster function is given below. The idea is simple, there is one parameter, h over which we want to maximize an objective function. So, it is a unidimensional optimization and the command *optimize* in R will do the rest.

```
mkde.tune <- function( x, low = 0.1, up = 3, s = cov(x) ) {
  ## x contains the multivariate data
  ## low and up are the limits within which the
  ## search is conducted
  n <- dim(x)[1]
  d <- dim(x)[2]  ## sample and dimensionality of x
  s <- s ## can put a robust covariance matrix here if you want
  eig <- eigen(s)
  lam <- eig$values  ## eigenvalues of the covariance matrix
  vec <- eig$vectors  ## eigenvectors of the covariance matrix
  B <- vec %*% ( t(vec) / sqrt(lam) )
```

```
  z <- x %*% B
  a2a <- Rfast::Dist( z, square = TRUE )
  a2a <- exp(-0.5 * a2a)
  ds <- 1 / prod(lam)^0.5

  tune <- function(h) {
    a <- a2a^( 1 / h^2 )
    - n * d * log(h) + mean( log( rowSums( a ) - 1 ) )
  }
  low <- low     ;    up <- up
  bar <- optimize(tune, c(low, up), maximum = TRUE)
  list( hopt = bar$maximum, maximum = bar$objective + log(ds) -
  d/2 * log(2 * pi) - log(n - 1) )
}
```

5.1.7 Bivariate Poisson distribution

This is a discrete distribution, nevertheless, it falls within the multivariate context, even in the two dimensions. In order to generate values from this distribution one needs three independent Poisson variables, $X_1 \sim \text{Po}(\lambda_1)$, $X_2 \sim \text{Po}(\lambda_2)$ and $X_3 \sim \text{Po}(\lambda_3)$. Then, $(X, Y) = (Y_1, Y_2) = (X_1 + X3, X_2 + X_3) \sim \text{BP}(\lambda_1, \lambda_2, \lambda_3)$. This was a way to simulate random values from the bivariate Poisson whose representation is given by

$$P(X = x, Y = y)\, e^{-(\lambda_1+\lambda_2+\lambda_3)} \frac{\lambda_1^x}{x!} \frac{\lambda_2^y}{y!} \sum_{k=0}^{\min(x,y)} \binom{x}{k} \binom{y}{k} k! \left(\frac{\lambda_3}{\lambda_1 \lambda_2} \right)^k. \tag{5.7}$$

The above form is found in Karlis and Ntzoufras (2003). This bivariate distribution allows for dependence between the two random variables. Marginally each random variable follows a Poisson distribution with $E(X) = \lambda_1 + \lambda_3$ and $E(Y) = \lambda_2 + \lambda_3$. In addition, $Cov(X, Y) = \lambda_3$. If $\lambda_3 = 0$, (5.7) becomes a product of two Poisson distributions. Hence, λ_3 is a measure of dependence between the two random variables. For more information about this distribution one can see Kocherlakota and Kocherlakota (1998) and for multivariate generalisations see Karlis (2003) and Karlis and Meligkotsidou (2005).

The next R code has the following stuff

Maximum likelihood estimation of the parameters

In order to estimate the triplet of parameters $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \lambda_3)$ we have to maximise the corresponding log-likelihood. Karlis and Ntzoufras (2003) implements an EM algorithm for this reason. I have chosen to use the function *optim* in R. Kawamura (1984) mentions that

the MLE of $\lambda_1 + \lambda_3$ and $\lambda_2 + \lambda_3$ are given by $\frac{\sum_{i=1}^{n} x_i}{n}$ and $\frac{\sum_{i=1}^{n} y_i}{n}$ respectively, where n denotes the sample size. Therefore, we can rewrite (5.7) as a function of λ_3 only and thus maximise the log-likelihood with respect to λ_3 only. Then $\hat{\lambda}_1 = \frac{\sum_{i=1}^{n} x_i}{n} - \hat{\lambda}_3$ and $\hat{\lambda}_2 = \frac{\sum_{i=1}^{n} y_i}{n} - \hat{\lambda}_3$.

Correlation coefficient

This form of the bivariate Poisson (5.7) allows for non negative correlation. The correlation coefficient is given by

$$\rho(X, Y) = \frac{\lambda_3}{\sqrt{(\lambda_1 + \lambda_3)(\lambda_2 + \lambda_3)}}. \tag{5.8}$$

The sample correlation coefficient comes from substituting the parameters by their sample estimates. We had done some simulations (Tsagris et al., 2012) comparing the Pearson correlation coefficient with the MLE based one (5.8) and we found that the Pearson behaves very well in terms of coverage of the confidence interval. The thing with the Pearson correlation coefficient is that it can take negative values as well, wheres (5.8) cannot. But, it can happen, that even under this model, negative sample correlation is observed.

Covariance matrix of the parameters

The covariance matrix of $(\lambda_1, \lambda_2, \lambda_3)$ is (Kocherlakota and Kocherlakota, 1998)

$$\frac{1}{n}\begin{pmatrix} \lambda_1 + \lambda_3 & \lambda_3 & \lambda_3 \\ \lambda_3 & \lambda_2 + \lambda_3 & \lambda_3 \\ \lambda_3 & \lambda_3 & \frac{\lambda_1\lambda_2 + \lambda_3(\lambda_1+\lambda_2)[\lambda_3(\tau-1)-1]}{\delta_2} \end{pmatrix},$$

where

$$\begin{aligned} \delta_2 &= -(\lambda_1 + \lambda_2) + \left[(\lambda_1 + \lambda_3)(\lambda_2 + \lambda_3) - \lambda_3^2\right](\tau - 1) \quad \text{and} \\ \tau &= \sum_{r=1}^{\infty}\sum_{s=1}^{\infty} \frac{P(X = r-1, Y = s-1)^2}{P(X = r, Y = s)}. \end{aligned}$$

Hypothesis testing for independence

Hypothesis testing for $H_0 : \lambda_3 = 0$ versus $H_1 : \lambda_3 > 0$ is performed via the Wald test statistic Kocherlakota and Kocherlakota (1998) $W = \frac{\hat{\lambda}_3}{\sqrt{\text{Var}(\hat{\lambda}_3)}}$. As for the variance of $\hat{\lambda}_3$ we can take it from the covariance matrix we saw before or using the observed information matrix (available from the *optim* function). Under the null hypothesis, the asymptotic distri-

bution of W is $N(0,1)$. Alternatively we can use the log-likelihood ratio test with a χ^2_1.

Confidence intervals for λ_3

In order for the log-likelihood ratio test not to reject H_0 at $\alpha = 5\%$, the following must hold true $2(\ell_1 - \ell_0) \leq \chi^2_{1,0.95}$, where ℓ_1 and ℓ_0 are the maximised log-likelihood values under H_1 and H_0 respectively. Thus, an asymptotic 95% confidence interval for λ_3 consists of the values of $\hat{\lambda}_3$ for which the log-likelihood is more than $\ell_1 - \frac{\chi^2_{1,0.95}}{2}$. Alternatively, asymptotic standard normal confidence intervals can be used, since we know the variance of $\hat{\lambda}_3$.

```
bp <- function(x1, x2, plot = FALSE) {
  ## x1 and x2 are the two variables
  n <- length(x1)  ## sample size
  m1 <- sum(x1) / n    ;    m2 <- sum(x2) / n
  ##  m1 and m2 estimates of lambda1* and lambda2* respectively
  ##  funa is the function to be maximised over lambda3
  ind <- Rfast::rowMins( cbind(x1, x2), value = TRUE )
  max1 <- max(x1)
  max2 <- max(x2)
  mm <- max( max1, max2 )
  mn <- min(max1, max2)
  omn <- 0:mn
  fac <- factorial( omn )
  ch <- matrix(numeric( (mm + 1)^2 ), nrow = mm + 1, ncol = mm + 1 )
  rownames(ch) <- colnames(ch) <- 0:mm

  for ( i in 1:c(mm + 1) ) {
    for ( j in c(i - 1):c(mm + 1) ) {
      ch[i, j] <- choose(j, i - 1)
    }
  }
  ly1 <- lgamma(x1 + 1)
  ly2 <- lgamma(x2 + 1)

  funa <- function(l3, n) {
    f <- f1 <- f2 <- numeric(n)
    con <-  - m1 - m2 + l3
    expo <- ( l3/( (m1 - l3) * (m2 - l3) ) )^omn
    l1 <- log(m1 - l3)
    l2 <- log(m2 - l3)
```

```
    for (j in 1:n) {
      f1[j] <- x1[j] * l1 - ly1[j] + x2[j] * l2 - ly2[j]
      f2[j] <- log( sum( ch[ 1:c(ind[j] + 1), x1[j] ] *
      ch[ 1:c(ind[j] + 1), x2[j] ] * fac[1:c(ind[j] + 1)] *
      expo[ 1:c(ind[j] + 1) ] ) )
    }
     n * con + sum(f1) + sum( f2[abs(f2) < Inf] )
  }

  if ( plot ) {  ## should a plot of the log-likelihood appear
    a <- b <- seq(0, min(m1, m2) - 0.1, by = 0.01)
    for (i in 1:length(a) )  b[i] <- funa(a[i])
    plot(a, b, ylab = 'Log-likelihood', type = 'l',
    xlab = expression(paste("Values of ", lambda[3])) )
    abline(v = a[which.max(b)], col = 2, lty = 2)
    cl <- max(b) - 1.92
    abline(h = cl, col = 2)
    a1 <- min(a[b >= cl]) ; a2 <- max(a[b >= cl])
    abline(v = a1, col = 3, lty = 2)
abline(v = a2, col = 3, lty = 2)
  }

  bar <- optim( cov(x1, x2), funa, n = n, control = list(fnscale = -1),
  method = "L-BFGS-B", lower = 0, upper = min(m1, m2) - 0.05, hessian = TRUE )
  l1 <- bar$value  ## maximum of the log-likelihood
  l3 <- bar$par  ## lambda3 estimate
  rho <- l3 / sqrt(m1 * m2)  ## correlation coefficient
  names(rho) <- "correlation"
  l0 <- funa(0, n)  ## log-likelihood with lam3=0, independence
  test <- 2 * (l1 - l0)  ## log-likelihood ratio test
  pval1 <- pchisq(test, 1, lower.tail = FALSE)
  ma <- mm + 20
  f1 <- f2 <- matrix(nrow = ma, ncol = ma)
  con <-  - m1 - m2 + l3

  for (r in 1:ma) {
    for (s in 1:ma) {
      i <- 0:min(r, s)
  comon <- factorial(i) * ( l3/( (m1 - l3) * (m2 - l3) ) )^i
```

```
    f1[r, s] <-  con + (r - 1) * log(m1 - l3) - lgamma(r) +
    (s - 1) * log(m2 - l3) - lgamma(s) + log(sum( choose(r - 1, i) *
    choose(s - 1, i) * comon ) )

    f2[r, s] <- con + r * log(m1 - l3) - lgamma(r + 1) +
    s * log(m2 - l3) - lgamma(s + 1) + log(sum( choose(r, i) *
choose(s, i) * comon ) )
  }
}

tau <- sum( exp(f1)^2/exp(f2) )
d2 <-  -m1 + l3 - m2 + l3 + (m1 * m2 - l3^2) * (tau - 1)
s <- matrix(c(m1, l3, l3, l3, m2, l3, l3, l3,
( (m1 - l3) * (m2 - l3) + l3 * (m1 - l3 + m2 - l3) *
(l3 * (tau - 1) - 1) )/d2) , ncol = 3) / n
v1 <-  -1/bar$hessian ## variance of l3 using the observed information matrix
v2 <- s[3, 3] ## variance of l3 using the asymptotic covariance matrix
t1 <- l3 / sqrt(v1)  ## Wald test 1
t2 <- l3 / sqrt(v2)  ## wald test 2
pval2 <- pnorm(-t1)
pval3 <- pnorm(-t2)
pvalue <- c(pval1, pval2, pval3)

names(pvalue) <- c('LLR', 'Wald 1', 'Wald 2')
ci <- rbind( c(l3 - 1.96 * sqrt(v1), l3 + 1.96 * sqrt(v1)),
      c(l3 - 1.96 * sqrt(v2), l3 + 1.96 * sqrt(v2)) )
colnames(ci) <- c('2.5%', '97.5%')
rownames(ci) <- c('Observed I', 'Asymptotic I')

if ( plot ) {
  ci <- rbind( c(a1, a2), ci )
  rownames(ci)[1] <- 'Log-likelihood'
}

loglik <- c(l1, l0)
names(loglik) <- c('loglik1', 'loglik0')
lambda <- c(m1 - l3, m2 - l3, l3)
names(lambda) <- c('Lambda1', 'Lambda2', 'Lambda3')
```

```
  list(lambda = lambda, rho = rho, ci = ci, loglik = loglik, pvalue = pvalue)
}
```

5.1.8 A goodness of fit test for the bivariate Poisson

Kocherlakota and Kocherlakota (1998) mention the following a goodness of fit test for the bivariate Poisson distribution, the index of dispersion test. It is mentioned in Kocherlakota and Kocherlakota (1998) that Loukas and Kemp (1986) developed this test as an extension of the univariate dispersion test. They test for departures from the bivariate Poisson againsta alternatives which involve an increase in the generalised variance, the determinant of the covariance matrix of the two variables.

Rayner et al. (2009) mention a revised version of this test whose test statistic is now given by

$$I_{B^*} = \frac{n}{1-r^2}\left(\frac{S_1^2}{\bar{x}_1} - 2r^2\sqrt{\frac{S_1^2}{\bar{x}_1}\frac{S_2^2}{\bar{x}_2}} + \frac{S_2^2}{\bar{x}_2}\right),$$

where n is the sample size, r is the sample Pearson correlation coefficient, S_1^2 and S_2^2 are the two sample variances and $\bar{x}_1$ and $\bar{x}_2$ are the two sample means. Under the null hypothesis the I_{B^*} follows asymptotically a χ^2 with $2n-3$ degrees of freedom. However, I did some simulations and I saw that it does not perform very well in terms of the type I error. If you see the simulations in their book Rayner et al., 2009, pg. 132 you will see this. For this reason, the next R function calculates the p-value of the I_{B^*} using Monte Carlo.

```
bp.gof <- function(x1, x2, R = 999) {
  ## x1 and x2 are the two variables
  runtime <- proc.time()
  n <- length(x1)  ## sample size
  r <- cor(x1, x2)  ## Pearson correlation coefficient
  m1 <- sum(x1) / n     ;     m2 <- sum(x2)/n
  v1 <- ( sum(x1^2) - n * m1^2 ) / (n - 1)
  v2 <- ( sum(x2^2) - n * m2^2 ) / (n - 1)

  Ib <- n/(1 - r^2) * ( v1 / m1 + v2 / m2 -
  2 * r^2 * sqrt(v1 / m1 * v2 / m2) )  ## test statistic
  tab <- table(x1, x2)
  tb <- numeric(R)
  lambda <- bp(x1, x2, plot = FALSE)$lambda

  for (i in 1:R) {
```

```
    z3 <- rpois(n, lambda[3])
    z1 <- rpois(n, lambda[1]) + z3
    z2 <- rpois(n, lambda[2]) + z3
    r <- cor(z1, z2)
    m1 <- sum(z1)/n    ;    m2 <- sum(z2)/n
    s1 <- ( sum(z1^2) - n * m1^2 ) / (n - 1)
    s2 <- ( sum(z2^2) - n * m2^2 ) / (n - 1)
    tb[i] <- n/(1 - r^2) * ( s1 / m1 + s2 / m2 -
    2 * r^2 * sqrt( s1 / m1 * s2 / m2 ) )
  }

  pvalue <- (sum(tb > Ib) + 1)/(R + 1)
  runtime <- proc.time() - runtime
  list(runtime = runtime, pvalue = pvalue, tab = tab)
}
```

The function *biv.gof* given below is the vectorised version of *bp.gof*. The work similar to the one described in te bootstrap correlation coefficient. There are 5 terms required for the correlation and then 4 of them are used in the test statistic. Note, that to calculate the correlation coefficient we have used an alternative form of the correlation formula, which suited here better. The results are the same as before. *biv.gof* is much faster for small to moderate sample sizes, but for bigger samples the time differences with *bp.gof* become smaller. In any case, it's good to be here, so that you can see this one as well and think how to vectorise (if possible) your functions.

```
biv.gof <- function(x1, x2, R = 999) {
  ## x1 and x2 are the two variables
  runtime <- proc.time()
  n <- length(x1)  ## sample size
  r <- cor(x1, x2)  ## Pearson correlation coefficient
  m1 <- mean(x1)     ;    m2 <- mean(x2)
  Ib <- n/(1 - r^2) * ( var(x1) / m1 + var(x2) / m2 -
  2 * r^2 * sqrt( var(x1) / m1 * var(x2) / m2 ) )  ## test statistic
  tab <- table(x1, x2)

  lambda <- bp(x1, x2, plot = FALSE)$lambda
  z3 <- matrix( rpois(R * n, lambda[3]), ncol = R )
  z1 <- matrix( rpois(R * n, lambda[1]), ncol = R ) + z3
  z2 <- matrix( rpois(R * n, lambda[2]), ncol = R ) + z3
  m1 <- Rfast::colmeans(z1)     ;    m2 <- Rfast::colmeans(z2)
```

```
  v1 <- Rfast::colVars(z1)   ;    v2 <- Rfast::colVars(z2)
  sxy <- Rfast::colsums(z1 * z2)
  rb <- (sxy - n * m1 * m2) / ( (n - 1) * sqrt( v1 * v2 ) )
  tb <- n/(1 - rb^2) * ( v1 / m1 + v2 / m2 - 2 * rb^2 *
  sqrt( v1 / m1 * v2 / m2 ) )

  pvalue <- ( sum(tb > Ib) + 1 ) / (R + 1)
  runtime <- proc.time() - runtime
  list(runtime = runtime, pvalue = pvalue, tab = tab)
}
```

5.1.9 Estimating the parameters of a Dirichlet-Multinomial distribution

Minka (2000) suggested a nice and fast way to estimate the parameters of the Dirichlet-multinomial distribution. But, at first let us see what is this distribution. Assume you have multinomial data, as shown in Table 5.1, where each row is a vector. The multinomial is the generalisation of the binomial to more than two categories. that Note that for the multinomial distribution, the row sums are assumed to be the same. For the Dirichlet-multinomial this is not the case.

Table 5.1: An example of multinomial data

X_1	X_2	X_3
4	5	6
3	2	0
5	4	3
$\vdots$	$\vdots$	$\vdots$

We can say that $\mathbf{X} \sim Mult\left(p_1, p_2, p_3\right)$. If we assume that the probabilities $\mathbf{p} = \left(p_1, p_2, p_3\right)$ follow a Dirichlet distribution with some parameters $\left(a_1, a_2, a_3\right)$ (prior distribution), then the posterior distribution with the probabilities $\mathbf{p}$ integrated out is a Dirichlet-multinomial and its density is given by

$$f\left(\mathbf{x}|\mathbf{a}\right) = \frac{\Gamma\left(\sum_{i=1}^{D} a_i\right)}{\Gamma\left(\sum_{i=1}^{D} x_i + \sum_{i=1}^{D} a_i\right)} \prod_{i=1}^{D} \frac{\Gamma\left(x_i + a_i\right)}{\Gamma\left(a_i\right)}$$

In some, the vector of parameters $\mathbf{a}$ can be estimated via the fixed-point iteration

$$a_i^{(k+1)} = a_i^{(k)} \frac{\sum_{j=1}^{n} \psi\left(x_{ij} + a_i\right) - n\psi\left(a_i\right)}{\sum_{j=1}^{n} \psi\left[\sum_{i=1}^{D}\left(x_{ij} + a_i\right)\right] - n\psi\left(\sum_{i=1}^{D} a_i\right)},$$

where $\psi(y) = (\log \Gamma(y))' = \frac{\Gamma'(y)}{\Gamma(y)}$ is the digamma function.

```
dirimultinom2 <- function(x, tol = 1e-07) {
  ## x is the data, integers
  ## tol is the tolerannce level, set to 10^(-7) by default
  p <- dim(x)[2]  ## dimensionality
  n <- dim(x)[1]  ## sample size
  lik <- NULL

  runtime <- proc.time()

  rs <- Rfast::rowsums(x)
  a1 <- Rfast::colmeans(x)
  sa <- sum(a1)
  x <- t(x)
  y <- x + a1
  lik[1] <- n * lgamma( sa ) - sum( lgamma( rs + sa ) ) +
  sum( lgamma( y ) ) - n * sum( lgamma( a1 ) )

  up <- Rfast::rowsums( digamma( y ) ) - n * digamma(a1)
  down <- sum( digamma( rs + sa  ) ) - n * digamma( sa )
  a2 <- a1 * up / down
  sa <- sum(a2)

  lik[2] <- n * lgamma( sa ) - sum( lgamma( rs + sa ) ) +
  sum( lgamma( x + a2 ) ) - n * sum( lgamma( a2 ) )

  i <- 2
  while ( (lik[i] - lik[i-1] > tol) ) {
    i <- i + 1
    a1 <- a2
    up <- Rfast::rowsums( digamma( x + a1 ) ) -
    n * digamma(a1)
    down <- sum( digamma( rs + sum(sa) ) ) -
    n * digamma( sa )
    a2 <- a1 * up / down
    sa <- sum(a2)
    lik[i] <- n * lgamma( sa ) - sum( lgamma( rs + sa ) ) +
    sum( lgamma( x + a2 ) ) - n * sum( lgamma( a2 ) )
  }
```

```
  runtime <- proc.time() - runtime
  list(runtime = runtime, iter = i, loglik = lik[i], param = a2)
}
```

Below is a faster algorithm based on Newton-Raphson. All we need to do is calculate the first and second derivatives.

```
dirimultinom <- function(x, tol = 1e-07) {
  ## x is the data, integers
  ## tol is the tolerannce level, set to 10^(-7) by default
  x <- as.matrix(x)  ## makes sure x is a matrix
  p <- dim(x)[2]  ## dimensionality
  n <- dim(x)[1]  ## sample size
  lik <- NULL

  rs <- Rfast::rowsums(x)
  a1 <- Rfast::colmeans(x)
  x <- t(x)
  y <- x + a1
  sa <- sum(a1)
  lik1 <- n * lgamma( sa ) - sum( lgamma( rs + sa ) ) - n * sum( lgamma( a1 ) ) +
  sum( lgamma( y ) )
  f <- n * digamma(sa) - sum( digamma(rs + sa) ) - n * digamma(a1) +
  rowsums( digamma(y) )
  f2 <- matrix(n * trigamma(sa) - sum( trigamma(rs + sa) ), p, p)
  diag(f2) <- diag(f2) - n * trigamma(a1) + rowsums( trigamma(y) )
  a2 <- a1 - solve(f2, f)
  sa <- sum(a2)
  lik2 <- n * lgamma( sa ) - sum( lgamma( rs + sa ) ) - n * sum( lgamma( a2 ) ) +
  sum( lgamma( x + a2 ) )

  i <- 2
  while ( sum( lik2 - lik1 ) > tol ) {
    i <- i + 1
    lik1 <- lik2
    a1 <- a2
    y <- x + a1
    f <- n * digamma(sa) - sum( digamma(rs + sa) ) - n * digamma(a1) +
    rowsums( digamma(y) )
```

```
    f2 <- matrix(n * trigamma(sa) - sum( trigamma(rs + sa) ), p, p)
    diag(f2) <-  diag(f2) - n * trigamma(a1) + rowsums( trigamma(y) )
    a2 <- a1 - solve(f2, f)
    sa <- sum(a2)
    lik2 <- n * lgamma( sa ) - sum( lgamma( rs + sa ) ) - n * sum( lgamma( a2 ) ) +
    sum( lgamma( x + a2 ) )
  }

  list(iters = i, loglik = lik2, param = a2)
}
```

5.2 Random values generation

5.2.1 Random values generation from a multivariate normal distribution

The previous function gives rise to a way to simulate from a multivariate normal with some specific parameters. The idea is simple. Suppose we want to generate n values from a p-variate normal with parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ using the *rnorm* function only. The algorithm is described below

1. Construct the eigenvalue decomposition of the covariance matrix

$$\boldsymbol{\Sigma} = \mathbf{V}\text{diag}\left(\lambda_1, \ldots, \lambda_p\right)\mathbf{V}^T.$$

2. Take the square root of the covariance matrix $\boldsymbol{\Sigma}^{1/2} = \mathbf{V}\text{diag}\left(\lambda_1^{1/2}, \ldots, \lambda_p^{1/2}\right)\mathbf{V}^T$.
3. Generate $n \times p$ values from a standard normal distribution $N\left(0, 1\right)$.
4. Put the generated values in a matrix with n rows and p columns randomly. We will call this matrix $\mathbf{X}$.
5. Construct $\mathbf{Y} = \mathbf{X}\boldsymbol{\Sigma}^{1/2} + \boldsymbol{\mu}$.

The columns in the $\mathbf{Y}$ matrix follow the multivariate normal with the specified parameters. Bear in mind that the covariance matrix needs to be of full rank. The function is faster than the *mvrnorm* funciton in the MASS library.

```
rmvnorm <- function(n, mu, sigma) {
   p <- length(mu)
   x <- matrix(RcppZiggurat::zrnorm(n * p), ncol = p)
   x %*% chol(sigma) + rep(mu, rep(n, p) )
}
```

5.2.2 Random values generation of covariance matrices (Wishart distribution)

I have written a simple code to generate covariance matrices based on the Wishart distribution. If $\mathbf{X}_i \ N_p\left(\mathbf{0}, \mathbf{\Sigma}\right)$, then $\mathbf{A} = \sum_{i=1}^{\nu} \mathbf{X}_i \mathbf{X}_i^T$ follows a p-variate Wishart distribution with parameters ν and $\mathbf{\Sigma}$, $\mathbf{A} \sim \mathbf{W}_p\left(\mathbf{\Sigma}, \nu\right)$ (Anderson, 2003). Its density is given by

$$f\left(\mathbf{A}\right) = \frac{\left|\mathbf{A}\right|^{\frac{\nu p}{2}} e^{-\frac{\text{tr}\left(\mathbf{\Sigma}^{-1}\mathbf{A}\right)}{2}}}{2^{\frac{\nu p}{2}} \pi^{\frac{p(p-1)}{4}} \left|\mathbf{\Sigma}\right|^{\frac{\nu}{2}} \prod_{i=1}^{p} \Gamma\left(\frac{\nu+1-i}{2}\right)}$$

The algorithm to generate covariance matrices from a Wishart distribution with expected value equal to $\nu\mathbf{\Sigma}$ is

1. Generate say ν random values $\mathbf{X}_i$ from a $N_p\left(\mathbf{0}, \mathbf{\Sigma}\right)$. Note, ν must be greater than p. So, if you have more dimensions than ν, change this number.

2. Thee matrix $\mathbf{X}_i \mathbf{X}_i^T$ is a random matrix from a Wishart distribution with ν degrees of freedom.

3. Repeat the two previous steps n times to get a sample of size n.

The ν parameter is the degrees of freedom of the Wishart distribution. Suppose you have a sample of p multivariate normal vectors $\mathbf{X}$ of size nu and calculate the $\mathbf{X}^T\mathbf{X}$. The degrees of freedom of the Wishart distribution is nu. I found out that there is a similar built-in function in R the *rWishart(n, df, Sigma)*. There is also a function in the package bayesm written by Rossi (2015) for those who are more interested in these.

```
rwishart <- function(n, Sigma, dof) {
  ## n is the number of matrices you want
  ## Sigma is the mean of the Wishart distribution
  ## df is the degrees of freedom of the Wishart
  ## df must be an integer
  if ( dof - round(dof) != 0 ) {
    sim <- cat("dof is not an integer", "\n")
  } else {
    p <- dim(Sigma)[2]  ## dimension of Sigma
    mu <- numeric(p)
    sim <- array(dim = c(p, p, n))
    for (i in 1:n) {
      x <- rmvnorm(dof, mu, Sigma)  ## generate multivariate normal values
      sim[, , i] <- crossprod(x) ## Wishart values with nu degrees of freedom
    }
```

```
  }
  sim
}
```

5.2.3 Random values generation from a multivariate t distribution

There is a command available through the *mvtnorm* package for generating from a multivariate t distribution with some given parameters. We also provide a function for doing that.

The basic relationship one needs to generate values from a multivariate t distribution with parameters $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ and ν is the following

$$\mathbf{Y} = \boldsymbol{\mu} + \sqrt{\frac{\nu}{\chi^2_\nu}}\boldsymbol{\Sigma}^{1/2}\mathbf{Z},$$

where $\mathbf{Z} \sim \mathbf{N_p}\left(\mathbf{0}, \mathbf{I_p}\right)$. So, basically, the algorithm is the same as in the multivariate normal distribution. The difference is the extra parameter ν.

```
rmvt <- function(n, mu, sigma, v) {
  ## n is the sample size
  ## mu is the mean vector
  ## sigma is the covariance matrix
  ## sigma does not have to be of full rank
  ## v is the degrees of freedom
  p <- length(mu)
  x <- matrix( RcppZiggurat::zrnorm(n * p), ncol = p )
  w <- sqrt( v / rchisq(n, v) )
  w * x %*% chol(sigma) + rep(mu, rep(n, p) )
}
```

5.2.4 Random values generation from a multivariate Laplace distribution

We will now provide the code to generate random values from the multivariate Laplace distribution whose density is given in (ref5.2). The basic equation is (Eltoft et al., 2006)

$$\mathbf{Y} = \boldsymbol{\mu} + \sqrt{W}\boldsymbol{\Gamma}^{1/2}\mathbf{X},$$

where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\Gamma}$ is a covariance type matrix whose determinant is 1. $\mathbf{X}$ is a multivariate normal distribution with zero mean vector and a covariance matrix equal to the identity matrix. Finally, W is a univariate exponential distribution. So basically we need a multivariate normal, a mean vector and a covariance matrix and a univariate exponential distribution to generate values from the multivariate Laplace distribution (5.2).

```
rmvlaplace <- function(n, lam, mu, G) {
  ## n is the sample size
  ## lam is the parameter of the exponential distribution
  ## mu is the mean vector
  ## G is a d x d covariance matrix with determinant 1
  if ( summary( det(G) )[1] == 1 ) {
    y <- paste("The determinant of the covariance matrix is not 1.")
  } else {
    d <- length(mu)  ## dimensionality of the data
    z <- rexp(n, lam)
    x <- matrix( RcppZiggurat::zrnorm(n * d), ncol = d )
    y <- sqrt(z) * x %*% chol(G) + rep(mu, rep(n, d) )## the simulated sample
  }
  y
}
```

5.2.5 Random values generation from a Dirichlet or an inverted Dirichlet distribution

Simulation of a random value from this distribution in p dimensions is straightforward. All we need is $p+1$ independent and equally scaled Gamma distributed random values. If $y_i \sim \text{Gamma}(\alpha_i, 1)$ and

$$x_i = \frac{y_i}{y_{p+1}},$$

then $\mathbf{x} = (x_1, \ldots, x_p)$ follows the inverted Dirichlet distribution with parameters $(\alpha_1, \ldots, \alpha_{p+1})$

If on the other hand we do this $x_i = \frac{y_i}{\sum_{j=1}^{p+1} y_j}$, then the $\mathbf{x} = (x_1, \ldots, x_{p+1})$ follows the Dirichlet distribution with parameters $(\alpha_1, \ldots, \alpha_{p+1})$.

```
rdir <- function(n, a, inv = FALSE) {
  ##  n is the sample size
  ##  a is the parameters vector
  ## if inv == FALSE, Dirichelt values will be generated
  ## and inverted Dirichelt otherwise
  D <- length(a)
  d <- D - 1  ## space of the inverted Dirichlet
  y1 <- matrix(rgamma(n * D, a, 1), ncol = D, byrow = TRUE)
  if ( inv ) {
    y <- y1[, 1:d] / y1[, D]  ## inverted Dirichlet simulated values
  } else  y <- y1 / Rfast::rowsums(y1)  ## Dirichlet simulated values
  y
```

```
}
```

5.3 Contour plots

5.3.1 Contour plot of the bivariate normal, t and skew-normal distributions

We will provide a function to obtain the parameters of the fitted distribution, plot the bivariate data and then add contour lines on the same plot. For the t distribution we require the MASS library and the function we presented before to calculate its associated parameters.

The idea is to take a grid of points along the two axis and for each point to calculate the value of the fitted density. Then, use the ready built-in function in R *contour* and that's it.

The skew-normal distribution

An alternative distribution which can also be used to model compositional data is the multivariate skew-normal distribution (Azzalini and Valle, 1996). The density of the skew-normal distribution is (Azzalini, 2005)

$$f_d(\mathbf{y}) = \frac{2}{\left|2\pi\boldsymbol{\Omega}^{1/2}\right|} e^{-\frac{1}{2}(\mathbf{y}-\boldsymbol{\xi})\boldsymbol{\Omega}^{-1}(\mathbf{y}-\boldsymbol{\xi})^T} \Phi\left[\boldsymbol{\alpha}^T\boldsymbol{\omega}^{-1}(\mathbf{y}-\boldsymbol{\xi})\right], \tag{5.9}$$

where $\Phi(\cdot)$ is the cumulative distribution of the standard normal distribution, $\boldsymbol{\omega}$ is the diagonal matrix containing the square root of diag $(\boldsymbol{\Omega})$ and α is the shape parameter ($\alpha \in \mathbb{R}^d$).

If $\alpha = \mathbf{0}$, then we end up with the multivariate normal distribution. The parameter δ_i is related to the i-th skewness coefficient as well. The skew normal can only model low skewness since the skewness coefficient cannot exceed the value 0.99527 in absolute value. Thus, for the numerical maximization of the log-likelihood of (5.9), good initial values for the vector δ are the skewness coefficients. If any of the coefficient exceeds the cut-off value 0.99527, in either direction, the initial starting value is set equal to this value.

The expected value and variance matrix of the skew-normal distribution are expressed as follows

$$E(\mathbf{y}) = \boldsymbol{\xi} + (2/\pi)^{1/2}\boldsymbol{\delta} \text{ and } Var(\mathbf{y}) = \boldsymbol{\Omega} - \boldsymbol{\omega}\boldsymbol{\mu}_z\boldsymbol{\mu}_z^T\boldsymbol{\omega},$$

where $\boldsymbol{\mu}_z = \sqrt{2/\pi}\boldsymbol{\delta}$, $\boldsymbol{\delta} = \left(1 + \boldsymbol{\alpha}^T\bar{\boldsymbol{\Omega}}\boldsymbol{\alpha}\right)^{-1/2}\bar{\boldsymbol{\Omega}}\boldsymbol{\alpha}$ and $\bar{\boldsymbol{\Omega}} = \boldsymbol{\omega}^{-1}\boldsymbol{\Omega}\boldsymbol{\omega}^{-1}$ is the correlation coefficient associated with $\boldsymbol{\Omega}$.

Azzalini (2011) has created an R package, called sn which fits the skew-normal distribution and this is what we use here.

```
den.contours <- function(x, type = 'normal') {
  ## x is a bivariate dataset
  ## type can be either 'normal', 't' or 'skewnorm'
```

```
## the user must make sure he/she has bivariate data. If the data are
## not bivariate the function will not work
## the default distribution in normal, but there are other options, such as
## t and skew normal
m <- Rfast::colmeans(x) ## mean vector
s <- cov(x) ## covariance matrix
n1 <- 100
n2 <- 100
x1 <- seq( min(x[, 1]) - 0.5, max(x[, 1]) + 0.5, length = n1 )
x2 <- seq( min(x[, 2]) - 0.5, max(x[, 2]) + 0.5, length = n1 )
## if for example the y axis is longer than the x axis, then you might
## want to change n2

if (type == 'normal') {
  r <- cor(x[, 1], x[, 2])
  con <-  - log(2 * pi) - log( det(s) )  ## constant part
  z1 <- ( x1 - m[1] ) / sqrt( s[1, 1] )
  z2 <- ( x2 - m[2] ) / sqrt( s[2, 2] )
  mat1 <- outer(z1^2, rep(1, n1), "*")
  mat2 <- outer(rep(1, n2), z2^2,  "*")
  mat3 <- tcrossprod(z1, z2)
  mat <- con - 0.5 /(1 - r^2) * (mat1 + mat2 - 2 * r * mat3)
  mat <- exp(mat)
  ind <- ( mat < Inf )
  ind[ ind == FALSE ] <- NA
  mat <- mat * ind
  ## we did this to avoid any issues with high numbers
  contour(x1, x2, mat, nlevels = 10, col = 2, xlab = colnames(x)[1],
  ylab = colnames(x)[2])
  points(x[, 1], x[, 2])
  points(m[1], m[2], pch = 10, col = 2, cex = 1.5)
  param = list(mesos = m, covariance = s)

} else  if (type == 't') {
  ## we will use the previous function 'multivt' to
  ## estimate the parameters of the bivariate t first
  f <- multivt(x)
  m <- f$center
  s <- f$covariance
```

```
  v <- f$df
  con <- lgamma( (v + 2) / 2 ) - lgamma(v / 2) - 0.5 * log( det(pi * v * s) )
  z1 <- ( x1 - m[1] ) / sqrt(s[1, 1])
  z2 <- ( x2 - m[2] ) / sqrt(s[2, 2])
  mat1 <- outer(z1^2, rep(1, n1), "*")
  mat2 <- outer(rep(1, n2), z2^2,  "*")
  mat3 <- tcrossprod(z1, z2)
  mat <- con - 0.5 * (v + 2) * log1p( 1 /(1 - r^2) *
  (mat1 + mat2 - 2 * r * mat3) / v )
  mat <- exp(mat)
  ind <- ( mat < Inf )
  ind[ind == FALSE] <- NA
  mat <- mat * ind
  ## we did this to avoid any issues with high numbers
  contour(x1, x2, mat, nlevels = 10, col = 2, xlab = colnames(x)[1],
  ylab = colnames(x)[2])
  points(x[, 1], x[, 2])
  points(m[1], m[2], pch = 10, col = 2, cex = 1.5)
  param = list(center = m, scatter = s, df = v)

} else if (type == 'skewnorm') {
  mat <- matrix(nrow = n1, ncol = n2)
  require(sn, quiet = TRUE, warn.conflicts = FALSE)
  para <- msn.mle(y = x)$dp
  for (i in 1:n1) {
    for (j in 1:n2) {
      can <- dmsn( c(x1[i], x2[j] ), dp = para)
      if (abs(can)<Inf)  mat[i,j] = can
    }
  }

  contour(x1, x2, mat, nlevels = 10, col = 2, xlab = colnames(x)[1],
  ylab = colnames(x)[2])
  points(x[, 1], x[, 2])
  para <- dp2cp(para, "SN") ## change the parameters to mean,
  ## covariance and gamma
  m <- para$beta
  points(m[1], m[2], pch = 10, col = 2, cex = 1.5)
  param <- list(mesos = para$beta, covariance = para$var.cov,
```

```
    gamma = para$gamma1)
  }

  param

}
```

5.3.2 Contour plot of a bivariate log-normal distribution

The task is the same as before, the difference now is that the fitted distribution is the bivariate log-normal.

```
lnorm.contours <- function(x) {
  ## x is a bivariate dataset
  x <- as.matrix(x)  ## dimensionality of the data is 2
  ## the user must make sure he/she has bivariate data.
  ## If the data are not bivariate the function will not work
  n1 <- 100
  n2 <- 100  ## n1 and n2 specify the number of points taken at each axis
  y <- log(x)
  m <- Rfast::colmeans(y)  ## fitted mean vector of the logged data
  s <- var(y) ## estimated covariance matrix of the logged data
  r <- cor(y[, 1], y[, 2])
  con <-  -log(2 * pi) - 0.5 * log(det(s))  ## contant part
  x1 <- seq( max(min(x[, 1]) - 0.5, 0.01), max(x[, 1]) + 0.5, length = n1 )
  x2 <- seq( max(min(x[, 2]) - 0.5, 0.01), max(x[, 2]) + 0.5, length = n2 )
  z1 <- ( log(x1) - mean(y[, 1]) ) / sd(y[, 1])
  z2 <- ( log(x2) - mean(y[, 2]) ) / sd(y[, 2])
  xat1 <- matrix( rep(x1, n2), ncol = n2 )
  xat2 <- matrix( rep(x2, n1), ncol = n2, byrow = TRUE )
  mat3 <- tcrossprod(z1,z2)
  mat <- con - xat1 - xat2 - 0.5 /(1 - r^2) * (mat1 + mat2 - 2 * r * mat3)
  mat <- exp(mat)
  contour(x1, x2, mat, nlevels = 10, col = 2, xlab = "x1", ylab = "x2")
  points(x[, 1], x[, 2])
  mvlnorm.mle(x)
}
```

5.3.3 Contour plot of a bivariate inverted Dirichlet distribution

The next function shows how to the contour plots of an inverted Dirichlet distribution look like.

```
invdir.contours <- function(x) {
  ## x is a bivariate dataset
  x <- as.matrix(x)  ## dimensionality of the data is 2
  ## the user must make sure he/she has bivariate data.
  ## If the data are not bivariate the function will not work

  n1 <- 100
  n2 <- 100  ## n1 and n2 specify the number of points taken at each axis
  da <- invdir.est(x)
  a <- da$alpha
  x1 <- seq(max(min(x[, 1]) - 1, 0.01), max(x[, 1]) + 1, length = n1)
  x2 <- seq(max(min(x[, 2]) - 1, 0.01), max(x[, 2]) + 1, length = n2)
  mat <- matrix(NA, nrow = n1, ncol = n2)
  con <- lgamma( sum(a) ) - sum( lgamma(a) )
  suma <- sum(a)
  ra <- a[1:2] - 1
  for (i in 1:n1) {
    for (j in 1:n2) {
      z <- c(x1[i], x2[j])
      f <- con + log(z) %*% ra - suma * log(1 + sum(z))
      if (exp(f) < Inf)  mat[i, j] <- exp(f)
    }
  }
  contour(x1, x2, mat, nlevels = 10, col = 2, xlab = "x1", ylab = "x2")
  points(x[, 1], x[, 2])
  da
}
```

5.3.4 Contour plot of a kernel density estimate

The idea is the same as before, take a grid of points and for each point calculate its kernel density estimate.

```
kern.contours <- function(x, h, thumb = FALSE) {
  ## x is a bivariate dataset
  ## h is the h you want, which is either a 2x2 diagonal matrix or a scalar
```

```
  ## thumb can be either FALSE so the specified h is used or TRUE, so
  ## that the Scott (or Silverman) rule is used.
  n <- dim(x)[1]  ## sample size
  ## the user must make sure he/she has bivariate data.
  ## If the data are not bivariate the function will not work
  if ( thumb ) {
    s <- Rfast::colVars(x, std = TRUE)
    h <- diag(s * n^(-1/6))
  }  else if ( !is.matrix(h) ) {
    h <- diag(rep(h, 2))
  } else h <- h

  ha <- solve(h^2)
  con <- prod( diag(h) )
  n1 <- 100  ## n1 and n2 specify the number of points taken at each axis
  x1 <- seq( min(x[, 1]) - 1, max(x[, 1]) + 1, length = n1 )
  x2 <- seq( min(x[, 2]) - 1, max(x[, 2]) + 1, length = n1 )
  mat <- matrix(NA, nrow = n1, ncol = n1)

  for (i in 1:n1) {
    for (j in 1:n1) {
      a <- as.vector( mahala(x, c( x1[i], x2[j] ), ha, inverted = TRUE ) )
      can <- 1/(2 * pi) * ( 1/con ) * sum( exp(-0.5 * a) )/n
      if ( abs(can) < Inf )  mat[i, j] <- can
    }
  }

  contour(x1, x2, mat, nlevels = 10, col = 2,
  xlab = colnames(x)[1], ylab = colnames(x)[2])
  points(x[, 1], x[, 2])
}
```

5.3.5 Contour plot of the bivariate Poisson distribution

The task is the same as before. It is not very usual to do contour plots for discrete distributions, but I did it so that anyone can see how they look like. The function needs the bivariate data and the estimates of the three parameters. The R function is given below.

```
bp.contour <- function(x1, x2, lambda) {
  ## x1 and x2 are the two variables
```

```
  ## lambda contains the three values of the three parameters
  lam1 <- lambda[1]
  lam2 <- lambda[2]
  lam3 <- lambda[3]
  z1 <- seq(max(min(x1) - 3, 0), max(x1) + 3)
  n1 <- length(z1)
  z2 <- seq(max(min(x2) - 3, 0), max(x2) + 3)
  n2 <- length(z2)
  mat <- matrix(nrow = n1, ncol = n2)
  l1 <- log(lam1)
  l2 <- log(lam2)
  ls <-  -(lam1 + lam2 + lam3)
  rho <- lam3/(lam1 * lam2)

  for (i in 1:n1) {
    for (j in 1:n2) {
      f1 <- ls + z1[i] * l1 - lgamma(z1[i] + 1) + z2[j] * l2 - lgamma(z2[j] + 1)
      k <- 0:min(z1[i], z2[j])
      f2 <- log( sum( choose(z1[i], k) * choose(z2[j], k) * factorial(k) * rho^k ) )
      f <- f1 + f2
      mat[i, j] <- exp(f)
    }
  }

  contour(z1, z2, mat, nlevels = 10, col = 2, xlab = "x1", ylab = "x2")
  points(x1, x2)
}
```

6 Covariance, principal component analysis and singular value decomposition

6.1 Fast covariance and correlation matrices

I found this information in stackoverflow, written by Shabalin (Shabalin, 2012) and I advertise it here. There are two functions, one for covariance and one for the correlation matrix. Use big sample sizes and or many variables to see differences of up to 50% when compared with R's standard *cov* and *cor* functions, when you have more than 1,000 dimensions. Try it with 5,000 variables to see. If you have small matrices, these functions might take a bit longer, but the difference is almost negligible.

```
cova_old <- function(x) {
  ## x must be a matrix
  n <- dim(x)[1]  ## sample size
  mat <- t(x) - Rfast::colmeans(x)
  tcrossprod( mat ) / (n - 1)
}
```

The next function is the one we have in **Rfast** (Papadakis et al., 2019).

```
cova <- function (x) {
  n <- dim(x)[1]
  m <- sqrt(n) * colmeans(x)
  (crossprod(x) - tcrossprod(m))/(n - 1)
}


cora <- function(x) {
  ## x must be a matrix
  mat <- t(x) - Rfast::colmeans(x)
  mat <- mat / sqrt( Rfast::rowsums(mat^2) )
  tcrossprod( mat )
}
```

6.2 Fast Mahalanobis distance

The function below works only for matrices. I took R's built-in *mahalanobis* function and modified it a bit. You need to define x as a matrix. You also need to supply a mean vector and a covariance matrix and also state whether the covariance matrix is already inverted or not. Almost like the *mahalanobis* function, but it is 2 or more times faster.

```
mahala <- function (x, m, s, inverted = FALSE) {
  ## x must be a matrix
  ## m is the mean vector and
  ## s is the covariance matrix
  ## if s is the inverse of the covariance matrix
  ## put inverted = TRUE
  y <- t(x) - m
  if ( !inverted ) {
    di <-  Rfast::colsums( y * crossprod( chol2inv( chol(s) ), y ) )
  }  else di <- Rfast::colsums(y * crossprod(s, y))
  di
}
```

6.3 Fast column-wise variances or standard deviations

Below I have a vectorised function to calculate column-wise variances or standard deviations from a matrix. To save time I do not put *x ¡- as.matrix(x)* as this functions is to be used internally inside some other function you will already have and *x* should be a matrix anyway. I found this in stackoverflow and was created by David Arenburg so the credits should go to him.

```
colVars <- function(x, suma = NULL, std = FALSE) {
  ##  x is a matrix
  ##  if you want standard deviations set std = TRUE
  if ( !is.null(suma) ) {
    m <- suma
  } else  m <- Rfast::colsums(x)
  n <- dim(x)[1]
  x2 <- Rfast::colsums(x^2)
  s <- ( x2 - m^2/n ) / (n - 1)
  if ( std )  s <- sqrt(s)
  s
}
```

6.4 Multivariate standardization

This is probably the transformation to which the term suits better. This function transforms the data such that they have zero mean vector and the identity as the covariance matrix. We used this function to perform hypothesis testing for zero correlation using bootstrap but did

not pay too much attention. At first we have to subtract the mean vector from the data and then multiply by the square root of the inverse of the covariance matrix

$$\mathbf{Z} = (\mathbf{X} - \boldsymbol{\mu})\,\boldsymbol{\Sigma}^{-1/2}.$$

The key thing is to decompose the covariance matrix, using Cholesky or eigen decomposition. We prefer the latter for simplicity and convenience. The spectral decomposition of the covariance matrix (or any positive definite square matrix in general) is

$$\boldsymbol{\Sigma} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T = \mathbf{V}\text{diag}\left(\lambda_1, \ldots, \lambda_p\right)\mathbf{V}^T,$$

where $\mathbf{V}$ is the matrix containing the eigenvectors, an orthogonal matrix and $\lambda_1, \ldots, \lambda_p$ are the p eigenvalues (the number of dimensions), where $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p > 0$. The square root, the inverse of $\boldsymbol{\Sigma}$ and its inverse square root can be written as

$$\begin{aligned}
\boldsymbol{\Sigma}^{1/2} &= \mathbf{V}\text{diag}\left(\lambda_1^{1/2}, \ldots, \lambda_p^{1/2}\right)\mathbf{V}^T \\
\boldsymbol{\Sigma}^{-1} &= \mathbf{V}\text{diag}\left(\lambda_1^{-1}, \ldots, \lambda_p^{-1}\right)\mathbf{V}^T \text{ and} \\
\boldsymbol{\Sigma}^{-1/2} &= \mathbf{V}\text{diag}\left(\lambda_1^{-1/2}, \ldots, \lambda_p^{-1/2}\right)\mathbf{V}^T
\end{aligned}$$

respectively.

Actually any power of a positive definite matrix can be calculated this way. If the covariance matrix is not of full rank (equal to p), that is if there is at least one eigenvalue equal to zero, it becomes clear why the inverse does not exist. Another thing to highlight is that the number of non zero eigenvalues is equal to the rank of the matrix (or vice versa). The following function performs this transformation using eigen decomposition of the covariance matrix.

Alternatively another standardization is simply to center the variables (subtract from each variable each mean) and then divide by its standard deviation $z_i = \frac{x_i - m_i}{s_i}$, for $i = 1, ..., p$. A similar, but robust, way is to use the median and the median absolute deviation instead.

Note that the built in command in R *scale* is used to center the data and make their standard deviations equal to 1. See its help for the options it offers.

```
rizamat <- function(s) {
  ## s must have positive eigenvalues
  eig <- eigen(s)
  lam <- eig$values
  vec <- eig$vectors
  vec %*% ( t(vec) * sqrt(lam ) )
}

multivstand <- function(x, type = "matrix") {
```

```
  ## x is the data
  ## type is either 'matrix', 'mean' or 'median'
  if (type == "matrix") {
     s <- cov(x)  ## covariance matrix
     B <- solve( chol(s) )
     m <- Rfast::colmeans(x)
     y <- t(x) - m
     z <- crossprod(y, B)
  }  ## multivariate standardization
  if (type == "mean") {
     m <- Rfast::colmeans(x)
     s <- Rfast::colVars(x, std = TRUE)
     z <- t( ( t(x) - m ) / s )
  }
  if (type == "median") {
    m <- Rfast::colMedians(x)
    y <- t( t(x) - m )
    s <- Rfast::colMedians( abs(y) ) / qnorm(3/4)
    z <- t( t(y) / s )
  }

  z
}
```

6.5 Choosing the number of principal components using SVD

SVD stands for Singular Value Decomposition of a rectangular matrix. That is any matrix, not only a square one in contrast to the Spectral Decomposition with eigenvalues and eigenvectors, produced by principal component analysis (PCA). Suppose we have a $n \times p$ matrix $\mathbf{X}$. Then using SVD we can write the matrix as

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T, \tag{6.1}$$

where $\mathbf{U}$ is an orthonormal matrix containing the eigenvectors of $\mathbf{X}\mathbf{X}^T$, the $\mathbf{V}$ is an orthonormal matrix containing the eigenvectors of $\mathbf{X}^T\mathbf{X}$ and D is an $p \times p$ diagonal matrix containing the r non zero singular values $d_1, \ldots, d_r$ (square root of the eigenvalues) of $\mathbf{X}\mathbf{X}^T$ (or $\mathbf{X}^T\mathbf{X}$) and the remaining $p - r$ elements of the diagonal are zeros. We remind that the maximum rank

of an $n \times p$ matrix is equal to $\min\{n, p\}$. Using (6.1), each column of $\mathbf{X}$ can be written as

$$\mathbf{x}_j = \sum_{k=1}^{r} \mathbf{u}_k d_k \mathbf{v}_{jk}.$$

This means that we can reconstruct the matrix $\mathbf{X}$ using less columns (if $n > p$) than it has.

$$\tilde{\mathbf{x}}_j^m = \sum_{k=1}^{m} \mathbf{u}_k d_k \mathbf{v}_{jk}, \text{ where } m < r.$$

The reconstructed matrix will have some discrepancy of course, but it is the level of discrepancy we are interested in. If we center the matrix $\mathbf{X}$, subtract the column means from every column, and perform the SVD again, we will see that the orthonormal matrix $\mathbf{V}$ contains the eigenvectors of the covariance matrix of the original, the un-centred, matrix $\mathbf{X}$.

Coming back to the a matrix of n observations and p variables, the question was how many principal components to retain. We will give an answer to this using SVD to reconstruct the matrix. We describe the steps of this algorithm below.

1. Center the matrix by subtracting from each variable its mean $\mathbf{Y} = \mathbf{X} - \mathbf{m}$
2. Perform SVD on the centred matrix $\mathbf{Y}$.
3. Choose a number from 1 to r (the rank of the matrix) and reconstruct the matrix using (6.1). Let us denote by $\widetilde{\mathbf{Y}}^m$ the reconstructed matrix.
4. Calculate the sum of squared differences between the reconstructed and the original values

$$PRESS\,(m) = \sum_{i=1}^{n} \sum_{j=1}^{p} \left(\tilde{y}_{ij}^m - y_{ij} \right)^2, m = 1, .., r.$$

5. Plot $PRESS\,(m)$ for all the values of m and choose graphically the number of principal components.

The graphical way of choosing the number of principal components is not the best and there alternative ways of making a decision (see for example **?**). The code in R is given below.

```
choose.pc <- function(x) {
  ## x contains the data
  n <- dim(x)[1]
  runtime <- proc.time()
  x <- Rfast::standardise(x, center = TRUE, scale = FALSE)  ## center the matrix
```

```
  A <- svd(x)  ## SVD of the centred matrix
  u <- A$u
  d <- A$d
  v <- t( A$v )
  p <- length(d)
  press <- numeric(p)

  y <- 0
  for (i in 1:p) {
    y <- y + u[, i, drop = FALSE] %*% ( d[i] * v[i, , drop = FALSE] )
    press[i] <- sqrt( sum( (y - x)^2 ) )  ## calculation of the PRESS
  }
  runtime <- proc.time() - runtime

  plot(press, type = "b", pch = 9, xlab = "Number of components",
  ylab = "Reconstruction error")
  val <-  d^2 / (n - 1)
  prop <- cumsum(val) / sum(val)
  diffa <- diff( c(prop, 1) )

  dev.new()
  plot(val, type = "b", pch = 9, xlab = "Number of components",
  ylab = "Eigenvalues")
  dev.new()

  plot( prop, type = "b", pch = 9, xlab = "Number of components",
  ylab = "Cumulative eigenvalue proportions" )
  dev.new()

  plot(diffa , type = "b", pch = 9, xlab = "Number
  of components", ylab = "Cumulative proportions differences")
  list(values = val, proportion = prop, differences = diffa, press = press,
  runtime = runtime)
}
```

6.6 Choosing the number of principal components using probabilistic PCA

Probabilistic PCA assumes that the principal components come from a multivariate normal distribution. In fact one can decompose the covariance matrix as $\mathbf{\Sigma} = \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}_p$, where $\mathbf{I}_p$ is the identity matrix in p dimensions. The maximum likelihood solution is given by Tipping and Bishop (1999)

$$\mathbf{W}_{ML} = \mathbf{U}_q \left(\mathbf{\Lambda}_q - \sigma_{ML}^2 \mathbf{I}_q\right)^{1/2} \text{ and } \sigma_{ML}^2 = \frac{1}{p-q}\sum_{j=q+1}^{p} \lambda_j,$$

where λ_j are the eigenvalues of the covariance matrix, $\mathbf{\Lambda}_q$ is a diagonal matrix with these eigenvalues and $\mathbf{U}_q$ is the matrix containing the corresponding eigenvectors.

At first one performs eigen analysis on the covariance matrix and extracts the eigenvalues λ_j and the matrix with the eigenvectors $\mathbf{U}$. Then, estimate the variance "lost" over the projection σ_{ML}^2 and finally the matrix $\mathbf{W}_{ML}$.

This is the closed form solution, but also an EM based solution can be found for the case where there are missing data (Tipping and Bishop, 1999) and expanded to the multivariate t distribution by Zhao and Jiang (2006). The matrix $\mathbf{W}_{ML}$ does not contain unit vectors, but if normalised, they are the same as the eigenvectors. Notealso that the covariance matrix using eigen analysis (or spectral decomposition) is written as $\mathbf{\Sigma} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^T$. What is offered though by this approach is an automated method for selection of principal components (Minka, 2001).

The first log-likelihood is the BIC approximation given by

$$BIC = -\frac{n}{2}\sum_{j=1}^{q}\log\lambda_j - \frac{n\left(p-q\right)}{2}\log\sigma_{ML}^2 - \frac{m+q}{2}\log n,$$

where n is the sample size and $m = p(p-1)/2 - (p-q)(p-q-1)/2$. The second log-likelihood is a modification of the log-likelihood (Minka, 2001) as proposed by Rajan and Rayner (1997)

$$RR = -\frac{np}{2}\log\left(2\pi\right) - \frac{np}{2}\sum_{j=1}^{q}\log\frac{\lambda_j}{q} - \frac{n\left(p-q\right)}{2}\log\sigma_{ML}^2 - \frac{np}{2}.$$

```
ppca.choose <- function(x) {
  ## x is matrix with the data

  runtime <- proc.time()
  p <- dim(x)[2]  ## number of variables
  n <- dim(x)[1]  ## sample size
```

```
  eig <- prcomp(x)
  lam <- eig$sd^2
  vec <- eig$rotation
  sigma <- cumsum( sort(lam) ) / (1:p)
  sigma <- sort(sigma, decreasing = TRUE)[-1]
  lsigma <- log(sigma)
  #for (i in 1:p) {
  # H <- vec[, 1:i] %*% ( lam[1:i] * diag(i) - sigma[i] * diag(i) )
  #}
  m <- p * (p - 1) / 2 - ( p - 1:c(p - 1) ) * ( p - 1:c(p - 1) - 1 ) / 2
  bic <-  - n / 2 * cumsum( log(lam)[ 1:c(p - 1) ] ) -
  n * ( p - 1:c(p - 1) ) / 2 * lsigma - ( m + 1:c(p - 1) ) / 2 * log(n)
  rr <-  -n * p / 2 * log(2 * pi) - n * p / 2 - n * ( 1:c(p - 1) ) / 2 *
  log( mean( lam[ 1:c(p - 1) ] ) ) - n * ( p - 1:c(p - 1) ) / 2 * lsigma
  runtime <- proc.time() - runtime

  plot(1:c(p - 1), bic, type = "b", pch = 9, xlab = "Number of components",
  ylab = "BIC")
  dev.new()
  plot(1:c(p - 1), rr, type = "b", pch = 9, xlab = "Number of components",
  ylab = "RR log-likelihood")
  names(bic) <- names(rr) <- paste("PC", 1:c(p - 1), sep = " ")
  list(bic = bic, rr = rr, runtime = runtime)
}
```

The function to perform probabilistic PCA is given below.

```
ppca <- function(x, k) {
  ## x is matrix with the data
  ## k is an integer between 1 and the number of columns of x
  ## k denotes the number of PCs to be returned
  p <- dim(x)[2] ## number of variables
  n <- dim(x)[1] ## sample size
  eig <- prcomp(x)
  lam <- eig$sd^2
  vec <- eig$rotation
  sigmaml <- sum( lam[ -c(1:k) ] ) / (p - k)
  wml <- t(vec[, 1:k]) *  sqrt( lam[1:k] - sigmaml )
  sigma <- c( sum(lam), sigmaml, sigmaml/sum(lam) )
  names(sigma) <- c("trace", "sigmaml", "prop.lost")
```

```
  list(sigma = sigma, wml = t(wml) )
}
```

6.7 Confidence interval for the percentage of variance retained by the first κ components

The algorithm is taken by Mardia et al., 1979, pg. 233-234. The percentage retained by the fist κ principal components denoted by $\hat{\psi}$ is equal to

$$\hat{\psi} = \frac{\sum_{i=1}^{\kappa} \hat{\lambda}_i}{\sum_{j=1}^{p} \hat{\lambda}_j}$$

ψ is asymptotically normal with mean ψ and variance

$$\begin{aligned} \tau^2 &= \frac{2}{(n-1)\left(tr\boldsymbol{\Sigma}\right)^2}\left[(1-\psi)^2\left(\lambda_1^2+...+\lambda_k^2\right)+\psi^2\left(\lambda_{\kappa+1}^2+...\lambda_p^2\right)\right] \\ &= \frac{2tr\boldsymbol{\Sigma}^2}{(n-1)\left(tr\boldsymbol{\Sigma}\right)^2}\left(\psi^2-2\alpha\psi+\alpha\right), \end{aligned}$$

where

$$\alpha = \left(\lambda_1^2+...+\lambda_k^2\right) / \left(\lambda_1^2+...+\lambda_p^2\right) \quad \text{and} \quad \text{tr}\boldsymbol{\Sigma}^2 = \lambda_1^2+...+\lambda_p^2$$

The bootstrap version provides an estimate of the bias, defined as $\hat{\psi}_{boot} - \hat{\psi}$ and confidence intervals calculated via the percentile method and via the standard (or normal) method (Efron and Tibshirani, 1993). The code below gives the option to perform bootstrap or not by making the (B) equal to or greater than 1.

```
lamconf <- function(x, k, a = 0.05, B = 1000) {
  ## x contains the data
  ## k is the number of principal components to keep
  ## a denotes the lower quantile of the standard normal distribution
  ## thus 0.95 confidence intervals are constructed
  ## R is the number of bootstrap replicates
  n <- dim(x)[1]
  p <- dim(x)[2]
  lam <- prcomp(x)$sd^2  ## eigenvalues of the covariance matrix
  psi <- sum(lam[1:k]) / sum(lam)  ## the percentage retained by the
  ## first k components
  if (B == 1) {
    trasu <- sum(lam)
```

```
    trasu2 <- sum(lam^2)
    alpha <- sum( (lam^2)[1:k] ) / trasu2
    t2 <- ( 2 * trasu2 * (psi^2 - 2 * alpha * psi + alpha) )/
    ( (n - 1) * (trasu^2) )
    ci <- c(psi - qnorm(1 - a/2) * sqrt(t2), psi +
    qnorm(1 - a/2) * sqrt(t2))
    result <- c(psi, ci = ci)
    names(result) <- c( 'psi', paste(c( a/2 * 100, (1 - a/2) * 100 ),
    "%", sep = "") )
  }
  if (B > 1) {
    ## bootstrap version
    tb <- numeric(B)
    for (i in 1:B) {
      b <- sample(1:n, n, replace = TRUE)
      lam <- prcomp(x[b, ])$sd^2
      tb[i] <- sum( lam[1:k] ) / sum(lam)
    }
    conf1 <- c( psi - qnorm(1 - a/2) * sd(tb), psi + qnorm(1 - a/2) * sd(tb) )
    conf2 <- quantile(tb, probs = c(a/2, 1 - a/2))
    hist(tb, xlab = "Bootstrap percentages", main = "")
    abline(v = psi, lty = 2, lwd = 2)
    abline(v = mean(tb), lty = 1, lwd = 3)
    ci <- rbind(conf1, conf2)
    legend( conf2[1], B/10, cex = 0.8, c("psi", "bootstrap psi"),
    lty = c(2, 1), lwd = c(2, 3) )
    colnames(ci) <- paste(c( a/2 * 100, (1 - a/2) * 100 ), "%", sep ="")
    rownames(ci) <- c("standard", "empirical")
    res <- c(psi, mean(tb), mean(tb) - psi )
    names(res) <- c('psi', 'psi.boot', 'est.bias')
    result <- list(res = res, ci = ci)

  }
  result
}
```

6.8 A metric for covariance matrices

A metric for covariance matrices is the title of a paper by Förstner and Moonen (2003) and this is what we will show here now. The suggested metric between two $p \times p$ non-singular covariance matrices is

$$d^2(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{p} \left[\log \lambda_i \left(\mathbf{A}\mathbf{B}^{-1}\right)\right]^2,$$

where λ_i stands for the i-th eigenvalue and note that the order of the multiplication of the matrices is not important.

```
cov.dist <- function(A, B) {
  ## A and B are two covariance matrices
  ## the order is irrelevant, A,B or B,A is the same
  S <- solve(B, A)
  sqrt( sum( log( eigen(S)$values )^2 ) )
}
```

6.9 The Helmert matrix

We can chose to put another d x D matrix in the choice of F as well. A good choice could be the Helmert sub-matrix. It is the Helmert matrix (Lancaster, 1965) with the first row deleted. This is defined as a $d \times D$ matrix with orthonormal rows that are orthogonal to $\mathbf{1}_\mathbf{D}^\mathbf{T}$, that is $HH^T = \mathbf{I_d}$ and $H\mathbf{1_D} = \mathbf{0_d}$. The $i-th$ row of the matrix is defined as $\frac{1}{\sqrt{i(i+1)}}$ until the $i-th$ column. The $(i+1)-th$ column is the negative sum of the i (first) elements of this row. The next columns of this row have zeros. Note that the Helmert sub-matrix is usually used to remove the singularity of the matrix (if the matrix has one zero eigenvalue) and it is also an isometric transformation (the distances between two row vectors is the same before and after the multiplication by the Helmert matrix).
An example of the form of the $(D-1) \times D$ Helmert sub-matrix is

$$\mathbf{H} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & \dots & \dots & 0 \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & -\frac{2}{\sqrt{6}} & 0 & \dots & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{1}{\sqrt{i(i+1)}} & \dots & \frac{1}{\sqrt{i(i+1)}} & -\frac{i}{\sqrt{i(i+1)}} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{dD}} & \dots & \dots & \dots & \dots & \frac{1}{\sqrt{dD}} & -\frac{dD}{\sqrt{dD}} \end{pmatrix} \tag{6.2}$$

The R-code for the Helmert sub-matrix is

```
helm <- function(n) {
  mat <- matrix(0, n - 1, n)
  i <- 2:n
  r <- 1 / sqrt( i * (i - 1) )
  for ( j in 1:(n - 1 ) ) {
    mat[j, 1: c(j + 1) ] <- c( rep(r[j], j), - j * r[j]  )
  }
  mat
}
```

6.10 The Moore-Penrose pseudo-inverse matrix

The Moore-Penrose pseudo-inverse matrix (Moore, 1920, Penrose, 1956) was created in the case when the square matrix is not of full rank and thus not invertible (determinant is zero). A pseudo-inverse matrix $\mathbf{A}^{-}$ satisfies the following criteria

- $\mathbf{A}\mathbf{A}^{-}\mathbf{A} = \mathbf{A}$
- $\mathbf{A}^{-}\mathbf{A}\mathbf{A}^{-} = \mathbf{A}^{-}$
- $(\mathbf{A}\mathbf{A}^{-})^{*} = \mathbf{A}\mathbf{A}^{-}$
- $(\mathbf{A}^{-}\mathbf{A})^{*} = \mathbf{A}^{-}\mathbf{A}$,

where $\mathbf{A}^{*}$ stands for the Hermitian transpose (or conjugate transpose) of $\mathbf{A}$. (Schaefer et al., 2007) have written the corpcor package which finds the Moore-Penrose pseudo-inverse matrix. There is the command *ginv* inside MASS and I am doing the same stuff. Let us now see how we calculate the pseudo-inverse matrix using singular value decomposition (SVD).

We will repeat the same stuff as in Section 6.5. Suppose we have a $p \times p$ matrix $\mathbf{X}$, which is of rank $r \leq p$. Using SVD we can write the matrix as

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{T},$$

where $\mathbf{U}$ is an orthonormal matrix containing the eigenvectors of $\mathbf{X}\mathbf{X}^{T}$, the $\mathbf{V}$ is an orthonormal matrix containing the eigenvectors of $\mathbf{X}^{T}\mathbf{X}$ and D is an $r \times r$ matrix containing the r non zero singular values $d_1, \ldots, d_r$ (square root of the eigenvalues) of $\mathbf{X}\mathbf{X}^{T}$ (or $\mathbf{X}^{T}\mathbf{X}$). Using (6.1), each column of $\mathbf{X}$ can be written as

$$\mathbf{A}_j = \sum_{k=1}^{p} \mathbf{u}_k d_k \mathbf{v}_{jk}.$$

This is exactly the key point we need, the r positive singular values and thus, the r columns and rows of the $\mathbf{U}$ and $\mathbf{V}$ respectively. The pseudo-inverse of $\mathbf{A}$ is then given by

$$\mathbf{A}^{-} = \mathbf{U}\mathbf{D}_r^{-1}\mathbf{V}^T,$$

where $\mathbf{D}_r = \text{diag}\,(u_1, \ldots, u_r, 0, \ldots, 0)$ is a diagonal matrix containing the r positive singular values and the remaining $p - r$ diagonal elements are zero.

```
pseudoinv <- function(x) {
  ## x is a not full rank covariance matrix
  ## if it is full rank, this will still work of course
  p <- dim(x)[2]  ## number of columns of the matrix
  a <- svd(x)
  i <- qr(x)$rank  ## rank of the matrix
  lam <- a$d[1:i]  ## singular or eigen values
  vec <- a$u  ## unit vectors
  su <- vec %*% ( t(vec) * c(1/lam, numeric(p - i)) )
  list(rank = i, su = su)  ## rank is the rank of the matrix
}
```

6.11 A not so useful pseudo-inverse matrix

We will give a very simple way to evaluate a pseudo-inverse matrix of a square D x D singular matrix whose rank is $n - 1$. Let $\mathbf{\Gamma}$ be such a singular matrix Aitchison, 2003, pg. 99. We need another matrix which reduces the dimensions of the matrix by one. One choice can be the following d x D $\mathbf{F}$ matrix with rank equal to d.

$$\mathbf{F} = [\mathbf{I}_d : -\mathbf{j}_d]. \tag{6.3}$$

This is simply the identity matrix with one extra column to the right with all elements equal to -1. Then the pseudo-inverse Γ^{-} is equal to:

$$\Gamma^{-} = \mathbf{F}^T \left(\mathbf{F}\mathbf{\Gamma}\mathbf{F}^T \right)^{-1} \mathbf{F}$$

```
ginv <- function(A) {
  d <- ncol(A) - 1
  ## F <- helm(ncol(A))
  F <- cbind(matrix(diag(d), ncol = d), matrix(rep(-1, d), ncol = 1))
  t(F) %*% solve( F %*% A %*% t(F), F )
}
```

You can also use the Helmert matrix of **A** in *ginv* function as the **F** matrix.

6.12 Exponential of a square matrix

R does not have a built in function for the exponential of a matrix. This can be found in the package *expm* (Goulet et al., 2013). We provide a simple formula for the case of a symmetric matrix following Moler and Van Loan (2003) using the eigenvectors and the eigenvalues of the matrix

$$e^{\mathbf{A}} = \mathbf{V}\text{diag}\left(e^{\lambda_1}, \ldots, e^{\lambda_p}\right)\mathbf{V}^{\mathbf{T}},$$

where $\mathbf{V}$ is the matrix containing the eigenvectors of the matrix $\mathbf{A}$, $\lambda_1, \ldots, \lambda_p$ are the eigenvalues of $\mathbf{A}$ and p is the rank of $\mathbf{A}$ assuming it is of full rank. A nice explanation of this can be found at Joachim Dahl' course webpage (slide No 10).

If on the other hand the matrix is not symmetric, but still square, Chang (1986) gives the formula via a Taylor series of the exponential function presented below. However, this holds true only if the matrix has real eivengectors and eigenvalues. If you have complex (or imaginary numbers) I how do not know how you deal with them.

$$e^{\mathbf{A}} = \mathbf{I_p} + \mathbf{A} + \mathbf{A^2}/\mathbf{2!} + \mathbf{A^3}/\mathbf{3} + \ldots$$

As for the power of a matrix, Mardia et al., 1979, pg. 470 provides a simple formula

$$\mathbf{A}^n = \mathbf{V}\text{diag}\left(e^n, \ldots, e^n\right)\mathbf{V}^{\mathbf{T}}.$$

The reader is addressed to Moler and Van Loan (2003) for a a review and thorough discussion on ways to calculate the matrix exponential.

```
expmat <- function(A, tol = 1e-07) {
  ## A has to be a symmetric matrix
  eig <- eigen(A)
  vec <- eig$vectors

  if ( all(t(A) - A != 0) ) {  ## non symmetric matrix
    tvec <- t(vec)
lam <- diag( eig$values )
    a1 <- diag( nrow(A) ) + A
    a2 <- a1 + 0.5 * vec %*% lam^2 %*% tvec
    i <- 2
    while ( sum( abs( a1 - a2 ) )  > tol ) {
```

```
      i <- i + 1
      a1 <- a2
      a2 <- a1 + ( vec %*% lam^i %*% tvec ) / factorial(i)
    }
    res <- list(iter = i, mat = a2)
  } else if ( all(t(A) - A == 0) ) {  ## symmetric matrix
    explam <-
    mat <- vec %*% ( t(vec) * exp( eig$values ) )
    res <- list( mat = mat )
  }

  res
}
```

7 Robust statistics

7.1 Approximate likelihood trimmed mean

Fraiman and Meloche (1999) defined a multivariate trimmed mean which he calls Approximate Likelihood trimmed mean or APL-trimmed mean. In sum, the APL-trimmed mean for a dataset $\mathbf{X}$ consisting of n observations is defined as

$$\boldsymbol{\mu}_{\gamma} = \frac{\sum_{i=1}^{n} \mathbf{x}_i \mathbf{1}\left\{\hat{f}\left(\mathbf{x}_i\right) \geq \gamma\right\}}{\sum_{i=1}^{n} \mathbf{1}\left\{\hat{f}\left(\mathbf{x}_i\right) \geq \gamma\right\}},$$

where $\mathbf{1}\{.\}$ is the indicator function and $\hat{f}\left(\mathbf{x}_i\right)$ is the kernel density estimate using (5.3). I have to note that Fraiman and Meloche (1999) used a slightly different way in the calculation of the kernel density. For the calculation of the kernel density estimate of each point, that point does not contribute to the kernel density.

The tuning parameter, γ, which determines the amount of trimming one requires, can be adjusted by solving the equation

$$\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\left\{\hat{f}\left(\mathbf{x}_i\right) \geq \gamma\right\} = 1 - \alpha,$$

where α is the percentage of observations to be used for the calculation of the APL-trimmed mean. Based on this mean, or one selected observations, one can calculate the relevant covariance matrix. The next R function offers this possibility.

```
apl.mean <- function(x, a = 0.2) {
  x <- as.matrix(x)  ## makes sure x is a matrix
  ## h is a vector containing grid values of the bandwidth
  ## a is the trimming level, by default is set to 0.2
  ## so, 80% of the data will be used
  hopt <- mkde.tune(x)$hopt
  f <- mkde(x, hopt)
  y <- x[(f > quantile(f, prob = a)), ]
  nu <- dim(y)[1]
  mu <- Rfast::colmeans(y)
  covar <- var(y)
  list(hopt = hopt, mu = mu, covar = covar)
}
```

When we mentioned the multivariate kernel density estimation we referred to Silverman (1986) who mentioned that instead of the classical covariance matrix, one can use a robust

version of it, when choosing the bandwidth parameter h.

7.2 Spatial median

The so called spatial median, is the vector $\boldsymbol{\gamma}$ which minimizes the sum $\sum_{i=1}^{n} \parallel \mathbf{y}_i - \boldsymbol{\gamma} \parallel$ (Möttönen et al., 2010), where $\parallel \cdot \parallel$ is the Euclidean norm, and it has a very long history. Gini and Galvani (1929) and Haldane (1948) have independently considered the spatial median as a generalization of the univariate median, as Möttönen et al. (2010) informs us. For more information you can see Möttönen et al. (2010) and Kärkkäinen and Äyrämö (2005) showed some methods of computing this location measure. From Kärkkäinen and Äyrämö (2005) I will take a nice iterative method he mentions and cites Kuhn (1973). The following equation is not the exact one, I put a zero probability in a point being the spatial median (see Kärkkäinen and Äyrämö, 2005 for more information and other iterative methods as well). The spatial median $\boldsymbol{\gamma}$ at step $k+1$ is equal to

$$\boldsymbol{\gamma}^{k+1} = \frac{\sum_{i=1}^{n} \frac{\mathbf{y}_i}{\|\boldsymbol{\gamma}^k - \mathbf{y}_i\|}}{\sum_{i=1}^{n} \frac{1}{\|\boldsymbol{\gamma}^k - \mathbf{y}_i\|}}.$$

The above iterative procedure is much much faster than using *nlm* or *optim* in R. The stopping criterion I have put is if the sum of the absolute difference between two successive iterations is less than 10^{-9}. If at some step k, the spatial median is equal to any of the vectors, the final result is the spatial median at the $k-1$ stp.

```
spat.med <- function(y, tol = 1e-09) {
  ## contains the data
  u1 <- Rfast::colMedians(y)
  y <- t(y)
  z <- y - u1
  ww <- 1 / sqrt( Rfast::colsums(z^2) )
  wei <- ww / sum(ww)
  u2 <-  as.vector( y %*% wei )
  while ( sum( abs(u2 - u1) ) > tol ) {
    z <- y - u2
    u1 <- u2
    ww <- 1 / sqrt( Rfast::colsums(z^2) )
    if ( max( ww ) < Inf ) {
      wei <- ww / sum(ww)
      u2 <- as.vector( y %*% wei )
    }
  }
```

```
  u2
}
```

The function below is an older version but I kept it so that you can measure the time with both of them. Of course the above is to be preferred as it is much faster.

```
spat.med_old <- function(x) {
  ## x contains the data
  p <- dim(x)[2]
  s <- diag(p)
  u1 <- apply(x, 2,median)
  ww <- 1 / sqrt( mahalanobis(x, u1, s, inverted = TRUE ) )
  u2 <- colSums(x * ww) / sum( ww)
  i <- 2
  while ( sum( abs(u2 - u1) ) > 1e-9 ) {
    i <- i +1
    u1 <- u2
    ww <- 1 / sqrt( mahalanobis(x, u1, s, inverted = TRUE ) )
    u2 <- colSums(x * ww) / sum( ww)
    if ( any( is.na( u2 ) ) )  u2 <- u1
  }
  list(iters = i, sm = u2)
}
```

7.3 Spatial sign covariance matrix

Now that we have seen the spatial median, we can see the spatial sign covariance matrix. It is defined as

$$SSC = \frac{1}{n}\sum_{i=1}^{n} s\left(\mathbf{x}_i - \boldsymbol{\gamma}\right) s\left(\mathbf{x}_i - \boldsymbol{\gamma}\right)^T,$$

where γ is the spatial median we saw before and $s\left(\mathbf{x}\right) = \frac{\mathbf{x}}{\|\mathbf{x}\|}$. So at first, we subtract the spatial median from all observations. Then we normalize each vector (make it unit vector) and then calculate the classical covariance matrix.

```
sscov <- function(x, me = NULL, tol = 1e-09) {
  ## x contains the data
  n <- dim(x)[1]  ## sample size
  p <- dim(x)[2]
```

```
  if ( is.null(me) ) me <- spat.med(x, tol)  ## spatial median of x
  y <- x - rep( me, rep(n, p) )
  rs <- sqrt ( Rfast::rowsums(y^2) )
  crossprod( y / rs ) / n  ## SSCM
}
```

7.4 Spatial median regression

If we substitute the spatial median $\boldsymbol{\gamma}$ we saw before with a linear function of covariates we end up with the spatial median regression (Chakraborty, 2003). So then, we want to find the $\mathbf{B}$ matrix of parameters which minimize the following sum

$$\sum_{i=1}^{n} \| \mathbf{y}_i - \mathbf{B}\mathbf{x}_i \| .$$

If the dependent variable is a vector or a matrix with one column, then the univariate median regression will be implemented. For more information on the univariate median regression see the package quantreg created by Koenker (2015). I use this package in order to obtain coefficients for the univariate regressions of every Y_i on the X_s. These would serve as initial values in the optimization procedure. I was using the OLS coefficients for this purpose. In some examples I had done, this did not seem to matter. If you have large sample, or many variables, or many outliers, maybe it will matter. The function *spatmed.reg* is the most efficient of them three below.

```
spatmed.reg_1 <- function(y, x, xnew = NULL) {
  ## y contains the dependent variables
  ## x contains the independent variable(s)

  runtime <- proc.time()
  y <- as.matrix(y)
  x <- as.matrix(x)
  d <- dim(y)[2]  ## dimensionality of y
  x <- cbind(1, x)  ## add the constant term
  p <- dim(x)[2]  ## dimensionality of x
  z <- list(y = y, x = x)
  ## medi is the function to perform median regression
  medi <- function(beta, z) {
    y <- z$y
    x <- z$x
    p <- dim(x)[2]
```

```
  be <- matrix(beta, nrow = p)
  est <- x %*% be
  sum( sqrt( rowSums((y - est)^2) ) )
}
## we use nlm and optim to obtain the beta coefficients
ini <- matrix(nrow = p, ncol = d)
for (i in 1:d)  ini[, i] <- coef( quantreg::rq(y[, i] ~ x[, -1]) )
ini <- as.vector(ini)
qa <- nlm(medi, ini, z = z, iterlim = 10000)
qa <- optim(qa$estimate, medi, z = z, control = list(maxit = 20000))
qa <- optim(qa$par, medi, z = z, control = list(maxit = 20000),
hessian = TRUE)
beta <- matrix( qa$par, ncol = dim(y)[2] )

if ( is.null(xnew) ) {
  est = x %*% beta
} else {
  xnew <- cbind(1, xnew)
  xnew <- as.matrix(xnew)
  est <- xnew %*% beta
}

seb <- sqrt( diag( solve(qa$hessian) ) )
seb <- matrix( seb, ncol = dim(y)[2] )

if ( is.null(colnames(y)) ) {
  colnames(seb) <- colnames(beta) <- paste("Y", 1:d, sep = "")
} else  colnames(seb) <- colnames(beta) <- colnames(y)

if ( is.null(colnames(x)) ) {
  p <- dim(x)[2] - 1
  rownames(beta) <- c("constant", paste("X", 1:p, sep = "") )
  if ( !is.null(seb) ) {
    rownames(seb) <- c("constant", paste("X", 1:p, sep = "") )
  }
} else {
  rownames(beta)  <- c("constant", colnames(x)[-1] )
  if  ( !is.null(seb) ) rownames(seb) <- c("constant", colnames(x)[-1] )
}
```

```
  if (d == 1)  est <- as.vector(est)
  runtime <- proc.time() - runtime

  list(runtime = runtime, beta = beta, seb = seb, est = est)
}
```

The second algorithm is

```
spatmed.reg_2 <- function(y, x, xnew = NULL, tol = 1e-07) {
  d = dim(y)[2]
  x = cbind(1, x)
  p = dim(x)[2]

   medi <- function(be, z) {
     y <- z$y
     x <- z$x
     p <- dim(x)[2]
     be <- matrix(be, nrow = p)
     est <- x %*% be
     sum( sqrt( rowSums( (y - est)^2 ) ) )
   }

  tic = proc.time()
  B1 = .lm.fit(x, y)$coefficients
  est = x %*% B1
  res = y - est
  di2 = sqrt( rowSums( res^2 ) )
  z = x /di2
  der = crossprod(z, res)
  der2 = - crossprod(x, z) + tcrossprod(der)
  B2 = B1 - solve(der2, der)

  i = 2
  while ( sum( abs( B2- B1 ) ) > tol ) {
    B1 = B2
    est = x %*% B1
    res = y - est
    di2 = sqrt( rowSums( res^2 ) )
    z = x / di2
```

```
    der = crossprod(z, res)
    der2 = - crossprod(x, z) + tcrossprod(der)
    i = i + 1
    B2 = B1 - solve(der2, der)
  }

  be <- B2
  ## we use nlm and optim to obtain the standard errors
  z <- list(y = y, x = x)
  qa <- nlm(medi, as.vector(be), z = z, iterlim = 1000, hessian = TRUE)
  seb <- sqrt( diag( solve(qa$hessian) ) )
  seb <- matrix(seb, ncol = d)
  if ( is.null(xnew) ) {
    est <- x %*% be
  } else {
    xnew <- cbind(1, xnew)
    xnew <- as.matrix(xnew)
    est <- xnew %*% be
  }
  if ( is.null(colnames(y)) ) {
    colnames(seb) <- colnames(be) <- paste("Y", 1:d, sep = "")
  } else  colnames(seb) <- colnames(be) <- colnames(y)
  if ( is.null(colnames(x)) ) {
    p <- dim(x)[2] - 1
    rownames(be) <- c("constant", paste("X", 1:p, sep = "") )
    rownames(seb) <- c("constant", paste("X", 1:p, sep = "") )
  } else {
    rownames(be)  <- c("constant", colnames(x)[-1] )
    rownames(seb) <- c("constant", colnames(x)[-1] )
  }
  runtime = proc.time() - tic
  list(iter = i, runtime = runtime, beta = be, seb = seb, est = est)
}
```

Finally, the one I suggest (for speed purposes) is

```
spatmed.reg <- function(y, x, xnew = NULL, tol = 1e-07, ses = TRUE) {
  n <- dim(y)[1]
  ## the desing matrix is created
  x <- model.matrix(y ~ ., as.data.frame(x) )
```

```
p <- dim(x)[2]
d <- dim(y)[2]
 medi <- function(be, y, x, p) {
   be <- matrix(be, nrow = p)
   est <- x %*% be
   sum( sqrt( rowSums( (y - est)^2 ) ) )
 }
tic <- proc.time()

B1 <- .lm.fit(x, y)$coefficients
est <- y - x %*% B1
ww <- sqrt( Rfast::rowsums( est^2 )
z <- x / ww
B2 <- solve( crossprod(x, z), crossprod(z, y) )
i <- 2
while ( sum( abs(B2 - B1) ) > tol ) {
  i <- i + 1
  B1 <- B2
  est <- y - x %*% B1
  ww <- sqrt( Rfast::rowsums( est^2 ) )
  ela <- which( ww == 0 )
  z <- x / ww
  if ( length(ela) > 0 )  z[ela, ] <- 0
  B2 <- solve( crossprod(x, z), crossprod(z, y) )
}

be <- B2
seb <- NULL
if ( ses ) {
  ## we use nlm and optim to obtain the standard errors
  qa <- nlm(medi, as.vector(be), y = y, x = x, p = p, iterlim = 5000,
  hessian = TRUE)
  seb <- sqrt( diag( solve(qa$hessian) ) )
  seb <- matrix(seb, ncol = d)
  if ( is.null(colnames(y)) ) {
    colnames(seb) <- colnames(be) <- paste("Y", 1:d, sep = "")
  } else  colnames(seb) <- colnames(be) <- colnames(y)
  if ( is.null(colnames(x)) ) {
    p <- dim(x)[2] - 1
```

```
      rownames(be) <- c("constant", paste("X", 1:p, sep = "") )
      rownames(seb) <- c("constant", paste("X", 1:p, sep = "") )
    } else {
      rownames(be)  <- c("constant", colnames(x)[-1] )
      rownames(seb) <- c("constant", colnames(x)[-1] )
    }

  }

  if ( is.null(xnew) ) {
    est <- x %*% be
  } else {
    xnew <- model.matrix(y ~ ., as.data.frame(xnew) )
    est <- xnew %*% be
  }
  if ( is.null(colnames(y)) ) {
    colnames(be) <- paste("Y", 1:d, sep = "")
  } else  colnames(be) <- colnames(y)
  if ( is.null(colnames(x)) ) {
    p <- dim(x)[2] - 1
    rownames(be) <- c("constant", paste("X", 1:p, sep = "") )
  } else {
    rownames(be)  <- c("constant", colnames(x)[-1] )
  }
  runtime <- proc.time() - tic
  list(iter = i, runtime = runtime, be = be, seb = seb, est = est)
}
```

7.5 Robust correlation analysis and other analyses

Should someone want to estimate a robust correlation coefficient, all he has to do is calculate the robust covariance matrix using the function *cov.mcd* available in the *MASS* library. Then, by turning the covariance matrix into a correlation matrix (*cov2cor*) the job is done.

In the case of robust principal component analysis one can do the same, perform an eigen analysis of the robust covariance (or correlation) matrix. This idea expands to principal components regression and discriminant analysis as well.

7.6 Detecting multivariate outliers graphically with the forward search

The forward search is a way to identify multivariate outliers graphically. A possible multivariate outlier is an observation whose squared Mahalanobis distance is grater than the $\chi^2_{0.975,p}$, where p denotes the number of dimensions. If the covariance matrix though is not estimated robustly this can lead to the masking effect. Outliers whose effect is masked and they are seen as not outliers. For this reason robust estimation of the covariance matrix is necessary. The Mahalanobis distance of a multivariate observation $\mathbf{x}$ is given by

$$MD\left(\mathbf{x}\right) = \left(\mathbf{x} - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1} \left(\mathbf{x} - \boldsymbol{\mu}\right),$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the mean vector and covariance matrix.

Robust estimation of the covariance matrix on the other hand can lead to what is called swamping effect. Outliers which are not outliers are detected as possible outliers. Filzmoser (2005) introduced a new method of robust detection of multivariate outliers following the idea of Gervini (2003) to increase the efficiency of the robust estimation of scatter (covariance matrix) and location (mean vector). The method is again based on the MCD we saw in the robust multivariate regression analysis. This method can be found in the R package mvoutlier written by Filzmoser and Gschwandtner (2014).

The forward search (FS) is a graphical method which shows the effect of the outliers in a graph. The reference book for this method is written by Atkinson et al. (2004). A paper explaining nicely the steps of the algorithm is written by Mavridis and Moustaki (2008). Let us now briefly explain the steps of the forward search.

First step of the FS

In the first step of the search a *good* subset must be chosen. This means that an outlier-free subset must be found in order to provide robust estimators of some parameters. After the subset size is determined a large number (e.g. 1000) of subsets of that size are determined. Let n denote the number of multivariate observations and n_g denote the initial subset size. This means that there are $\binom{n}{n_g}$ possible subsets. Once a *good* subset is determined the search consists of $n - n_g$ steps; the number of observations that will enter the initial subset.

Many ways have been suggested in the literature so as to find the best subset with which to start the search. The MCD is used here and the fraction required is actually chosen by the MCD and is equal to $[(n + p + 1)/2]$, where n and p indicate the sample size and the number of variables or dimensions, respectively and $[x]$ means the the largest integer not greater than x. So, the idea is to estimate initially robust estimates of scatter and location and then use these to calculate the Mahalanobis distances of the selected observations (based on which the robust estimates are calculated). Then keep the n_g observations with the smallest

Mahalanobis distances.

The initial subset size is another issue also. Atkinson et al. (2004) proposed a size of $3p$. However the sample size is not crucial as long as it is outlier-free. I believe that the initial subset size should be determined taking into account the dimensions of the data matrix (both the number of variables and the sample size). However, in the function presented here, the default value is 20% of the sample size.

Finally, the mean and the variance of the observations in the subset are estimated. If there are no outliers in the data, the estimates are very robust.

Second step of the FS

Given a subset of size n_g observations one must find a way to progress in the search, which is to find a way to include all the $m = n - n_g$ remaining multivariate observations. The subset size is also called *basic* set (at each step its size is increased) and the set with all the other data is called *non-basic* set (at each step its size is decreased). One good way is to calculate the Mahalanobis distances of the observations not in the initial subset from the robust estimates of scatter and location provided by the *basic* set and order them from the smallest to the largest. The observation with the smallest Mahalanobis is the one to leave the *non-basic* set and enter the *basic* set and the estimates of scatter and location are re-estimated.

The size of *basic* set is now $n_g + 1$ and there are $m - 1$ remaining steps of the FS and hence $m - 1$ observations in the *non-basic* set. The Mahalanobis distances of the observations in the *non-basic* set are calculated and ordered again in an ascending order and the observation with the smallest distance enters the *basic* set. This procedure is repeated until all observations from the *non-basic* set enter the *basic* set.

One observation is added at each step, but the inclusion of an outlier can cause the ordering of the Mahalanobis distances of the points not in the basic set to change. This change of the data ordering during the FS is a feature of the multivariate data and not of the univariate data as mentioned by Atkinson et al. (2004).

At this point we must say that this is the non standard FS. In the standard FS a point can be included in the set at a step and be removed at a later step.

Third step of the FS

The last step of the FS involves monitoring some statistics of interest during the search which are helpful in the identification of outliers or observations that have a larger effect than expected. One statistic of interest could be the minimum Mahalanobis distance of the observations not in the *basic* set. If the distance is large, this is an indication that an outlier is about to enter the *basic* set. If however a cluster of outliers join the set successively, these minimum distances will decrease. Another way is to monitor the change between two

successive minimum Mahalanobis distances or the scaled by the determinant covariance matrices Mahalanobis distances (Atkinson et al., 2004).

If one's concern lies in estimating the influence of an observation in a model (multiple regression or factor analysis for instance) then the parameter estimates, the residuals and other goodness of fit tests are likely to be of more interest. It is true, that even a single outlier can cause a factor analysis model to go wrong or a test of multivariate normality to fail.

The output of the *forward.ns* function has two components, a) the order of entrance all the observations and b) the minimum Mahalanobis distances of the initial step and the minimum Mahalanobis distances as described in step 2.

```
forward.ns <- function(z, quan = 0.2, graph = TRUE) {
  dm <- dim(z)
  n <- dm[1]  ## sample size
  p <- dm[2]  ## dimensionality
  z <- cbind(1:n, z)  ## this will identify the sequence of entrance
  arxi <- quan * n  ## initial subset size
  if (arxi < 0.5 * p * (p + 1) + 1)  arxi <- 0.5 * p * (p + 1) + 1
  ## n the final sample we will see the order of entrance
  Xmcd <- robust::covRob(z[, -1], method = "weighted")  ## search 5000 subsets
  disa <- Rfast::mahala(z[, -1], Xmcd$center, Xmcd$cov)
  names(disa) <- 1:n
  disa <- sort(disa)
  b <- as.integer( names( disa[1:c(arxi + 1)] ) )
  ini <- z[b, ]  ## initial subset
  z3 <- z[-b, ]
  vim <- nrow(z3)  ## steps of the FS
  dis <- numeric(vim)
  nt <- arxi + 1
  mn <- colmeans(ini[, -1])
  mat <- t(ini[, -1]) - mn
  sn <- tcrossprod(mat)

  for ( j in 1:c(vim - 1) ) {
    d <- Rfast::mahala( z3[, -1], mn, sn/(nt - 1) )
    a <- which.min(d)
    dis[j] <- min(d)
    y <- z3[a, -1]
    nt <- nt + 1
    sn <- (sn + tcrossprod(y - mn) * (nt - 1)/ nt )
```

```
    mn <- ( mn * (nt - 1) + y )/nt
    ini <- rbind(ini, z3[a, ])
    z3 <- z3[-a, ]
  }

  z3 <- matrix(z3, ncol = length(z3))
  ini <- rbind(ini, z3)
  dis[vim] <- Rfast::mahala( z3[, -1], Rfast::colmeans(ini[1:(n - 1), -1]),
  cov(ini[1:(n - 1), -1]) )
  nama <- ini[, 1]
  ini <- ini[, -1]
  if ( graph )  plot(dis, type = "l")
  MD <- as.vector( c(disa[1:c(arxi + 1)], dis) )
  list(order = nama, MD = MD)
}
```

7.7 Detecting high-dimensional multivariate outliers with a diagonal MCD

Very recently (2015) a paper appeared in Biometrika regarding outliers in high-dimensional data sets written by Kwangil et al. (2015). When Kwangil et al. (2015) say high-dimensional they do not mean thousands (or hundreds of thousands) of variables, but rather the $n \ll p$ case, when the sample size (n) is less or much much less than the number of variables (p). It seems to work reasonably well, but it takes some time, a few seconds. Of course, robust statistics require more time and if we increase the dimensions to thousands or tens of thousands this would take a few minutes. Nonetheless, it is suggested. I will now try to describe the idea, as usual, very briefly and hopefully concisely, so that you can understand what this paper is all about.

Step 1 Take $m = 100$ (you can increase if you like) small samples (subsets) of size 2 (this is suggested by the authors and is used in their codes).

Step 2 Calculate the covariance matrix of these m small samples and take the product of their diagonal elements (variances).

Step 3 Select the subset with the minimum diagonal product value.

Step 4 Calculate the Mahalanobis distances of the all the points using the mean vector and the diagonal covariance matrix calculated from the selected subset. When I say diagonal I mean that you only calculate the variances of the variables. No covariance terms are calculated.

In the second phase, the refined algorithm, presented below, is performed.

Step 1 Order the Mahalanobis distances and keep the $h = [n/2] + 1$ observations corresponding to the $h = [n/2 + 1]$ smallest distances, where $[x]$ is the integer part of x.

Step 2 Use this subset and calculate the mean vector and the diagonal covariance matrix.

Step 3 Calculate the Msquared ahalanobis distances of all observations MD_i^2 and scale them using $MD_i^2 = \frac{pMD_i^2}{\text{median}\left(MD_i^2\right)}$.

Step 4 Calculate

$$\text{tr}\left(R_h^2\right),$$

where R_h is the correlation matrix of the h observations and tr denotes the trace of a matrix and

$$\begin{aligned} \text{tr}\left(R_h^2\right)_{MDP} &= \text{tr}\left(R_h^2\right) - \frac{p^2}{h} \\ \hat{c}_{pn} &= 1 + \frac{\text{tr}\left(R_h^2\right)}{p^{3/2}} \end{aligned}$$

Step 5 Put weights to each observation

$$w_i = \left\{ \begin{array}{ll} 1 & if \quad MD_i^2 \leq p + z_\delta \sqrt{2\text{tr}\left(R_h^2\right)_{MDP}} \\ 0 & if \quad else \end{array} \right\}$$

Step 6 For the observations with positive weight repeat Steps 2-5. That is, for this selected subset of observations, calculate the updated mean vector and covariance matrix, then calculate the scaled squared Mahalanobis distances and calculate the weights once more. These will be the final weights. If an observation has a zero weight it means it is a possible outlier.

I was given the function below by Changliang Zou who is one of the authors of the Biometrika 2015 paper I have just described (Kwangil et al., 2015), so any deeper in understanding questions should be addressed to him (or any other of his co-authors).

```
rmdp_old <- function(y, alpha = 0.05, itertime = 500) {
  ## y is the data
  ## alpha is the significance level
  ## itertime is the number of iterations for the first step of the algorithm
  n <- dim(y)[1]  ## sample size
  p <- dim(y)[2]  ## dimensionality
  h <- round(n/2) + 1  ## subset of data to be used for the location
```

```
ty <- t(y)

## and scatter estimators
init_h <- 2  ## initial sample size for the first step of the algorithm
delta <- alpha/2
bestdet <- 0
jvec <- numeric(n)
runtime <- proc.time()

for ( A in 1:itertime ) {
    id <- sample(n, init_h, replace = FALSE)
    ny <- y[id, ]
    mu_t <- Rfast::colmeans(ny)
    var_t <- Rfast::colVars(ny)
    sama <- ( ty - mu_t )^2 / var_t
    disa <- Rfast::colsums(sama)
    crit <- 10
    l <- 0
    while (crit != 0 & l <= 15) {
       l <- l + 1
       ivec <- numeric(n)
       dist_perm <- order(disa)
       ivec[ dist_perm[1:h] ] <- 1
       crit <- sum( abs(ivec - jvec) )
       jvec <- ivec
       newy <- y[dist_perm[1:h], ]
       mu_t <- Rfast::colmeans(newy)
       var_t <- Rfast::colVars(newy)
       sama <- ( ty - mu_t )^2 / var_t
       disa <- Rfast::colsums(sama)
    }
    tempdet <- prod(var_t)
    if(bestdet == 0 | tempdet < bestdet) {
        bestdet <- tempdet
        final_vec <- jvec
    }
  }

  submcd <- (1:n)[final_vec != 0]
```

```
    mu_t <- Rfast::colmeans( y[submcd, ] )
    var_t <- Rfast::colVars( y[submcd, ] )
    sama <- ( ty - mu_t )^2 / var_t
    disa <- Rfast::colsums(sama)
    disa <- disa * p / Rfast::med(disa)
    ER <- cora( y[submcd, ] )
    tr2_h <- sum( ER^2 )  ## trace of ER %**% ER
    tr2 <- tr2_h - p^2 / h
    cpn_0 <- 1 + (tr2_h) / p^1.5
    w0 <- (disa - p) / sqrt( 2 * tr2 * cpn_0 ) < qnorm(1 - delta)
    nw <- sum(w0)
    sub <- (1:n)[w0]
    ysub <- y[sub, ]
    mu_t <- Rfast::colmeans( ysub )
    var_t <- Rfast::colVars( ysub )
    ER <- cora( ysub )
    tr2_h <- sum( ER^2 )  ## trace of ER %**% ER
    tr2 <- tr2_h - p^2 / nw
    sama <- ( ty - mu_t )^2 / var_t
    disa <- Rfast::colsums(sama)
    scal <- 1 + 1/sqrt( 2 * pi) * exp( - qnorm(1 - delta)^2 / 2 ) /
    (1 - delta) * sqrt( 2 * tr2) / p
    disa <- disa / scal
    cpn_1 <- 1 + (tr2_h) / p^1.5
    dis <- (disa - p) / sqrt(2 * tr2 * cpn_1)
    wei <- dis < qnorm(1 - alpha)
    runtime <- proc.time() - runtime
    list(runtime = runtime, dis = dis, wei = wei)
}
```

The above version is the original one with some modifications to make it faster. However if you have $10,000$ or even $40,000$ variables and not enough memory this is unlikely to work. The reason for this is that after the for (A in 1:itertime) loop the correlation matrix must be calculated, twice. For this reason I created a memory efficient version of the above. The function below is taken directly from the package **Rfast** (Papadakis et al., 2019). The *for* loop was converted in C++ and the part after that became memory efficient using the following mathematical trick

$$\text{tr}\left(RR\right) = \sum_{i=1}^{p} \lambda_i^2,$$

where R is the correlation matrix of the $n \times p$ matrix $\mathbf{X}$ and λ_i is the i-th eigenvalue of $\mathbf{X}$

```
rmdp <- function(y, alpha = 0.05, itertime = 100) {
  ## y is the data
  ## alpha is the significance level
  ## itertime is the number of iterations for the first step of the algorithm
  dm <- dim(y)
  n <- dm[1]  ## sample size
  p <- dm[2]  ## dimensionality
  h <- round(n/2) + 1  ## subset of data to be used for the location
  ## and scatter estimators
  init_h <- 2  ## initial sample size for the first step of the algorithm
  delta <- alpha/2
  runtime <- proc.time()
  ########################
  ########################
  id <- replicate( itertime, sample.int(n, 2) ) - 1
  final_vec <- as.vector(.Call('Rfast_rmdp', PACKAGE = 'Rfast',y,h,id,itertime))
  ########################
  ########################
  submcd <- seq(1, n)[final_vec != 0]
  mu_t <- Rfast::colmeans( y[submcd, ] )
  var_t <- Rfast::colVars( y[submcd, ] )
  sama <- ( t(y) - mu_t )^2 / var_t
  disa <- Rfast::colsums(sama)
  disa <- disa * p / Rfast::med(disa)
  b <- Rfast::hd.eigen(y[submcd, ], center = TRUE, scale = TRUE)
  tr2_h <- sum(b^2)
  tr2 <- tr2_h - p^2 / h
  cpn_0 <- 1 + (tr2_h) / p^1.5
  w0 <- (disa - p) / sqrt( 2 * tr2 * cpn_0 ) < qnorm(1 - delta)
  nw <- sum(w0)
  sub <- seq(1, n)[w0]
  mu_t <- Rfast::colmeans( y[sub, ] )
  var_t <- Rfast::colVars( y[sub, ], suma = nw * mu_t )
  sama <- ( t(y) - mu_t )^2 / var_t
  disa <- colsums(sama)
  b <- Rfast::hd.eigen(y[sub, ], center = TRUE, scale = TRUE)
  tr2_h <- sum(b^2)
  tr2 <- tr2_h - p^2 / nw
```

```
  scal <- 1 + exp( - qnorm(1 - delta)^2 / 2 ) /
  (1 - delta) * sqrt( tr2) / p / sqrt(pi)
  disa <- disa / scal
  cpn_1 <- 1 + (tr2_h) / p^1.5
  dis <- (disa - p) / sqrt(2 * tr2 * cpn_1 )
  wei <- dis < qnorm(1 - alpha)
  ####
  runtime <- proc.time() - runtime
  list(runtime = runtime, dis = dis, wei = wei)
}
```

8 Compositional data

Compositional data are a special type of multivariate data in which the elements of each observation vector are non-negative and sum to a constant, usually taken to be unity. Data of this type arise in biological settings, for instance, where the researcher is interested in the proportion of megakaryocytes in ploidy classes. Other areas of application of compositional data analysis include geology, where the metal composition of a rock specimen is of interest; archaeometry, where the composition of ancient glasses for instance is of interest; and economics, where the focus is on the percentage of the household expenditure allocated to different products. Other fields are political sciences, forensic sciences, ecology and sedimentology.

The main book suggested to the reader for familiarizing himself with compositional data is Aitchison's book (Aitchison, 2003). For more information one can look at these Lecture notes on Compositional Data Analysis and Van Den Boogaart and Tolosana-Delgado (2013). The functions described here exist as an R package as well Compositional Tsagris and G. (2016).

In mathematical terms, we can define the relevant sample space as

$$\mathbb{S}^d = \left\{ (x_1, ..., x_D) | x_i \geq 0, \sum_{i=1}^{D} x_i = 1 \right\}, \tag{8.1}$$

where $d = D - 1$. When $D = 3$, the best way to visualize them is the ternary diagram (or a three edged pyramid when $D = 4$), which is essentially a triangle. If we plot the simplex in three dimensions what we will see is a two dimensional triangle, therefore a projection to two dimensions under the unity sum constraint is convenient. The result is the already mentioned ternary diagram. The higher the value of the component, the closer it is to the corresponding vertex.

8.1 Some introductory stuff

8.1.1 Ternary plot

Suppose we have a composition $\mathbf{X}$ where $\mathbf{x}_i = (x_1, x_2, x_3)^T \in \mathbb{S}^2$. The matrix $\mathbf{X}$ consists of n rows and 3 columns, thus every row vector consists of 3 proportions. In order to plot the points on a ternary diagram we need to left multiply the composition by the following matrix:

$$P = \begin{bmatrix} 0 & 1 & 0.5 \\ 0 & 0 & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{8.2}$$

The columns of (8.2) represent the vertices of an equilateral triangle in the Cartesian

coordinates (Schnute and Haigh, 2007). In this way the length of each side of the triangle is equal to 1. Watson and Nguyen (1985) gave a different representation of an equilateral triangle, in which case the barycentre lies on the origin and the height of the triangle is equal to 1, resulting in the length of the sides being greater than 1. Viviani's theorem concerns any point within the triangle and the three lines from that point which are perpendicular to the sides of the triangle. The sum of the lengths of the lines is a fixed value, regardless of the position of the point and is equal to the height of the triangle. Below we present the code to produce a ternary plot.

The pair of coordinates of every composition in $\mathbb{R}^2$ after multiplying by the P matrix (8.2) is given by

$$\mathbf{y} = (y_1, y_2) = \left(x_2 + \frac{x_3}{2}, \frac{x_3\sqrt{3}}{2} \right) \tag{8.3}$$

Below is the code to produce the ternary plot with the the compositional vectors plotted in $\mathbb{R}^2$. The code plots the closed geometric mean (Aitchison, 1989) and the simple arithmetic mean of the data as well. The closed geometric mean of a composition $\mathbf{X}$ is defined as

$$\boldsymbol{\mu}_0 = \left(\frac{g_1}{g_1 + \ldots + g_D}, \ldots, \frac{g_D}{g_1 + \ldots + g_D} \right), \tag{8.4}$$

where

$$g_i = \prod_{j=1}^{n} x_{ij}^{1/n},\ i = 1, \ldots, D.$$

The simple arithmetic mean is defined as

$$\boldsymbol{\mu}_1 = \left(\frac{1}{n} \sum_{j=1}^{n} x_{1j}, \ldots, \frac{1}{n} \sum_{j=1}^{n} x_{Dj} \right) \tag{8.5}$$

We have added an extra option, the plotting of the first principal component on $\mathbb{S}^2$. Let us see this option a bit more. If you use the package *compositions* this option is available there. But here we show how it's constructed. In addition, MASS has a function for the ternary diagram. type *?Skye* to see the dataset *Skye*. In the help, is the function *ternary*. At first let use transform the data using the centred log-ratio transformation

$$\mathbf{y} = \left(\log x_1 - \frac{1}{D} \sum_{i=1}^{D} \log x_i, \ldots, \log x_D - \frac{1}{D} \sum_{i=1}^{D} \log x_i \right) = \left(\log \frac{x_1}{g(\mathbf{x})}, \ldots, \log \frac{x_D}{g(\mathbf{x})} \right), \tag{8.6}$$

where $g(\mathbf{x}) = \prod_{j=i}^{D} x_i^{1/D}$ is the geometric mean of each compositional vector. Then we will calculate the eigenvectors ($\mathbf{V}$) of the covariance matrix of the centred log-ratio transformed

data as Aitchison (1983) suggests. We will take the first eigenvector $\mathbf{v}_1$ only and the mean of the transformed data ($\hat{\boldsymbol{\mu}}$), so that the beginning of this unit vector is the not the origin $(0, 0, 0)$ but the mean vector ($\hat{\boldsymbol{\mu}}$).

So the eigenvector starts from ($\hat{\boldsymbol{\mu}}$) and has its direction pointed by its values. So this vector has a beginning and an end, or two points on the Euclidean coordinate system which define it. Let's call them A (the $\hat{\boldsymbol{\mu}}$) and B. In general a line segment on the Euclidean hyper plane is defined by two points and a scalar

$$\lambda A + (1 - \lambda) B.$$

We calculate the scores of the first principal component to see their range so that we adjust the values of λ more or less to it. Thus, all we have to do now is choose m different values of λ and calculate points on the straight line defined by the eigenvector. A and B have three elements each, so in the end we will have a matrix of some rows and of 3 columns. Let's call this matrix Z. Now we will calculate the inverse of (8.6) for each row of Z in order to map the line segment back into the simplex $\mathbb{S}^2$.

$$\mathbf{c}_j = \left(\frac{e^{z_1 j}}{\sum_{k=1}^{D} e^{z_{kj}}}, \ldots, \frac{e^{z_D j}}{\sum_{k=1}^{D} e^{z_{kj}}} \right), \quad j = 1, \ldots, m$$

The matrix $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_m)^T$ contains m points of the first principal component inside the simplex. We just have to put these points in the ternary diagram.

```
ternary <- function(x, means = TRUE, pca = FALSE) {
  ## x contains the composiitonal data
  ## if means==TRUE it will plot the arithmetic and the
  ## closed geometric mean
  ## if pca==TRUE it will plot the first principal component
  if ( !is.null( colnames(x) ) ) {
    nam <- colnames(x)
  } else nam <- paste("X", 1:3, sep = " ")
  n <- dim(x)[1]
  ina <- numeric(n) + 1
  ## m1 is the closed geometric mean
  g1 <- colMeans( log(x[, -1] / x[, 1]) )
  g2 <- c( 1, exp(g1) )
  m1 <- g2 / sum(g2)
  ## m2 is the simple arithmetic mean
  m2 <- Rfast::colmeans(x)
  x <- rbind(x, m1, m2)
```

```
## the next code checks for zeros
ina[ rowSums(x == 0) > 0 ] <- 3
b1 <- c(1/2, 0, 1, 1/2)
b2 <- c(sqrt(3)/2, 0, 0, sqrt(3)/2)
b <- cbind(b1, b2)
plot(b[, 1], b[, 2], type = "l", xlab = " ", ylab = " ", pty = "s",
xaxt = "n", yaxt = "n", bty = "n")
proj <- matrix(c(0, 1, 1/2, 0, 0, sqrt(3)/2), ncol = 2)
d <- x %*% proj
points( d[1:n, 1], d[1:n, 2], col = ina )
text( b[1, 1], b[1, 2] + 0.02, nam[3], cex = 1 )
text( b[2:3, 1], b[2:3, 2] - 0.02, nam[1:2], cex = 1 )
if ( means ) {
  ## should the means appear in the plot?
  points( d[c(n + 1), 1], d[c(n + 1), 2], pch = 2, col = 2 )
  points( d[c(n + 2), 1], d[c(n + 2), 2], pch = 3, col = 3 )
  legend(0.57, 0.9, c("closed geometric mean"," arithmetic mean"),
  pch = c(2, 3), col = c(2, 3), bg = 'gray90')
}
if (pca   &  min(x) > 0 ) {
  ## should the first principal component appear?
  zx <- log(x[1:n, ])
  z <- zx - Rfast::rowmeans( zx ) ## clr transformation
  m <- Rfast::colmeans(z)  ## mean vector in the clr space
  a <- eigen( cov(z) )$vectors[, 1] + m  ## move the unit vector a bit
  sc <- z %*% a
  lam <- seq( min(sc) - 1.5, max(sc) + 1.5, length = n )
  x1 <- cbind( a[1] * lam, a[2] * lam, a[3] * lam) + cbind( m[1] * (1 - lam),
  m[2] * (1 - lam), m[3] * (1 - lam) )
  expx1 <- exp(x1)
  wa1 <- expx1 / Rfast::rowsums( expx1 )  ## first principal component in S^2
  wa <- wa1 %*% proj
  lines(wa, lwd = 2, lty = 2)
}
mu <- rbind(m1, m2)
rownames(mu) <- c("closed geometric", "arithmetic mean")
colnames(mu) <- nam
mu
```

8.1.2 Log-ratio transformations

The Dirichlet distribution (8.9) we will see later is a natural parametric model on the simplex but not very rich though. Alternative distributions are the multivariate normal and skew normal and the multivariate t distribution. We will show two transformation which allow us to map $\mathbb{S}^d$ onto $\mathbb{R}^d$.

Aitchison (2003) suggested a log-ratio transformation for compositional data. He termed it additive log-ratio transformation and is the generalised logistic transformation

$$\mathbf{y} = \left(\log \frac{x_1}{x_D}, \ldots, \log \frac{x_d}{x_D} \right), \tag{8.7}$$

where x_D indicates the last component (any other component can play the role of the common divisor). Another log-ratio transformation we saw before, also suggested by Aitchison (1983) is the centred log-ratio transformation (8.6). The additive log-ratio transformation maps the data from $\mathbb{S}^d$ to $\mathbb{R}^d$, in contrast to the centred log-ratio transformation (8.6) which maps the $\mathbb{S}^d$ onto $\mathbb{Q}^d$

$$\mathbb{Q}^d = \left\{ (x_1, ..., x_D)^T : \sum_{i=1}^{D} x_i = 0 \right\}.$$

However, if we left multiply the centred log-ratio transformation (8.6) by the Helmert sub-matrix (6.2) the result is the isometric log-ratio transformation (Egozcue et al., 2003) which maps the data from $\mathbb{Q}^d$ onto $\mathbb{R}^d$.

$$\mathbf{z} = \mathbf{H}\mathbf{y} \tag{8.8}$$

The multiplication by the Helmert matrix is often met in shape analysis and it was applied also in simplex shape spaces by Le and Small (1999). It was also known to Aitchison (2003) who knew the relationship between the covariance matrix of (8.6) and (8.8) transformations. In fact, the multiplication by the Helmert sub-matrix leads to what he called standard orthogonal contrasts.

We will skip the technical details here and just say that the road is open now to fit multivariate distributions whose support is the whole of $\mathbb{R}^d$. To be more accurate, we also need the Jacobians of the log-ratio transformations, but in the contour plot we will not use them. For more information the reader is addressed to Aitchison (2003) and Pawlowsky Glahn et al. (2007). We can apply either the additive log-ratio transformation (8.7) or the isometric log-ratio transformation (8.8) and in the transformed data fit a multivariate distribution defined in $\mathbb{R}^d$.

8.2 Estimating location and scatter parameters for compositional data

I provide a general function which allows for fitting a multivariate normal, t and skew-normal distribution to compositional data and hence estimating their parameters. I left the multivariate skew-t outside because of its complexity. In addition robust estimation of the mean and covariance matrix via the MCD method are offered. Sharp (2006) used the graph median as a measure of central tendency for compositional data. We will provide a function to calculate the spatial median instead of the graph median, along with the spatial sign covariance matrix. We saw the spatial median function in Section 7.4. In all cases, either the additive log-ratio (8.7) or the isometric log-ratio transformation (8.8) can be used.

```
comp.den <- function(x, type = "alr", dist = "normal", tol = 1e-09) {
 ## x is the compositional data
 ## type is the type of transformation, "alr", "ilr"
 ## dist is the distribution to be fitted,
 ## "normal", "rob", "spatial", "t", "skewnorm"
 x <- as.matrix(x)  ## makes sure x is a matrix
 x <- x / rowSums(x)  ## makes sure x is compositional data
 ## if type = "none" or dist = "dirichlet" the Dirichlet is fitted
 if (dist == "normal") {
   if (type == "alr") {  ## additive log-ratio transformation
     y <- log(x[, -1]/ x[, 1])
     m <- colMeans(y)
     mu <- c( 1, exp(m) )
     mu <- mu / sum(mu)
     s <- cov(y)
   } else {
     y <- alfa(x, 0)
     m <- colMeans(y)
     mu <- alfainv(m, 0)
     s <- cov(y)
   }
   result <- list(mean = m, comp.mean = mu, covariance = s)

 } else if (dist == "t") {
     if (type == "alr") {  ## additive log-ratio transformation
       y <- log(x[, -1]/ x[, 1])
       mod <- multivt(y)
       m <- mod$center
       mu <- c( 1, exp(m) )
```

```
      mu <- mu / sum(mu)
      s <- mod$scatter
      dof <- mod$dof
    } else {
      y <- alfa(x, 0)
      mod <- multivt(y)
      m <- mod$center
      mu <- alfainv(m, 0)
      s <- mod$scatter
      dof <- mod$dof
    }
    result <- list(mean = m, comp.mean = mu, covariance = s, dof = dof)

} else if (dist == "rob") {
    if (type == "alr") {  ## additive log-ratio transformation
      y <- log(x[, -1]/ x[, 1])
      mod <- MASS::cov.rob(y, method = "mcd")
      m <- mod$center
      mu <- c( 1, exp(m) )
      mu <- mu / sum(mu)
      s <- mod$cov
      best <- mod$best
    } else {
      y <- alfa(x, 0)
      mod <- MASS::cov.rob(y, method = "mcd")
      m <- mod$center
      mu <- alfainv(m, 0)
      s <- mod$cov
      best <- mod$best
    }
    result <- list(mean = m, comp.mean = mu, covariance = s, best = best)

} else if (dist == "spatial") {
    if (type == "alr") {  ## additive log-ratio transformation
      y <- log(x[, -1]/ x[, 1])
      delta <- spat.med(y, tol = tol)
      comp.delta <- c( 1, exp( delta ) )
      comp.delta <- delta / sum( delta )
      s <- sscov(y, delta)
```

```
    } else {
      y <- alfa(x, 0)
      delta <- spat.med(y)
      comp.delta <- alfainv(delta, 0)
      s <- sscov(y, delta)
    }
    result <- list(spat.med = delta, comp.spat.med = comp.delta, ssc = s)

  } else if (dist == "skewnorm") {
    if (type == "alr") {  ## additive log-ratio transformation
      y <- log(x[, -1]/ x[, 1])
      mod <- sn::msn.mle(y = y)
      beta <- as.vector( mod$dp$beta )
      Omega <- as.matrix( mod$dp$Omega )
      alpha <- as.vector(mod$dp$alpha)
      cobeta <- c( 1, exp( beta) )
      cobeta <- cobeta / sum(cobeta)
    } else {
      y <- alfa(x, 0)
      mod <- sn::msn.mle(y = y)
      beta <- as.vector( mod$dp$beta )
      Omega <- as.matrix( mod$dp$Omega )
      alpha <- as.vector(mod$dp$alpha)
      cobeta <- alfainv(beta, 0)
    }
    result <- list(beta = beta, Omega = Omega, alpha = alpha,
    comp.beta = cobeta)
  }

  result
}
```

8.3 The Dirichlet distribution

The Dirichlet distribution is a distribution whose support is the simplex (8.1). The density of the Dirichlet distribution is the following

$$f\left(x_1,\ldots,x_D;\alpha_1,\ldots,\alpha_D\right)=\frac{1}{\mathbf{B}\left(\boldsymbol{\alpha}\right)}\prod_{i=1}^{D}x_i^{\alpha_i-1} \tag{8.9}$$

where

$$\mathbf{B}(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{D}\Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{D}\alpha_i\right)} \text{ and } \boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_D)$$

In the next two section we see how to estimate the parameters of the Dirichlet distribution.

8.3.1 Estimating the parameters of the Dirichlet

The log-likelihood of the Dirichlet has the following form:

$$l = n\log\Gamma\left(\sum_{i=1}^{D}\alpha_i\right) - n\sum_{i=1}^{D}\log\Gamma(\alpha_i) + \sum_{j=1}^{n}\sum_{i=1}^{D}(\alpha_i - 1)\log x_{ij}$$

- Classical MLE. We can use the "optim" function to maximize the log-likelihood. The argument "hessian=T" we will see in the function *diri.est* calculates the hessian matrix and the inverse of the hessian matrix serves as the observed information matrix of the parameters. This way can also be found at the package VGAM (Yee, 2010). The extra feature offered by the package is the ability to include covariates.

- An alternative parametrization. An alternative form of the Dirichlet density is via the precision parameter ϕ:

$$f(\mathbf{x}) = \frac{\Gamma\left(\sum_{i=1}^{D}\phi a_i^*\right)}{\prod_{i=1}^{D}\Gamma(\phi a_i^*)}\prod_{i=1}^{D}x_i^{\phi a_i^* - 1}, \tag{8.10}$$

  where $\phi = \sum_{i=1}^{D} a_i$ and $\sum_{i=1}^{D} a_i^* = 1$.

  Maier (2011) has created and R package (*DirichletReg*) which performs Dirichlet estimation (with or without covariates) with both parameter formulations. Furthermore, in this parametrization he offers the possibility of modelling the parameter ϕ with the covariates as well. The relative log-likelihood is

$$\ell = n\log\Gamma(\phi) - \sum_{j=1}^{n}\sum_{i=1}^{D}\log\Gamma(\phi a_i^*) + \sum_{j=1}^{n}\sum_{i=1}^{D}(\phi a_i^* - 1)\log x_{ij}, \tag{8.11}$$

- Estimation via the entropy. We will make use of the following relationship

$$E[\log X_i] = \psi(\alpha_i) - \psi(\alpha_0), \tag{8.12}$$

where ψ is the digamma function defined as

$$\psi(x) = \frac{d}{dx}\log\Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)} \text{ and } \alpha_0 = \sum_{i=1}^{D}\alpha_i$$

Instead of trying to maximize the log-likelihood of the Dirichlet distribution we will try to solve the k simultaneous equations imposed by 8.12. If you notice, these are just the first derivatives of the log-likelihood with respect to each of the parameters. In other words, their are the score statistics, since the expectation is in the game. I then opened up a book I have by Ng et al. (2011) about the Dirichlet distribution and I saw that they show that this approach is the generalised method of moments (GMM). No matter what the method is called, we will use the package BB (Varadhan and Gilbert, 2009). A disadvantage of the "entropy style" estimation is that the log-likelihood maximization is very stable and you can compare the results with the package VGAM (Yee, 2010).

Below is the code offering all three options, classical MLE, MLE with the precision parameter ϕ and via the entropy. For the classical MLE type I take the exponential of the parameters, to avoid negative solutions. In the alternative parameterization I take the exponential of the ϕ and the other parameters. This is a classical trick for such cases. Instead of using constrained optimisation, to avoid negative values, use the exponential.

```
diri.est <- function(x, type = 'mle') {
  ## x is the compositional data
  ## type indicates how to estimate parameters
  ## type = 'mle' means the classical mle case
  ## type = 'prec' means to use the precision parameter phi
  ## type = 'ent' means to use the entropy for the estimation

  x <- as.matrix(x)  ## makes sure x is a matrix
  x <- x/rowSums(x)  ## makes sure x is compositional data
  n <- dim(x)[1]  ## sample size
  z <- log(x)

  ## loglik is for the 'mle' type
  loglik <- function(param, z) {
     param <- exp(param)
    -( n * lgamma( sum(param) ) - n * sum( lgamma(param) ) +
     sum( z %*% (param - 1) ) )
  }

  ## diri is for the 'prec' type
```

```
diriphi <- function(param, z) {
 phi <- exp(param[1])
 b <- c(1, exp(param[-1]) )
 b <- b / sum(b)
 f <-  -( n * lgamma(phi) - n * sum( lgamma(phi * b) ) +
 sum( z %*% (phi * b - 1) ) )
 f
}

## entro is for the 'ent' type
entro <- function(param) {
 f <- numeric(length(param))
 param <- exp(param)
 for (i in 1:length(f)) {
   f[i] <- ma[i] - digamma(param[i]) + digamma( sum(param) )
 }
 f
}

if (type == 'mle') {
  runtime <- proc.time()
  oop <- options(warn = -1)
  on.exit(options(oop))
  da <- nlm(loglik, colMeans(x) * 10, z = z, iterlim = 10000)
  da <- nlm(loglik, da$estimate, z = z, iterlim = 10000)
  da <- nlm(loglik, da$estimate, z = z, iterlim = 10000)
  da <- optim(da$estimate, loglik, z = z, control = list(maxit = 2000),
  hessian = TRUE)

  runtime <- proc.time() - runtime
  result <- list( loglik = -da$value, param = exp(da$par),
  std = sqrt( diag( solve(da$hessian) ) ), runtime = runtime  )
}

if (type == 'prec') {
  runtime <- proc.time()
  oop <- options(warn = -1)
  on.exit(options(oop))
  da <- nlm(diriphi, c(10, colMeans(x)[-1]), z = z, iterlim = 2000)
```

```
    da <- nlm(diriphi, da$estimate, z = z, iterlim = 2000)
    da <- nlm(diriphi, da$estimate, z = z, iterlim = 2000, hessian = TRUE)
    da <- optim(da$estimate, diriphi, z = z, control = list(maxit = 3000),
    hessian = TRUE)
    phi <- exp(da$par[1])
    a <- c( 1, exp(da$par[-1]) )
    a <- a/sum(a)

    runtime <- proc.time() - runtime
    result <- list( loglik = -da$value, phi = phi, a = a,
    b = phi * a, runtime = runtime )
  }

  if (type == 'ent') {
    runtime <- proc.time()
    ## this requires the BB package
    ma <- colMeans(z)
    da <- BB::BBsolve(colMeans(x) * 10, entro, control =
    list(maxit = 20000, tol = 1e-10))
    da <- BBsolve( da$par, entro, control = list(maxit = 20000, tol = 1e-10) )
    da <- BBsolve( da$par, entro, control = list(maxit = 20000, tol = 1e-10) )
    da <- BBsolve( da$par, entro, control = list(maxit = 20000, tol = 1e-10) )
    param <- exp(da$par)
    lik <- n * lgamma( sum(param) ) - n * sum( lgamma(param) ) +
    sum( z %*% (param - 1) )

    runtime <- proc.time() - runtime
    result <- list( loglik = lik, param = param, runtime = runtime )
  }

  result
}
```

Let me now do the same calculations using the Newton-Raphson algorithm as suggested by Minka (2000). Apart from the algorithm for estimating the parameters of the Dirichlet-multinomial distribution (see §5.1.9), Minka (2000) described the whole Newton-Rapshon algorithm for the Dirichlet distribution, but we only need some lines from that. If you want to know more, read his technical report. All you need to know, the resume, is the final

Newton-Raphson algorithm given by

$$\boldsymbol{\alpha}^{new} = \boldsymbol{\alpha}^{old} - \frac{\mathbf{g} - \mathbf{J}_p b}{\mathbf{q}},$$

where

$$\begin{aligned} g_k &= n\psi\left(\sum_{i=k}^{p}\alpha_k\right) - n\psi\left(\alpha_k\right) + \sum_{i=1}^{n}\log x_{ik}, \;\; k = 1, \ldots, p \\ q_k &= -n\psi'\left(\alpha_k\right), \;\; k = 1, \ldots, p \\ b &= \frac{\sum_{j=1}^{p} g_j/q_j}{1/z + \sum_{j=1}^{p} 1/q_j} \quad \text{and} \quad z = n\psi'\left(\sum_{i=k}^{p}\alpha_k\right). \end{aligned}$$

The function $\psi\left(t\right) = \frac{\Gamma'(t)}{\Gamma(t)}$ is the digamma function and $\psi'\left(t\right)$ is its derivative. The $\mathbf{J}$ is the p-dimensional vector of ones, since we have p dimensions and n is the sample size of the compositional data $\mathbf{X}$. The initial value for the precision parameter, as suggested by Minka (2000) is given by

$$\phi' = \frac{\left(D-1\right)/2}{-\sum_{i=1}^{D}\frac{\bar{x}_i}{n}\frac{1}{n}\sum_{j=1}^{n}\log\frac{x_{ij}}{\bar{x}_i}}.$$

Hence, initial estimates for α are given by $\boldsymbol{\alpha}^{ini} = \phi'\left(\bar{x}_1, \ldots, \bar{x}_1\right)$. This makes the estimation a bit faster. I tried using the previous function (*diri.est*) with these initial values, but it was much slower(!!).

The Newton-Raphson algorithm was implemented in (8.12) and the results are equally fast. The truth, is that these two are exactly the same, the difference is that **??** did a very good job in simplifying the calculations to vectors only, i.e. no matrices are involved and hence no matrix inversions.

If you want to see more codes for the Dirichlet distributions check the S-plus/R Codes used in the book Dirichlet and Related Distributions: Theory, Methods and Applications by Ng et al. (2011). The only problem I have seen with this method is that if the data are concentrated around a point, say the center of the simplex, it will be hard for this and the previous methods to give estimates of the parameters. In this extremely difficult scenario I would suggest the use of the previous function with the precision parametrisation *diri.est(x, type = "prec")*. It will be extremely fast and accurate. Another option is the the function offered by the R package *VGAM vglm(x 1, dirichlet)* (Yee, 2010).

```
diri.nr <- function(x, type = 1, tol = 1e-07) {
  ## x is compositional data
  ## x can be either "lik" or "ent"
  if (type == 1) {
```

```
  runtime <- proc.time()
  x <- as.matrix(x)  ## makes sure x is a matrix
  x <- x / Rfast::rowsums(x)  ## makes sure x is compositional data
  n <- dim(x)[1]  ## the sample size
  p <- dim(x)[2]  ## dimensionality
  m <- Rfast::colmeans(x)
  zx <- t( log(x) )
  down <-  - sum( m * ( Rfast::rowmeans( zx ) - log(m) ) )
  sa <- 0.5 * (p - 1) / down  ## initial value for precision
  a1 <- sa * m  ## initial values
  gm <- Rfast::rowsums(zx)
  z <- n * digamma( sa )
  g <- z - n * digamma(a1) + gm
  qk <-  - n * trigamma(a1)
  b <- ( sum(g / qk) ) / ( 1/z - sum(1 / qk) )
  a2 <- a1 - (g - b)/qk
  i <- 2

  while( sum( abs( a2 - a1 ) ) > tol ) {
    i <- i + 1
    a1 <- a2
    z <- n * digamma( sum(a1) )
    g <- z - n * digamma(a1) + gm
    qk <-  - n * trigamma(a1)
    b <- sum(g / qk) / ( 1/z - sum(1 / qk) )
    a2 <- a1 - (g - b) / qk
  }

  loglik <- n * lgamma( sum(a2) ) - n * sum( lgamma(a2) ) +
    sum( zx * (a2 - 1) )
  runtime <- proc.time() - runtime

} else if (type == 2) {

 runtime <- proc.time()
  x <- as.matrix(x)  ## makes sure x is a matrix
  x <- x / Rfast::rowsums(x)  ## makes sure x is compositional data
  n <- dim(x)[1]  ## sample size
```

```
    p <- dim(x)[2]
    zx <- t( log(x) )
    ma <- Rfast::rowmeans(zx)
    m <- Rfast::colmeans(x)
    down <-  - sum( m * ( ma - log(m) ) )
    sa <- 0.5 * (p - 1) / down  ## initial value for precision
    a1 <- sa * m  ## initial values
    f <- ma - digamma(a1) + digamma( sa )
    f2 <- matrix( trigamma(sa), p, p)
    diag(f2) <- diag(f2) - trigamma(a1)
    a2 <- a1 - solve(f2, f)
    i <- 2

    while ( sum( abs( a2 - a1 ) ) > tol ) {
      a1 <- a2
      i <- i + 1
      sa <- sum( a1)
      f <- ma - digamma(a1) + digamma( sa )
      f2 <- matrix( trigamma(sa), p, p)
      diag(f2) <- diag(f2) - trigamma(a1)
      a2 <- a1 - solve(f2, f)

    }
    loglik <- n * lgamma( sum(a2) ) - n * sum( lgamma(a2) ) +
              sum( zx * (a2 - 1) )
    runtime <- proc.time() - runtime
  }
  if ( is.null(colnames(x)) ) {
    names(a2) <- paste("X", 1:p, sep = "")
  } else  names(a2) <- colnames(x)

  list(iter = i, loglik = loglik, param = a2, runtime = runtime)
}
```

8.3.2 Symmetric Dirichlet distribution

The symmetric Dirichlet distribution arises when all of its parameters are equal. To test this assertion we will use the log-likelihood ratio test statistic. The relevant R code is given below

```
sym.test <- function(x) {
  ## x contains the data
  n <- dim(x)[1]  ## the sample size
  D <- dim(x)[2]  ## the dimensionality of the data
  zx <- log(x)

   sym <- function(a, zx) {
    n * lgamma(D * a) - n * D * lgamma(a) +
    sum( zx * (a - 1) )
   }

  t0 <- optimize(sym, c(0, 1000), zx = zx, maximum = TRUE)
  t1 <- diri.nr(x)
  a0 <- t0$maximum
  a1 <- t1$param
  h1 <- t1$loglik
  h0 <- as.numeric(t0$objective)
  test <- 2 * (h1 - h0)
  pvalue <- pchisq(test, D - 1, lower.tail = FALSE)
  if ( is.null(colnames(x)) ) {
    names(a1) <- paste("X", 1:D, sep = "")
  } else  names(a1) <- colnames(x)
  res <- c(h1, h0, test, pvalue, D - 1)
  names(res) <- c('loglik1', 'loglik0', 'test', 'pvalue', 'df')
  list(est.par = a1, one.par = a0, res = res )
}
```

8.3.3 Kullback-Leibler divergence and Bhattacharyya distance between two Dirichlet distributions

We show a function to calculate the Kullback-Leibler divergence between two Dirichlet distributions. The proof of the Kullback-Leibler divergence between $Dir(a)$ and $Dir(b)$ is available at this technical report. This divergence is equal to

$$KL(D_1(a) \parallel D_2(b)) = \sum_{i=1}^{D}(a_i - b_i)[\Psi(a_i) - \Psi(a_0)] + \sum_{i=1}^{D}\log\frac{\Gamma(b_i)}{\Gamma(a_i)} + \log\frac{\Gamma(a_0)}{\Gamma(b_0)},$$

where $a_0 = \sum_{i=1}^{D} a_i$, $b_0 = \sum_{i=1}^{D} b_i$ and $\Psi(.)$ is the digamma function.

In Rauber et al. (2008) is mentioned that the the Kullback-Leibler divergence is inappropriate as a divergence since it is not defined when there is a zero value. For this reason

we will give below the code to calculate the Bhattacharyya ditance between two Dirichlet distributions. The Bhattacharyya distance between two Dirichlet distributions is defined as

$$\begin{aligned} J_B\left(D_1\left(a\right), D_2\left(b\right)\right) &= \log\Gamma\left(\sum_{i=1}^{D}\frac{a_i+b_i}{2}\right)+\frac{1}{2}\sum_{i=1}^{D}\left[\log\Gamma\left(a_i\right)+\log\Gamma\left(b_i\right)\right] \\ &\quad -\sum_{i=1}^{D}\log\Gamma\left(\frac{a_i+b_i}{2}\right)-\frac{1}{2}\left[\log\Gamma\left(\sum_{i=1}^{D}a_i\right)+\log\Gamma\left(\sum_{i=1}^{D}b_i\right)\right] \end{aligned} \tag{8.13}$$

The code to calculate (8.13) is given below

```
kl.diri <- function(a, b, type = "KL") {
  ## a and b are the two vectors of parameters of the two Dirichlets
  ## if type == "KL" the KL-Divergence between Dir(a) and Dir(b) is calculated
  ## if type == "bhatt" the Bhattacharyya distance between Dir(a) and
  ## Dir(b) is calcualted
  if (type == "KL") {
    a0 <- sum(a)
    b0 <- sum(b)
    f <- sum( (a - b) * (digamma(a) - digamma(a0)) ) + sum(lgamma(b) -
    lgamma(a) ) + lgamma(a0) - lgamma(b0)
  } else {
    f <- lgamma(0.5 * sum(a + b)) + 0.5 * sum(lgamma(a) + lgamma(b)) -
sum(lgamma(0.5 * (a + b))) - 0.5 * (lgamma(sum(a)) + lgamma(sum(b)))
  }
  f
}
```

8.4 Contour plots of distributions on $\mathbb{S}^2$

In section 8.1.1 we showed how construct a ternary plot by making use of a matrix (8.2). In this case, we need to do the opposite. The contour plot presented here needs parameter values. The idea is the same as in Section 5.3.1.

8.4.1 Contour plot of the Dirichlet distribution

What the user has to do is to fit a parametric model (Dirichlet distributions for example, or the normal, t or skew normal distribution in the log-ratio transformed data) and estimate the parameters. Then add a couple of extra lines to all the next functions where he plots his compositional data.

We take a grid of points in $\mathbb{R}^2$ and see if it lies within the triangle (or the ternary plot seen in (8.1.1)). If it lies, then it comes from a composition. To find the composition we need to work out the opposite of (8.3). The coordinates of a compositional vector in $\mathbb{R}^2$ taken from (8.3) are

$$(y_1, y_2) = \left(x_2 + \frac{x_3}{2}, \frac{x_3\sqrt{3}}{2} \right).$$

We have the pair (y_1, y_2) and want to calculate (x_1, x_2, x_3) at first. The result is

$$\left\{ \begin{array}{rc} x_3 = & \frac{2y_2}{\sqrt{3}} \\ x_2 = & y_1 - \frac{y_2}{\sqrt{3}} \\ x_1 = & 1 - x_2 - x_3 \end{array} \right\}$$

Thus $(x_1, x_2, x_3) \in \mathbb{S}^2$ when (y_1, y_2) fall within the interior of the triangle. If you plot the ternary plot from section 8.1.1 you will see that the top of the triangle is located at $\left(0.5, \frac{\sqrt{3}}{2}\right)$ and the other two vertices are located at $(0, 0)$ and $(1, 0)$ given in (8.2). Thus, the three lines which define the triangle are

$$\begin{aligned} y_2 &= 0 \text{ with } 0 \leq y_1 \leq 1 \\ y_2 &= \sqrt{3}y_1 \text{ with } 0 \leq y_1 \leq 0.5 \\ y_2 &= \sqrt{3} - \sqrt{3}y_1 \text{ with } 0.5 \leq y_1 \leq 1. \end{aligned}$$

Thus, only the points inside the interior of the triangle come from a composition. Once we have calculated (x_1, x_2, x_3) from the pair of y_s which lie inside the interior of the triangle we will plug them in (8.9). In this way we will calculate the density of the Dirichlet with some given parameter (estimated or not) at that point. We will do this for all points and in the end we will plot the contour lines along with the triangle. There is the option to plot the data as well. The code is given below.

```
diri.contour <- function(a, n = 100, x = NULL) {
  ## a are the estimated Dirichlet parameters
  ## n shows the number of points at which the density is calculated
  ## so, n^2 points are used.
  ## x should be a 3-part compositional data or NULL for no data
  x1 <- seq(0.001, 0.999, length = n)  ## coordinates of x
  sqrt3 <- sqrt(3)
  x2 <- seq(0.001, sqrt3/2 - 1e-03, length = n)  ## coordinates of y
  mat <- matrix(nrow = n, ncol = n)
  beta <- prod( gamma(a)) / gamma(sum(a) )  ## beta function
```

```
for ( i in 1:c(n/2) ) {
  for (j in 1:n) {
    if ( x2[j] < sqrt3 * x1[i] ) {
      ## This checks if the point will lie inside the triangle
      ## the next three lines invert the points which lie inside
      ## the triangle back into the composition in S^2
      w3 <- 2 * x2[j]/sqrt3
      w2 <- x1[i] - x2[j]/sqrt3
      w1 <- 1 - w2 - w3
      w <- c(w1, w2, w3)
      can <- (1 / beta) * prod( w^(a - 1) )
      if (abs(can) < Inf)  mat[i, j] <- can  else  mat[i, j] <- NA
    } else  mat[i, j] <- NA
  }
}
for (i in c(n/2 + 1):n) {
  for (j in 1:n) {
    ## This checks if the point will lie inside the triangle
    if ( x2[j] < sqrt3 - sqrt3 * x1[i] ) {
      ## the next three lines invert the points which lie inside
      ## the triangle back into the composition in S^2
      w3 <- 2 * x2[j]/sqrt3
      w2 <- x1[i] - x2[j]/sqrt3
      w1 <- 1 - w2 - w3
      w <- round(c(w1, w2, w3), 6)
      can <- (1 / beta) * prod(w^(a - 1))
      if (abs(can) < Inf)  mat[i, j] <- can  else  mat[i, j] <- NA
    } else  mat[i, j] <- NA
  }
}

contour(x1, x2, mat, col = 3)  ## contour plots
b1 <- c(1/2, 0, 1, 1/2)
b2 <- c(sqrt3/2, 0, 0, sqrt3/2)
b <- cbind(b1, b2)
## the next line draws the triangle in the two dimensions
points(b[, 1], b[, 2], type = "l", xlab = " ", ylab = " ")
```

```
  if ( !is.null(x) ) {
    x <- as.matrix(x)
    x <- x / Rfast::rowsums(x)
    proj <- matrix(c(0, 1, 1/2, 0, 0, sqrt3/2), ncol = 2)
    xa <- x %*% proj
    points(xa[, 1], xa[, 2])
  }

}
```

8.4.2 Contour plot of the normal distribution in $\mathbb{S}^2$

The density of the multivariate normal is

$$f(\mathbf{y}) = \frac{e^{-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{y}-\boldsymbol{\mu})}}{|2\pi\boldsymbol{\Sigma}|^{1/2}} \tag{8.14}$$

We will repeat Section 8.4.1 with the only difference that we will give the code for the contour plot of the bivariate multivariate normal distribution. The idea is the same, we choose a grid of points and for each pair of points we see whether it falls within the triangle. If yes, we calculated the density of the bivariate normal at that point by plugging it at (8.14). There is the option to make the data appear or not.

```
norm.contour <- function(x, type = "alr", n = 100, appear = "TRUE") {
  ## the type parameter determines whether the additive or
  ## the isometric log-ratio transformation will be used.
  ## If type='alr' (the default) the additive
  ## log-ratio transformation is used.
  ## If type='ilr', the isometric log-ratio is used
  ## n is the number of points of each axis used
  x <- as.matrix(x)
  x <- x / rowSums(x)
  x1 <- seq(0.001, 0.999, length = n)
  sqrt3 <- sqrt(3)
  x2 <- seq(0.001, sqrt3/2 - 0.001, length = n)
  mat <- matrix(NA, nrow = n, ncol = n)
  ha <- t( helm(3) )

  if (type == "alr") {
    ya <- log( x[, -3] / x[, 3] )
```

```
} else {
  ya <- log(x)
  ya <- ya - Rfast::rowmeans( ya )
  ya <- as.matrix( ya %*% ha )
}

m <- colMeans(ya)  ## mean vector
s <- var(ya)  ## covariance matrix
down <- det(2 * pi * s)^(-0.5)
st <- solve(s)

for ( i in 1:c(n/2) ) {
  for ( j in 1:n ) {
    if ( x2[j] < sqrt3 * x1[i] ) {
      ## This checks if the point will lie inside the triangle
      ## The next 4 lines calculate the composition
      w3 <- 2 * x2[j] / sqrt3
      w2 <- x1[i] - x2[j] / sqrt3
      w1 <- 1 - w2 - w3
      w <- c(w1, w2, w3)
      if (type == "alr") {
        y <- log( w[-3] / w[3] )  ## additive log-ratio transformation
      } else {
        y <- log(w) - mean( log(w) )
        y <- as.vector( y %*% ha )
      }  ## isometric log-ratio transformation
      can <- down * exp( -0.5 * ( ( y - m ) %*% st %*% ( y - m ) ) )
      if (abs(can) < Inf)  mat[i, j] <- can
    }
  }
}

for ( i in c(n/2 + 1):n ) {
  for ( j in 1:n ) {
    ## This checks if the point will lie inside the triangle
    if ( x2[j] < sqrt3 - sqrt3 * x1[i] ) {
      ## The next 4 lines calculate the composition
      w3 <- 2 * x2[j] / sqrt3
      w2 <- x1[i] - x2[j] / sqrt3
```

```
        w1 <- 1 - w2 - w3
        w <- c(w1, w2, w3)
        if (type == "alr") {
          y <- log( w[-3] / w[3] )  ## additive log-ratio transformation
        } else {
          y <- log(w) - mean( log(w) )
          y <- as.vector( y %*% ha )
        }  ## isometric log-ratio transformation
        can <- down * exp( -0.5 * ( ( y - m ) %*% st %*% ( y - m ) ) )
        if (abs(can) < Inf)  mat[i, j] <- can
      }
    }
  }

  contour( x1, x2, mat, nlevels = 7, col = 3, pty = "s", xaxt = "n",
  yaxt = "n", bty = "n" )
  b1 <- c( 1/2, 0, 1, 1/2 )
  b2 <- c( sqrt3/2, 0, 0, sqrt3/2 )
  b <- cbind(b1 ,b2)
  points( b[, 1], b[, 2], type = "l", xlab = " ", ylab = " " )
  nam <- colnames(x)
  if ( is.null(nam) )  nam <- paste("X", 1:3, sep = "")
  text( b[1, 1], b[1, 2] + 0.01, nam[3], cex = 1)
  text( b[2:3, 1] + 0.01, b[2:3, 2] - 0.01, nam[1:2], cex = 1)

  if ( appear ) {
    proj <- matrix(c(0, 1, 1/2, 0, 0, sqrt(3)/2), ncol = 2)
    x <- as.matrix(x)
    x <- x/rowSums(x)
    xa <- x %*% proj
    points(xa[, 1], xa[, 2])
  }

}
```

8.4.3 Contour plot of the multivariate t distribution in $\mathbb{S}^2$

The density of the multivariate t distribution is given in (5.1). After applying the additive log-ratio (8.7) or the isometric log-ratio transformation (8.8) to the compositional data we

can estimate the parameters of the multivariate t distribution via numerical optimization. In Section 5.1.3 we provided a function to perform this task.

The way to produce a contour plot of the bivariate t distribution on the simplex is similar to the normal distribution. The code is given below. There is the option to make the data appear or not.

```
bivt.contour <- function(x, type = 'alr', n = 100, appear = TRUE) {
  ## x is the compositional data
  ## the type parameter determines whether the additive
  ## or the isometric log-ratio transformation will be used.
  ## If type='alr' (the default) the additive
  ## log-ratio transformation is used.
  ## If type='ilr', the isometric log-ratio is used
  ## n is the number of points of each axis used
  x <- as.matrix(x)
  x <- x / Rfast::rowsums(x)
  sqrt3 <- sqrt(3)

  if (type == 'alr') {
    y <- log( x[, -3] / x[, 3] ) ## additive log-ratio transformation
  } else {
    ha <- t( helm(3) )
    y <- log(x)
    y <- y - Rfast::rowmeans( y )
    y <- as.matrix( y %*% ha )
  }

  mod <- multivt(y)
  m <- mod$center
  s <- mod$scatter
  v <- mod$df
  p <- 2
  x1 <- seq(0.001, 0.999, length = n)
  x2 <- seq(0.001, sqrt3/2 - 0.001, length = n)
  mat <- matrix(nrow = n, ncol = n)
  st <- solve(s)

  for (i in 1:c(n/2) ) {
    for (j in 1:n) {
```

```
    if (x2[j] < sqrt3 * x1[i]) { ## This checks if the point lies
    ## inside the triangle
    ## The next 4 lines calculate the composition
      w3 <- 2 * x2[j] / sqrt3
      w2 <- x1[i] - x2[j] / sqrt3
      w1 <- 1 - w2 - w3
      w <- c(w1, w2, w3)

      if (type == 'alr') {
    y <- log(w[-3] / w[3]) ## additive log-ratio transformation
      } else {
        y <- log(w) - mean(log(w))
        y <- as.vector( y %*% ha )
      }  ## isometric log-ratio transformation

      ca <- lgamma( (v + p)/2 ) - lgamma(v/2) - 0.5 * log( det(pi * v * s) )-
      0.5 * (v + p) * ( log1p( ( y - m ) %*% st %*% ( y - m) )/v )
      can <- exp(ca)
     if (abs(can) < Inf)  mat[i, j] <- can

}
  }
}

for (i in c(n/2 + 1):n) {
  for (j in  1:n) {

    ## This checks if the point will lie inside the triangle
    if (x2[j] < sqrt3 - sqrt3 * x1[i]) {
      ## The next 4 lines calculate the composition
      w3 <- 2 * x2[j] / sqrt3
      w2 <- x1[i] - x2[j] / sqrt3
      w1 <- 1 - w2 - w3
      w <- c(w1, w2, w3)

      if (type == 'alr') {
      y <- log(w[-3]/w[3]) ## additive log-ratio transformation
      } else  {
        y <- log(w) - mean(log(w))
```

```
          y <- as.vector( y %*% ha )
        }  ## isometric log-ratio transformation

        ca <- lgamma( (v + p)/2 ) - lgamma(v/2) - 0.5 * log( det(pi * v * s) )-
        0.5 * (v + p) * ( log1p( ( y - m ) %*% st %*% ( y - m) )/v )
        can <- exp(ca)
        if (abs(can) < Inf)  mat[i, j] <- can
      }
    }
  }

  contour(x1, x2, mat, nlevels = 7, col = 3, pty = "s", xaxt = "n",
  yaxt = "n", bty = "n")
  b1 <- c(0.5, 0, 1, 0.5)
  b2 <- c(sqrt3/2, 0, 0, sqrt3/2)
  b <- cbind(b1, b2)
  points(b[, 1], b[, 2], type = "l", xlab = " ", ylab = " ")

  if ( appear ){
    nam <- colnames(x)
    if ( is.null(nam) )  nam <- paste("X", 1:3, sep = "")
    text(b[1, 1], b[1, 2] + 0.02, nam[3], cex = 1)
    text(b[2:3, 1], b[2:3, 2] - 0.02, nam[1:2], cex = 1)
    proj <- matrix(c(0, 1, 1/2, 0, 0, sqrt3/2), ncol = 2)
    x <- as.matrix(x)   ;   x <- x/rowSums(x)
    xa <- x %*% proj
    points(xa[, 1], xa[, 2])
  }

}
```

8.4.4 Contour plot of the skew-normal distribution in $\mathbb{S}^2$

In order to fit the skew-normal distribution (5.9) to a compositional dataset we first apply either the additive log-ratio (8.7) or the isometric log-ratio transformation (8.8). Using the transformed data we need to estimate the parameters of the skew-normal distribution.

The code to produce a contour plot for the bivariate skew-normal distribution on the simplex is given below. There is also the option to make the data appear or not.

```
skewnorm.contour <- function(x, type = 'alr', n = 100, appear = FALSE) {
```

```
 ## the type parameter determines whether the additive
 ## or the isometric log-ratio transformation will be used.
 ## If type='alr' (the default) the additive log-ratio transformation is used.
 ## If type='ilr', the isometric log-ratio is used
 ## n is the number of points of each axis used
 x <- as.matrix(x)
 x <- x / rowSums(x)
 ha <- t( helm(3) )

 if (type == "alr") {
   ya <- log( x[, -3] / x[, 3] )
 } else {
   ya <- log(x)
   ya <- ya - Rfast::rowmeans( ya )
   ya <- as.matrix( ya %*% ha )
 }
 sqrt3 <- sqrt(3)
 mod <- sn::msn.mle(y = ya)
 param <- mod$dp
 x1 <- seq(0.001, 0.999, length = n)
 x2 <- seq(0.001, sqrt3/2 - 0.001, length = n)
 mat <- matrix(nrow = n, ncol = n)

 for ( i in 1:c(n/2) ) {
  for ( j in 1:n ) {
    ## This checks if the point lies inside the triangle
   if ( x2[j] < sqrt3 * x1[i] ) {
    ## The next 4 lines calculate the composition
    w3 <- 2 * x2[j] / sqrt3
    w2 <- x1[i] - x2[j] / sqrt3
    w1 <- 1 - w2 - w3
    w <- c(w1, w2, w3)
    if (type == 'alr') {
  y <- log(w[-3] / w[3])  ## additive log-ratio transformation
    } else {
      y <- log(w) - mean(log(w))
      y <- as.vector( y %*% ha )
}  ## isometric log-ratio transformation
    can <- sn::dmsn(y, dp = param)
```

```
    if ( abs(can) < Inf )  mat[i, j] <- can
   }
  }
 }

 for ( i in c(n/2+1):n ) {
  for ( j in  1:n )       {
  ## This checks if the point will lie inside the triangle
   if ( x2[j] < sqrt3 - sqrt3 * x1[i] ) {
    ## The next 4 lines calculate the composition
    w3 <- 2 * x2[j] / sqrt3
    w2 <- x1[i] - x2[j] / sqrt3
    w1 <- 1 - w2 - w3
    w <- c(w1, w2, w3)
    if (type == 'alr') {
  y <- log(w[-3] / w[3])  ## additive log-ratio transformation
    } else {
      y <- log(w) - mean(log(w))
      y <- as.vector( y %*% ha )
 }  ## isometric log-ratio transformation
    can <- sn::dmsn(y, dp = param)
   if ( abs(can) < Inf )  mat[i, j] <- can
}
  }
 }

 contour(x1, x2, mat, nlevels = 7, col = 3, pty = "s", xaxt = "n",
 yaxt = "n", bty = "n")
 b1 <- c(1/2, 0, 1, 1/2)
 b2 <- c(sqrt3/2, 0, 0, sqrt3/2)
 b <- cbind(b1, b2)
 points(b[, 1], b[, 2], type = "l", xlab = " ", ylab = " ")

 if ( appear ) {
  nam <- colnames(x)
  if ( is.null(nam) )  nam <- paste("X", 1:3, sep = "")
  text(b[1, 1], b[1, 2] + 0.02, nam[3], cex = 1)
  text(b[2:3, 1], b[2:3, 2] - 0.02, nam[1:2], cex = 1)
  proj <- matrix(c(0, 1, 1/2, 0, 0, sqrt3/2), ncol = 2)
```

```
   x <- as.matrix(x) ; x = x/rowSums(x)
   xa <- x %*% proj
   points(xa[, 1], xa[, 2])
  }

}
```

8.4.5 Contour plot of a normal mixture model in $\mathbb{S}^2$

We need again the R package *mixture* (Browne et al., 2015) for the contour plots.

```
mixnorm.contour <- function(x, mod) {
  ## mod is a mixture model containing all the parameters
  ## the type parameter determines whether the additive or the isometric
  ## log-ratio transformation will be used. If type='alr' (the default) the
  ## additive log-ratio transformation is used. If type='ilr', the isometric
  ## log-ratio is used

  x <- as.matrix(x)  ## makes sure x is matrix
  x <- x/rowSums(x)  ## make sure x compositional data
  prob <- mod$prob  ## mixing probabilitiy of each cluster
  mu <- mod$mu
  su <- mod$su
  type <- mod$type  ## the type of the log-ratio transformation, "alr" or "ilr"
  g <- length(mod$prob)  ## how many clusters are there
  n <- 100  ## n is the number of points of each axis used
  sqrt3 <- sqrt(3)
  x1 <- seq(0.001, 0.999, length = n)
  x2 <- seq(0.001, sqrt3/2 - 0.001, length = n)
  mat <- matrix(nrow = n, ncol = n)
  ha <- t( helm(3) )

  ldet <- numeric(g)
  for (k in 1:g) {
    ldet[k] <-  -0.5 * log(det(2 * pi * su[, , k]))
  }

  for ( i in 1:c(n/2) ) {
    for (j in 1:n) {
      if ( x2[j] < sqrt3 * x1[i] ) {
```

```
      ## This checks if the point will lie inside the triangle
      ## The next 4 lines calculate the composition
      w3 <- 2 * x2[j] / sqrt3
      w2 <- x1[i] - x2[j] / sqrt3
      w1 <- 1 - w2 - w3
      w <- c(w1, w2, w3)
      if (type == "alr") y <- log( w[-3]/w[3] )  ## alr transformation
      if (type == "ilr") {  ## isometric log-ratio transformation
        y <- log(w) - mean(log(w))
        y <- as.vector( y %*% ha )
      }
      ta <- numeric(g)
      for (k in 1:g) {
        ta[k] <-  ldet[k] - 0.5 * mahalanobis(y, mu[k, ], su[, , k])
      }
      can <- sum( prob * exp(ta) )
      if (abs(can) < Inf)   mat[i, j] <- can
    }
  }
}

for ( i in c(n/2 + 1):n ) {
  for ( j in 1:n ) {
    ## This checks if the point will lie inside the triangle
    if ( x2[j] < sqrt3 - sqrt3 * x1[i] ) {
      ## The next 4 lines calculate the composition
      w3 <- 2 * x2[j] / sqrt3
      w2 <- x1[i] - x2[j] / sqrt3
      w1 <- 1 - w2 - w3
      w <- c(w1, w2, w3)
      if (type == "alr") y <- log( w[-3]/w[3] )  ## alr transformation
      if (type == "ilr") {  ## isometric log-ratio transformation
        y <- log(w) - mean(log(w))
        y <- as.vector( y %*% ha )
      }
      ta <- numeric(g)
      for (k in 1:g) {
        ta[k] <-  ldet[k] - 0.5 * mahalanobis(y, mu[k, ], su[, , k])
      }
```

```
        can <- sum( prob * exp(ta) )
        if ( abs(can) < Inf )  mat[i, j] <- can
      }
    }
  }

  contour( x1, x2, mat, nlevels = 7, col = 3, pty = "s", xaxt = "n",
  yaxt = "n", bty = "n" )
  b1 <- c(1/2, 0, 1, 1/2)
  b2 <- c( sqrt3/2, 0, 0, sqrt3/2 )
  b <- cbind(b1, b2)
  points(b[ , 1], b[ , 2] , type = "l", xlab = " ", ylab = " ")

  nam <- colnames(x)
  if ( is.null(nam) )  nam <- paste("X", 1:3, sep = "")
  text( b[1, 1], b[1, 2] + 0.02, nam[3], cex = 1 )
  text( b[2:3, 1], b[2:3, 2] - 0.02, nam[1:2], cex = 1 )
  proj <- matrix(c(0, 1, 1/2, 0, 0, sqrt3/2), ncol = 2)
  xa <- x %*% proj
  points(xa[, 1], xa[, 2])

}
```

8.4.6 Contour plot of a kernel density estimation in $\mathbb{S}^2$

The idea is the same as before, but a bit different now. Instead of providing the parameters of a distribution, the user provides the dataset itself and decides whether the additive or the isometric log-ratio transformation is to be used. Then, the best bandwidth parameter h is obtained via tuning (see Section 5.1.6)and then for a grid of points the kernel density estimation takes place. The difference with the previous contour plots, is that now we can see the data plotted on the simplex, along with the contours of the kernel density estimate.

```
comp.kerncontour <- function(x, type = "alr", n = 100) {
  ## x contains the compositional data
  ## type determines which log-ratio transformation will be used.
  ## If type='alr' (the default) the additive
  ## log-ratio transformation is used.
  ## If type='ilr', the isometric log-ratio is used
  ## n is the number of points of each axis used
```

```
  x <- as.matrix(x)
  x <- x / Rfast::rowsums(x)  ## makes sure x is compositional data
  nu <- dim(x)[1]  ## sample size
  sqrt3 <- sqrt(3)
  ha <- t( helm(3) )

  if (type == "alr")  z <- log(x[, -3]/x[, 3])  ## alr transformation
  if (type == "ilr") {  ## isometric log-ratio transformation
      zx <- log(x)
      z <- zx - Rfast::rowmeans( zx )
      z <- z %*% ha
  }

  hopt <- mkde.tune(z)$hopt
  con <- hopt^2
  ts <- diag( hopt^2, 2 )
  x1 <- seq(0.001, 0.999, length = n)
  x2 <- seq(0.001, sqrt3/2 - 0.001, length = n)
  mat <- matrix(nrow = n, ncol = n)

  for ( i in 1:c(n/2) ) {
    for ( j in 1:n ) {
      if (x2[j] < sqrt3 * x1[i]) {
        ## This checks if the point will lie inside the triangle
        ## The next 4 lines calculate the composition
        w3 <- 2 * x2[j] / sqrt3
        w2 <- x1[i] - x2[j]/sqrt3
        w1 <- 1 - w2 - w3
        w <- c(w1, w2, w3)
        if (type == "alr")  y <- log(w[-3]/w[3])  ## alr transformation
        if (type == "ilr") {  ## isometric log-ratio transformation
            y <- log(w) - mean(log(w))
            y <- as.vector(y %*% ha )
        }
a <- Rfast::mahala(z, y, ts)
        can <- 1/(2 * pi) * (1/con) * sum( exp(-0.5 * a) )/nu
        if ( abs(can) < Inf )   mat[i, j] <- can

      }
```

```
    }
  }

  for ( i in c(n/2 + 1):n ) {
    for ( j in 1:n ) {
      ## This checks if the point will lie inside the triangle
      if (x2[j] < sqrt3 - sqrt3 * x1[i]) {
        ## The next 4 lines calculate the composition
        w3 <- 2 * x2[j]/sqrt3
        w2 <- x1[i] - x2[j]/sqrt3
        w1 <- 1 - w2 - w3
        w <- c(w1, w2, w3)
        if (type == "alr") y <- log(w[-3]/w[3])  ## alr transformation
        if (type == "ilr") {  ## isometric log-ratio transformation
          y <- log(w) - mean(log(w))
          y <- as.vector( y %*% ha )
        }
    a <- Rfast::mahala(z, y, ts)
        can <- 1/(2 * pi) * (1/con) * sum( exp(-0.5 * a) )/nu
        if (abs(can) < Inf)  mat[i, j] <- can
      }
    }
  }

  contour( x1, x2, mat, nlevels = 7, col = 3, pty = "s", xaxt = "n",
  yaxt = "n", bty = "n" )
  proj <- matrix(c(0, 1, 1/2, 0, 0, sqrt3/2), ncol = 2)
  da <- x %*% proj
  points(da[, 1], da[, 2])
  b1 <- c(1/2, 0, 1, 1/2)
  b2 <- c(sqrt3/2, 0, 0, sqrt3/2)
  b <- cbind(b1, b2)
  points(b[, 1], b[, 2], type = "l", xlab = " ", ylab = " ")
  nam <- colnames(x)
  if ( is.null(nam) )  nam <- paste("X", 1:3, sep = "")
  text( b[1, 1], b[1, 2] + 0.02, nam[3], cex = 1 )
  text( b[2:3, 1], b[2:3, 2] - 0.02, nam[1:2], cex = 1 )

}
```

8.5 The α-transformation for compositional data

This Section is about a recently suggested Box-Cox type power transformation for compositional data termed the α-transformation (Tsagris et al., 2011). Discriminant analysis and regression using this transformation will be covered, but I decided to put everything related to the α-transformation in one Section.

8.5.1 The α-transformation

The α-transformation of a compositional vector $\mathbf{x} \in \mathbb{S}^d$ (Tsagris et al., 2011) is defined by

$$\mathbf{z}_\alpha(\mathbf{x}) = \mathbf{H} \cdot \left(\frac{D\,\mathbf{u}_\alpha(\mathbf{x}) - \mathbf{1}_D}{\alpha} \right), \tag{8.15}$$

where

$$\mathbf{u}_\alpha(\mathbf{x}) = \left(\frac{x_1^\alpha}{\sum_{j=1}^D x_j^\alpha}, \ldots, \frac{x_D^\alpha}{\sum_{j=1}^D x_j^\alpha} \right)^T \tag{8.16}$$

is the compositional power transformation (Aitchison, 2003), $\mathbf{1}_D$ is the D-dimensional vector of ones, and $\mathbf{H}$ is any d-by-D matrix consisting of orthonormal rows, each of which is orthogonal to $\mathbf{1}_D$. A common choice of $\mathbf{H}$ is the Helmert sub-matrix (see (6.2)). The purpose of $\mathbf{H}$ is to remove the redundant dimension which is present due to the compositional constraint. In particular, the vector $\left(D\,\mathbf{u}_\alpha(\mathbf{x}) - \mathbf{1}_D\right)/\alpha$ has components which sum to zero and therefore it lies in a subspace of $\mathbb{R}^D$; left-multiplication by $\mathbf{H}$ is an isometric one-to-one mapping from this subspace into $\mathbb{R}^d$. This orthonormal matrix has been used by Egozcue et al. (2003).

The Jacobian determinant of the α-transformation is

$$|\mathbf{J}| = D^d \prod_{i=1}^D \frac{x_i^{\alpha-1}}{\sum_{j=1}^D x_j^\alpha}. \tag{8.17}$$

The image $\mathcal{V}_\alpha = \left\{ \mathbf{z}_\alpha(\mathbf{x}) : \mathbf{x} \in \mathbb{S}^d \right\}$ of transformation (8.15) is $\mathbb{R}^d$ in the limit $\alpha \to 0$ but a strict subset of $\mathbb{R}^d$ for $\alpha \neq 0$. Transformation (8.15) is invertible: for $\mathbf{v} \in \mathcal{V}_\alpha$ the inverse of $\mathbf{z}_\alpha(\mathbf{x})$ is

$$\mathbf{z}_\alpha^{-1}(\mathbf{v}) = \mathbf{u}_\alpha^{-1}\left(\alpha \mathbf{H}^\top \mathbf{v} + \mathbf{1}_D \right) \in \mathbb{S}^d, \tag{8.18}$$

where

$$\mathbf{u}_\alpha^{-1}(\mathbf{x}) = \left(\frac{x_1^{1/\alpha}}{\sum_{j=1}^D x_j^{1/\alpha}}, \ldots, \frac{x_D^{1/\alpha}}{\sum_{j=1}^D x_j^{1/\alpha}} \right).$$

If one is willing to exclude from the sample space the boundary of the simplex, which corresponds to observations that have one or more components equal to zero, then the α-transformation (8.15) and its inverse (8.18) are well defined for all $\alpha \in \mathbb{R}$. (Excluding the boundary is standard practise in LRA because the definition is used to sidestep the problem of having data with zeros.) The motivation for the α-transformation (8.15) is that the case $\alpha = 0$ corresponds to LRA, since at the limit as $\alpha \to 0$, (8.15) tends to (8.8). The case of $\alpha = 1$ corresponds to a analysing compositional data as if they were Euclidean (Baxter, 2001, Baxter et al., 2005, Woronow, 1997). In this case $\alpha = 1$, (8.15) is just a linear transformation of the simplex $\mathbb{S}^d$. Thus, (8.15) is a more general transformation than the isometric (or the centred) log-ratio one.

Power transformations similar to (8.15) were considered by Greenacre (2009) and Greenacre (2011), in the somewhat different context of correspondence analysis. A Box–Cox transformation applied to each component of $\mathbf{x} \in \mathbb{S}^d$ so that $\mathbf{x}$ is transformed to

$$\left[\theta^{-1}\left(x_1^{\theta} - 1\right), \ldots, \theta^{-1}\left(x_D^{\theta} - 1\right)\right]^T, \tag{8.19}$$

has the limit $(\log x_1, \ldots, \log x_D)^T$ as $\theta \to 0$. We favour transformation (8.15) in view of its closer connection, via (8.8), to Aitchison's centred logratio transformation. In addition, the α-transformation can be defined even in the case of zero values present, but in that case α must be non-negative ($\alpha > 0$). Different values of α might lead to better results, but the problem is that if for some components these values go to zero, the transformation *"breaks down"*. The two functions below calculate the α-transformation and its inverse .Note that the *alfa* function also calculates the term $\sum_{i=1}^{n} \log\left(\sum_{j=1}^{D} x_{ij}^{\alpha}\right)$ which is part of the Jacobian determinant (8.17). I do this, because in the function *profile* is is required. This was the function *profile* is faster.

```
alfa <- function(x, a, h = TRUE) {
  ## x contains the compositional data
  ## a is the power parameter, usually between -1 and 1
  ## if h is TRUE the multiplication with the Helmert matrix takes place
  x <- as.matrix(x)  ## makes sure x is a matrix
  D <- dim(x)[2] ## number of components
  if ( D == 1 )   x <- t(x)
  x <-  x / Rfast::rowsums(x)  ## makes sure x is compositional data
  if (a != 0) {
    z <- x^a
    ta <- Rfast::rowsums(z)
    z <- z / ta
    z <- D/a * z - 1/a
    sa <- sum( log(ta) )
```

```
  } else {  ## if a=0 the ilr is calculated
    xa <- log(x)
    z <- xa - Rfast::rowmeans( xa )  ## this is the clr
    sa <- dim(x)[1] * log(D)
  }
  if ( h ) {
    aff <- tcrossprod(z, helm( D ) ) ## multiply by the Helmert sub-matrix
    res <- list(sa = sa, aff = aff)
  } else  res <- list(sa = sa, aff = z)
  res
}
```

And below is the inverse of the α-transformation.

```
alfainv <- function(x, a, h = TRUE) {
  ## x is the data, not compositional
  ## a is the power parameter
  x <- as.matrix(x)
  D <- dim(x)[2]
  if ( D == 1)   x <- t(x)
  if ( h )  {
    h <- helm( D + 1 )  ## multiply with the Helmert
    ## sub-matrix to bring them onto Q^D
    y <- x %*% h
  }  else y <- x
  if (a != 0) {
    z <- ( a * y + 1 )^( 1/a )
    z <- z / Rfast::rowsums(z)
    } else {
    ## is a=0, the inverse of the clr is calculated
    ey <- exp(y)
    z <- ey / rowSums( ey )
    }
  z
}
```

8.5.2 The α-distance

For a given α we can define a simplicial distance (Tsagris et al., 2011)

$$\Delta_{\alpha}(\mathbf{x}, \mathbf{w}) = \frac{D}{|\alpha|} \left[\sum_{i=1}^{D} \left(\frac{x_i^{\alpha}}{\sum_{j=1}^{D} x_j^{\alpha}} - \frac{w_i^{\alpha}}{\sum_{j=1}^{D} w_j^{\alpha}} \right)^2 \right]^{1/2}. \tag{8.20}$$

Note, that (8.20) is simply the Euclidean distance applied to the α-transformed data. Also, as $\alpha \to 0$, (8.20) tends to the Euclidean distance applied to the centred log-ratio transformed data (Aitchison, 1983)

$$\Delta_{0}(\mathbf{x}, \mathbf{w}) = \left[\sum_{i=1}^{D} \left(\log \frac{x_i}{g(\mathbf{x})} - \log \frac{x_i}{g(\mathbf{x})} \right)^2 \right]^{1/2}, \tag{8.21}$$

where $g(\mathbf{x})$ is the geometric mean of $\mathbf{x}$ we saw in (8.6). So this means, that in this case, the centred log-ratio transformation is applied to both compositional vectors and then Euclidean distance is calculated. If the isometric log-ratio transformation (8.8) the result would be the same, because we said before that the name isometric comes from the fact, that the distances remain the same.

```
alfadist <- function(x, a) {
  ## x contains the compositional data
  ## a is the power parameter, usually between -1 and 1
  x <- as.matrix(x)  ## makes sure x is a matric
  y <- alfa(x, a, h = FALSE)$aff
  disa <- Rfast::Dist(y)
  disa
}
```

8.5.3 The Fréchet mean

Associated with this one-parameter family of distances (8.20) is the family of Fréchet means (Tsagris et al., 2011)

$$\boldsymbol{\mu}_{(\alpha)} = \mathcal{C} \left\{ \left\{ \left(\frac{1}{n} \sum_{j=1}^{n} \frac{x_{ij}^{\alpha}}{\sum_{k=1}^{D} x_{kj}^{\alpha}} \right)^{1/\alpha} \right\}_{i=1,\ldots,D} \right\}. \tag{8.22}$$

This agrees with (8.5) when $\alpha = 1$ and with (8.4) when $\alpha = 0$. Now you can go to the *ternary* function we saw before and add the Fréchet mean as well to see how it works. For an example of this with real and simulated data see Tsagris et al. (2011).

```
frechet <- function(x, a) {
  ## x contains the compositional data
  ## a is the power parameter, usually between -1 and 1
  if (a == 0) {
     m1 <- exp( Rfast::colmeans(Rfast::Log(x)) )
     m <- m1 / sum( m1 )  ## closed geometric mean
  }  else {
     xa <- x^a
     z <- xa / Rfast::rowsums(xa)
     m1 <- Rfast::colmeans(z) ^ ( 1 / a )
     m <- m1 / sum(m1)  ## frechet mean in general
  }
  m
}
```

8.5.4 Profile log-likelihood of α

Similarly to the Box-Cox transformation the most classical situation of choosing the value of the transformation parameter is through maximisation of the profile log-likelihood of the parameter. The most widely used multivariate parametric model is the multivariate normal.

The assumption we impose onto the data is that after the α-transformation they can be modelled by a multivariate normal distribution. The two versions lead to equivalent versions of the multivariate normals. The density of the d-multivariate normal after the α-transformation is

$$f(\mathbf{B}_\alpha(\mathbf{x})) = \frac{(2\pi)^{-d/2}}{|\boldsymbol{\Sigma}_\alpha|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{B}_\alpha - \bar{\mathbf{B}}_\alpha)^T \boldsymbol{\Sigma}_\alpha^{-1}(\mathbf{B}_\alpha - \bar{\mathbf{B}}_\alpha)\right\} \left|\mathbf{J}'(\mathbf{x})_\alpha\right|, \tag{8.23}$$

where $(|\mathbf{J}'_\alpha|)$ is the Jacobian determinant of the α-transformation (8.17). So, essentially, (8.23) is simply the density of the multivariate normal with an extra part, the Jacobian determinant. What remains now is to maximize the log-likelihood version of (8.23) with respect to α. This task is feasible via two ways; either by using an optimisation algorithm such as Nelder-Mead (Nelder and Mead, 1965) (using the command *optim*), or by evaluating the profile log-likelihood of α for a range of values of α

Step 1. Choose a value of α and transform the data using (8.15).

Step 2. Calculate the sample mean vector and sample covariance matrix of the transformed data.

Step 3. Evaluate the log-likelihood at the sample estimates of the mean vector and covariance matrix.

Step 4. Repeat Steps 1-3 for a range of values of α.

Step 5. Choose the α which maximises the log-likelihood calculated at Step 3.

```
alfa.profile <- function(x, a = seq(-1, 1, by = 0.01) ) {
  ## x contains the data
  ## a is the grid of values of the power parameter

  x <- as.matrix(x)  ## makes the data in a matrix form
  x <- x / Rfast::rowsums(x)  ## makes sure the data are compositional
  D <- dim(x)[2]  ## number of components
  d <- D - 1  ## dimensionality of the simplex
  n <- dim(x)[1]  ## sample size of the data
  f <- (n - 1) / n
  qa <- numeric( length(a) )  ## the log-likelihood values will be stored here
  ja <- sum( log(x) )  ## part of the Jacobian of the alpha transformation
  con <-  - n/2 * d * log(2 * pi * f) - (n - 1) * d/2 + n * (d + 1/2) * log(D)

  for ( i in 1:length(a) ) {
    trans <- alfa( x, a[i] )
    aff <- trans$aff  ## the alpha-transformation
    sa <- trans$sa  ## part of the Jacobian determinant as well
    qa[i] <-  - n/2 * log( abs( det( cov(aff) ) ) ) + (a[i] - 1) * ja - D * sa
  }

  qa <- qa + con
  ## the green lines show a 95% CI for the true value of
  ## alpha using a chi-square distribution
  b <- max(qa) - qchisq(0.95, 1)/2
  plot(a, qa, type = "l", xlab = expression( paste(alpha, " values", sep = "") ),
  ylab = "Profile log-likelihood")
  abline(h = b, col = 2)
  ci <- c( min(a[qa >= b]), max(a[qa >= b]) )
  names(ci) <- paste(c("2.5", "97.5"), "%", sep = "")
  abline(v = ci[1], col = 3, lty = 2)
  abline(v = ci[2], col = 3, lty = 2)

  res <- c(a[which.max(qa)], max(qa), qa[a == 0])
  names(res) <- c('alfa', 'max.log.lik', 'log.lik0')
  list(result = res, ci = ci)
```

```
}
```

Below is a faster function, which allows for bootstrap confidence intervals (percentile method as Efron and Tibshirani (1993) calls it) as well. Parallel computation is an option to be used preferably with big samples and not so much when there are many components. Parallel computation is advised in large sample sizes, many components and or combinations of both.

```
alfa.tune <- function(x, B = 1, ncores = 1) {
  ## x is the compositional data
  ## x must not contain any zeros

  x <- as.matrix(x)
  x <- x / Rfast::rowsums(x)
  n <- dim(x)[1]  ## sample size
  f <- (n - 1) / n
  D <- dim(x)[2]  ## number of components
  d <- D - 1  ## dimensionality of the simplex
  ja <- sum( log(x) )  ## part of the Jacobian of the alpha transformation
  con <-  -n / 2 * d * log(2 * pi) - (n - 1) * d/2 + n * (d + 1/2) * log(D)

  pa <- function(a, x) {
    trans <- alfa(x, a)
    z <- trans$aff  ## the alpha-transformation
    sa <- trans$sa  ## part of the Jacobian determinant as well
    -n/2 * log( abs( det( f * cov(z) ) ) ) + (a - 1) * ja - D * sa
  }

  if (B == 1) {
    ell <- optimize(pa, c(-1, 1), x = x, maximum = TRUE )
    aff0 <- alfa(x, 0)
    z0 <- aff0$aff
    sa <- aff0$sa  ## part of the Jacobian determinant as well
    lik0 <- con -  n/2 * log( abs( det( f * cov(z0) ) ) ) -
            ja - D * sa
    result <- c(ell$maximum, ell$objective + con, lik0)
    names(result) <- c("best alpha", "max log-lik", "log-lik at 0")

  } else {  ## bootstrap confidence intervals
    ell <- optimize(pa, c(-1, 1), x = x, maximum = TRUE )
```

```
    ab <- numeric(B)

    if (ncores == 1) {
      runtime <- proc.time()
      for (i in 1:B) {
        ind <- sample(1:n, n, replace = TRUE)
        ab[i] <- optimize(pa, c(-1, 1), x = x[ind, ], maximum = TRUE )$maximum
      }
      runtime <- proc.time() - runtime

    } else {
      runtime <- proc.time()
      cl <- makePSOCKcluster(ncores)
      registerDoParallel(cl)
      ww <- foreach::foreach( i = 1:B, .combine = rbind,
      .export = c("alfa", "helm") ) %dopar% {
        ind <- sample(1:n, n, replace = TRUE)
        ab[i] <- optimize(pa, c(-1, 1), x = x[ind, ], maximum = TRUE )$maximum
      stopCluster(cl)
      ab <- as.vector( ww )
      runtime <- proc.time() - runtime
    }

    param <- c(ell$maximum, ell$objective + con,
    quantile( ab, c(0.025, 0.975) ) )
    names(param)[1:2] <- c("best alpha", "max log-lik")
    hist(ab, main = "Bootstrapped alpha values",
         xlab = expression( paste(alpha, " values", sep = "") ) )
    abline(v = ell$maximum, col = 3)
    abline(v = mean(ab), lty = 2, col = 4)
    message <- paste("The green is the best alpha value. The blue line is
               the bootstrap mean value of alpha.")

    result <- list(param = param, message = message, runtime = runtime )
  }
  result
}
```

8.6 Regression for compositional data

8.6.1 Regression using the additive log-ratio transformation

The additive log-ratio transformation (8.7) will be used for the implementation of regression for compositional data. we could of course use the isometric log-ratio transformation (8.8) but the interpretation of the parameters is really hard and as the dimensions increase it can become impossible. The idea is simple. Apply the additive log-ratio transformation and then do multivariate regression. In the end close the fitted values back into the simplex using the inverse of the transformation.

The multivariate regression we have as option in the current function is either standard multivariate regression (see function *multivreg*) or robust multivariate regression (see function *rob.multivreg*). Section 4.2 has more functions for multivariate regression analysis. Should the user wish to use them, he/she can simply change the function *comp.reg* and incorporate the other regression functions.

$$\log\left(\frac{y_i}{y_D}\right) = \mathbf{x}^T\boldsymbol{\beta}_i \Leftrightarrow \log y_i = \log y_D + \mathbf{x}^T\boldsymbol{\beta}_i, \;\; i = 1, \ldots, d \tag{8.24}$$

where $\mathbf{x}^T$ is a column vector of the design matrix $\mathbf{X}$, D is the number of components, $d = D - 1$, $\mathbf{y}_D$ is the last component playing the role of the common divisor and

$$\boldsymbol{\beta}_i = \left(\beta_{0i}, \beta_{1i}, ..., \beta_{pi}\right)^T, \; i = 1, ..., d$$

are the regression coefficients and where p is the number of independent variables.

We see from (8.24) that when the dependent variable is the logarithm of any component, the logarithm of the common divisor component can be treated as an offset variable; an independent variable with coefficient equal to 1. But this is not something to worry about. The only issue is that no zero values are allowed.

Let us now see an example in order to make this compositional regression a bit more clear. Suppose we have the arctic lake dat from Aitchison (2003), where there are 39 measurements of three elements, sand, silt and clay from different depths (in meters) of an arctic lake. The logarithm of the depth is the independent variable (it's a good idea to use the logarithm of the independent variables, especially when these have high values). The result of the regression is

$$\begin{aligned} \log\left(sand/clay\right) &= 9.697 - 2.743\log\left(depth\right) + e_1 \\ \log\left(silt/clay\right) &= 4.805 - 1.096\log\left(depth\right) + e_2 \end{aligned}$$

We can see that the clay plays the role of the common divisor component. If the depth is 1 meter, so $\log 1 = 0$, then we can say that the percentage of sand is higher than that of

clay and the percentage of silt is higher than that of clay as well. The percentage of sand is also higher than the percentage of silt (the constant term in the first line is higher than the constant term in the second line). To find out what is the value of the composition at 1 meter of water depth we do

$$\mathcal{C}\left(e^{9.697}, e^{4.805}, 1\right) = (0.9925, 0.007, 50.0001),$$

where $\mathcal{C}(.)$ is the closure operation which means that we must divide by the sum of the vector, so that is becomes compositional, i.e. its elements sum to 1. The negative coefficient in the first line means that sand reduces relatively to clay as the water depth increases. The same is true for silt relatively to clay. A good way to understand these coefficients is to plot the logarithms of the ratios as a function of the independent variable. And then you will see why there is a negative sign.

The next function

```
comp.reg <- function(y, x, type = "classical", xnew = NULL, yb = NULL) {
  ## y is dependent variable, the compositional data
  ## x is the independent variable(s)
  ## type takes three values, either 'classical' or
  ## 'spatial' for spatial median regression.

  ## alr transformation with the first component being the base
  if ( is.null(yb) )  {
    z <- log( y[, -1] / y[, 1] )
  } else {
    z <- yb
  }

  if (type == "classical") {
    runtime <- proc.time()
    mod <- multivreg(z, x, plot = FALSE, xnew = xnew)  ## multivariate regression
    res <- mod$suma
    di <- ncol(z)
    beta <- seb <- matrix(nrow = NCOL(x) + 1, ncol = di)
    for (i in 1:di) {
     beta[, i] <- res[, 1, i]
     seb[, i] <- res[, 2, i]
    }
    rownames(seb) <- rownames(beta) <- rownames(res[, , 1])
    colnames(seb) <- colnames(beta) <- colnames(mod$fitted)
```

```
    est1 <- mod$est
    runtime <- proc.time() - runtime
  }

  if (type == "spatial") {
    mod <- spatmed.reg(z, x, xnew = xnew)  ## spatial median regression
    beta <- mod$beta
    seb <- mod$seb
    est1 <- mod$est
    runtime <- mod$runtime
  }

  est2 <- cbind( 1, exp(est1) )
  est <- est2 / Rfast::rowsums(est2)
  list(runtime = runtime, beta = beta, seb = seb, est = est)

}
```

8.6.2 Simple Dirichlet regression

An alternative method for regression is to use the Dirichlet distribution (8.9) and (8.10). The second form though (8.10) is more convenient and the estimated parameters have the same interpretation as in the additive logistic regression (8.24).

We mentioned before that Maier (2011) has created an R package for Dirichlet regression. For more information the reader is addressed to Maier's report (Maier, 2014). The next function does not come to substitute Maier's functions, by no means. Maier (2011) allows the possibility of modelling ϕ as well, linking it with the same covariates, where an exponential link is necessary to ensure that the fitted $\phi_i s$ are always positive. This is presented in the next Section.

Influence diagnostics are provided by Hijazi (2006) who suggested using a scaled Pearson χ^2 statistic to identify influential observations. This was first introduced in Boyles (1997). The idea is simple, use the following approximation.

$$(\phi + 1) \sum_{i=1^D} \frac{(y_i - \hat{y}_i)^2}{\hat{y}_i} \sim \chi^2_{D-1}. \tag{8.25}$$

So, one has to calculate the above statistic for all observations. Those observations exceeding the cut-off point of χ^2_{D-1} are considered to have possibly high influence on the regression model.

The Dirichlet density (the same as in (8.10)) is

$$f(\mathbf{x}) = \frac{\Gamma\left(\sum_{i=1}^{D} \phi a_i^*\right)}{\prod_{i=1}^{D} \Gamma\left(\phi a_i^*\right)} \prod_{i=1}^{D} y_i^{\phi a_i^* - 1},$$

where $\phi = \sum_{i=1}^{D} a_i$ and $\sum_{i=1}^{D} a_i^* = 1$. The link function used for the parameters (except for ϕ) is

$$a_1^* = \frac{1}{\sum_{j=1}^{D} e^{\mathbf{x}^T \boldsymbol{\beta}_j}}$$
$$a_i^* = \frac{e^{\mathbf{x}^T \boldsymbol{\beta}_i}}{\sum_{j=1}^{D} e^{\mathbf{x}^T \boldsymbol{\beta}_j}} \quad \text{for } i = 2, ..., D.$$

So, the the corresponding log-likelihood (a function of the β_is) is

$$\ell = n \log \Gamma(\phi) - \sum_{j=1}^{n} \sum_{i=1}^{D} \log \Gamma\left(\phi a_i^*\right) + \sum_{j=1}^{n} \sum_{i=1}^{D} \left(\phi a_i^* - 1\right) \log y_{ij},$$

The next function offers Dirichlet regression and produces an informative output. It is important for the compositional data (dependent variable) to have column names otherwise the function will not produce an output. If you do not want this, then simply remove the lines in the codes which refer to the column names of the compositional data.

```
diri.reg <- function(y, x, plot = TRUE, xnew = NULL) {
  ## y is the compositional data

  dm <- dim(y)
  n <- dm[1]  ## sample size
  ## the design matrix is created
  x <- model.matrix(y ~ ., as.data.frame(x) )
  d <- dm[2] - 1  ## dimensionality of the simplex
  z <- log(y)

    dirireg <- function(param, z, x, n, d) {
      phi <- exp( param[1] )  ## this avoids negative values in phi
      para <- param[-1]
      be <- matrix(para, ncol = d)  ## puts the beta parameters in a matrix
      mu1 <- cbind( 1, exp(x %*% be) )
      ma <- mu1 / rowSums(mu1)  ## the fitted values
      ba <- phi * ma
      - n * lgamma(phi) + sum( lgamma(ba) ) - sum( z * (ba - 1 ) )
    }
```

```
runtime <- proc.time()
rla <- z[, -1] - z[, 1]   ##  log(y[, -1] / y[, 1])  ## alr transformation
ini <- as.vector( coef( lm.fit(x, rla) ) )  ## initial values
## based on the logistic normal
## the next lines optimize the dirireg function and
## estimate the parameter values

el <- NULL
options(warn = -1)
qa <- nlm(dirireg, c(3, ini), z = z, x = x, n = n, d = d)
el[1] <- -qa$minimum
qa <- nlm(dirireg, qa$estimate, z = z, x = x, n = n, d = d)
el[2] <- -qa$minimum
vim <- 2

while (el[vim] - el[vim - 1] > 1e-06) {
  ## the tolerance value can of course change
  vim <- vim + 1
  qa <- nlm(dirireg, qa$estimate, z = z, x = x, n = n, d = d)
  el[vim] <- -qa$minimum
}

qa <- nlm(dirireg, qa$estimate, z = z, x = x, n = n, d = d, hessian = TRUE)
log.phi <- qa$estimate[1]
para <- qa$estimate[-1]  ## estimated parameter values
beta <- matrix(para, ncol = d)  ## matrix of the betas
colnames(beta) <- colnames(y[, -1])  ## names of the betas
seb <- sqrt( diag( solve(qa$hessian) ) )  ## std of the estimated betas
std.logphi <- seb[1]  ## std of the estimated log of phi
seb <- matrix(seb[-1], ncol = d)  ## std of the estimated betas

if ( !is.null( colnames(y) ) ) {
  colnames(seb) <- colnames(y[, -1])
} else  colnames(seb) <- paste("Y", 1:d, sep = "")

if ( !is.null(xnew) ) {
  xnew <- model.matrix(~., data.frame(xnew) )
  mu <- cbind( 1, exp(xnew %*% beta) )
```

```
    est <- mu / Rfast::rowsums(mu)

  } else {
    mu <- cbind( 1, exp(x %*% beta) )
    est <- mu / Rfast::rowsums(mu)  ## fitted values
    lev <- ( exp(log.phi) + 1 ) * Rfast::rowsums( (y - est)^2 / mu )

    if ( plot ) {
      plot(1:n, lev, main = "Influence values", xlab = "Observations",
      ylab = expression( paste("Pearson ", chi^2, "statistic") ) )
      lines(1:n, lev, type = "h")
      abline(h = qchisq(0.95, d), lty = 2, col = 2)
    }
  }

  runtime <- proc.time() - runtime

  if ( is.null( colnames(x) ) ) {
    p <- dim(x)[2] - 1
    rownames(beta) <- c("constant", paste("X", 1:p, sep = "") )
    if ( !is.null(seb) ) {
      rownames(seb) <- c("constant", paste("X", 1:p, sep = "") )
    }
  } else {
    rownames(beta)  <- c("constant", colnames(x)[-1] )
    if  ( !is.null(seb) ) rownames(seb) <- c("constant", colnames(x)[-1] )
  }

  list(runtime = runtime, loglik = -qa$minimum, phi = exp(log.phi),
  log.phi = log.phi, std.logphi = std.logphi, beta = beta, seb = seb,
  lev = lev, est = est)
}
```

8.6.3 Mixed Dirichlet regression

In the previous section we linked the parameters with some covariates. The mixed Dirichlet regression refers to the case when the parameter ϕ is also linked to the same covariates. So, instead of having the same value of ϕ for all compositional vectors, we allow it to vary as a function of the covariates.

The link function, used here, is the logarithm, to ensure that it's always positive

$$\phi_j^* = e^{\sum_{k=1}^{p} \mathbf{x}_j^T \boldsymbol{\gamma}_k} \tag{8.26}$$

This type of regression can also be found in Maier's report (Maier, 2014) as we have mentioned again before.

This means that we have to substitute the precision parameter ϕ in (8.11) with (8.26).

$$\ell = \sum_{j=1}^{n} \log \Gamma\left(\phi_j^*\right) - \sum_{j=1}^{n} \sum_{i=1}^{D} \log \Gamma\left(\phi_j^* a_i^*\right) + \sum_{j=1}^{n} \sum_{i=1}^{D} \left(\phi_j^* a_i^* - 1\right) \log y_{ij}.$$

The next function offers Dirichlet regression and produces an informative output. It is important for the compositional data (dependent variable) to have column names otherwise the function will not produce an output. If you do not want this, then simply remove the lines in the codes which refer to the column names of the compositional data.

```
diri.reg2 <- function(y, x, xnew = NULL) {
  ## y is the compositional data

  n <- dim(y)[1]  ## sample size
  x <- model.matrix(y ~ ., as.data.frame(x) )
  p <- dim(x)[2]  ## dimensionality of x
  d <- dim(y)[2] - 1
  ly <- log(y)  ## dimensionality of the simplex

    dirireg2 <- function(param) {
      ## param contains the parameter values
      phipar <- param[1:p]
      para <- param[ -c(1:p) ]
      phi <- exp( x %*% phipar )  ## phi is a function of the covariates
      be <- matrix(para, nrow = p)  ## puts the beta parameters in a matrix
      mu1 <- cbind( 1, exp(x %*% be) )
      ma <- mu1 / rowSums(mu1)  ## the fitted values
      ba <- as.vector(phi) * ma
      - sum( lgamma(phi) ) + sum( lgamma(ba) ) - sum( ly * (ba - 1) )
    }

  runtime <- proc.time()
  rla <- ly[, -1] - ly[, 1]    ##  log( y[, -1] / y[, 1] )  ## alr transformation
  ini <- as.vector( coef( lm.fit(x, rla) ) )  ## initial values
```

```
## based on the logistic normal
## the next lines optimize the dirireg2 function and
## estimate the parameter values

el <- NULL
qa <- nlm(dirireg2, c(rnorm(p, 0, 0.1), as.vector( t(ini) ) ) )
el[1] <- -qa$minimum
qa <- nlm(dirireg2, qa$estimate)
el[2] <- -qa$minimum
vim <- 2
while (el[vim] - el[vim - 1] > 1e-06) {
  ## the tolerance value can of course change
  vim <- vim + 1
  qa <- nlm(dirireg2, qa$estimate)
  el[vim] <- -qa$minimum
}

qa <- nlm(dirireg2, qa$estimate, hessian = TRUE)
phipar <- qa$estimate[1:p]
para <- qa$estimate[-c(1:p)]  ## estimated parameter values
beta <- matrix(para, nrow = p)  ## matrix of the betas
mu1 <- cbind( 1, exp(x %*% beta) )
ma <- mu1 / Rfast::rowsums(mu1)  ## fitted values
phi <- as.numeric( exp(x %*% phipar) )  ## estimated beta parameters of phi
s <- sqrt( diag( solve(qa$hessian) ) )  ## std of the estimated parameters
std.phi <- s[1:p]  ## std of the estimated beta parameters of the phi
seb <- matrix( s[-c(1:p)], ncol = d )  ## std of the estimated betas
V <- solve(qa$hessian)  ## covariance matrix of the parameters

runtime <- proc.time() - runtime

if ( !is.null( colnames(y) ) ) {
  colnames(beta) <- colnames(seb) <- colnames(y[, -1])
} else  colnames(beta) <- colnames(seb) <- paste("Y", 1:d, sep = "")

if ( !is.null(xnew) ) {
  xnew <- model.matrix(~., data.frame(xnew) )
  mu <- cbind( 1, exp(xnew %*% beta) )
  est <- mu / Rfast::rowsums(mu)
```

```
  } else {
    mu <- cbind( 1, exp(x %*% beta) )
    est <- mu / Rfast::rowsums(mu)  ## fitted values
  }


  if ( is.null(colnames(x)) ) {
    p <- dim(x)[2] - 1
    rownames(beta) <- c("constant", paste("X", 1:p, sep = "") )
    if ( !is.null(seb) )  rownames(seb) <- c("constant", paste("X", 1:p, sep = "") )
  } else {
    rownames(beta)  <- c("constant", colnames(x)[-1] )
    if  ( !is.null(seb) ) rownames(seb) <- c("constant", colnames(x)[-1] )
  }

  list(runtime = runtime, loglik = -qa$minimum, phipar = phipar,
       std.phi = std.phi, beta = beta, seb = seb, sigma = V, phi = phi, est = est)

}
```

8.6.4 OLS regression for compositional data

The next regression method is simply an OLS, like the *comp.reg* but applied to the raw compositional data, i.e. without log-ratio transforming them. This approach I saw it in Murteira and Ramalho (2014), where they mention that $\hat{\mathbf{B}}$, the matrix of the estimated regression coefficients, is consistent and asymptotically normal. How is $\hat{\mathbf{B}}$ calculated? Simply by minimizing the sum of squares of the residuals

$$\sum_{i=1}^{n} \mathbf{u}_i^T \mathbf{u}_i, \text{ where } \mathbf{u}_i = \mathbf{y}_i - \mathbf{G}_i\left(\mathbf{B}\right) \text{ and}$$

$$\mathbf{G}_i\left(\mathbf{B}\right) = \left(\frac{1}{\sum_{j=1}^{D} e^{\mathbf{x}_i^T \boldsymbol{\beta}_j}}, \frac{e^{\mathbf{x}_i^T \boldsymbol{\beta}_2}}{\sum_{j=1}^{D} e^{\mathbf{x}_i^T \boldsymbol{\beta}_j}}, \ldots, \frac{e^{\mathbf{x}_i^T \boldsymbol{\beta}_d}}{\sum_{j=1}^{D} e^{\mathbf{x}_i^T \boldsymbol{\beta}_j}} \right),$$

with $\mathbf{y}_i \in \mathbb{S}^d$ and $d = D - 1$, where D denotes the number of components. The link function ise same as before, the inverse of the additive log-ratio transformation.

The next R function offers the possibility of bootstrapping the standard errors of the betas. If no bootstrap is selected no standard errors will be produced.

```
ols.compreg <- function(y, x, B = 1, ncores = 1, xnew = NULL) {
```

```
## y is dependent variable, the compositional data
## x is the independent variable(s)
## B is the number of bootstrap samples used to obtain
## standard errors for the bes
## if B==1 no bootstrap is performed and no standard errors reported
## if ncores=1, then 1 processor is used, otherwise
## more are used (parallel computing)

runtime <- proc.time()
x <- model.matrix(y ~ ., as.data.frame(x) )
n <- dim(y)[1]  ## sample size
d <- dim(y)[2] - 1  ## dimensionality of the simplex
z <- list(y = y, x = x)

olsreg <- function(para, y, x, d) {
  be <- matrix(para, byrow = TRUE, ncol = d)
  mu1 <- cbind(1, exp(x %*% be))
  mu <- mu1 / rowSums(mu1)
  sum( (y - mu)^2 )
}

## the next lines minimize the reg function and obtain the betas
ini <- as.vector( t( coef(lm.fit(x, y[, -1]) ) ) )  ## initial values
oop <- options(warn = -1)
on.exit(options(oop))
qa <- nlm(olsreg, ini, y = y, x = x, d = d)
qa <- nlm(olsreg, qa$estimate, y = y, x = x, d = d)
qa <- nlm(olsreg, qa$estimate, y = y, x = x, d = d)
beta <- matrix(qa$estimate, byrow = TRUE, ncol = d)
seb <- NULL
runtime <- proc.time() - runtime

if (B > 1) {
  nc <- ncores
  if (nc == 1) {
    runtime <- proc.time()
    betaboot <- matrix(nrow = B, ncol = length(ini))
    for (i in 1:B) {
      ida <- sample(1:n, n, replace = TRUE)
```

```
      yb <- y[ida, ]
      xb <- x[ida, ]
      ini <- as.vector( t( coef(lm.fit(xb, yb[, -1]) ) ) )  ## initial values
      qa <- nlm(olsreg, ini, y = yb, x = xb, d = d)
      qa <- nlm(olsreg, qa$estimate, y = yb, x = xb, d = d)
      qa <- nlm(olsreg, qa$estimate, y = yb, x = xb, d = d)
      betaboot[i, ] <- qa$estimate
    }
    s <- Rfast::colVars(betaboot, std = TRUE)
    seb <- matrix(s, byrow = TRUE, ncol = d)
    runtime <- proc.time() - runtime

  } else {
    runtime <- proc.time()
    betaboot <- matrix(nrow = B, ncol = length(ini) )
    cl <- makePSOCKcluster(ncores)
    registerDoParallel(cl)
    ww <- foreach::foreach(i = 1:B, .combine = rbind, .export="olsreg")
  %dopar% {
      ida <- sample(1:n, n, replace = TRUE)
      yb <- y[ida, ]
      xb <- x[ida, ]
      ini <- as.vector( t( coef(lm.fit(xb, yb[, -1]) ) ) )  ## initial values
      qa <- nlm(olsreg, ini, y = yb, x = xb, d = d)
      qa <- nlm(olsreg, qa$estimate, y = yb, x = xb, d = d)
      qa <- nlm(olsreg, qa$estimate, y = yb, x = xb, d = d)
      betaboot[i, ] <- qa$estimate
    }
    stopCluster(cl)
    s <- Rfast::colVars(ww, std = TRUE)
    seb <- matrix(s, byrow = TRUE, ncol = d)
    runtime <- proc.time() - runtime
  }
}

if ( is.null(xnew) ) {
  mu <- cbind( 1, exp(x %*% beta) )
  est <- mu / Rfast::rowsums(mu)
} else {
```

```
    xnew <- model.matrix(~., data.frame(xnew) )
    mu <- cbind(1, exp(xnew %*% beta))
    est <- mu / Rfast::rowsums(mu)
  }

  if ( is.null(colnames(x)) ) {
    p <- dim(x)[2] - 1
    rownames(beta) <- c("constant", paste("X", 1:p, sep = "") )
    if ( !is.null(seb) )  rownames(seb) <- c("constant",
                      paste("X", 1:p, sep = "") )
  } else {
    rownames(beta)  <- c("constant", colnames(x)[-1] )
    if  ( !is.null(seb) ) rownames(seb) <- c("constant", colnames(x)[-1] )
  }

  list(runtime = runtime, beta = beta, seb = seb, est = est)
}
```

8.6.5 Multinomial logit regression (or Kullback-Leibler divergence based regression for compositional data)

Murteira and Ramalho (2014) studied the multinomial logit regression using a more general transformation than the inverse of the additive logistic transformation. In fact (8.27) corresponds to the maximisation of the multinomial log-likelihood. However, here we will use the the inverse of the additive logistic transformation as the link function. The goal is to minimize the KL divergence with respect to the regression parameters. In logistic regression, this method is called estimation by minimum discrimination information (Agresti, 2002).

$$\min_{\boldsymbol{\beta}_i} \left\{ \sum_{j=1}^{n} \sum_{i=1}^{D} y_{ij} \log \frac{y_{ij}}{f(\mathbf{x})} \right\} = \min_{\boldsymbol{\beta}_i} \left\{ -\sum_{j=1}^{n} \sum_{i=1}^{D} y_{ij} \log f(\mathbf{x}) \right\} =$$
$$\max_{\boldsymbol{\beta}_i} \left\{ \sum_{j=1}^{n} \sum_{i=1}^{D} y_{ij} \log f(\mathbf{x}) \right\}, \tag{8.27}$$

where $\mathbf{x}$ stands for the design matrix and f is the function defined below. For every value of the composition y_{ij} there corresponds a fitted value $\hat{y}_{ij}$ which is a function of some covariates through an exponential form.

$$f(\mathbf{x}) = \left\{ \begin{array}{ll} \hat{y}_{1j} = \frac{1}{1+\sum_{l=2}^{D} e^{\mathbf{x}_l^T \boldsymbol{\beta}_l}} & \\ \hat{y}_{ij} = \frac{e^{\mathbf{x}_i^T \boldsymbol{\beta}_i}}{1+\sum_{l=2}^{D} e^{\mathbf{x}_l^T \boldsymbol{\beta}_l}} & \text{for } i = 2, ..., D. \end{array} \right\}$$

As for the properties of the coefficients, Murteira and Ramalho (2014) shows that are asymptotically normal, so this is good news. However, I saw that their standard errors are not similar to the ones obtained from the other methods. So, I offer the option of bootstrap estimation of their standard errors in the next R function.

The second key thing about this regression method is that even if there are zeros in the observed values, there is absolutely no problem. This advantage of this method was not highlighted by Murteira and Ramalho (2014) and I am making this point now. Just think about it, no need for zero value imputation, so no extra added bias or variability.

```
kl.compreg <- function(y, x, B = 1, ncores = 1, xnew = NULL) {
  ## y is dependent variable, the compositional data
  ## x is the independent variable(s)
  ## B is the number of bootstrap samples used to obtain
  ## standard errors for the betas
  ## if B==1 no bootstrap is performed and no standard errors are reported
  ## if ncores=1, then 1 processor is used, otherwise
  ## more are used (parallel computing)

  n <- dim(y)[1]  ## sample size
  x <- model.matrix(y ~ ., as.data.frame(x) )
  d <- dim(y)[2] - 1  ## dimensionality of the simplex

   klreg <- function(para, y, x, d) {
     be <- matrix(para, byrow = TRUE, ncol = d)
     mu1 <- cbind( 1, exp(x %*% be) )
     mu <- mu1 / rowSums(mu1)
     - sum(y * log(mu), na.rm = TRUE)
   }

  ## the next lines minimize the reg function and obtain the estimated betas
  ini <- as.vector( t( coef( lm.fit(x, y[, -1]) ) ) )  ## initial values

  runtime <- proc.time()
  oop <- options(warn = -1)
  on.exit(options(oop))
  qa <- nlm(klreg, ini, y = y, x = x, d = d)
  qa <- nlm(klreg, qa$estimate, y = y, x = x, d = d)
  qa <- nlm(klreg, qa$estimate, y = y, x = x, d = d)
  beta <- matrix(qa$estimate, byrow = TRUE, ncol = d)
  seb <- NULL
```

```
runtime <- proc.time() - runtime

if (B > 1) {
nc <- ncores
  if (nc == 1) {
    runtime <- proc.time()
    betaboot <- matrix(nrow = B, ncol = length(ini))
    for (i in 1:B) {
      ida <- sample(1:n, n, replace = TRUE)
      yb <- y[ida, ]
      xb <- x[ida, ]
      ini <- as.vector( t( coef( lm.fit(xb, yb[, -1]) ) ) )  ## initial values
      qa <- nlm(klreg, ini, y = yb, x = xb, d = d)
      qa <- nlm(klreg, qa$estimate, y = yb, x = xb, d = d)
      qa <- nlm(klreg, qa$estimate, y = yb, x = xb, d = d)
      betaboot[i, ] <- qa$estimate
    }
    s <- Rfast::colVars(betaboot, std = TRUE)
    seb <- matrix(s, byrow = TRUE, ncol = d)
    runtime <- proc.time() - runtime

  } else {
    runtime <- proc.time()
    betaboot <- matrix(nrow = B, ncol = length(ini) )
    cl <- makePSOCKcluster(ncores)
    registerDoParallel(cl)
    ww <- foreach::foreach(i = 1:B, .combine = rbind) %dopar% {
      ida <- sample(1:n, n, replace = TRUE)
        yb <- y[ida, ]
        xb <- x[ida, ]
        ini <- as.vector( t( coef( lm.fit(xb, yb[, -1]) ) ) )  ## initial values
        qa <- nlm(klreg, ini, y = yb, x = xb, d = d)
        qa <- nlm(klreg, qa$estimate, y = yb, x = xb, d = d)
        qa <- nlm(klreg, qa$estimate, y = yb, x = xb, d = d)
        betaboot[i, ] <- qa$estimate
     }
    stopCluster(cl)

    s <- Rfast::colVars(ww, std = TRUE)
```

```
      seb <- matrix(s, byrow = TRUE, ncol = d)
      runtime <- proc.time() - runtime
    }
  }

    if ( is.null(xnew) ) {
    mu <- cbind( 1, exp(x %*% beta) )
    est <- mu / Rfast::rowsums(mu)
  } else {
    xnew <- model.matrix(~., data.frame(xnew) )
    mu <- cbind(1, exp(xnew %*% beta))
    est <- mu / Rfast::rowsums(mu)
  }

  if ( is.null(colnames(x)) ) {
    p <- dim(x)[2] - 1
    rownames(beta) <- c("constant", paste("X", 1:p, sep = "") )
    if ( !is.null(seb) )  rownames(seb) <- c("constant", paste("X", 1:p, sep = "") )
  } else {
    rownames(beta)  <- c("constant", colnames(x)[-1] )
    if  ( !is.null(seb) ) rownames(seb) <- c("constant", colnames(x)[-1] )
  }

  list(runtime = runtime, beta = beta, seb = seb, est = est)
}
```

8.6.6 ESOV (Kullback-Leibler divergence based) regression

I have recently suggested (Tsagris, 2015a) that as a measure of the distance between two compositions we can use a special case of the Jensen-Shannon divergence

$$\text{ES-OV}\left(\mathbf{x}, \mathbf{y}\right) = \sum_{j=1}^{D} \left(x_j \log \frac{2x_j}{x_j + y_j} + y_j \log \frac{2y_j}{x_j + y_j} \right), \tag{8.28}$$

where $\mathbf{x}$ and $\mathbf{y} \in \mathbb{S}^d$. Endres and Schindelin (2003) and Österreicher and Vajda (2003) proved, independently, that (8.28) satisfies the triangular identity and thus it is a metric. The names ES-OV comes from the researchers' initials. In fact, (8.28) is the square of the metric, still a metric, and we will use this version.

The idea is simple and straightforward, minimization of the ES-OV metric between the

observed and the fitted compositions with respect to the beta coefficients

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{D} \left(\mathbf{y}_i \log \frac{2\mathbf{y}_i}{\mathbf{y}_i + \hat{\mathbf{y}}_i} + \hat{\mathbf{y}}_i \log \frac{2\hat{\mathbf{y}}_i}{\mathbf{y}_i + \hat{\mathbf{y}}_i} \right), \tag{8.29}$$

```
js.compreg <- function(y, x, B = 1, ncores = 1, xnew = NULL) {
  ## y is dependent variable, the compositional data
  ## x is the independent variable(s)
  ## B is the number of bootstrap samples used to obtain
  ## standard errors for the betas
  ## if B==1 no bootstrap is performed and no standard errors are reported
  ## if ncores=1, then 1 processor is used, otherwise
  ## more are used (parallel computing)

  n <- dim(y)[1]  ## sample size
  x <- model.matrix(y ~ ., as.data.frame(x) )
  d <- dim(y)[2] - 1  ## dimensionality of the simplex

  jsreg <- function(para, y, x, d){
    be <- matrix(para, byrow = TRUE, ncol = d)
    mu1 <- cbind( 1, exp(x %*% be) )
    mu <- mu1 / rowSums(mu1)
    M <- ( mu + y ) / 2
    sum( - y * log1p(mu / y) + mu * log(mu / M), na.rm = TRUE )
  }

  ## the next lines minimize the kl.compreg function and obtain the estimated betas
  ini <- as.vector( t( kl.compreg(y, x[, -1])$beta ) )

  runtime <- proc.time()
  oop <- options(warn = -1)
  on.exit(options(oop))
  qa <- nlm(jsreg, ini, y = y, x = x, d = d)
  qa <- nlm(jsreg, qa$estimate, y = y, x = x, d = d)
  qa <- nlm(jsreg, qa$estimate, y = y, x = x, d = d)
  beta <- matrix(qa$estimate, byrow = TRUE, ncol = d)
  seb <- NULL
  runtime <- proc.time() - runtime

  if (B > 1) {
```

```
betaboot <- matrix( nrow = B, ncol = length(ini) )
nc <- ncores
  if (nc == 1) {
    runtime <- proc.time()
    for (i in 1:B) {
      ida <- sample( 1:n, n, replace = TRUE )
      yb <- y[ida, ]
      xb <- x[ida, ]
      ini <- as.vector( t( kl.compreg(yb, xb[, -1])$beta ) ) ## initial values
      qa <- nlm(jsreg, ini, y = yb, x = xb, d = d)
      qa <- nlm(jsreg, qa$estimate, y = yb, x = xb, d = d)
      qa <- nlm(jsreg, qa$estimate, y = yb, x = xb, d = d)
      betaboot[i, ] <- qa$estimate
    }
    s <- Rfast::colVars(betaboot, std = TRUE)
    seb <- matrix(s, byrow = TRUE, ncol = d)
    runtime <- proc.time() - runtime

  } else {
    runtime <- proc.time()
    cl <- makePSOCKcluster(ncores)
    registerDoParallel(cl)
    ww <- foreach::foreach(i = 1:B, .combine = rbind, .export="jsreg") %dopar% {
      ida <- sample(1:n, n, replace = TRUE)
      yb <- y[ida, ]
      xb <- x[ida, ]
      ini <- as.vector( t( kl.compreg(yb, xb[, -1])$beta ) ) ## initial values
      qa <- nlm(jsreg, ini, y = yb, x = xb, d = d)
      qa <- nlm(jsreg, qa$estimate, y = yb, x = xb, d = d)
      qa <- nlm(jsreg, qa$estimate, y = yb, x = xb, d = d)
      betaboot[i, ] <- qa$estimate
    }
    stopCluster(cl)
    s <- Rfast::colVars(ww, std = TRUE)
    seb <- matrix(s, byrow = TRUE, ncol = d)
    runtime <- proc.time() - runtime
  }
}
```

```
  if ( is.null(xnew) ) {
    mu <- cbind( 1, exp(x %*% beta) )
    est <- mu / Rfast::rowsums(mu)
  } else {
    xnew <- model.matrix(~., data.frame(xnew) )
    mu <- cbind(1, exp(xnew %*% beta))
    est <- mu / Rfast::rowsums(mu)
  }

  if ( is.null(colnames(x)) ) {
    p <- dim(x)[2] - 1
    rownames(beta) <- c("constant", paste("X", 1:p, sep = "") )
    if ( !is.null(seb) )  rownames(seb) <- c("constant", paste("X", 1:p, sep = "") )
  } else {
    rownames(beta)  <- c("constant", colnames(x)[-1] )
    if  ( !is.null(seb) ) rownames(seb) <- c("constant", colnames(x)[-1] )
  }

  list(runtime = runtime, beta = beta, seb = seb, est = est)
}
```

8.6.7 The α-regression

We will use the inverse of the additive logistic transformation (8.7), combined with the α-transformation (8.15), as a link function. This is a new regression using the α-transformation which allows for more flexibility even in the presence of zero values (Tsagris, 2015b). Another feature of this method is that the line is always curved (unless α is far away from zero) and so it can be seen not only as a generalization of the log-ratio regression but also as a flexible type compositional regression in the sense that the curvature of the line is chosen based on some discrepancy criteria, examined later.

In order for the fitted values to satisfy the constraint imposed by the simplex we model the inverse of the additive logistic transformation of the mean response. Hence, the fitted values will always lie within S^d and we also retain the flexibility the α-transformation offers.

We assume that the conditional mean of the observed composition can be written as a non-linear function of some covariates

$$\begin{aligned} \mu_1 &= \frac{1}{1+\sum_{j=1}^{d} e^{\mathbf{x}^T \boldsymbol{\beta}_j}} \\ \mu_i &= \frac{e^{\mathbf{x}^T \boldsymbol{\beta}_i}}{1+\sum_{j=1}^{d} e^{\mathbf{x}^T \boldsymbol{\beta}_j}} \quad \text{for } i = 2, ..., D, \end{aligned} \tag{8.30}$$

where

$\boldsymbol{\beta}_i = \left(\beta_{0i}, \beta_{1i}, ..., \beta_{pi}\right)^T$, $i = 1, ..., d$ and p denotes the number of covariates.

Then a multivariate linear regression is applied to the α-transformed data

$$l(\alpha) = -\frac{n}{2}\log\left|\hat{\boldsymbol{\Sigma}}\right| - \frac{1}{2}tr\left[\left(\mathbf{Y}_\alpha - \mathbf{M}_\alpha\right)\hat{\boldsymbol{\Sigma}}_\alpha^{-1}\left(\mathbf{Y}_\alpha - \mathbf{M}_\alpha\right)^T\right], \tag{8.31}$$

where $\mathbf{Y}_\alpha$ and $\mathbf{M}_\alpha$ are the α-transformed response and fitted compositional vectors. We have ignored the Jacobian determinant of the α-transformation since it plays no role in the optimization process and the choice of α For each value of α we maximize the value of this objective function (8.31). The $\hat{\boldsymbol{\Sigma}}$ needs not be numerically estimated, since $\hat{\mathbf{B}}$, the matrix of the estimates and $\hat{\boldsymbol{\Sigma}}$ are statistically independent (Mardia et al., 1979). The maximum likelihood estimator of $\boldsymbol{\Sigma}$ is (Mardia et al., 1979)

$$\hat{\boldsymbol{\Sigma}}_\alpha = n^{-1}\mathbf{Y}_\alpha^T\mathbf{P}\mathbf{Y}_\alpha,$$

where $\mathbf{P} = \mathbf{I}_n - \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T$. But since this covariance is not unbiased we will use the unbiased estimator

$$\hat{\boldsymbol{\Sigma}}_\alpha = \left(n - p - 1\right)^{-1}\mathbf{Y}_\alpha^T\mathbf{P}\mathbf{Y}_\alpha,$$

where $\mathbf{X}$ is the design matrix and p is the number of independent variables.

The consistency of the estimators of the parameters is not an issue in our case since we focus on prediction inference. Since the estimation of the parameters depends upon the value of α, the estimates will not be consistent, unless that is the true assumed model. The multivariate normal is defined in the whole of $\mathbb{R}^d$ but the α-transformation maps the data onto a subset of $\mathbb{R}^d$. Thus, unless there is not too much probability left outside the simplex, the multivariate normal distribution might not be the best option.

What the α-transformation does essentially, is to contract the simplex, center it to the origin and then project it on a subspace of $\mathbb{R}^d$ by using the Helmert sub-matrix (Lancaster, 1965). So if the fitted multivariate normal has high dispersion that will lead to probability left outside the simplex. The multivariate t distribution was used by Lange et al. (1989) as a more robust, in comparison to the multivariate normal, model but even so, it will not be the best option, mainly for two reasons. Even if the multivariate t distribution could provide flatter tails, there would still be some probability (even less than the normal) left outside the simplex. Secondly, in a regression setting, the number of parameters we would have to estimate numerically is increased and this would make the maximization process more difficult. Let alone the fact that it is not as robust as one may think or believe (Breusch et al., 1997).

A final key feature we have to note is that when $\alpha \to 0$ we end up with the additive log-ratio regression (8.24). In the next function the last argument ("yb") allows you to put the α-transformed data directly. This is useful in the case of the cross-validation, to avoid transforming the data every time, for every fold, only for every value of α.

```
alfa.reg <- function(y, x, a, xnew = NULL, yb = NULL, seb = FALSE) {
  ## y is the compositional data (dependent variable)
  ## x is the independent variables
  ## a is the value of alpha
  ## internal function for the alfa-regression
  reg <- function(para, ya, x, ha, d, n, D) {
    be <- matrix(para, ncol = d)
    mu1 <- cbind( 1, exp(x %*% be) )
    zz <- mu1^a
    ta <- rowSums(zz)
    za <- zz / ta
    za <- D / a * za - 1/a
    ma <- za %*% ha
    esa <- ya - ma
    sa <- crossprod(esa)
    log( det(sa) )
  }

  D <- dim(y)[2]
  d <- D - 1  ## dimensionality of the simplex
  dm <- dim(x)
  p <- dm[2]  ;    n <- dm[1]

  if ( is.null(yb) ) {
    ya <- alfa(y, a)$aff
  } else  ya <- yb
  x <- model.matrix(ya ~., data.frame(x) )

  if ( a == 0 ) {
    mod <- comp.reg(y, x[, -1], yb = yb)
    be <- mod$be
    seb <- mod$seb
    runtime <- mod$runtime

  } else {
```

```
    runtime <- proc.time()

    ha <- t( helm(D) )
    ini <- as.vector( solve(crossprod(x), crossprod(x, ya) ) )
    qa <- nlminb( ini, reg, ya = ya, x = x, ha = ha, d = d, n = n,
    D = D, control = list(iter.max = 2000) )
    qa <- optim( qa$par, reg, ya = ya, x = x, ha = ha, d = d, n = n,
    D = D, control = list(maxit = 5000) )
    qa <- optim( qa$par, reg, ya = ya, x = x, ha = ha, d = d, n = n,
    D = D, control = list(maxit = 5000) )
    qa <- optim( qa$par, reg, ya = ya, x = x, ha = ha, d = d, n = n,
    D = D, control = list(maxit = 5000), hessian = TRUE )
    be <- matrix(qa$par, ncol = d)
    runtime <- proc.time() - runtime

    if ( seb ) {
      seb <- sqrt( diag( Rfast::spdinv(qa$hessian) ) )
      seb <- matrix(seb, ncol = d)
    } else  seb <- NULL
  }  ## end if (a == 0)

  est <- NULL
  if ( !is.null(xnew) ) {
    xnew <- model.matrix(~., data.frame(xnew) )
    est <- cbind( 1, exp(xnew %*% be) )
est <- est/Rfast::rowsums(est)
  }

  if ( is.null( colnames(x) ) ) {
    p <- dim(x)[2] - 1
    rownames(be) <- c("constant", paste("X", 1:p, sep = "") )
    if ( !is.null(seb) )  rownames(seb) <- c("constant", paste("X", 1:p, sep = "") )
  } else {
    rownames(be)  <- c("constant", colnames(x)[-1] )
    if ( !is.null(seb) )  rownames(seb) <- c("constant", colnames(x)[-1] )
  }

  list(runtime = runtime, be = be, seb = seb, est = est)
}
```

The disadvantage of the profile log-likelihood of α (should you use it), for choosing the value of α, is that it does not allow zeros. On the other hand, it provides the maximum likelihood estimates which are asymptotically normal. But the latter is not entirely true, since the resulting normal is not defined on whole of $\mathbb{R}^d$.

I suggest an alternative and perhaps better way of choosing the value of α. Better in the sense that it is trying to take into account the proximity between the observed and the fitted values. The criterion is to choose the α which minimizes twice the Kullback-Leibler divergence (Kullback, 1997)

$$KL = 2 \sum_{j=1}^{n} \sum_{i=1}^{D} y_{ij} \log \frac{y_{ij}}{\hat{y}_{ij}}, \tag{8.32}$$

where y_{ij} is the observed compositional point and $\hat{y}_{ij}$ is the corresponding fitted value. The form of the deviance for the log-linear models and the logistic regression has the same expression as well. Hence, I transfer the same form of divergence to compositional data. For every value of α we estimate the parameters of the regression and choose the value of α which minimizes (8.32).

The number 2 is there because in the case of $D = 2$ we end up with the log-likelihood of the binary logistic regression. The Kullback-Leibler divergence (8.32) takes into account the divergence or the distance of each of the observed values from the fitted values.

Since I am interested in prediction analysis I use a K-fold cross-validation to choose the value of α. I split the data into K sets (fold). Every time I leave a set out and fit the model in the remaining sample (chose the best value of α and so on). Then, I predict the values of the compositional data for the set left outside and calculate the Kullback-Leibler divergence (8.32) in order to measure the performance. This is repeated for all K sets (folds) of data and the average Kullback-Leibler divergence is obtained.

In the published paper Tsagris (2015b) I just used the performance obtained using the fitted values in order to save time. I could have used a repeated hold-out cross validation, but neither of them is the appropriate one.

```
alfareg.tune <- function(y, x, a = seq(0.1, 1, by = 0.1), nfolds = 10,
                         folds = NULL, nc = 1, seed = FALSE, graph = FALSE) {
  ## y is the compositional data (dependent variable)
  ## x is the independent variables
  ## a is a range of values of alpha
  ## nfolds is the number of folds for the K-fold cross validation
  ## nc is how many cores you want to use, default value is 2
  if ( min(y) == 0 )  a <- a[a>0]
  la <- length(a)
  n <- dim(y)[1]
```

```
  ina <- 1:n
  x <- model.matrix(y ~., data.frame(x) )
  if ( is.null(folds) )  folds <- Compositional::makefolds(ina,
                         nfolds = nfolds, stratified = FALSE, seed = seed)
  nfolds <- length(folds)

  if (nc <= 1) {
    apa <- proc.time()
    kula <- matrix(nrow = nfolds, ncol = la)
    for (j in 1:la) {
      ytr <- alfa(y, a[j])$aff
      for (i in 1:nfolds) {
        xu <- x[ folds[[ i ]], -1 , drop = FALSE]
        yu <- y[ folds[[ i ]], ]
        xa <- x[ -folds[[ i ]], -1]
        yb <- ytr[ -folds[[ i ]], ]
        mod <- alfa.reg(yu, xa, a[j], xnew = xu, yb = yb)
        yest <- mod$est
        kula[i, j] <- 2 * mean(yu * log(yu / yest), na.rm = TRUE)
      }
    }

    kl <- Rfast::colmeans(kula)
    opt <- a[ which.min(kl) ]
    val <- which.min(kl)
    per <- min(kl, na.rm = TRUE)
    pera <- Rfast::rowMins(kula, value = TRUE)
    apa <- proc.time() - apa

  } else {
    apa <- proc.time()
    oop <- options(warn = -1)
on.exit( options(oop) )
    val <- matrix(a, ncol = nc) ## if the length of a is not equal to the
    ## dimensions of the matrix val a warning message should appear
    ## but with oop <- options(warn = -1) you will not see it
    cl <- parallel::makePSOCKcluster(nc)
    doParallel::registerDoParallel(cl)
    if ( is.null(folds) )  folds <- Compositional::makefolds(ina, nfolds = nfolds,
```

```
                                    stratified = FALSE, seed = seed)
  kula <- foreach::foreach(j = 1:nc, .combine = cbind, .packages = "Rfast",
          .export = c("alfa.reg",  "alfa", "helm", "comp.reg", "multivreg",
          "rowsums", "colmeans", "colVars") ) %dopar% {
     ba <- val[, j]
     ww <- matrix(nrow = nfolds, ncol = length(ba) )
     for ( l in 1:length(ba) ) {
        ytr <- alfa(y, ba[l])$aff
        for (i in 1:nfolds) {
          xu <- x[ folds[[ i ]], -1 , drop = FALSE]
          yu <- y[ folds[[ i ]], ]
          xa <- x[ -folds[[ i ]], -1]
          yb <- ytr[ -folds[[ i ]], ]
          mod <- alfa.reg(yu, xa, ba[l], xnew = xu, yb = yb)
          yest <- mod$est
          ww[i, l] <- 2 * mean(yu * log(yu / yest), na.rm = TRUE)
        }
     }
     return(ww)
  }
  parallel::stopCluster(cl)

  kula <- kula[, 1:la]
  kl <- Rfast::colmeans(kula)
  opt <- a[ which.min(kl) ]
  val <- which.min(kl)
  per <- min(kl, na.rm = TRUE)
  pera <- Rfast::rowMins(kula, value = TRUE)   ## apply(kula, 1, min)
  apa <- proc.time() - apa
}

if ( graph ) {
  plot( a, kula[1, ], type = 'l', ylim = c( min(kula), max(kula) ),
       xlab = expression(alpha), ylab = 'Twice the Kullback Leibler divergence',
       cex.lab = 1.3 )
  for (i in 2:nfolds)  lines(a, kula[i, ])
  lines(a, kl, col = 2, lty = 2, lwd = 2)
}
```

```
  list(runtime = apa, kula = kula, kl = kl, opt = opt, value = per)
}
```

8.6.8 Regression for compositional data with compositional covariates

We will now see a simple approach to the case of both dependent and independent variables being compositional variables. The key thing is principal component regression. Transform the independent compositional variables using the isometric log-ratio transformation (8.8). You can of course use the additive log-ratio transformation (8.7), but I chose the first one as I do not think it makes that much of a difference.

Perform principal component analysis on the transformed data and calculate the scores. Use them as the independent variables and do the compositional data regression, either the classical multivariate regression of the additively log-ratio transformed data (*comp.reg* function), the Dirichlet regression (*diri.reg* or *diri.reg2* functions), the OLS (*ols.compreg* function) or the multinomial logit regression (*kl.compreg* function). In addition, you can choose how many principal components you want to keep. The drawback of this way, is that the regression coefficients are not unbiased and consistent, since, we are using the principal component scores. So, this way is mostly for prediction purposes.

In addition, we would not want the independent variables to have zeros. If the dependent variables have zero, we can deal with it, use zero value replacement (package*robCompositions*) and then the regression models or use the *kl.compreg* or *ols.compreg* functions which work even in the presence of zeros.

```
comp.compreg <- function(y, x, type = "classical", k) {
  ## y and x are compositional data
  ## y and x are the dependent and independent variables respectively
  ## type is the type of regression to be used
  ## type can be 'classical', 'spatial', 'diri_1', 'diri_2',
  ## 'ols' or 'kl'
  ## k is the number of principal components to use
  p <- dim(x)[2]  ## dimensionality of x
  n <- dim(y)[1]  ## sample size

  ## note that the maximum number of k is p-1
  zx <- log(x)
  z1 <- zx - Rfast::rowmeans(zx)
  z <- tcrossprod( z1, helm(p) )  ## the ilr transformation for the x
  z <- Rfast::standardise(z)  ## standardize the independent variables
  eig <- eigen( crossprod(z), symmetric = TRUE )
  values <- eig$values  ## eigenvalues
```

```
  per <- values/sum(values)  ## proportion of each eigenvalue
  vec <- eig$vectors  ## eigenvectors, or principal components
  pc <- z %*% vec[, 1:k]  ## PCA scores

  if (type == "classical" | type == "spatial" ) {
    mod <- comp.reg(y, pc, type = type)
  }

  if (type == "diri_1")   mod <- diri.reg(y, pc)
  if (type == "diri_2")   mod <- diri.reg2(y, pc)
  if (type == "ols")      mod <- ols.compreg(y, pc, B = 1)
  if (type == "kl")       mod <- kl.compreg(y, pc, B = 1)

  list(percentage = per, mod = mod)
}
```

The way to choose k, the number of principal components to use is the same as in the case of the principal component regression. Split the data into training and test set. Use the training set to estimate the parameters of the model and then use the test set for prediction purposes. Calculate the Kullback-Leibler divergence of the observed from the fitted compositional vectors

$$\sum_{j=1}^{n} \sum_{i=1}^{D} y_{ij} \log \frac{y_{ij}}{\hat{y}_{ij}}.$$

Repeat this process say 200 times and calculate the average for different number of principal components. Note, that the maximum number of components you can have is $p - 1$, where p stands for the number of components of the independent compositional variables.

```
compcompreg.tune <- function(y, x, type = "classical", fraction = 0.2,
  R = 200, seed = TRUE) {
  ## y and x are the dependent and independent variables resepctively
  ## y and x are compositional data
  ## type can be 'classical', 'spatial', 'diri_1', 'diri_2',
  ## 'ols' or 'kl'
  ## fraction is the percentage of data to be used as test set
  ## R is the number of iterations
  n <- dim(y)[1]  ## sample size
  p <- dim(x)[2] - 1  ## dimensionality of x - 1
  ## p is the maximum number of principal components to be used
```

```
  nu <- round(fraction * n)  ## test sample size
  deigma <- matrix(nrow = R, ncol = nu)
  ## deigma will contain the positions of the test set
  ## this is stored but not showed in the end
  ## the user can access it though by running
  ## the commands outside this function
  crit <- matrix(nrow = R, ncol = p)
  ## if seed==TRUE then the results will always be the same
  if ( seed )  set.seed(1234567)
  for (vim in 1:R)  deigma[vim, ] <- sample(1:n, nu)
  for (j in 1:p) {
    for (i in 1:R) {
      ytest <- y[deigma[i, ], ]
      xtest <- x[deigma[i, ], ]
      ytrain <- y[-deigma[i, ], ]
      xtrain <- x[-deigma[i, ], ]
      be <- comp.compreg(ytrain, xtrain, type = type, j)$mod$beta
      mu1 <- cbind(1, exp(cbind(1, xtest) %*% be))
      mu <- mu1/rowSums(mu1)
      crit[i, j] <- sum(ytest * log(ytest/mu), na.rm = T)
    }
  }
  mspe <- Rfast::colmeans(crit)
  names(mspe) <- paste( "PC ", 1:p )
  plot( mspe, type = 'b', ylab = "MSPE values",
  xlab = "Number of principal components" )
  list(optimal = which.min(mspe), mspe = mspe)
}
```

8.6.9 Univariate regression where the independent variables are compositional data using the α-transformation

The α-transformation can be used again in the univariate regression when the independent variables are actually compositional data. The idea is again simple, apply the α-transformation (8.15) to the compositional data (independent variables) and then perform principal component regression. If you perform the isometric log-ratio transformation for example (8.8) or the α-transformation in general, and then regression directly, you are neglecting the collinearity issues. That is why I propose (Tsagris, 2015b) to use principal components regression. What is more, is that if you have zeros in the data, the α-transformation will still

work without the need for imputation. In that case, α must be non negative. The next function is more general than the one found in Tsagris (2015b) in the sense that principal component regression for binary (binomial) and count data (poisson) are now offered.

```
alfa.pcr <- function(y, x, a, k, xnew = NULL) {
  ## y is dependent univariate variable. It can be matrix or vector
  ## x are compositional data, the covariates
  ## a is the value of a for the alpha-transformation
  ## k is the number of principal components to use
  ## oiko can be either "normal", "binomial" or "poisson"
  ## depending on the type of the independent variable
  ## "normal" is set by default
  z <- alfa(x, a, h = TRUE)$aff ## apply the alpha-transformation

  if ( length(unique(y)) == 2 ) {
    oiko <- "binomial"
  } else if ( sum( y - round(y) ) == 0 ) {
    oiko <- "poisson"
  } else oiko <- "normal"

  if (oiko == 'normal') {
    mod <- pcr(y, z, k, xnew = xnew)
  } else  mod <- glm.pcr(y, z, k, xnew = xnew)
  mod ## principal component regression with the alpha-transformed
  ## compositional data
}
```

The task now is to choose the optimal pair of (α, k) values. To do so, cross validation is to employed once again. For a grid of values of α every time α-transform the data and then find the optimal number of principal components via cross validation. Optimal in the sense of minimizing the mean squared error of the predictions. For every value of α, it transforms the data and then performs principal component regression according to the distribution set by the user.

```
alfapcr.tune <- function(y, x, model = "gaussian", nfolds = 10, maxk = 50,
                         a = seq(-1, 1, by = 0.1), folds = NULL, ncores = 1,
                          graph = TRUE, col.nu = 15, seed = FALSE) {
  ## model can be either "normal", "binomial" or "poisson"
  ## depending on the type of the independent variable
  ## "normal" is set by default
```

```
n <- dim(x)[1]
d <- dim(x)[2] - 1
if ( min(x) == 0 )   a <- a[ a > 0 ]  ## checks for zero values in the data.
da <- length(a)
ina <- 1:n
if ( is.null(folds) )  folds <- Compositional::makefolds(ina, nfolds = nfolds,
                                                stratified = FALSE, seed = seed)
nfolds <- length(folds)
mspe2 <- array( dim = c( nfolds, d, da) )

if ( model == 'gaussian' ) {
  tic <- proc.time()
  for ( i in 1:da ) {
    z <- Compositional::alfa(x, a[i])$aff
    mod <- Compositional::pcr.tune(y, z, nfolds = nfolds,
    maxk = maxk, folds = folds, ncores = ncores, seed = seed, graph = FALSE)
    mspe2[, , i] <- mod$msp
  }
  toc <- proc.time() - tic

} else if ( model == "multinomial" )  {
  tic <- proc.time()
  for ( i in 1:da ) {
    z <- Compositional::alfa(x, a[i])$aff
    mod <- Compositional::multinompcr.tune(y, z, nfolds = nfolds,
    maxk = maxk, folds = folds, ncores = ncores, seed = seed, graph = FALSE)
    mspe2[, , i] <- mod$msp
  }
  toc <- proc.time() - tic

} else if ( model == "binomial"  |  model == "poisson" ) {
  tic <- proc.time()
  for ( i in 1:da ) {
    z <- Compositional::alfa(x, a[i])$aff
    mod <- Compositional::glmpcr.tune(y, z, nfolds = nfolds,
    maxk = maxk, folds = folds, ncores = ncores, seed = seed, graph = FALSE)
    mspe2[, , i] <- mod$msp
  }
  toc <- proc.time() - tic
```

```
  }

  dimnames(mspe2) <- list(folds = 1:nfolds, PC = paste("PC", 1:d, sep = ""), a = a)
  mspe <- array( dim = c(da, d, nfolds) )
  for (i in 1:nfolds)  mspe[, , i] <- t( mspe2[i, , 1:da] )
  dimnames(mspe) <- list(a = a, PC = paste("PC", 1:d, sep = ""), folds = 1:nfolds )
  mean.mspe <- t( colMeans( aperm(mspe) ) )   ## apply(mspe, 1:2, mean)
  if ( model == "multinomial" ) {
    best.par <- which(mean.mspe == max(mean.mspe), arr.ind = TRUE)[1, ]
  } else  best.par <- which(mean.mspe == min(mean.mspe), arr.ind = TRUE)[1, ]
  performance <- mean.mspe[ best.par[1], best.par[2] ]
  names(performance) <- "mspe"
  rownames(mean.mspe) <- a
  colnames(mspe) <- paste("PC", 1:d, sep = "")

  if ( graph )  filled.contour(a, 1:d, mean.mspe, xlab =
   expression( paste(alpha, " values") ), ylab = "Number of PCs", cex.lab = 1.3)
  best.par <- c( a[ best.par[1] ], best.par[2] )
  names(best.par) <- c("alpha", "PC")
  list(mspe = mean.mspe, best.par = best.par, performance = performance, runtime = toc)
}
```

8.7 Model based clustering for compositional data

This Section is about model based clustering and we will see how to use the EM algorithm for this purpose, simulate random data from a mixture model, choose the number of components using BIC and finally plot the contours of any model for compositional data in $\mathbb{S}^2$. We will make use of the R package *mixture* (Browne et al., 2015). The idea is simple, apply the additive (8.7) or the isometric (8.8) log-ratio transformation to the compositional data and then perform model based clustering on the transformed data.

8.7.1 Fitting a mixture model

The mixture model comprising of g components is written as

$$h\left(\mathbf{x}|\boldsymbol{\Theta}\right) = \sum_{i=1}^{g} \pi_i f_i\left(\mathbf{x}|\boldsymbol{\theta}_i\right),$$

where $\boldsymbol{\Theta} = \left(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_g\right)$ with $\boldsymbol{\theta}_i = \left(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)$ and $\mathbf{x} \in \mathbb{R}^p$. The π_is are the mixing probabilities, need to be estimated also. I will describe the EM algorithm briefly, because I am not an

expert, for this example.

The EM stands for *Expectation* and *Maximization*, the two steps of the algorithm. The key idea behind this algorithm is to perform likelihood maximization or parameter estimation when some information is missing. In our case, the missing information is the mixture probabilities, how many populations are there and which are their mixing probabilities from which the data were generated. The E step comes here, it calculates an expected value for this missing information. Then, with this knowledge, we can maximize the objective function and estimate its parameters.

The t-th step of the algorithm is briefly described below

E step. Estimate the probability of each observation belonging to a component by

$$p_{ij}^{t} = \frac{\pi_i^{t-1} f_i\left(\mathbf{x}|\boldsymbol{\theta}_i\right)}{\sum_{m=1}^{D} \pi_m^{t-1} f_m\left(\mathbf{x}|\boldsymbol{\theta}_m\right)}$$

M step. Update the parameters

$$\boldsymbol{\mu}_i^t = \frac{\sum_{j=1}^{n} p_{ij}\mathbf{x}_j}{\sum_{j=1}^{n} p_{ij}} \quad \text{and} \quad \boldsymbol{\Sigma}_i^t = \frac{\sum_{j=1}^{n} p_{ij}\left(\mathbf{x}_{ij}-\boldsymbol{\mu}_i^t\right)\left(\mathbf{x}_{ij}-\boldsymbol{\mu}_i^t\right)^T}{\sum_{j=1}^{n} p_{ij}} \quad (i=1,\ldots,g)$$

Step 3. Repeat the E and M steps until the log-likelihood does not increase any more.

(Browne et al., 2015) perform a K-means algorithm for initialization of the EM algorithm. Another point that is worthy to mention is that when (Fraley et al., 2012) wrote their R package *mclust* based on a paper by Fraley and Raftery (2002) allowed for 10 possible models. Browne et al. (2015) include all 14 possible models. When we say models, we mean different types of covariance matrices, listed below

1. "EII": All groups have the same diagonal covariance matrix, with the same variance for all variables.
2. "VII": Different diagonal covariance matrices, with the same variance for all variables within each group.
3. "EEI": All groups have the same diagonal covariance matrix.
4. "VEI": Different diagonal covariance matrices. If we make all covariance matrices have determinant 1, (divide the matrix with the p-th root of its determinant) then all covariance matrices will be the same.
5. "EVI": Different diagonal covariance matrices with the same determinant.
6. "VVI": Different diagonal covariance matrices, with nothing in common.

7. "EEE": All covariance matrices are the same.
8. "EEV": Different covariance matrices, but with the same determinant and in addition, if we make them have determinant 1, they will have the same trace.
9. "VEV": Different covariance matrices but if we make the matrices have determinant 1, then they will have the same trace.
10. "VVV": Different covariance matrices with nothing in common.
11. "EVE": Different covariance matrices, but with the same determinant. In addition, calculate the eigenvectors for each covariance matrix and you will see the extra similarities.
12. "VVE": Different covariance matrices, but they have something in common with their directions. Calculate the eigenvectors of each covariance matrix and you will see the similarities.
13. "VEE": Different covariance matrices, but if we make the matrices have determinant 1, then they will have the same trace. In addition, calculate the eigenvectors for each covariance matrix and you will see the extra similarities.
14. "EVV": Different covariance matrices, but with the same determinant.

As we can see, there are many combinations of similarities when the covariance matrices are diagonal and non diagonal. Below is the functions which utilises the *gpcm* function within the *mixture* R package

```
mix.compnorm <- function(x, g, model, type = "alr") {
  ## x is the compositional data
  ## g is the number of components to be used
  ## model is the type of model to be used
  ## type is either 'alr' or 'ilr'
  x <- as.matrix(x)  ## makes sure x is a matrix
  x <- x / Rfast::rowsums(x)  ## makes sure x is compositional
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size

  if (type == "alr") {
    y <- log(x[, -p]/x[, p])
  } else {
    y0 <- log(x)
    y1 <- y0 - Rfast::rowmeans( y0 )
```

```
    y <- tcrossprod( y1, helm(p) )

  }

  mod <- mixture::gpcm(y, G = g, mnames = model, start = 0, atol = 0.01)
  param <- mod$gpar
  mu <- matrix(nrow = g, ncol = length(param[[1]]$mu))
  su <- array(dim = c(length(param[[1]]$mu), length(param[[1]]$mu), g))
  for (i in 1:g) {
    mu[i, ] <- param[[i]]$mu  ## mean vector of each component
    su[, , i] <- param[[i]]$sigma  ## covariance of each component
  }
  prob <- param$pi  ## mixing probability of each component
  colnames(mu) <- colnames(su) <- colnames(y)
  ta <- matrix(nrow = n, ncol = g)
  for (j in 1:g) {
    ta[, j] <- -0.5 * log(det(2 * pi * su[, , j])) -
      0.5 * mahala(y, mu[j, ], su[, , j])
  }
  probexpta <- prob * exp(ta)
  pij <- probexpta / Rfast::rowsums(probexpta)
  est <- Rfast::rowMaxs(pij)
  list(type = type, mu = mu, su = su, prob = prob, est = est)
}
```

8.7.2 Choosing the optimal mixture model via BIC

BIC is used to choose the optimal model. So, first one has to run the next function (*bic.mixcompnorm*) and see which model has the lowest BIC and then use the *mix.compnorm* function for model based clustering.

```
bic.mixcompnorm <- function(x, A, type = "alr") {
  ## x is the compositional data
  ## A is the maximum number of components to be considered
  ## type is either 'alr' or 'ilr'
  x <- as.matrix(x)  ## makes sure x is a matrix
  x <- x / Rfast::rowsums(x)  ## makes sure x is compositional
  p <- dim(x)[2]  ## dimensionality of the data

  if (type == "alr") {
```

```
    y <- log(x[, -p]/x[, p])
  } else {
    y0
    y1 <- y0 - Rfast::rowmeans( y0 )
    y <- tcrossprod( y1, helm(p) )
  }

  mod <- mixture::gpcm(y, G = 1:A, start = 0, atol = 0.01)
  bic <- mod$BIC[, , 3]  ## BIC for all models
  ## Next, we plot the BIC for all models
  plot(1:A, bic[, 1], type = "b", pch = 9, xlab = "Number of components",
  ylab = "BIC values", ylim = c(min(bic, na.rm = T), max(bic, na.rm = T)))
  for (i in 2:nrow(bic)) lines(1:A, bic[, i], type = "b", pch = 9, col = i)
  list(mod = mod, BIC = bic)
}
```

8.7.3 Simulation of random values from a normal mixture model

In order to simulate random values from a normal mixture model for compositional data, the following R code is to be used.

```
rmixcomp <- function(n, prob, mu, sigma, type = "alr") {
  ## n is the sample size
  ## p is a vector with the mixing probabilities
  ## mu is a matrix with with the mean vectors
  ## sigma is an array with the covariance matrices
  p2 <- c(0, cumsum(prob))
  p <- ncol(mu)  ## dimensionality of the data
  u <- runif(n)
  g <- nrow(mu)  ## how many clusters are there
  ina <- as.numeric(cut(u, breaks = p2))  ## the cluster of each observation
  ina <- sort(ina)
  nu <- as.vector(table(ina))  ## frequency table of each cluster
  y <- array(dim = c(n, p, g))
  for (j in 1:g) {
    y[1:nu[j], , j] <- Rfast::rmvnorm( nu[j], mu[j, ], sigma[ , , j])
  }
  x <- y[1:nu[1], , 1]
  for (j in 2:g) {
    x <- rbind(x, y[1:nu[j], , j])
```

```
  }
  if (type == "alr") {
    x1 <- cbind(1, exp(x) )
    x <- x1 / Rfast::rowsums(x1)
  } else {
    x1 <- tcrossprod( x, helm( p + 1) )
    x2 <- exp(x1)
    x <- x2 / Rfast::rowsums( x2 )
  }
  ## x is the simulated data
  ## data come from the first cluster, then from the second and so on
  list(id = ina, x = x)
}
```

8.8 Discriminant analysis (classification) for compositional data

8.8.1 The k-NN algorithm with the power transformation for compositional data

The first algorithm we will see is the k-NN where the power transformation (8.16) will be used (not the α-transformation (8.15) basically). The algorithm needs a metric to be calculated. Tsagris et al. (2016b) advocates that as a measure of the distance between two compositions we can use the square root of the Jensen-Shannon divergence

$$ES - OV(\mathbf{x}, \mathbf{w}) = \left[\sum_{i=1}^{D} \left(x_i \log \frac{2x_i}{x_i + w_i} + w_i \log \frac{2w_i}{x_i + w_i} \right) \right]^{1/2}, \tag{8.33}$$

where $\mathbf{x}, \mathbf{w} \in S^d$.

Endres and Schindelin (2003) and Österreicher and Vajda (2003) proved, independently, that (8.33) satisfies the triangular identity and thus it is a metric. For this reason we will refer to it as the ES-OV metric.

We will use the power transformation (8.15) to define a more general metric termed ES-OV$_\alpha$ metric

$$ES - OV_\alpha(\mathbf{x}, \mathbf{w}) = \left[\sum_{i=1}^{D} \left(\frac{x_i^\alpha}{\sum_{j=1}^{D} x_j^\alpha} \log \frac{2 \frac{x_i^\alpha}{\sum_{j=1}^{D} x_j^\alpha}}{\frac{x_i^\alpha}{\sum_{j=1}^{D} x_j^\alpha} + \frac{w_i^\alpha}{\sum_{j=1}^{D} w_j^\alpha}} + \frac{w_i^\alpha}{\sum_{j=1}^{D} w_j^\alpha} \log \frac{2 \frac{w_i^\alpha}{\sum_{j=1}^{D} w_j^\alpha}}{\frac{x_i^\alpha}{\sum_{j=1}^{D} x_j^\alpha} + \frac{w_i^\alpha}{\sum_{j=1}^{D} w_j^\alpha}} \right) \right]^{1/2}. \tag{8.34}$$

The taxicab metric is also known as L_1 (or Manhattan) metric and is defined as

$$TC(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{D} |x_i - w_i| \tag{8.35}$$

We will again employ the power transformation (8.15) to define a more general metric which

we will term the TC_α metric

$$TC_\alpha(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{D} \left| \frac{x_i^\alpha}{\sum_{j=1}^{D} x_j^\alpha} - \frac{w_i^\alpha}{\sum_{j=1}^{D} w_j^\alpha} \right| \tag{8.36}$$

The last two power transformed metrics were suggested by Tsagris (2014), but only the case for $\alpha = 1$ in (8.34) was examined in Tsagris et al. (2016b). Aitchison (2003) suggested the Euclidean metric applied to the log-ratio transformed data as a measure of distance between compositions

$$Ait(\mathbf{x}, \mathbf{w}) = \left[\sum_{i=1}^{D} \left(\log \frac{x_i}{g(\mathbf{x})} - \log \frac{w_i}{g(\mathbf{w})} \right)^2 \right]^{1/2}, \tag{8.37}$$

where $g(\mathbf{z}) = \prod_{i=1}^{D} z_i^{1/D}$ stands for the geometric mean.

The power transformed compositional vectors still sum to 1 and thus the ES-OV_α (8.34) is still a metric. It becomes clear that when $\alpha = 1$ we end up with the ES-OV metric (8.33). If on the other hand $\alpha = 0$, then the distance is zero, since the compositional vectors become equal to the centre of the simplex. An advantage of the ES-OV_α metric (8.34) over the Aitchisonian metric (8.37) is that the the first one is defined even when zero values are present. In this case the Aitchisonian metric (8.37) becomes degenerate and thus cannot be used. We have to note that we need to scale the data so that they sum to 1 in the case of the ES-OV metric, but this is not a requirement of the taxicab metric.

Alternative metrics could be used as well, such as

1. the Hellinger metric (Owen, 2001)

$$H(\mathbf{x}, \mathbf{w}) = \frac{1}{\sqrt{2}} \left[\sum_{i=1}^{D} \left(\sqrt{x_i} - \sqrt{w_i} \right)^2 \right]^{1/2} \tag{8.38}$$

2. or the angular metric if we treat compositional data as directional data (for more information about this approach see Stephens (1982) and Scealy and Welsh (2014, 2011a,b))

$$Ang(\mathbf{x}, \mathbf{w}) = \arccos \left(\sum_{i=1}^{D} \sqrt{x_i} \sqrt{w_i} \right) \tag{8.39}$$

The algorithm is described below.

1. Separate the data into the training and the test dataset.
2. Choose a value of k, the number of nearest neighbours.
3. There are two possibilities here (not mentioned in Tsagris (2014)). One can use either the standard version or the non-standard version of the algorithm.
   (a) *Standard version*. Calculate the distances of a point in the test set $\mathbf{z}_0$ from all the points in the training set (there are ways to avoid all these calculations, but I did

this) and keep the k points in the training set which have the k smallest distances. Allocate the point $\mathbf{z}_0$ to the group which has the most of these k points. In case of ties, for example, 2 observations from group 1 and two observations from group 2 then, do the allocation randomly (again there are better ways, instead of randomness).

(b) *Non-Standard version*. This is what has been done in Tsagris (2014). Calculate the distances of $\mathbf{z}_0$ from all the points in the training set. For every group, keep the k points with the smallest distances and then calculate the average of these k distances for each group. Allocate $\mathbf{z}_0$ to the group with the minimum average of k distances. Another option offered here is to use the median of the k distances.

4. Classify the test data using either the ES-OV$_\alpha$ (8.34), the TC$_\alpha$ (8.36) for a range of values of α and each time calculate the percentage of correct classification. Alternatively, if the Aitchisonian (8.37), the Hellinger (8.38) or the angular distance (8.39) are used, the value of α is 1.

5. Repeat steps $2-3$ for a different value of k.

6. Repeat steps $1-4$ B (I did it with $B = 200$) times and for each α and k and estimate the percentage of correct classification by averaging over all B times.

The next function takes some compositional data whose groupings are known. For given values of α and for one or more values of k it will allocate some new data. You can specify which version to use, the standard or the non-standard, which metric of the aforementioned to use and whether the mean or the median will be used (in the case of the non-standard version).

```
comp.knn <- function(xnew, x, ina, a = 1, k = 5, type = "S", apostasi = "ESOV",
  mesos = TRUE) {
  n <- dim(x)[1]
  p <- dim(x)[2]
  ina <- as.numeric(ina)
  xnew <- as.matrix(xnew)
  xnew <- matrix(xnew, ncol = p ) ## makes sure xnew is a matrix
  nc <- max(ina)  ## The number of groups
  nu <- dim(xnew)[1]

  if (apostasi == "CS" & a == 0)  apostasi = "Ait"

  if ( (apostasi == "taxicab" | apostasi == "Ait" | apostasi == "Hellinger")  &
      type == "S" ) {
```

```
  if ( apostasi == "taxicab" ) {
    xa <- x^a
    zx <- xa / Rfast::rowsums( xa )  ## The power transformation is applied
    za <- xnew^a
    znew <- za / Rfast::rowsums( za )  ## The power transformation is applied
    g <- Rfast::knn(znew, ina, zx, k = k, dist.type = "mahattan", type = "C",
    freq.option = 1)
  } else if ( apostasi == "Ait" ) {
    xa <- Rfast::Log(x)
    zx <- xa - Rfast::rowmeans( xa )
    za <- Rfast::Log(xnew)
    znew <- za - Rfast::rowmeans( za )
    g <- Rfast::knn(znew, ina, zx, k = k, dist.type = "euclidean", type = "C",
    freq.option = 1)

  } else if ( apostasi == "Hellinger" ) {
    g <- Rfast::knn(sqrt(xnew), ina, sqrt(x), k = k, dist.type = "euclidean",
    type = "C", freq.option = 1)
  }

} else {
  ## all other methods
  if ( apostasi == "taxicab" ) {
    xa <- x^a
    zx <- xa / Rfast::rowsums( xa )  ## The power transformation is applied
    za <- xnew^a
    znew <- za / Rfast::rowsums( za )  ## The power transformation is applied
    disa <- Rfast::dista(znew, zx, "manhattan", trans = FALSE)

  } else if ( apostasi == "Ait" ) {
    xa <- Rfast::Log(x)
    zx <- xa - Rfast::rowmeans( xa )
    za <- Rfast::Log(xnew)
    znew <- za - Rfast::rowmeans( za )
    disa <- Rfast::dista(znew, zx, trans = FALSE)

  } else if ( apostasi == "Hellinger" ) {
    disa <- Rfast::dista(sqrt(xnew), sqrt(x), "euclidean", trans = FALSE)
```

```
} else if ( apostasi == "angular" ) {
  zx <- sqrt(x)
  znew <- sqrt(xnew)
  disa <- tcrossprod(zx, znew )
  disa[disa >= 1] <- 1
  disa <- acos(disa)

} else if ( apostasi == "ESOV" ) {
  disa <- matrix(0, n, nu)
  xa <- x^a
  zx <- xa / Rfast::rowsums( xa )  ## The power transformation is applied
  za <- xnew^a
  znew <- za / Rfast::rowsums( za )  ## The power transformation is applied
  tzx <- t(zx)
  for (i in 1:nu) {
    zan <- znew[i, ]
    ma <- tzx + zan
    disa[, i] <- colSums( zan * log( 2 * zan / ma ) + tzx * log( 2 * tzx/ma ),
    na.rm = TRUE )
  }
} else if ( apostasi == "CS" ) {
  xa <- x^a
  zx <- xa / Rfast::rowsums( xa )  ## The power transformation is applied
  za <- xnew^a
  znew <- za / Rfast::rowsums( za )  ## The power transformation is applied
  tzx <- t(zx)
  for (i in 1:nu) {
    znewi <- znew[i, ]
    sa <- ( tzx - znewi )^2 / ( zx + znewi )
    sa[is.infinite(sa)] <- 0
    disa[, i] <- Rfast::colsums( sa )
  }
  ## disa <- sqrt(disa) / abs(a) * sqrt(2 * p) not necessary to
  ## take the sqrt and then divide and multiply with constants everywhere
}

klen <- length(k)
if ( klen == 1 ) {
```

```
  if (type == "NS") {      ## Non Standard algorithm
    ta <- matrix(nrow = nu, ncol = nc)
    for (m in 1:nc) {
      apo <- disa[ina == m, ]
      apo <- Rfast::sort_mat(apo)
      if ( mesos ) {
        ta[, m] <- Rfast::colmeans( apo[1:k, , drop = FALSE] )
      } else  ta[, m] <- Rfast::colhameans( apo[1:k, , drop = FALSE] )
    }
    g <- as.matrix( Rfast::rowMins(ta) )

  } else {   ## if type is "S"   ## Standard algorithm
    g1 <- Rfast::colnth( disa, rep(k, nu) )
    g <- g1
    for (l in 1:nu) {
      ind <- which(disa[, l] <= g1[l] )
      a <- Rfast::Table( ina[ind] )
      b <- as.numeric( names(a) )
      g[l] <- b[which.max(a)]
    }
    g <- as.matrix(g)
  }  ## end if (type == "NS")

} else {  ## k has many values
  g <- matrix(0, nu, klen)
  if (type == "NS") {      ## Non Standard algorithm
    ta <- matrix(nrow = nu, ncol = nc)
    for (j in 1:klen) {
      for (m in 1:nc) {
        apo <- disa[ina == m, ]
        apo <- Rfast::sort_mat(apo)
        if ( mesos ) {
          ta[, m] <- Rfast::colmeans( apo[1:k[j], , drop = FALSE] )
        } else  ta[, m] <- Rfast::colhameans( apo[1:k[j], , drop = FALSE] )
      }
      g[, j] <- Rfast::rowMins(ta)
    }
```

```
      } else {   ## if type is "S"   ## Standard algorithm
        for (j in 1:klen) {
          g1 <- Rfast::colnth( disa, rep( k[j], nu) )
          for (l in 1:nu) {
            ind <- which(disa[, l] <= g1[l] )
            a <- Rfast::Table( ina[ind] )
            b <- as.numeric( names(a) )
            g[l, j] <- b[which.max(a)]
          }  ## end inner for
        } ## end outer for
      }  ## end if (type == "NS")
    }  ## end if (length(k) == 1)
  }  ## end of other methods
  colnames(g) <- paste("k=", k, sep = "")
  g
}
```

In order to choose the optimal pair of α and k, the metric and any extra arguments, you can use the next function. This function requires the fields library for the graphics, created by Nychka et al. (2015), but never the less, R has a built-in function for image plots should you wish not to download this package.

The idea, is to use a grid of values of α and k and for every combination of these two values to estimate the percentage of correct classification. Then, you can do the same for the different metrics and mean or median and so on. The estimation of the rate of correct classification is done in the same way as in the previous functions. K-fold cross-validation is performed and the estimated bias of the rate of correct classificationTibshirani and Tibshirani (2009) is subtracted from the highest estimated rate.

```
compknn.tune <- function(x, ina, nfolds = 10, A = 5, type = "S", mesos = TRUE,
                a = seq(-1, 1, by = 0.1), apostasi = "ESOV", folds = NULL,
                stratified = FALSE, seed = FALSE, graph = FALSE) {
  n <- dim(x)[1]  ## sample size
  ina <- as.numeric(ina)
  if ( A >= min(table(ina)) )  A <- min( table(ina) ) - 3  ## The maximum
  ## number  of nearest neighbours to use
  if ( min(x) == 0 )  a <- a[ a > 0 ]
  if ( is.null(folds) )  folds <- Compositional::makefolds(ina, nfolds = nfolds,
                                  stratified = stratified, seed = seed)
  nfolds <- length(folds)
  ## The algorithm is repated R times and each time the estimated
```

```
## percentages are stored in the array per.
if (apostasi == "ESOV" | apostasi == "taxicab" | apostasi == "CS") {
  a <- a[ a != 0 ]
}
per <- array( dim = c(nfolds, A - 1, length(a)) )
runtime <- proc.time()

for (vim in 1:nfolds) {
  id <- ina[ folds[[ vim ]] ]  ## groups of test sample
  ina2 <- ina[ -folds[[ vim ]] ]   ## groups of training sample
  aba <- as.vector( folds[[ vim ]] )
  aba <- aba[aba > 0]
  for ( i in 1:length(a) ) {
z <- x^a[i]
    z <- x / Rfast::rowsums( z )
    g <- comp.knn(z[aba, , drop = FALSE], z[-aba, ], ina2, a = NULL,
    k = 2:A, type = "S", apostasi = apostasi, mesos = mesos)
    be <- g - id
    per[vim, , i] <- Rfast::colmeans(be == 0)
  }
}
runtime <- proc.time() - runtime

if (apostasi == "Ait" | apostasi == "Hellinger" | apostasi == "angular" ) {

  ela <- Rfast::colmeans(per)
  performance <- max(ela)
  names(performance) <- "rate"
  names(ela) <- paste("k=", 2:A, sep = "")
  best_k <- which.max(ela) + 1

  if (graph)  plot(2:A, ela, type = "b", xlab = "k nearest neighbours",
                   pch = 9, col = 2, ylab = "Estimated percentage of correct
                   classification", cex.lab = 1.3)
  results <- list(ela = ela, performance = performance,
  best_k = which.max(ela) + 1, runtime = runtime)

} else {
  ela <- t( colMeans(per) )
```

```
    ## The ela matrix contains the averages of the R
    ## repetitions over alpha and k
    colnames(ela) <- paste("k=", 2:A, sep = "")
    rownames(ela) <- paste("alpha=", a, sep = "")
    ## The code for the heat plot of the estimated percentages
    if (graph)  fields::image.plot(a, 2:A, ela, col = grey(1:11/11),
                ylab = "k nearest-neighbours", xlab = expression(paste(alpha,
                " values")), cex.lab = 1.3 )

    performance <- max(ela)
    names(performance) <- c( "rate")
    confa <- which(ela == performance, arr.ind = TRUE)[1, ]
    results <- list( ela = ela, performance = performance, best_a =
    a[ confa[1] ], best_k = confa[2] + 1, runtime = runtime )
  }
  results
}
```

The function above allows for 1 processor to be used only. If your computer has more cores, then you can use the next function which uses the previous function. Also, make sure that R (the number of repetitions in the cross validation) is a multiple of *nc*, the number of cores to be used.

8.8.2 The k-NN algorithm with the α-metric

The idea is the same as before, but now we will use the α-distance (8.20) as a measure of distance between two compositional vectors. We remind the reader that the α-distance converges to the Euclidean distance applied to the centred log-ratio transformed data (8.21). See the relevant paper also (Tsagris et al., 2016b).

```
alfa.knn <- function(xnew, x, ina, a = 1, k = 5, type = "S", mesos = TRUE,
                     apostasi = "euclidean") {
  ## x is the matrix containing the data
  ## ina indicates the groups
  ## a is the value of the power parameter
  ## k in the number of nearest neighbours
  ## apostasi is the type of metric used, either "euclidean" or "manhattan"
  ## type is either S or NS. Should the standard k-NN be use or not
  ## if mesos is TRUE, then the arithmetic mean distance of the
  ## k nearest points will be used
```

```
  ## Both of these apply for the non-standard,
  ## algorithm, that is when type=NS
  ## xnew is the new dataset. It can be a single vector or a matrix
  p <- dim(x)[2]
  xnew <- as.matrix(xnew)
  xnew <- matrix(xnew, ncol = p)  ## makes sure xnew is a matrix
  ina <- as.numeric(ina)
  nc <- max(ina) ## The number of groups
  nu <- dim(xnew)[1]
  znew <- alfa(xnew, a, h = FALSE)$aff
  z <- alfa(x, a, h = FALSE)$aff

  if (type == "NS") {
    ## Non Standard algorithm
    klen <- length(k)
    g <- matrix(0, nu, klen)
    ta <- matrix(nrow = nu, ncol = nc)
    apo <- list()
    for (m in 1:nc) {
      disa <- Rfast::dista(znew, z[ina == m,], type = apostasi, trans = FALSE)
      apo[[ m ]] <- Rfast::sort_mat(disa)[1:max(k), ]
    }
    for (j in 1:klen) {
      for (m in 1:nc) {
        if ( mesos ) {
          ta[, m] <- Rfast::colmeans( apo[[ m ]][1:k[j], , drop = FALSE] )
        } else  ta[, m] <- Rfast::colhameans( apo[[ m ]][1:k[j], , drop = FALSE] )
      }
      g[, j] <- Rfast::rowMins(ta)
    }

  } else if (type == "S") {
    ## Standard algorithm
    g <- Rfast::knn(xnew = znew, y = ina, x = z, k = k, dist.type = apostasi,
    type = "C", freq.option = 1)
  }
  colnames(g) <- paste("k=", k, sep = "")
  g
}
```

The next function tunes the parameters α and k via cross-validation.

```
alfaknn.tune <- function(x, ina, M = 10, A = 5, type = "S", mesos = TRUE,
                a = seq(-1, 1, by = 0.1), apostasi = "euclidean",
                mat = NULL, graph = FALSE) {
  if ( min(x) == 0 )  a <- a[a>0]  ## checks for any zeros in the data
  n <- dim(x)[1]  ## sample size
  if ( A >= min( table(ina) ) )    A <- min( table(ina) ) - 3  ## The maximum
  ina <- as.numeric(ina) ## makes sure ina is numeric
  ela <- matrix(nrow = length(a), ncol = A - 1)
  colnames(ela) <- paste("k=", 2:A, sep = "")
  rownames(ela) <- paste("alpha=", a, sep = "")
  if ( is.null(mat) ) {
    nu <- sample(1:n, min( n, round(n / M) * M ) )
    ## It may be the case this new nu is not exactly the same
    ## as the one specified by the user
    ## to a matrix a warning message should appear
    suppressWarnings()
    mat <- matrix( nu, ncol = M ) # if the length of nu does not fit
  } else  mat <- mat
  M <- dim(mat)[2]

  if ( type == "S" ) {
    runtime <- proc.time()
    folds <- list()
    for (i in 1:M)  folds[[ i ]] <- mat[, i]
    ## Standard algorithm
    for (i in 1:length(a) ) {
      z <- alfa(x, a[i], h = FALSE)$aff
      ela[i, ] <- Rfast::knn.cv(folds = folds, nfolds = M, y = ina, x = z, k = 2:A,
      dist.type = apostasi, type = "C", freq.option = 1)$crit
    }
    runtime <- proc.time() - runtime
    if ( graph )  fields::image.plot(a, 2:A, ela, col = grey(1:11/11),
    ylab = "k nearest-neighbours", xlab = expression(paste(alpha, " values")) )
    opt <- max(ela)
    confa <- as.vector( which(ela == opt, arr.ind = TRUE)[1, ] )
    res <- list( ela = ela, performance = max(ela), best_a = a[ confa[1] ],
    best_k = confa[2] + 1, runtime = runtime )
    ## Non standard method
```

```
} else {
  per <- array( dim = c( M, A - 1, length(a) ) )  ## The estimated percentages
  for ( i in 1:length(a) ) {
    for (vim in 1:M) {
      id <- ina[ mat[, vim] ]   ## groups of test sample
      ina2 <- ina[ -mat[, vim] ]   ## groups of training sample
      aba <- as.vector( mat[, vim] )
      aba <- aba[aba > 0]
      g <- alfa.knn(x[aba, ], x[-aba, ], ina = ina2, a = a[i], k = 2:A, type = "NS",
      mesos = mesos, apostasi = apostasi)
      be <- g - id
      per[vim, , i] <- Rfast::colmeans(be == 0)
    }
  }
  for ( i in 1:length(a) )  ela[i, ] <- colMeans(per[, , i])
  runtime <- proc.time() - runtime
  if ( graph )  fields::image.plot(a, 2:A, ela, col = grey(1:11/11),
  ylab = "k nearest-neighbours", xlab = expression(paste(alpha, " values")) )
  opt <- max(ela)
  confa <- as.vector( which(ela == opt, arr.ind = TRUE)[1, ] )
  bias <- numeric(M)
  for (i in 1:M)  bias[i] <- opt - per[ i, confa[2], confa[1] ]
  bias <- mean(bias)
  performance <- c(opt - bias, bias)
  names(performance) <- c( "rate", "bias" )
  res <- list( ela = ela, performance = performance, best_a = a[ confa[1] ],
  best_k = confa[2] + 1, runtime = runtime )
}  ## end if (type == "S")
res
```

8.8.3 Regularised discriminant analysis with the α-transformation

Tsagris et al. (2016b) proposed a more general discriminant analysis for compositional data by employing the α-transformation (8.15). The idea is simple, apply the α-transformation and then use the classical regularised discriminant analysis we saw in Section 4.3.5. The function below predicts the group of new observations for some given values of α, δ and γ.

```
alfa.rda <- function(xnew, x, ina, a, gam = 1, del = 0) {
  ## xnew is the new compositional observation
  ## x contains the compositional data
```

```
  ## ina is the grouping variable
  ## a is the value of the alpha
  ## gam is between pooled covariance and diagonal
  ## gam*Spooled+(1-gam)*diagonal
  ## del is between QDA and LDA
  ## del*QDa+(1-del)*LDA
  y <- alfa(x, a)$aff  ## apply the alpha-transformation
  ynew <- alfa(xnew, a)$aff
  rda(xnew = ynew, x = y, ina = ina, gam = gam, del = del)
 }
```

In order to tune the values of the parameters we perform a cross validation. Parallel computing is an option in the function *rda.tune* and thus is an option here as well.

```
alfarda.tune <- function(x, ina, a = seq(-1, 1, by = 0.1), nfolds = 10,
                gam = seq(0, 1, by = 0.1), del = seq(0, 1, by = 0.1),
                ncores = 1, folds = NULL, stratified = TRUE, seed = FALSE) {
  ## x contains the compositonal data
  ## ina is the grouping variable
  ## a is the grid of values of a
  ## M is th number of folds
  ## ncores is the number of cores to be used
  ## if mat is NULL the folds happen internally
  ## if you already have folds, provide the indices of the data
  ## in a matrix form, each column corresponds to a fold
  ## gam is between pooled covariance and diagonal
  ## gam * Spooled+(1 - gam) * diagonal
  ## del is between QDA and LDA
  ## del * QDa + (1 - del) * LDA
  toc <- proc.time()
  n <- length(ina)
  if ( is.null(folds) )  folds <- Compositional::makefolds(ina,
       nfolds = nfolds, stratified = stratified, seed = seed)
  nfolds <- length(folds)
  ## if you have zero values, only positive alphas are allowed
  if ( min(x) == 0 )  a = a[ a > 0 ]
  info <- list()
  props <- ser <- array( dim = c( length(gam), length(del), length(a) ) )

  for ( k in 1:length(a) ) {
```

```
    z <- Compositional::alfa(x, a[k])$aff  ## apply the alpha-transformation
    mod <- Compositional::rda.tune(x = z, ina = ina, nfolds = nfolds,
    gam = gam, del = del, ncores = ncores,
                    folds = folds, stratified = stratified, seed = seed)
    ## since seed is TRUE, for every value of alpha, the same splits will occur
    ## thus, the percentages for all values of alpha are comparable
    props[, , k] <- mod$percent
    ser[, , k] <- mod$se
    info[[ k ]] <- mod$per
  }

  dimnames(props) <- list(gamma = gam, delta = del, a = a)
  opt <- apply(props, 3, max)
  names(opt) <- a
  percent <- props[ , , which.max(opt)]
  se <- ser[, , which.max(opt)]
  confa <- as.vector( which(props == max( props), arr.ind = TRUE )[1, ] )
  pera <- array( dim = c( length(gam), length(del), length(a) ) )
  opt <- props[ confa[1], confa[2], confa[3] ]
  seopt <- ser[ confa[1], confa[2], confa[3] ]
  res <- c( opt, seopt, a[ confa[3] ], gam[ confa[1] ], del[ confa[2] ] )
  names(res) <- c( "rate", "se of rate", "best_a", "best_gam", "best del" )
  runtime <- proc.time() - toc

  list(result = res,  percent = percent, se = se, runtime = runtime)
}
```

9 Circular (or angular) data

Another important field of statistics is the analysis of directional data. Directional data are data which lie on the circle, sphere and hypersphere (sphere in more than 3 dimensions). Some reference books include Fisher (1995) and Jammalamadaka and Sengupta (2001) for circular data, Fisher et al. (1987) for spherical data and Mardia and Mardia (1972) and Mardia and Jupp (2000) for directional statistics. A more recent book (for circular statistics only) written by Pewsey et al. (2013) contains a lot of R scripts as well. We will start with circular data and then move on to spherical and hyperspherical data. There are also some R packages, CircStats by Lund and Agostinelli (2012), circular by Agostinelli and Lund (2011) and NPCirc by Oliveira et al. (2013) (nonparametric smoothing methods) for circular data and movMF by Hornik and Grün (2014) for mixtures of von Mises-Fisher distribution (circular, spherical or hyper-spherical). The functions described here (and a few more) exist as an R package as well Directional Tsagris et al. (2016a).

The space of directional data is such that for any vector $\mathbf{x} \in \mathbb{R}^q$ with $q \geq 2$ we have that $\|\mathbf{X}\| = \mathbf{x}^T\mathbf{x} = 1$. This mean that $\mathbf{x}$ is a unit vector since its length is 1. The space of such vectors will be denoted by $\mathbb{S}^{q-1}$. If $q = 2$, the $\mathbf{x}$ lies on a circle and if $q = 3$ it lies on the surface of a sphere.

At first we start with the circular data analysis, that is, data defined on the circle. Thus their space is denoted by $\mathbb{S}^1$. Even though they can be treated as univariate I decided to include them here, first because they still need multivariate analysis some times and secondly, because they are good to know before we proceed to the spherical and hyper-spherical data.

9.1 Summary statistics

We will show how to calculate the sample mean direction, the sample mean resultant length and the sample circular variance.

Suppose we are given a sample of angular data $\mathbf{u} = (u_1, \ldots u_n)$ (angle of wind speed for example) in degrees or radians. We will suppose that the data are in radians (we provide a function to go from degrees to radians and backwards).

At first we have to transform the data to Euclidean coordinates $(\cos u_i, \sin u_i)^T$. Then we sum them component-wise to get

$$\bar{C} = \frac{1}{n}\sum_{i=1}^{n} \cos u_i \text{ and } \bar{S} = \frac{1}{n}\sum_{i=1}^{n} \sin u_i. \tag{9.1}$$

The sample circular mean, or mean direction is given by (Mardia and Jupp, 2000)

$$\bar{\theta} = \left\{ \begin{array}{cc} \tan^{-1}\left(\bar{S}/\bar{C}\right) & \text{if } \bar{C} > 0 \\ \tan^{-1}\left(\bar{S}/\bar{C}\right) + \pi & \text{if } \bar{C} < 0 \end{array} \right\}$$

If you do the above calculations in R, you will see that the outcome is not always correct. More conditions are required, see for example Jammalamadaka and Sengupta (2001). So, to avoid any confusion, I decided to use what Presnell et al. (1998) mentions; the angular mean will be given by

$$\hat{\theta} = \left[\tan^{-1}\left(\frac{\boldsymbol{\beta}_2^T \mathbf{x}_i}{\boldsymbol{\beta}_1^T \mathbf{x}_i} \right) + \pi I\left(\boldsymbol{\beta}_1^T \mathbf{x}_i < 0 \right) \right] mod 2\pi,$$

where I is the indicator function.

We will take the $\bar{C}$ and $\bar{S}$ and calculate the mean resultant length $\bar{R} = \sqrt{\bar{C}^2 + \bar{S}^2}$. The sample circular variance is given by $V = 1 - \bar{R}$ and thus $0 \leq V \leq 1$. Bear in mind that some authors multiply the variance by 2. The circular standard deviation is given by $\left(-2 \log \bar{R}\right)^{1/2}$ (Mardia and Jupp, 2000). Let us now construct a $(1-\alpha)\,\%$ confidence interval for the true mean angle μ. We can distinguish, two cases

- $\bar{R} \leq 2/3$

$$\bar{\theta} \pm \cos^{-1}\left\{ \left[\frac{2n\left(2R^2 - n\chi^2_{1,\alpha}\right)}{R^2\left(4n - \chi^2_{1,\alpha}\right)} \right]^{1/2} \right\}$$

- $\bar{R} > 2/3$

$$\bar{\theta} \pm \cos^{-1}\left\{ \frac{\left[n^2 - \left(n^2 - R^2\right)\exp\left(\chi^2_{1,\alpha}/n\right)\right]^{1/2}}{R} \right\}$$

9.2 The von Mises distribution

Another key thing we can estimate is the degree of concentration of the data. It is the analogue we can say of the variance in the data in the real. For this we have to assume that the data follow the von Mises distribution (Mardia and Jupp, 2000) whose density is

$$f(u) = \frac{1}{2\pi I_0(\kappa)} e^{\kappa \cos\left(u - \bar{\theta}\right)}, \tag{9.2}$$

where $u, \bar{\theta} \in [0, 2\pi]$, $\bar{theta}$ is the mean angle and $I_0(\kappa)$ is the modified Bessel function of the first function and order 0, defined as

$$I_0(\kappa) = \frac{1}{2\pi} \int_0^{2\pi} e^{\cos\phi} d\phi.$$

Maximization of the log-likelihood of (9.2) with respect to κ, which is strictly positive

and independent of $\bar{\theta}$, like the variance in the normal distribution, is what we have to do. We will see this distribution again in Section (10.7.1), but in its generalization for spherical and hyper-spherical data.

An option to plot the data on the unit circle is also given. We first construct a unit circle and then plot the $(\cos(u), \sin(u))$ pair of points. By default this option is set to *TRUE*. The R code with these summary measures is given below.

```
circ.summary <- function(u, rads = FALSE, fast = FALSE,
                         tol = 1e-09, plot = TRUE) {
  ## u is an angular variable
  if ( !rads )   u <- u * pi/180
  if (fast) {
    mod <- Rfast::vm.mle(u, tol = tol)
    res <- list(mesos = mod$param[1], kappa = mod$param[2], loglik = mod$loglik)
  } else {
    n <- length(u)  ## sample size
    ## if the data are in degrees we transform them into radians
    ## mesos contains the sample mean
    ## direction
    C <- sum( cos(u) ) / n
    S <- sum( sin(u) )/ n
    Rbar <- sqrt( C^2 + S^2 )  ## mean resultant length
    if (C > 0) {
      mesos <- atan(S/C)
    } else  mesos <- atan(S/C) + pi
    MRL <- Rbar  ## mean resultant length
    circv <- 1 - Rbar
    circs <- sqrt( -2 * log(Rbar) )  ## sample cicrular standard deviation
    ## lik is the von Mises likelihood
    lik <- function(k)   k * sum( cos(u - mesos) ) -
  n * ( log(besselI( k, 0, expon.scaled = TRUE) ) + k )
  mod <- optimize(lik, c(0, 50000), maximum = TRUE, tol = tol)
    kappa <- mod$maximum
    ## kappa is the estimated concentration (kappa)
    R <- n * Rbar
    if (Rbar < 2/3) {
      fact <- sqrt(2 * n * ( 2 * R^2 - n * qchisq(0.05, 1) ) /
  ( R^2 * ( 4 * n - qchisq(0.05, 1)) ) )
      ci <- c(mesos - acos(fact), mesos + acos(fact))
    } else  {
```

```
      fact <- sqrt( n^2 - (n^2 - R^2) * exp( qchisq(0.05, 1)/n ) )/R
      ci <- c(mesos - acos(fact), mesos + acos(fact))
    }
    if ( !rads ) {
      mesos <- mesos * 180/pi
      ci <- ci * 180/pi
    }
    if ( plot ) {
      r <- seq(0, 2 * pi, by = 0.01)
      plot(cos(r), sin(r), type = "l", xlab = "Cosinus", ylab = "Sinus")
      xx <- seq(-1, 1, by = 0.1)
      yy <- seq(-1, 1, by = 0.1)
      ta <- numeric(length(xx))
      lines(ta, xx, type = "l", lty = 2)
      lines(yy, ta, lty = 2)
      points(cos(u), sin(u))
    }
    res <- list( mesos = mesos, confint = ci, kappa = kappa, MRL = MRL,
circvariance = circv, circstd = circs, loglik = mod$objective - n * log(2 * pi) )
  }
  res
}
```

9.3 Simulation of random values from the von Mises distribution

We will provide a function for simulating random values from the von Mises distribution. Bear in mind that this distribution is a special case of the von Mises-Fisher distribution, when $p = 2$, that is when we are in the circle. For this reason we will use the *rvmf* function which simulates values from the von Mises-Fisher distribution. The values are expressed in the cartesian coordinates, they are unit vectors which have to be transformed to radians or degrees, regarding what the user specifies. For more information on the simulation the reader is requested to see Section 10.8.1. To turn the unit vector into radians we will use the same transformation we will see in (9.9) in the projected bivariate regression for circular data.

```
rvonmises <- function(n, m, k, rads = TRUE) {
  ## n is the sample size
  ## m is the mean angle expressed in radians or degrees
  ## k is the concentration parameter
  if ( !rads ) m <- m/180 * pi  ## turn degrees to radians
```

```
  mu <- c(cos(m), sin(m))
  if (k > 0) {  ## draw from a von Mises distribution
    x <- rvmf(n, mu, k)  ## sample from a von Mises
    ## x is expressed in cartesian coordinates. We have to
    ## transform the values into degrees or radians
    u <- (atan(x[, 2]/x[, 1]) + pi * I(x[, 1] < 0)) %% (2 * pi)
  } else u <- runif(n, 0, 2 * pi)  ## draw from a von Mises
  if ( !rads ) u <- u * pi/180  ## should the data be in degrees?
  u
}
```

9.4 Kernel density estimation using a von Mises kernel

The von Mises kernel is (García-Portugués, 2013, Taylor, 2008)

$$\hat{f}(\theta, h) = \frac{1}{n 2\pi I_0(1/h^2)} \sum_{i=1}^{n} e^{\frac{\cos(\theta - \theta_i)}{h^2}}, \tag{9.3}$$

where n denotes the sample size and h is the bandwidth parameter whose value we have to choose.

Taylor (2008) provided a "plug-in rule" for the value of h which is based on the minimum asymptotic mean integrated squared error (AMISE). Taylor (2008) mentions this is of a similar asymptotic form as the normal-scale plug in rule. I present the formula given by García-Portugués (2013), because it looks better

$$h_{TAY} = \left[\frac{4\pi^{\frac{1}{2}} I_0(\hat{\kappa})^2}{3\hat{\kappa}^2 n I_2(2\hat{\kappa})} \right]^{1/5}$$

García-Portugués (2013) provided a new rule of thumb which is the directional analogue of the rule of thumb of Silverman (1986)

$$h_{ROT} = \left\{ \frac{4\pi^{\frac{1}{2}} I_0(\hat{\kappa})^2}{\hat{\kappa} n \left[2I_1(2\hat{\kappa}) + 3\hat{\kappa} I_2(2\hat{\kappa}) \right]} \right\}^{1/5}.$$

In both cases, $\hat{\kappa}$ is the estimated (see the *circ.summary* function) concentration parameter. Alternatively, we can use maximum likelihood cross validation (5.5) just as in the multivariate kernel density estimation for Euclidean data we saw before. So, choose h which maximises

$$MLCV(h) = \frac{1}{n} \sum_{i=1}^{n} \log \left[\hat{f}_{-i}(\mathbf{x}_i; h) \right],$$

where $\hat{f}_{-i}(\mathbf{x}_i; h)$ is the von Mises kernel (9.3) of the i-th observation, calculated without it.

The *vm.kde* function calculates the density estimate of each point in a sample using either a rule of thumb or another h.

```
vm.kde <- function(u, h = NULL, thumb = "none", rads = TRUE) {
  ## u is the data
  ## h is the bandwidth you want
  ## thumb is either 'none' (defualt), or 'tay' (Taylor, 2008) or
  ## 'rot' (Garcia-Portugues, 2013)
  ## if the data are in degrees we transform them into radians

  if ( !rads )  u <- u/180 * pi
  n <- length(u)  ## sample size
  x <- cbind( cos(u), sin(u) )
  disa <- tcrossprod(x)
  diag(disa) <- 1
  expa <- exp(disa)

  if ( is.null(h) ) {

    if (thumb == "tay") {
      k <- circ.summary(u, rads = TRUE, plot = FALSE)$kappa
      h <- ( (4 * pi^0.5 * besselI(k, 0)^2)/(3 * n * k^2 *
      besselI(2 * k, 2)) )^0.2

    } else if (thumb == "rot") {
      k <- circ.summary(u, rads = TRUE, plot = FALSE)$kappa
      h <- ( (4 * pi^0.5 * besselI(k, 0)^2) /( k * n * ( 2 * besselI(2 * k, 1) +
      3 * k * besselI(2 * k, 2)) ) )^0.2

    } else if (thumb == "none") {
      h <- as.numeric( vmkde.tune(u, low = 0.1, up = 1)[1] )
    }

  } else h <- h

  f <- rowSums( expa^( 1 / h^2 ) ) / ( n * 2 * pi * besselI(1/h^2, 0) )

  list( h = h, f = as.vector(f) )
}
```

The *vmkde.tune* chooses h using maximum likelihood cross validation (5.5).

```
vmkde.tune_2 <- function(u, h = seq(0.1, 1, by = 0.01), rads = T,
  plot = TRUE, ncores = 4) {
  ## u is the data
  ## h is the bandwidth grid of values
  ## nc is the number of cores you want to use
  require(doParallel, quiet = TRUE, warn.conflicts = FALSE)
  n <- length(u)  ## sample size
  ## if the data are in degrees we transform them into radians
  if ( !rads ) u <- u/180 * pi
  x <- cbind( cos(u), sin(u) )
  disa <- tcrossprod(x)
  expa <- exp(disa)
  diag(expa) <-  -Inf  ## we do not want the diagonal elements

  options(warn = -1)
  nc <- ncores
  val <- matrix(h, ncol = nc) ## if the length of h is not equal to the
  ## dimensions of the matrix val a warning message should appear
  ## but with suppressWarnings() you will not see it

  cl <- makePSOCKcluster(nc)
  registerDoParallel(cl)
  ww <- foreach(j = 1:nc, .combine = cbind) %dopar% {
   ba <- val[, j]
    for (l in 1:length(val[, j])) {
      A <- expa^( 1 / ba[l]^2 )
      f <- rowSums(A)/((n - 1) * 2 * pi * besselI( 1/ba[l]^2, 0) )
      ba[l] <- mean( log(f) )
    }
   return(ba)
  }
  stopCluster(cl)

  cv <- as.vector(ww)[1:length(h)]
  if ( plot ) {
    plot(h, cv, type = "l")
  }
```

```
  list(hopt = h[which.max(cv)], cv = cv)
}
```

The next code is a faster version of the previous, especially if you do not have a multi-core computer.

```
vmkde.tune <- function(u, low = 0.1, up = 1, rads = TRUE) {
  ## u is the data
  n <- length(u)  ## sample size
  ## if the data are in degrees we transform them into radians
  if ( !rads )  u <- u/180 * pi
  x <- cbind( cos(u), sin(u) )
  disa <- tcrossprod(x)
  expa <- exp(disa)
  diag(expa) <-  - Inf  ## we do not want the diagonal elements
  con <- 2 * (n - 1) * pi

  funa <- function(h) {
    A <- expa^( 1 / h^2 )
    f <- rowSums( A )
    sum( log(f) ) - n * log( besselI(1/h^2, 0) )
  }

  bar <- optimize(funa, c(low, up), maximum = TRUE)
  res <- c( bar$maximum, bar$objective/n - log(con) )
  names(res) <- c("Optimal h", "cv")
  res
}
```

9.5 Analysis of variance for circular data

We will see three ways of performing hypothesis testing for the mean directions of two or more samples. In all cases, equality of the concentration parameters is assumed, similarly to the classical analysis of variance. The null hypothesis H_0 is

$$\bar{\theta}_1 = \ldots = \bar{\theta}_g,$$

where g denotes the number of samples we have.

9.5.1 High concentration F test

Let us define $C = \bar{C}$ and $S = \bar{S}$, where $\bar{C}$ and $\bar{S}$ are defined in (9.1). The resultant length is given by $R = \sqrt{C^2 + S^2}$ or as $R = n\bar{R}$. Let R_i define the resultant length of each group and R the resultant length of all the data together, assuming they are one sample. Finally, let n_i denote the sample size of the i-th sample, where $i = 1, \ldots, g$ and $n = \sum_{i=1}^{g} n_i$. The test statistic is

$$F_{ww} = \frac{(n-g)\left(\sum_{i=1}^{g} R_i - R\right)}{(g-1)\left(n - \sum_{i=1}^{g} R_i\right)}.$$

Under H_0, F_{ww} follows asymptotically an $F_{g-1,n-q}$. This approximation can be refined, as Mardia and Jupp (2000) mention, to

$$\left(1 + \frac{3}{8\hat{\kappa}}\right) F_{ww}.$$

Mardia and Jupp (2000) mention that this refinement is adequate for $\kappa \geq 1$, i.e. $\bar{R} \geq 0.45$ and that for $\hat{\kappa} \geq 10$ the refinement factor is negligible. They cite Stephens (1972) for this information. I wrote the following R code based on the information I found in Jammalamadaka and Sengupta (2001), who mention that if $1 \leq \hat{\kappa} \leq 2$, the refinement can be used.

How do we estimate this common $\hat{\kappa}$, or, this pooled estimate? In order to estimate κ in the one sample case we have to maximise the log-likelihood of the von Mises distribution with respect to κ. Mardia and Jupp (2000) use the following equation for estimating the $\hat{\kappa}$

$$A_1(\hat{\kappa}) = \bar{R}, \tag{9.4}$$

where $\bar{R}$, I remind the reader, is the mean resultant length of the combined sample, assuming that all samples are one sample. A solver, like *uniroot* in R, is used to find the solution to (9.4).

```
hcf.circaov <- function(u, ina, rads = FALSE) {
  ## u contains all the circular data in radians or degrees
  ## ina is an indicator variable of each sample

  n <- length(u)  ## sample size
  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there

  ## if the data are in degrees we transform them into radians
  if ( !rads )  u <- u * pi/180
```

```
  x1 <- cos(u)
  x2 <- sin(u)
  Ci <- rowsum(x1, ina)
  Si <- rowsum(x2, ina)

  Ri <- sqrt( Ci^2 + Si^2 )
  ## Ri is the resultant length of each group
  V <- sum(Ri)
  C <- sum(x1)
  S <- sum(x2)

  R <- sqrt(C^2 + S^2)  ## the resultant length based on all the data
  ## Next we stimate the common concentration parameter kappa

  kappa <- circ.summary(u, rads = T, plot = F)$kappa
  ## kappa is the estimated concentration parameter based on all the data
  if (kappa > 2) {
    Ft <- ( (n - g) * (V - R) )/( (g - 1) * (n - V) )
    pvalue <- pf(Ft, g - 1, n - g, lower.tail = FALSE)

  } else  if (kappa < 2 & kappa > 1) {
    Ft <- (1 + 3/(8 * kappa)) * ((n - g) * (V - R))/( (g - 1) * (n - V) )
    pvalue <- pf(Ft, g - 1, n - g, lower.tail = FALSE)

  } else {
    Ft <- NA
    pvalue <- NA
  }

  res <- c(Ft, pvalue, kappa)
  names(res) <- c('test', 'p-value', 'kappa')
  res

}
```

9.5.2 Log-likelihood ratio test

We have g sample of u_i observations. At first we transform them in the Euclidean coordinates $\mathbf{x}_i = (\cos u_i, \sin u_i)$. Let $\bar{\mathbf{x}}_{i.}$ and $\bar{\mathbf{x}}_{..}$ define the mean direction of the i-th sample and of

the combined sample respectively, defined as

$$\bar{\mathbf{x}}_{i.} = \frac{\bar{\mathbf{u}}_{i.}}{\|\bar{\mathbf{u}}_{i.}\|} \text{ and } \bar{\mathbf{x}}_{..} = \frac{\bar{\mathbf{u}}_{..}}{\|\bar{\mathbf{u}}_{..}\|}.$$

The log-likelihood ratio test takes the following form

$$w = \hat{\kappa} \sum_{i=1}^{g} R_i \|\bar{\mathbf{u}}_{i.} - \bar{\mathbf{u}}_{..}\|^2.$$

Alternatively, we can write it as $w = 2\hat{\kappa} \sum_{i=1}^{g} R_i \left[1 - \cos\left(\bar{\theta}_i - \bar{\theta}\right)\right]$, where $\bar{\theta}$ is the mean of the combined sample. Under the null hypothesis w follows asymptotically a χ^2_{g-1}. In the following R code I used the first form.

```
lr.circaov <- function(u, ina, rads = FALSE) {
  ## u contains all the circular data in radians or degrees
  ## ina is an indicator variable of each sample

  n <- length(u)  ## sample size
  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there

  ## if the data are in degrees we transform them into radians
  if (rads == F)  u <- u * pi/180
  x <- cbind(cos(u), sin(u))
  Ci <- rowsum(x[, 1], ina)
  Si <- rowsum(x[, 2], ina)
  Ri <- sqrt(Ci^2 + Si^2)  ## the resultant length of each group

  ni <- tabulate(ina)
  mi <- rowsum(x, ina) / ni
  mi <- mi/sqrt(rowSums(mi^2))  ## mean direction of each group

  m <- Rfast::colmeans(x)
  m <- m/sqrt(sum(m^2))  ## mean direction based on all the data
  m <- matrix(rep(m, g), nrow = g, byrow = T)

  R <- sqrt( sum( colSums(x)^2 ) )  ## the resultant length based on all the data

  ## Next we estimate the common concentration parameter kappa
  kappa <- circ.summary(u, rads = T, plot = F)$kappa
```

```
  ## kappa is the estimated concentration parameter based on all the data

  w <- kappa * sum(Ri * rowSums((mi - m)^2))
  pvalue <- 1 - pchisq(w, g - 1)
  res <- c(w, pvalue, kappa)
  names(res) <- c('test', 'p-value', 'kappa')
  res

}
```

9.5.3 Embedding approach

A third test statistic mentioned in Mardia and Jupp (2000) is based on the embedding approach. I give the test statistic straight away with not too much theory (the interested reader will see in Mardia and Jupp (2000) that it is less than 1 page) which is similar to the high concentration test statistic we saw before. Mardia and Jupp (2000) has a mistake in the form of this statistic (Equations (7.4.15) and (7.4.16)) on page 139. The factor n is missing and I show the correct form here.

$$F = \frac{(n-g)\left(\sum_{i=1}^{g} n_i \bar{R}_i^2 - n\bar{R}^2\right)}{(g-1)\left(n - \sum_{i=1}^{g} n_i \bar{R}_i^2\right)}$$

Under H_0, F follows asymptotically an $F_{g-1,n-g}$ distribution. They also mention a corrected version and that is what I use in the relevant R function

$$F_c = \left(1 - \frac{1}{5\hat{\kappa}} - \frac{1}{10\hat{\kappa}^2}\right) F,$$

where $\hat{\kappa}$ is the estimate of the common concentration parameter.

```
embed.circaov <- function(u, ina, rads = FALSE) {
  ## u contains all the circular data in radians or degrees
  ## ina is an indicator variable of each sample

  n <- length(u)  ## sample size
  ina <- as.numeric(ina)
  ni <- tabulate(ina)
  g <- max(ina)  ## how many groups are there
  ## if the data are in degrees we transform them into radians

  if ( !rads )  u <- u * pi/180
  x1 <- cos(u)
```

```
  x2 <- sin(u)
  Ci <- rowsum(x1, ina)
  Si <- rowsum(x2, ina)
  Rbi <- sqrt( Ci^2 + Si^2 )/ni

  C <- sum(Ci)
  S <- sum(Si)
  Rbar <- sqrt(C^2 + S^2)/n  ## the mean resultant length based on all the data
  ## Next we estimate the common concentration parameter kappa

  kappa <- circ.summary(u, rads = TRUE, plot = FALSE)$kappa
  ## kappa is the estimated concentration parameter based on all the data

  Fb <- ( (sum(ni * Rbi^2) - n * Rbar^2 )/(g - 1) ) /
    ( (n - sum(ni * Rbi^2) )/(n - g) )
  Fc <- ( 1 - 1/(5 * kappa) - 1/(10 * kappa^2) ) * Fb
  pvalue <- pf(Fc, g - 1, n - g, lower.tail = FALSE)
  res <- c(Fc, pvalue, kappa)
  names(res) <- c('test', 'p-value', 'kappa')
  res

}
```

9.5.4 A test for testing the equality of the concentration parameters

Mardia and Jupp (2000) provides a test for testing the equality of the concentration parameter among g samples, where $g \geq 2$. There are three distinct cases, based on the value of $\bar{R}$, the mean resultant length of all data.

- *Case I*. $\bar{R} < 0.45$. The test statistic has the following form

$$U_1 = \sum_{i=1}^{g} w_i g_1 \left(2\bar{R}_i\right)^2 - \frac{\left[\sum_{i=1}^{g} w_i g_1 \left(2\bar{R}_i\right)\right]^2}{\sum_{i=1}^{g} w_i},$$

  where $w_i = \frac{4(n_i - 4)}{3}$, with n_i denoting the sample size of the i-th group and $g_1(x) = \sin^{-1}\left(\sqrt{\frac{3}{8}}x\right)$. $\bar{R}_i$ is the mean resultant length of the i-th group.

- *Case II*. $0.45 \leq \bar{R} \leq 0.70$. The test statistic now becomes

$$U_2 = \sum_{i=1}^{g} w_i g_2 \left(\bar{R}_i\right)^2 - \frac{\left[\sum_{i=1}^{g} w_i g_2 \left(\bar{R}_i\right)\right]^2}{\sum_{i=1}^{g} w_i},$$

where $w_i = \frac{n_i - 3}{0.798}$ and $g_2(x) = \sinh^{-1}\left(\frac{x - 1.089}{0.258}\right)$.

- *Case III.* $\bar{R} > 0.70$. For this high concentration case the test statistic is

$$U_3 = \frac{1}{1+d}\left[\nu \log\left(\frac{n - \sum_{i=1}^{g} R_i}{\nu}\right) - \sum_{i=1}^{g} \nu_i \log\left(\frac{n_i - R_i}{\nu_i}\right)\right],$$

where $\nu_i = n_i - 1$, $\nu = n - g$ and $d = \frac{1}{3(g-1)}\left(\sum_{i=1}^{g} \frac{1}{\nu_i} - \frac{1}{\nu}\right)$.

Under H_0, each U_i, with regards to each case, follows asymptotically a χ^2_{g-1}.

```
conc.test <- function(u, ina, rads = FALSE) {
  ## u contains all the circular data in rads or degrees
  ## ina is an indicator variable of each sample
  n <- length(u)  ## sample size
  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there
  ni <- tabulate(ina)
  if ( !rads )  u <- u * pi/180

  ## if the data are in degrees we transform them into radians
  x1 <- cos(u)
  x2 <- sin(u)
  Ci <- rowsum(x1, ina) / ni
  Si <- rowsum(x2, ina) / ni
  Rbi <- sqrt( Ci^2 + Si^2 )

  ## Ri is the mean resultant length of each group
  C <- sum(x1)/n
  S <- sum(x2)/n
  Rb <- sqrt(C^2 + S^2)  ## the mean resultant length of all the data

  if (Rb < 0.45) {
    ## case 1
    g1 <- wi <- numeric(g)
    wi <- (4 * (ni - 4))/3
    g1 <- asin( sqrt(3/8) * 2 * Rbi )
    U1 <- sum(wi * g1^2) - ( sum(wi * g1) )^2 / sum(wi)
    pvalue <- pchisq(U1, g - 1, lower.tail = FALSE)
    mess <- paste('The mean resultant length is less than 0.45. U1 was calculated')
```

```
  } else if (Rb >= 0.45 & Rb <= 0.7) {
    ## case 2
    g2 <- wi <- numeric(g)
    wi <- (ni - 3)/0.798
    g2 <- asinh( (Rb - 1.089) / 0.258 )
    U2 <- sum( wi * g2^2) - (sum(wi * g2) )^2/sum(wi)
    pvalue <- pchisq(U2, g - 1, lower.tail = FALSE)
    mess <- paste('The mean resultant length is between 0.45 and 0.7.
    U3 was calculated')

  } else  if (Rb > 0.7) {
    ## case 3
    Ri <- Rbi * ni
    vi <- ni - 1
    v <- n - g
    d <- 1/(3 * (g - 1)) * (sum(1/vi) - 1/v)
    U3 <- 1/(1 + d) * (v * log((n - sum(Ri))/v) - sum(vi * log((ni - Ri)/vi)))
    pvalue <- pchisq(U3, g - 1, lower.tail = FALSE)
    mess <- paste('The mean resultant length is more than 0.7. U3 was calculated')
  }

  res <- c(U3, pvalue)
  names(res) <- c('test', 'p-value')
  list(mess = mess, res = res)

}
```

9.5.5 Tangential approach for testing the equality of the concentration parameters

Mardia and Jupp (2000) refer to Fisher (1995) who recommends this test on the grounds of its robustness against outliers and departure from the von Mises distribution. The test statistic is

$$F = \frac{(n-g)\sum_{i=1}^{g} n_i \left(\bar{d}_i - \bar{d}\right)^2}{(g-1)\sum_{i=1}^{g}\sum_{j=1}^{n_i}\left(d_{ij} - \bar{d}_i\right)^2},$$

where

$$d_{ij} = \left|\sin\left(u_{ij} - \bar{\theta}_i\right)\right|, \ j = 1, \ldots, n_i$$
$$\bar{d}_i = \frac{1}{n_i}\sum_{i=1}^{g} d_{ij} \text{ and } \bar{d} = \frac{1}{n}\sum_{i=1}^{g}\sum_{j=1}^{n_i} d_{ij}.$$

Under H_0, that the concentration parameters are equal, F follows asymptotically an $F_{g-1,n-g}$.

```
tang.conc <- function(u, ina, rads = FALSE) {
  ## u contains all the circular data in radians or degrees
  ## ina is an indicator variable of each sample
  n <- length(u)  ## sample size
  ina <- as.numeric(ina)
  ni <- tabulate(ina)
  g <- max(ina)  ## how many groups are there
  ## if the data are in degrees we transform them into radians
  if ( !rads )   u <- u * pi/180
  d <- NULL  ## will store the absolute sinus centred data here
  d2 <- dmi <- numeric(g)
  x1 <- cos(u)
  x2 <- sin(u)
  C <- rowsum(x1, ina)
  S <- rowsum(x2, ina)
  mi <- atan(S/C) + pi * as.numeric(C<0)
  for (i in 1:g) {
    b <- abs( sin( u[ ina == i ] - mi[i] ) )
    d <- c(d, b)
  }

  dmi <- Rfast::group.mean(d, ina)
  d2 <- Rfast::group.var(d, ina) * (ni - 1)
  mdm <- mean(d)
  Ft <- (n - g) * sum(ni * (dmi - mdm)^2) / ( (g - 1) * sum(d2) )
  pvalue <- pf(Ft, g - 1, n - g, lower.tail = FALSE)
  res <- c(Ft, pvalue)
  names(res) <- c('test', 'p-value')
  res
}
```

9.5.6 Analysis of variance without assuming equality of the concentration parameters

For the heterogeneous case, when the concentration parameters cannot be assumed to be equal, Mardia and Jupp (2000) provides us with a test statistic. The form of this statistic is

$$T = 2\left(\sum_{i=1}^{g} \hat{\kappa}_i R_i - R_w\right),$$

where

$$R_w = \left[\left(\sum_{i=1}^{g} \hat{\kappa}_i R_i \cos\bar{\theta}_i\right)^2 + \left(\sum_{i=1}^{g} \hat{\kappa}_i R_i \sin\bar{\theta}_i\right)^2\right]^{1/2}$$

and the $\hat{\kappa}_i$ and R_i quantities are estimated for each sample separately. Under H_0, T follows a χ^2_{g-1} distribution. Mardia and Jupp (2000) informs us that this test was introduced by Watson (1983a).

```
het.circaov <- function(u, ina, rads = FALSE) {
  ## u contains all the circular data in radians or degrees
  ## ina is an indicator variable of each sample
  n <- length(u)  ## sample size
  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there
  ni <- tabulate(ina)
  ## if the data are in degrees we transform them into radians

  if ( !rads )  u <- u * pi/180
  kappa <- numeric(g)
  x1 <- cos(u)
  x2 <- sin(u)
  C <- rowsum(x1, ina)
  S <- rowsum(x2, ina)

  mi <- atan(S/C) + pi * as.numeric(C<0)
  Ri <- sqrt(C^2 + S^2)  ## the resultant length of each group
  for (i in 1:g) {
    kappa[i] <- circ.summary( u[ina == i], rads = TRUE, plot = FALSE )$kappa
  }
  ## kappa contains the estimated concentration parameters of each group
  Rw <- sqrt( (sum(kappa * Ri * cos(mi)) )^2 + ( sum( kappa * Ri * sin(mi) ) )^2 )
  Ta <- 2 * ( sum(kappa * Ri) - Rw )
```

```
  pvalue <- pchisq(Ta, g - 1, lower.tail = FALSE)
  res <- c(Ta, pvalue)
  names(res) <- c('test', 'p-value')
  res

}
```

9.6 Circular correlation

9.6.1 Circular-circular correlation I

Jammalamadaka and Sarma (1988) suggested a correlation coefficient for a sample of pairs of angular data (α_i, β_i) with $i = 1, \ldots, n$. The correlation is defined as

$$r_c = \frac{\sum_{i=1}^{n} \sin\left(\alpha_i - \bar{\alpha}\right) \sin\left(\beta_i - \bar{\beta}\right)}{\sqrt{\sum_{i=1}^{n} \sin^2\left(\alpha_i - \bar{\alpha}\right) \sum_{i=1}^{n} \sin^2\left(\beta_i - \bar{\beta}\right)}}, \tag{9.5}$$

where $\bar{\alpha}$ and $\bar{\beta}$ are the mean directions of the two samples. We saw in the previous section how to calculate them. Jammalamadaka and Sengupta (2001) states that under a suitable transformation we can get asymptotic normality and thus perform the hypothesis testing of zero correlation. If the sample size n is large enough, then under the null hypothesis that the true correlation is zero we have that

$$\sqrt{n}\sqrt{\frac{\hat{\lambda}_{02}\hat{\lambda}_{20}}{\hat{\lambda}_{22}}} r_c \sim N\left(0, 1\right),$$

where

$$\hat{\lambda}_{ij} = \frac{1}{n}\sum_{i=1}^{n} \sin^i\left(\alpha_i - \bar{\alpha}\right) \sin^j\left(\beta_i - \bar{\beta}\right).$$

This is an asymptotic normality based test and below I provide the relevant R code.

```
circ.cor1 <- function(theta, phi, rads = FALSE) {
  ## theta and phi are angular data in degrees or radians
  ## by default they are in degrees
  n <- length(theta)  ## sample size

  ## if the data are in degrees we transform them into radians
  if ( !rads ) {
    theta <- theta * pi/180
    phi <- phi * pi/180
```

```
  }

  ## We calculate the mean of each vector
  m1 <- circ.summary( theta, rads = TRUE, plot = FALSE )$mesos
  m2 <- circ.summary( phi, rads = TRUE, plot = FALSE )$mesos
  sintheta <- sin(theta - m1)
  sinphi <- sin(phi - m2)

  up <- sum( sintheta * sinphi )
  down <- sqrt( sum( sintheta ^2 ) * sum( sinphi^2 ) )
  rho <- up/down  ## circular correlation
  lam22 <- sum( sintheta^2 * sinphi^2 ) / n
  lam02 <- sum( sinphi^2 ) / n
  lam20 <- sum( sintheta^2 ) / n
  zrho <- sqrt(n) * sqrt(lam02 * lam20/lam22) * rho
  pvalue <- 2 * ( 1 - pnorm( abs(zrho) ) )

  res <- c(rho, pvalue)
  names(res) <- c("rho", "p-value")
  res

}
```

9.6.2 Circular-circular correlation II

Mardia and Jupp (2000) mention another correlation of pairs of circular variables θ and ϕ. They say that it is a measure of dependence between $\mathbf{u}$ and $\mathbf{v}$, where $\mathbf{u} = (\cos\Theta, \sin\Theta)^T$ and $\mathbf{v} = (\cos\Phi, \sin\Phi)^T$. This is a squared correlation coefficient, so it only takes positive values and is defined as

$$r^2 = \frac{\left(r_{cc}^2 + r_{cs}^2 + r_{sc}^2 + r_{ss}^2\right) + 2\left(r_{cc}r_{ss} + r_{cs}r_{sc}\right)r_1 r_2 - 2\left(r_{cc}r_{cs} + r_{sc}r_{ss}\right)r_2 - 2\left(r_{cc}r_{sc} + r_{cs}r_{ss}\right)r_1}{\left(1 - r_1^2\right)\left(1 - r_2^2\right)} \quad (9.6)$$

where $r_{cc} = \text{corr}(\cos\theta, \cos\phi)$, $r_{cs} = \text{corr}(\cos\theta, \sin\phi)$, $r_{sc} = \text{corr}(\sin\theta, \cos\phi)$, $r_{ss} = \text{corr}(\sin\theta, \sin\phi)$, $r_1 = \text{corr}(\cos\theta, \sin\theta)$ and $r_2 = \text{corr}(\cos\phi, \sin\phi)$.

```
circ.cor2 <- function(theta, phi, rads = FALSE) {
  ## theta and phi are angular data in degrees or radians
  ## by default they are in degrees
  n <- length(theta)  ## sample size
  ## if the data are in degrees we transform them into radians
```

```
  if ( !rads ) {
    theta <- theta * pi/180
    phi <- phi * pi/180
  }

  costheta <- cos(theta)
  cosphi <- cos(phi)
  sintheta <- sin(theta)
  sinphi<- sin(phi)

  rcc <- cor(costheta, cosphi)
  rcs <- cor(costheta, sinphi)
  rss <- cor(sintheta, sinphi)
  rsc <- cor(sintheta, cosphi)
  r1 <- cor(costheta, sintheta)
  r2 <- cor(cosphi, sinphi)

  up <- rcc^2 + rcs^2 + rsc^2 + rss^2 + 2 * (rcc * rss + rcs * rsc) * r1 * r2 -
    2 * (rcc * rcs + rsc * rss) * r2 - 2 * (rcc * rsc + rcs * rss) * r1
  down <- (1 - r1^2) * (1 - r2^2)
  rho <- up/down
  test <- n * rho^2
  pvalue <- pchisq(test, 4, lower.tail = FALSE)
  res <- c(rho, pvalue)
  names(res) <- c("rho", "p-value")
  res

}
```

9.6.3 Circular-linear correlation

Mardia and Jupp (2000) mention a correlation coefficient when we have a euclidean variable (X) and a circular variable (Θ). The formula is the following

$$R^2_{x\theta} = \frac{r^2_{xc} + r^2_{xs} - 2r_{xc}r_{xs}r_{cs}}{1 - r^2_{cs}},$$

where $r_{xc} = \text{corr}(x, \cos\theta)$, $r_{xs} = \text{corr}(x, \sin\theta)$ and $r_{cs} = \text{corr}(\cos\theta, \sin\theta)$ are the classical Pearson sample correlation coefficients.

If X and Θ are independent and X is normally distributed then

$$\frac{(n-3)\, R_{x\theta}^2}{1 - R_{x\theta}^2} \sim F_{2,n-3}.$$

Since the F distribution is asymptotic we can a use non parametric bootstrap to calculate the p-value as well. In the following R function bootstrap is not implemented. But, the code works for many euclidean variables. If for example a matrix is supplied, all the correlations of the circular variable with the euclidean variables will be calculated.

```
circlin.cor <- function(theta, x, rads = FALSE) {
  ## theta is a angular variable in degrees by default
  ## x is euclidean variable or a matrix containing euclidean variables
  n <- length(theta)  ## sample size
  if ( !rads )  theta <- theta * pi/180
  costheta <- cos(theta)
  sintheta <- sin(theta)

  rxc <- as.numeric( cor(costheta, x) )  ## and cos(theta) and x correlation
  rxs <- as.numeric( cor(sintheta, x) )  ## sin(theta) and x correlation
  rcs <- cor(costheta, sintheta)  ## cos(theta) and sin(theta) correlation
  R2xt <- (rxc^2 + rxs^2 - 2 * rxc * rxs * rcs)/(1 - rcs^2)

  ## linear-circular correlation
  Ft <- (n - 3) * R2xt/(1 - R2xt)  ## F-test statistic value
  pvalue <- pf(Ft, 2, n - 3, lower.tail = FALSE)
  res <- cbind(R2xt, pvalue)
  colnames(res) <- c('R-squared', 'p-value')
  res

}
```

9.7 Regression for circular data

9.7.1 Regression for circular data using the von Mises distribution

Fisher and Lee (1992) used the von Mises distribution (defined on the circle) to link the mean of some angular data with a covariate. This means that the response variable is a circular variable and the explanatory variables are not defined on the circle.

The density of the von Mises distribution was defined in (9.2). Fisher and Lee (1992) suggested two models. The first one models the mean direction only and the second (the

mixed one) models the concentration parameter as well. In the first example the mean angle μ is linked with the explanatory variables ($\mathbf{X} = \left(x_1, \ldots, x_p\right)^T$) via

$$\mu = a + g\left(\boldsymbol{\beta}^T \mathbf{X}\right), \text{ where } g(x) = 2\tan^{-1}(x).$$

In the mixed model case the concentration parameter is also linked with the explanatory variables via an exponential function to ensure that it stays always positive

$$\kappa = e^{\gamma + \boldsymbol{\delta}^T \mathbf{X}}.$$

The estimates of the parameters are obtained via numerical optimisation of the log-likelihood of the von Mises distribution (9.2). We decided not to include a r function though since this model has some numerical problems (Pewsey et al., 2013). We mention the way though so that the reader is aware of this model also. The package circular offers this type of regression.

9.7.2 Projected bivariate normal for circular regression

Presnell et al. (1998) used the projected bivariate normal (Watson, 1983b) to perform circular regression. The density of the projected normal in the circular case can be written as

$$f(\theta) = \frac{1}{2\pi} e^{-\frac{\gamma^2}{2}} \left[1 + \frac{\gamma \cos(\theta - \omega)\, \Phi(\gamma \cos(\theta - \omega))}{\phi(\gamma \cos(\theta - \omega))}\right], \tag{9.7}$$

where θ represents the angle, ω is the mean direction and $\Phi(.)$ and $\phi(.)$ are the standard normal probability and density function respectively. An estimate of the concentration parameter is given by $\gamma = \|\boldsymbol{\mu}\|$ (Presnell et al., 1998), where $\boldsymbol{\mu} = \left(\frac{1}{n}\sum_{i=1}^n \cos u_i, \frac{1}{n}\sum_{i=1}^n \sin u_i\right)$ is the mean vector of the data in the Euclidean coordinates. However, I did some simulations and I think Presnell et al. (1998) was a bit wrong. The estimate of the concentration parameter is $\gamma = \|\boldsymbol{\mu}\|^2$. Generate some data from the von Mises distribution using the *rvonmises* function we saw before and apply the *circ.summary* we saw before and the *spml* function given below and you will see some similarities. This is not random and it was noticed in Watson (1983b).

However, (9.7) is the expression of the projected normal in the circular case. The general form of the density is

$$f(\mathbf{u}) = \frac{1}{2\pi} e^{-\frac{\gamma^2}{2}} \left[1 + \frac{\gamma \mathbf{u}^T \boldsymbol{\eta} \Phi\left(\gamma \mathbf{u}^T \boldsymbol{\eta}\right)}{\phi\left(\gamma \mathbf{u}^T \boldsymbol{\eta}\right)}\right], \tag{9.8}$$

where $\mathbf{u} = (\cos\theta, \sin\theta)$ and $\boldsymbol{\eta} = \boldsymbol{\mu}/\gamma$.

We will write its associated log-likelihood as

$$\ell(\mathbf{B}) = -\frac{1}{2}\sum_{i=1}^{n}\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i + \sum_{i=1}^{n}\log\left[1 + \frac{\mathbf{u}^T\boldsymbol{\mu}\Phi\left(\mathbf{u}^T\boldsymbol{\mu}\right)}{\phi\left(\mathbf{u}^T\boldsymbol{\mu}\right)}\right] - n\log(2\pi),$$

where $\boldsymbol{\mu}_i = \mathbf{B}^T\mathbf{x}_i$ is the bivariate mean vector of the projected normal linearly linked with some covariates $\mathbf{x}$, $\mathbf{B}$ is the matrix of parameters and n is the sample size. Thus, in order to apply the projected normal bivariate linear model we must first bring the angles θ_i onto the circle as $\mathbf{u}_i = (\cos(\theta_i), \sin(\theta_i))$.

The matrix of the parameters is a $(p+1) \times 2$ matrix, where p is the number of independents variables

$$\mathbf{B} = (\boldsymbol{\beta}_1, \boldsymbol{\beta}_2) = \begin{pmatrix} \beta_{01} & \beta_{02} \\ \beta_{11} & \beta_{12} \\ \vdots & \vdots \\ \beta_{p1} & \beta_{p2} \end{pmatrix}$$

The $\boldsymbol{\mu}_i$ lies in R^2 and so the fitted angular mean is given by

$$\hat{\theta}_i = \left[\tan^{-1}\left(\frac{\boldsymbol{\beta}_2^T\mathbf{x}_i}{\boldsymbol{\beta}_1^T\mathbf{x}_i}\right) + \pi I\left(\boldsymbol{\beta}_1^T\mathbf{x}_i < 0\right)\right] mod 2\pi, \tag{9.9}$$

where I is the indicator function.

In previous versions I was using *optim* or *nlm* to maximise the log-likelihood and estimate the coefficients. The issue with this way is time. For this reason I changed it into using the E-M algorithm as described in Presnell et al. (1998).

The algorithm iterates between two equations

$$\begin{aligned} \mathbf{M}^{(k)} &= \mathbf{M}\left(\hat{\mathbf{B}}^{(k)}\right) \text{ and} \\ \mathbf{B}^{(k+1)} &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{M}^{(k)}\mathbf{U}, \quad k = 0, 1, \ldots \end{aligned}$$

Let me now explain what these matrices stand for. $\mathbf{M}$ is a diagonal matrix

$$\mathbf{M} = \mathbf{M}\left(\hat{\mathbf{B}}\right) = \text{diag}\left(\dot{\psi}\left(\mathbf{u}_1^T\mathbf{B}^T\mathbf{x}_1\right), \ldots, \dot{\psi}\left(\mathbf{u}_n^T\mathbf{B}^T\mathbf{x}_n\right)\right),$$

with $\dot{\psi}(t) = t + \frac{\Phi(t)}{\phi(t)+\Phi(t)}$, where $\phi(t)$ and $\Phi(t)$ are the probability density and the cumulative distribution functions respectively of the standard normal distribution. The matrix $\mathbf{U}$ contains two columns

$$\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)^T,$$

where $\mathbf{u}_i$ are the cosinus and sinus of the angle, mentioned before.

The Hessian matrix (second derivative of the log-likelihood) is given by

$$\ddot{\ell}(\mathbf{B}) = \begin{bmatrix} \mathbf{X} & \mathbf{0} \\ \mathbf{0} & \mathbf{X} \end{bmatrix} \begin{bmatrix} -1 + \ddot{\psi}\left(\mathbf{u}^T\boldsymbol{\mu}\right) C^2 & \ddot{\psi}\left(\mathbf{u}^T\boldsymbol{\mu}\right) CS \\ \ddot{\psi}\left(\mathbf{u}^T\boldsymbol{\mu}\right) CS & -1 + \ddot{\psi}\left(\mathbf{u}^T\boldsymbol{\mu}\right) S^2 \end{bmatrix} \begin{bmatrix} \mathbf{X}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{X}^T \end{bmatrix},$$

where

$$\ddot{\psi}(t) = 2 - t\left[\frac{\Phi(t)}{\phi(t) + t\Phi(t)}\right] - \left[\frac{\Phi(t)}{\phi(t) + t\Phi(t)}\right]^2$$

and $C = [\cos(\theta_1), \ldots, \cos(\theta_n)]$ and $S = [\sin(\theta_1), \ldots, \sin(\theta_n)]$. The $2(p+1) \times 2(p+1)$ (we implement a bivariate regression with p covariates and 1 constant term) matrix with the stand errors is given by the inverse of the negative Hessian matrix

$$\text{Var}(\mathbf{B}) = \left[-\ddot{\ell}(\mathbf{B})\right]^{-1}.$$

As for a measure of fit of the model we provide a pseudo R^2 suggested by Lund (1999). We calculate the circular correlation coefficient (9.5) between the observed and the estimated angles, show the p-value of the hypothesis of a zero correlation and then square the correlation. This serves as an analogue of the R^2 in the classical linear models. Actually the paper by Lund (1999) describes another type of circular regression model, which we will not present here (at the moment) but the reader is encouraged to have a look. In addition you can predict the circular value of some new data if you want.

```
spml.reg <- function(y, x, tol = 1e-07, seb = FALSE) {
  ## y is the angular dependent variable
  ## x contains the independent variable(s)
  x <- model.matrix(~., data.frame(x) )
  ci <- cos(y)  ;   si <- sin(y)
  u <- cbind(ci, si)  ## bring the data onto the circle
  n <- dim(u)[1]
  XX <- solve( crossprod(x), t(x) )
  B1 <-  XX %*% u
  mu <- x %*% B1
  f <-  - 0.5   ;   con <- sqrt(2 * pi)
  tau <- rowsums(u * mu)
  ptau <- pnorm(tau)
  psit <- tau + ptau / ( exp(f * tau^2)/con + tau * ptau )
  B2 <- XX %*% (psit * u)
  mu <- mu * psit
```

```
  tau <- rowsums(u * mu)
  ptau <- pnorm(tau)
  i <- 2
  ## mono th while
  while ( sum( abs(B2 - B1) ) > tol ) {
    i <- i + 1
    B1 <- B2
    mu <- x %*% B1
    tau <- rowsums(u * mu)
    ptau <- pnorm(tau)
    rat <- ptau / ( exp(f * tau^2)/con + tau * ptau )
    psit <- tau + rat
    psit2 <- 2 - tau * rat - rat^2
    der <- as.vector( crossprod(x, - mu + psit * u) )
    a11 <- crossprod(x, x * (psit2 * ci^2 - 1) )
    a12 <- crossprod(x, x * (psit2 * ci * si ) )
    a22 <- crossprod(x, x * (psit2 * si^2 - 1 ) )
    der2 <- cbind( rbind(a11, a12), rbind(a12, a22) )
    B2 <- B1 - solve(der2, der)
  }
  ###
  if ( seb ) {
    seb <-  sqrt( diag( solve( -der2 ) ) )
    seb <- matrix(seb, ncol = 2)
    colnames(seb) <- c("Cosinus of y", "Sinus of y")
    rownames(seb) <- colnames(x)
  } else   seb <- NULL
  loglik <-  - 0.5 * sum( mu^2 ) + sum( log1p( tau * ptau * con / exp(f * tau^2) ) ) -
             n * log(2 * pi)
  colnames(B2) <- c("Cosinus of y", "Sinus of y")
  rownames(B2) <- colnames(x)
  list(iters = i, loglik = loglik, be = B2, seb = seb)
}
```

10 (Hyper-)spherical data

We continue with (hyper)spherical data analysis. Note that these techniques can also be applied to circular data. For example, the von Mises-Fisher distribution in two dimensions is simply the von Mises distribution. Thus, the following functions regarding the von Mises-

Fisher distribution can also be used for the von Mises. The space here is $\mathbb{S}^2$ if we are on the sphere and $\mathbb{S}^{q-1}$ if we are on the hypersphere. The functions described here (and a few more) exist as an R package as well Directional Tsagris et al. (2016a).

10.1 Change from geographical to Euclidean coordinates and vice versa

Imagine that we are given geographical coordinates and we want to perform directional statistical analysis. Say for example the coordinates of the earthquakes in some region over a period of time. In order to apply directional statistics we need to convert them to Euclidean (or Cartesian) coordinates ($\mathbb{S}^2$). So when we are given a pair of latitude and longitude in degrees say $(lat, long)$ the change to Euclidean coordinates is given by

$$\mathbf{u} = (x, y, z) = [\cos(lat), \sin(lat) * \cos(long), \sin(lat)\sin(long)]$$

At first we have to transform the latitude and longitude from degrees to radians and then apply the change to Euclidean coordinates. Note that the vector $\mathbf{u}$ is a unit vector (i.e. $\sum_{i=1}^{3} u_i^2 = 1$). Thus, the $\mathbf{u}$ lies on the unit radius sphere. Note, that this transformation was used by Kent (1982) and that is why I use it here. Chang (1986) used a more standard, I would say, transformation

$$\mathbf{u} = (x, y, z) = [\cos(lat) * \cos(long), \cos(lat) * \sin(long), \sin(lat)].$$

```
euclid <- function(u) {
  ## u is a matrix of two columns
  ## the first column is the latitude and the second the longitude
  u <- as.matrix(u)
  if (ncol(u) == 1)  u <- t(u)
  u <- pi * u/180  ## from degrees to rads
  a1 <- sin(u[, 1])
  U <- cbind(cos(u[, 1]), a1 * cos(u[, 2]), a1 * sin(u[, 2]))
  colnames(U) <- c("x", "y", "z")
  ## U are the cartesian coordinates of u
  U
}
```

The inverse transformation, from Euclidean coordinates to latitude and longitude is given by $\mathbf{u} = [\text{asin}(z), \text{atan2}(y/x)]$. And of course we have to transform back from radians to degrees.

```
euclid.inv <- function(U) {
  ## U is a 3-column matrix of unit vectors
```

```
  ## the cartesian coordinates
  U <- as.matrix(U)
  if (ncol(U) == 1)  U <- t(U)
  u <- cbind( acos(U[, 1]), ( atan(U[, 3]/U[, 2]) + pi * I(U[, 2]<0) )
  %% (2 * pi) )
  u <- u * 180/pi  ## from rads to degrees
  colnames(u) <- c("Lat", "Long")
  ## u is a matrix of two columns
  ## the first column is the latitude and the second the longitude in degrees
  u
}
```

10.2 Rotation of a unit vector

Suppose we have two unit vectors $\mathbf{a}$ and $\mathbf{b}$ on the hypersphere in $\mathbb{R}^d$ (or $\mathbb{S}^{d-1}$) and we wish to move $\mathbf{a}$ to $\mathbf{b}$ along the geodesic path on the hypersphere. Amaral et al. (2007) show, that provided $\| \mathbf{a}^T\mathbf{b} \| < 1$, a rotation matrix is determined in a natural way. Let

$$\mathbf{c} = \frac{\mathbf{b} - \mathbf{a}\left(\mathbf{a}^T\mathbf{b}\right)}{\| \mathbf{b} - \mathbf{a}\left(\mathbf{a}^T\mathbf{b}\right) \|}$$

Define $\alpha = cos^{-1}\left(\mathbf{a}^T\mathbf{b}\right) \in (0, 2\pi)$ and $\mathbf{A} = \mathbf{a}\mathbf{c}^T - \mathbf{a}^T\mathbf{c}$. The rotation matrix is then defined as

$$\mathbf{Q} = \mathbf{I}_p + \sin(\alpha)\,\mathbf{A} + [\cos(\alpha) - 1]\left(\mathbf{a}\mathbf{a}^T + \mathbf{c}\mathbf{c}^T\right) \tag{10.1}$$

Then $\mathbf{b} = \mathbf{Q}\mathbf{a}$. The R code is given below.

```
rotation <- function(a, b) {
  ## a and b are two unit vectors
  ## Calculates the rotation matrix
  ## to move a to b along the geodesic path
  ## on the unit sphere which connects a to b
  p <- length(a)
  ab <- sum(a * b)
  ca <- a - b * ab
  ca <- ca / sqrt( sum(ca^2) )
  A <- b %*% t(ca)
  A <- A - t(A)
  theta <- acos( ab )
  diag(p) + sin(theta) * A + (cos(theta) - 1) * ( b %*% t(b) + ca %*% t(ca) )
}
```

10.3 Rotation matrices on the sphere

We will see how we can obtain a rotation matrix in SO(3) when we have the rotation axis and the angle of rotation. The SO(3) space denotes the special orthogonal group of all 3×3 orthogonal matrices whose determinant is 1. In addition, the inverse of a rotation matrix is equal to its transpose. Suppose we have the rotation axis $\boldsymbol{\xi} = (\xi_1, \xi_2)$, where ξ_1 is the latitude and ξ_2 is the longitude and the angle of rotation θ in degrees or radians. If the angle is expressed in degrees we turn it into radians using $\phi = \frac{\theta\pi}{180}$. We then transform $\boldsymbol{\xi}$ to the Cartesian coordinates as $\mathbf{t} = (\cos\xi_1 \cos\xi_2, \cos\xi_1 \sin\xi_2, \sin\xi_1)$. Then as Chang (1986) mentions, we construct the following matrix

$$A(\theta) = \mathbf{I} + \sin(\theta)\mathbf{L} + (1 - \cos(\theta))\,\mathbf{L},$$

where

$$\mathbf{L} = \begin{pmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{pmatrix}$$

The R code is given below.

```
rot.matrix <- function(ksi, theta, rads = FALSE) {
  ## ksi is the rotation axis, where the first element is the
  ## latitude and the second is the longitude
  ## theta is the angle of rotation

  if ( rads ) {
    lat <- ksi[1]
    long <- ksi[2]
    the <- theta

  } else {
    lat <- ksi[1] * pi / 180
    long <- ksi[2] * pi / 180
    the <- theta * pi / 180
  }

  t1 <- cos(lat) * cos(long)
  t2 <- cos(lat) * sin(long)
  t3 <- sin(lat)
  L <- matrix( c(0, t3, -t2, -t3, 0, t1, t2, -t1, 0), ncol = 3 )
```

```
  diag(3) + L * sin(the) + L %*% L * ( 1 - cos(the) )
}
```

The inverse problem, when we have a rotation matrix in SO(3) and we want to find the rotation axis and the angle of rotation (in degrees, not radians) is not difficult to do. I took the next information from the course webpage of Howard E. Haber. Given a $3x3$ rotation matrix $\mathbf{A}$ we work as follows

- Calculate the angle of rotation (in radians) using the trace of $\mathbf{A}$

$$\phi = \cos^{-1}\left(\frac{tr\left(\mathbf{A}\right)-1}{2}\right)$$

- Transform the angle from radians to degrees

$$\theta = \frac{180\phi}{\pi}$$

- The rotation axis is

$$\boldsymbol{\xi} = \frac{1}{\sqrt{\left(3-tr\left(\mathbf{A}\right)\right)\left(1+tr\left(\mathbf{A}\right)\right)}}\left(\mathbf{A}_{32}-\mathbf{A}_{23}, \mathbf{A}_{13}-\mathbf{A}_{31}, \mathbf{A}_{21}-\mathbf{A}_{12},\right),$$

  where $tr\left(\mathbf{A}\right) \neq -1, 3$ and subscript (ij) denotes the (i,j) entry of the matrix $\mathbf{A}$.

Below is the relevant R code.

```
Arotation <- function(A) {
  ## A is a 3x3 rotation matrix
  con1 <- round(det(A), 15)
  con2 <- round( mean( abs( A %*% t(A) - diag(3) ) ), 15 )

  if ( con1 != 1 |  con2 > .Machine$double.eps ) {
    res <- paste("This is not a rotation matrix")

  } else {
    tr <- sum( diag(A) )
    rad <- acos(0.5 * (tr - 1))
    angle <- rad * 180 / pi  ## from rads to degrees
    ksi <- c(A[3, 2] - A[2, 3], A[1, 3] - A[3, 1], A[2, 1] - A[1, 2])/
    sqrt( (3 - tr) * (1 + tr) )
    axis <- c( asin(ksi[3]), atan2(ksi[2], ksi[1]) )
    axis <- c(axis / pi * 180)  ## from degrees to rads
```

```
    ## if the latitude or longitude are negative add 360 (degrees)
    axis[axis<0] <- axis[axis<0] + 360
    names(axis) <- c("latitude", "longitude")
    res <- list(angle = angle, axis = axis)
  }
  res

}
```

10.4 Spherical-spherical regression

Suppose we have pairs of data $(\mathbf{u}_i, \mathbf{v}_i)$ on the sphere (the constraint for any vector x which lies on the sphere is $\sum_{j=1}^{3} x_j^2 = 1$) and we know that $\mathbf{Y}$ was derived from $\mathbf{X}$ via a rotation matrix $\mathbf{A}$ (so $\mathbf{A}$ belongs to SO(3))

$$\mathbf{Y} = \mathbf{A}\mathbf{X}.$$

We wish to estimate this rotation matrix $\mathbf{A}$. Chang (1986) mentions that the estimate comes from the least squares method. He also mentions that the solution has already been given in closed form by Mackenzie (1957) and Stephens (1979). It is a singular value decomposition

$$\mathbf{X}\mathbf{Y}^T = \mathbf{O}_1\mathbf{\Lambda}\mathbf{O}_2^T,$$

where $\mathbf{O}_1$ and $\mathbf{O}_2$ belong to SO(3) and $\mathbf{\Lambda}$ is diagonal with entries $\lambda_1, \lambda_2, \lambda_3$ satisfying $\lambda_1 \geq \lambda_2 \geq |\lambda_3|$ (Chang, 1986). If $\mathbf{X}$ is of full rank (3 in our case), the determinant of $\mathbf{X}\mathbf{Y}^T$ is nonzero with probability 1 and in this case $\mathbf{A}$ is uniquely estimated (Chang, 1986)

$$\hat{\mathbf{A}} = \mathbf{O}_2\mathbf{O}_1^T$$

If $\det\left(\hat{\mathbf{A}}\right) = -1$, then an SVD is performed, $\hat{\mathbf{A}} = \mathbf{Y}\mathbf{\Lambda}\mathbf{X}^T$ and the elements of the third eigenvector of $\mathbf{X}^T$ change sign. The R code is given below.

```
spher.reg <- function(y, x, rads = FALSE) {
  ## x is the independent variable
  ## y is the dependent variable
  ## The first row of both matrices is the latitude
  ## and the second is the longitude
  n <- dim(x)[1]  ## sample size

  if ( dim(x)[2] == 2  &  dim(y)[2] == 2 ) {
```

```
    if ( !rads ) {
      x <- pi * x / 180  ## from degrees to rads
      y <- pi * y / 180
    }  ## from degrees to rads
    ## the first row of both matrices is the latitude and the second is the longitude
    ## the next two rows transform the data to Euclidean coordinates
    cosx1 <- cos(x[, 1])   ;  cosy1 <- cos(y[, 1])
    X <- cbind( cosx1 * cos(x[, 2]), cosx1 * sin(x[, 2]), sin(x[, 1]) )
    Y <- cbind( cosy1 * cos(y[, 2]), cosy1 * sin(y[, 2]), sin(y[, 1]) )
  } else if ( dim(x)[2] == 3  &  dim(y)[2] == 3 ) {
    X <- x
    Y <- y
  }

  XY <- crossprod(X, Y) / n
  b <- svd(XY)  ## SVD of the XY matrix
  A <- b$v %*% t(b$u)
  if ( det(A) < 0 ) {
    b$u[, 3] <-  - b$u[, 3]
    A <- tcrossprod(b$v, b$u )
  }

  est <- tcrossprod(X, A)
  list(A = A, fitted = est)
}
```

Since $\hat{\mathbf{A}}$ is a rotation matrix, we can then use the function we saw in the previous section (10.3) to calculate the rotation axis and the angle of rotation. If you want to predict the response value of some new data, you just do $\hat{\mathbf{Y}}_{new} = \hat{\mathbf{A}}\mathbf{X}_{new}$ or in R you type

```
Ynew <- Xnew %*% t(A),
```

where A is computed from the regression function.

10.5 (Hyper-)spherical correlation

Suppose we have two variables $\mathbf{X} \in \mathbb{S}^{p-1}$ and $\mathbf{Y} \in \mathbb{S}^{q-1}$ and we want to quantify their dependence. We will use the covariance matrices of the two variables. Denote by $\mathbf{S}$ their sample covariance

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}_{xx} & \mathbf{S}_{xy} \\ \mathbf{S}_{yx} & \mathbf{S}_{yy} \end{pmatrix}$$

Mardia and Jupp (2000) mentions that the circular-circular correlation type II we saw before (9.6) generalizes to

$$r^2 = \text{tr}\left(\mathbf{S}_{xx}^{-1}\mathbf{S}_{xy}\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx}\right),$$

provided that the block matrices $\mathbf{S}_{xx}$ and $\mathbf{S}_{yy}$ are non singular. Under the H_0 (independence) $nr^2 \sim \chi^2_{pq}$. The R code is given below.

```
spher.cor <- function(x, y) {
  ## x and y are two (hyper-)spherical variables
  p <- dim(x)[2]  ## dimension of x
  q <- dim(y)[2]  ## dimension of y
  n <- dim(x)[1]  ## sample size
  x <-  t(x) - Rfast::colmeans(x)    ## subtract the mean
  y <-  t(y) - Rfast::colmeans(y)   ## subtract the mean

  s11 <- tcrossprod(x) / n
  s12 <- tcrossprod( x, y ) / n
  s21 <- t( s12 )
  s22 <- tcrossprod(y) / n

  a1 <- solve(s11, s12)
  a2 <- solve(s22, s21)
  rsq <- sum( t(a1) * a2)
  test <- n * rsq
  pvalue <- pchisq(test, p * q, lower.tail = FALSE)
  res <- c(rsq, pvalue)

  names(res) <- c('R-squared', 'p-value')
  res

}
```

10.6 Analysis of variance for (hyper-)spherical data

10.6.1 High concentration F test

Similarly to the high concentration F test for the circular data, we have a version for data in $\mathbb{S}^{p-1}$ with $p \geq 3$.

$$F = \frac{(n-g)(p-1)\left(\sum_{i=1}^{g} R_i - R\right)}{(g-1)(p-1)\left(n - \sum_{i=1}^{g} R_i\right)}.$$

Under H_0, F follows asymptotically an $F_{(g-1)(p-1),(n-g)(p-1)}$. An improved approximation is given by Mardia and Jupp (2000)

$$F_c = \frac{\hat{\kappa}}{\hat{\gamma}} F,$$

where $\hat{\kappa}$ is the common estimate of the concentration parameter and is calculated by solving the equation $A_p(\hat{\kappa}) = \frac{R}{n}$, where R is the resultant length based on all the data. The factor $\frac{1}{\hat{\gamma}}$ is given by

$$\frac{1}{\hat{\gamma}} = \left\{ \begin{array}{ll} \frac{1}{\hat{\kappa}} - \frac{1}{5\hat{\kappa}^3} & \text{if } p = 3 \\ \frac{1}{\hat{\kappa}} - \frac{p-3}{4\hat{\kappa}^2} - \frac{p-3}{4\hat{\kappa}^3} & \text{if } p > 3. \end{array} \right\}$$

Mardia and Jupp (2000) mention that the corrected approximated test statistic above is adequate for $\hat{\kappa} \geq 1$.

```
hcf.aov <- function(x, ina, fc = TRUE) {
  ## x contains all the data
  ## ina is an indicator variable of each sample
  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there
  p <- dim(x)[2]
  n <- dim(x)[1]  ## dimensionality and sample size of the data

  S <- rowsum(x, ina)
  Ri <- sqrt( Rfast::rowsums(S^2) )  ## the resultant length of each group
  S <- Rfast::colsums(x)
  R <- sqrt( sum(S^2) )  ## the resultant length based on all the data

  ## Next we stimate the common concentration parameter kappa
  kappa <- vmf(x)$kappa
  ## kappa is the estimated concentration parameter based on all the data
```

```
  Ft <- ( (n - g) * (p - 1) * (sum(Ri) - R) )/( (g - 1) * (p - 1) * (n - sum(Ri)) )

  if ( fc ) {  ## correction is used
    if (p == 3) {
      Ft <- kappa * (1/kappa - 1/(5 * kappa^3)) * Ft
    } else if (p > 3)  {
      Ft <- kappa * (1/kappa - (p - 3)/(4 * kappa^2) - (p - 3)/(4 * kappa^3)) * Ft
    }
  }

  pvalue <- pf(Ft, (g - 1) * (p - 1), (n - g) * (p - 1), lower.tail = FALSE)
  res <- c(Ft, pvalue, kappa)
  names(res) <- c('test', 'p-value', 'kappa')
  res
}
```

10.6.2 Log-likelihood ratio test

The log-likelihood ratio test statistic is (Mardia and Jupp, 2000)

$$\Lambda = 2\left[\hat{\kappa}\sum_{i=1}^{g} R_i - \tilde{\kappa}R - n\alpha_p\left(\hat{\kappa}\right) + n\alpha_p\left(\tilde{\kappa}\right)\right],$$

where $\alpha_p\left(\kappa\right)$ is given in (10.5) and $\tilde{\kappa}$ and $\hat{\kappa}$ are the maximum likelihood estimates of κ under H_0 and H_1 respectively and are given by

$$A_p\left(\tilde{\kappa}\right) = \bar{R} \text{ and } A_p\left(\hat{\kappa}\right) = \frac{1}{n}\sum_{i=1}^{g} R_i.$$

Under H_0, $\Lambda \sim \chi^2_{(g-1)(p-1)}$.

```
lr.aov <- function(x, ina) {
  ## x contains all the data
  ## ina is an indicator variable of each sample

  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there
  x <- as.matrix(x)
  x <- x/sqrt(rowSums(x^2))  ## makes sure x are unit vectors
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data
```

```
S <- rowsum(x, ina)
Ri <- sqrt( rowSums(S^2) )  ## the resultant length of each group
S <- colSums(x)
R <- sqrt( sum(S^2) )  ## the resultant length based on all the data

## Next we stimate the common concentration parameter kappa under H0 and H1
Apk <- function(p, k) {
besselI(k, p/2, expon.scaled = T)/besselI(k, p/2 - 1, expon.scaled = T)
}

Rk <- R/n
k <- numeric(4)
j <- 1
k[j] <- Rk * (p - Rk^2)/(1 - Rk^2)
j <- 2
k[j] <- k[j - 1] - (Apk(p, k[j - 1]) - Rk)/(1 - Apk(p, k[j - 1])^2 -
(p - 1)/k[j - 1] * Apk(p, k[j - 1]))

while (abs(k[j] - k[j - 1]) > 1e-07) {
  j <- j + 1
  k[j] <- k[j - 1] - (Apk(p, k[j - 1]) - Rk)/(1 - Apk(p, k[j - 1])^2 -
  (p - 1)/k[j - 1] * Apk(p, k[j - 1]))
}

k0 <- k[j]  ## concentration parameter under H0
Rk <- sum(Ri)/n
k <- numeric(4)
j <- 1
k[j] <- Rk * (p - Rk^2)/(1 - Rk^2)
j <- 2
k[j] <- k[j - 1] - (Apk(p, k[j - 1]) - Rk)/(1 - Apk(p, k[j - 1])^2 -
(p - 1)/k[j - 1] * Apk(p, k[j - 1]))

while (abs(k[j] - k[j - 1]) > 1e-07) {
  j <- j + 1
  k[j] <- k[j - 1] - (Apk(p, k[j - 1]) - Rk)/(1 - Apk(p, k[j - 1])^2 -
  (p - 1)/k[j - 1] * Apk(p, k[j - 1]))
}
```

```
  k1 <- k[j]  ## concentration parameter under H1
  apk0 <- (1 - p/2) * log(k0/2) + lgamma(p/2) +
  log(besselI(k0, p/2 - 1, expon.scaled = T)) + k0
  apk1 <- (1 - p/2) * log(k1/2) + lgamma(p/2) +
  log(besselI(k1, p/2 - 1, expon.scaled = T)) + k1
  w <- 2 * (k1 * sum(Ri) - k0 * R - n * apk1 + n * apk0)

  pvalue <- 1 - pchisq(w, (g - 1) * (p - 1))
  res <- c(w, pvalue)
  names(res) <- c('w', 'p-value')
  res

}
```

10.6.3 Embedding approach

Similar to the circular data, we have the embedding approach for hyper-spherical data as well. Mardia and Jupp (2000) has a mistake in the form of this test statistic (Equation (10.6.19) on page 225). The factor n is missing and I show the correct form here.

$$F = \frac{(n-g)(p-1)\left(\sum_{i=1}^{g} n_i \bar{R}_i^2 - n\bar{R}^2\right)}{(g-1)(p-1)\left(n - \sum_{i=1}^{g} n_i \bar{R}_i^2\right)}.$$

Under H_0, $F \sim F_{(g-1)(p-1),(n-g)(p-1)}$.

```
embed.aov <- function(x, ina) {
  ## x contains all the data
  ## ina is an indicator variable of each sample

  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there
  ni <- as.vector(table(ina))
  x <- as.matrix(x)
  x <- x / sqrt( Rfast::rowsums(x^2) )  ## makes sure x are unit vectors
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data

  S <- rowsum(x, ina) / ni
  Rbi <- sqrt( Rfast::rowsums(S^2) ) ## the mean resultant length of each group
  S <- Rfast::colmeans(x)
```

```
  Rbar <- sqrt( sum(S^2) )  ## the mean resultant length based on all the data

  Ft <- ((n - g) * (p - 1) * ( sum(ni * Rbi^2) - n * Rbar^2) )/( (g - 1) *
  (p - 1) * (n - sum(ni * Rbi^2)) )

  pvalue <- pf(Ft, (g - 1) * (p - 1), (n - g) * (p - 1), lower.tail = FALSE)
  res <- c(Ft, pvalue)
  names(res) <- c('F', 'p-value')
  res

}
```

10.6.4 A test for testing the equality of the concentration parameters for spherical data only

Mardia and Jupp (2000) provides a test for testing the equality of the concentration parameter among g samples, where $g \geq 2$ in the case of spherical data only. There are three distinct cases, based on the value of $\bar{R}$, the mean resultant length of all data.

- *Case I*. $\bar{R} < 0.44$. The test statistic has the following form

$$U_1 = \sum_{i=1}^{g} w_i g_1 \left(3\bar{R}_i\right)^2 - \frac{\left[\sum_{i=1}^{g} w_i g_1 \left(3\bar{R}_i\right)\right]^2}{\sum_{i=1}^{g} w_i},$$

  where $w_i = \frac{5(n_i-5)}{3}$, with n_i denoting the sample size of the i-th group and $g_1(r) = \sin^{-1}\frac{r}{\sqrt{(5)}}$. $\bar{R}_i$ is the mean resultant length of the i-th group.

- *Case II*. $0.44 \leq \bar{R} \leq 0.67$. The test statistic now becomes

$$U_2 = \sum_{i=1}^{g} w_i g_2 \left(\bar{R}_i\right)^2 - \frac{\left[\sum_{i=1}^{g} w_i g_2 \left(\bar{R}_i\right)\right]^2}{\sum_{i=1}^{g} w_i},$$

  where $w_i = \frac{n_i-4}{0.394}$ and $g_2(x) = \sin^{-1}\left(\frac{r+0.176}{1.029}\right)$.

- *Case III*. $\bar{R} > 0.67$. For this high concentration case the test statistic is

$$U_3 = \frac{1}{1+d}\left[\nu \log\left(\frac{n - \sum_{i=1}^{g} R_i}{\nu}\right) - \sum_{i=1}^{g} \nu_i \log\left(\frac{n_i - R_i}{\nu_i}\right)\right],$$

  where $\nu_i = 2(n_i - 1)$, $\nu = 2(n-g)$ and $d = \frac{1}{3(g-1)}\left(\sum_{i=1}^{g}\frac{1}{\nu_i} - \frac{1}{\nu}\right)$.

Under H_0, each U_i, with regards to each case, follows asymptotically a χ^2_{g-1}.

```
spherconc.test <- function(x, ina) {
  ## x contains all the data
  ## ina is an indicator variable of each sample

  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there
  ni <- as.vector(table(ina))
  x <- as.matrix(x)
  x <- x / sqrt( Rfast::rowSums(x^2) )  ## makes sure x are unit vectors
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data

  if (p == 3) {
    S <- rowsum(x, ina) / ni
    Rbi <- sqrt( Rfast::rowsums(S^2) )   ## the mean resultant length of each group
    S <- Rfast::colmeans(x)
    Rb <- sqrt( sum(S^2) )  ## the mean resultant length of all the data

    if ( Rb < 0.44 ) {
      ## case 1
      g1 <- wi <- numeric(g)
      wi <- ( 5 * (ni - 5) ) / 3
      g1 <- asin(3 * Rbi/sqrt(5))
      U1 <- sum(wi * g1^2) - ( sum(wi * g1) )^2/sum(wi)
      stat <- U1
      pvalue <- pchisq(stat, g - 1, lower.tail = FALSE)
      mess <- paste('The mean resultant length is less than 0.44. U1 was calculated')

    } else if ( Rb >= 0.44 & Rb <= 0.67 ) {
      ## case 2
      g2 <- wi <- numeric(g)
      wi <- (ni - 4)/0.394
      g2 <- asin( (Rbi + 0.176)/1.029 )
      U2 <- sum(wi * g2^2) - ( sum(wi * g2) )^2/sum(wi)
      stat <- U2
      pvalue <- pchisq(stat, g - 1, lower.tail = FALSE)
      mess <- paste('The mean resultant length is between 0.44 and 0.67.
      U2 was calculated')
```

```
    } else if ( Rb > 0.67 ) {
      ## case 3
      Ri <- Rbi * ni
      vi <- 2 * (ni - 1)
      v <- 2 * (n - g)
      d <- 1/(3 * (g - 1)) * ( sum(1/vi) - 1/v )
      U3 <- 1/(1 + d) * ( v * log( (n - sum(Ri) )/v ) - sum( vi * log( (ni - Ri)/vi) ) )
      stat <- U3
      pvalue <- pchisq(U3, g - 1, lower.tail = FALSE)
      mess <- paste('The mean resultant length is more than 0.67. U3 was calculated')
    }

  } else {
    stat <- NA
    pvalue <- NA
    mess <- paste("This test is valid only for spherical data")
  }

  res <- c(stat, pvalue)
  names(res) <- c('test', 'p-value')
  list(mess = mess, res = res)

}
```

10.6.5 Analysis of variance without assuming equality of the concentration parameters

When the concentration parameters of the different samples cannot be assumed equal we can use the following test statistic (Mardia and Jupp, 2000)

$$T = 2\left(\sum_{i=1}^{g} \hat{\kappa}_i n_i \|\bar{\mathbf{x}}_i\| - \left\| \sum_{i=1}^{g} \hat{\kappa}_i n_i \bar{\mathbf{x}}_i \right\| \right),$$

where $\hat{\kappa}_i = A_p^{-1}(\bar{R}_i)$. Under H_0, the large sample asymptotic distribution of T is $\chi^2_{(g-1)(p-1)}$.

```
het.aov <- function(x, ina) {
  ## x contains all the data
  ## ina is an indicator variable of each sample

  ina <- as.numeric(ina)
  g <- max(ina)  ## how many groups are there
```

```
  x <- as.matrix(x)
  x <- x / sqrt( Rfast::rowsums(x^2) )  ## makes sure x are unit vectors
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data
  ni <- as.vector(table(ina))  ## group sample sizes

  kappa <- numeric(g)
  mi <- rowsum(x, ina) / ni

  for (i in 1:g) {
    kappa[i] <- vmf( x[ina == i, ] )$kappa
  }

  tw <- Rfast::colsums(kappa * ni * mi)
  Tt <- 2 * ( sum( kappa * ni * sqrt( rowSums(mi^2) ) ) - sqrt( sum(tw^2) ) )
  pvalue <- pchisq(Tt, (p - 1) * (g - 1), lower.tail = FALSE)
  res <- c(Tt, pvalue)
  names(res) <- c('test', 'p-value')
  res

}
```

10.7 Spherical and hyper-spherical distributions related stuff

10.7.1 Estimating the parameters of the the von Mises-Fisher distribution

The von Mises-Fisher distribution is the generalization of the von Mises distribution (on the circle) to the sphere in $\mathbb{R}^3$ (or $\mathbb{S}^2$) and the hypersphere in $\mathbb{R}^p$ (or $\mathbb{S}^{p-1}$) ($p > 3$). Its density is given by

$$f_p(\mathbf{x}; \mu, \kappa) = C_p(\kappa) \exp\left(\kappa \boldsymbol{\mu}^T \mathbf{x}\right), \tag{10.2}$$

where

$$\kappa \geq 0, \ \ \|\boldsymbol{\mu}\| = 1 \ \text{ and } \ C_p(\kappa) = \frac{\kappa^{p/2-1}}{(2\pi)^{p/2} I_{p/2-1}(\kappa)},$$

where $I_v(z)$ denotes the modified Bessel function of the first kind and order v calculated at z.

Maximum likelihood estimation of the parameters does not require numerical optimization of the corresponding log-likelihood. The estimated mean direction is available in closed

form given by

$$\hat{\boldsymbol{\mu}} = \frac{\bar{\mathbf{x}}}{\| \bar{\mathbf{x}} \|},$$

where $\| \cdot \|$ denotes the Euclidean norm on $\mathbb{R}^d$. The concentration parameter though needs two steps of a truncated Newton-Raphson algorithm (Sra, 2012).

$$\hat{\kappa}^{(t)} = \hat{\kappa}^{(t-1)} - \frac{A_p\left(\hat{\kappa}^{(t-1)}\right) - \bar{R}}{1 - \left[A_p\left(\hat{\kappa}^{(t-1)}\right)\right]^2 - \frac{p-1}{\hat{\kappa}^{(t-1)}} A_p\left(\hat{\kappa}^{(t-1)}\right)}, \tag{10.3}$$

where

$$A_p\left(\hat{\kappa}^{(t-1)}\right) = \frac{I_{p/2}\left(\hat{\kappa}\right)}{I_{p/2-1}\left(\hat{\kappa}\right)} = \frac{\| \sum_{i=1}^{n} \mathbf{x}_i \|}{n} = \bar{R}, \tag{10.4}$$

and $I_p\left(\hat{\kappa}\right)$ is the modified Bessel function of the first kind (see Abramowitz and Stegun (1970)). Similarly to Sra (2012) we will set $\hat{\kappa}^{(0)} = \frac{\bar{R}\left(p-\bar{R}^2\right)}{1-\bar{R}^2}$ to (10.3). The variance of $\hat{\kappa}$ is given by (Mardia and Jupp, 2000)

$$var\left(\hat{\kappa}\right) = \left[n\left(1 - \frac{A_p\left(\hat{\kappa}\right)}{\hat{\kappa}} - A_p\left(\hat{\kappa}\right)^2\right)\right]^{-1}$$

The modified Bessel function in R gives us the option to scale it exponentially. This means, that it calculates this quantity instead $I_p\left(\hat{\kappa}\right) exp^{-\hat{\kappa}}$. This is useful because when large numbers are plugged into the Bessel function, R needs the exponential scaling to calculate the ratio of the two Bessel functions. Note that we can use this to calculate the parameters of the von Mises distribution as well, since the von Mises distribution is simply the von Mises-Fisher distribution on the circle, with $p = 2$.

```
vmf <- function(x, tol = 1e-06) {
  ## x contains the data
  ## tol specifies the tolerance value for convergence
  ## when estimating the concentration parameter

  x <- as.matrix(x)
  x <- x / sqrt( Rfast::rowSums(x^2) )
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data

  Apk <- function(p, k) {
   besselI(k, p/2, expon.scaled = TRUE)/besselI(k, p/2 - 1, expon.scaled = TRUE)
```

```
  }

  m1 <- Rfast::colsums(x)
  R <- sqrt( sum(m1^2) )/n  ## mean resultant length
  m <- m1 / (n * R)
  k <- numeric(4)
  i <- 1
  k[i] <- R * (p - R^2)/(1 - R^2)

  if (k[i] > 100000) {
    k <- k[i]
  } else {
    i <- 2
    apk <- Apk(p, k[i - 1])
    k[i] <- k[i - 1] - (apk - R)/( 1 - apk^2 - (p - 1)/k[i - 1] * apk )

    while (abs(k[i] - k[i - 1]) > tol) {
      i <- i + 1
      apk <- Apk(p, k[i - 1])
      k[i] <- k[i - 1] - (apk - R)/( 1 - apk^2 - (p - 1)/k[i - 1] * apk )
    }
    k <- k[i]
  }

  loglik <- n * (p/2 - 1) * log(k) - 0.5 * n * p * log(2 * pi) -
  n * ( log( besselI(k, p/2 - 1, expon.scaled = TRUE) ) + k ) + k * sum(x %*% m)
  vark <- 1 / ( n * (1 - Apk(p, k)/k - Apk(p, k)^2) )
  list(mu = m, kappa = k, MRL = R, vark = vark, loglik = loglik)

}
```

Alternatively and perhaps easier, if you want to estimate the concentration parameter κ you can solve the equation (10.4) numerically (function *uniroot*) and thus substitute the Newton-Raphson algorithm from the above function. Another way is to optimize, numerically, the log-likelihood with respect to κ. After calculating the mean direction, simply use the function *optimize* and that's it. If you calculate the log-likelihood with respect to κ for a number of values of κ and then plot it, you will see its curve graphically.

10.7.2 (Hyper-)spherical median direction

Fisher (1985) introduced the idea of the spherical median direction. It is the unit vector $\mathbf{m}$ which minimizes the sum of the arc distances of all the points

$$\sum_{i=1}^{n} \cos^{-1}\left(\mathbf{x}_i^T \mathbf{m}\right).$$

The next function does the job.

```
mediandir_2 = function(x) {
  ## x is the directional data

    funa <- function(pa) {
      pa <- pa / sqrt( sum(pa^2) )
      mean( acos( x %*% pa ) )
    }

  pa <- Rfast::colMedians(x)
  bar <- nlm( funa, pa, iterlim = 10000 )
  bar <- nlm( funa, bar$estimate, iterlim = 10000 )
  bar <- optim( bar$estimate, funa, control = list(maxit = 10000) )
  med <- bar$par
  med / sqrt( sum(med^2) )

}
```

However, time can become an issue, especially with large scale data and given that this is a numerical optimiser errors can occur. For example, the correct estimated median might not be found, but a very near point can be returned. For these two reasons I will also provide a fixed-point iteration solution, which was given by Cabrera and Watson (1990) and is much faster and robust (or more reliable if you prefer).

$$\mathbf{m}^{(k+1)} = \text{unit vector parallel to} \sum_{i=1}^{n} \frac{\mathbf{x}_i^T}{\sqrt{1 - \left(\mathbf{x}_i^T \mathbf{m}^{(k)}\right)^2}}$$

The initial value is $\mathbf{m}^{(1)} = \bar{\mathbf{x}} / |\bar{\mathbf{x}}|$ (mean direction).

```
mediandir <- function(x) {
  ## x is the directional data
  n <- dim(x)[1]
```

```
  p <- dim(x)[2]

  pa <- Rfast::colmeans(x)
  u1 <- pa / sqrt( sum(pa^2) )
  ww <- as.vector( sqrt( 1 - ( x %*% u1 )^2 ) )
  u2 <- Rfast::colsums(x / ww )
  u2 <- u2 / sqrt( sum( u2^2 ) )

  while ( sum( abs (u2 - u1 ) ) > 1e-10 ) {
    u1 <- u2
    ww <- as.vector( sqrt( 1 - ( x %*% u1 )^2 ) )
    u2 <- Rfast::colsums (x / ww )
    u2 <- u2 / sqrt( sum( u2^2 ) )
  }

  u2
}
```

10.7.3 Kernel density estimation using a von Mises-Fisher kernel

The von Mises-Fisher kernel density estimate is given by

$$f(\mathbf{x}; h) = \frac{1}{n} \sum_{i=1}^{n} C_p(h) \exp\left(\frac{\mathbf{x}_i^T \mathbf{x}}{h^2} \right),$$

where

$$C_p(h) = \frac{\left(1/h^2\right)^{p/2-1}}{(2\pi)^{p/2} I_{p/2-1}\left(1/h^2\right)},$$

So, it's pretty much the same as (10.2), but instead of κ there is $1/h^2$ and instead of $\boldsymbol{\mu}$ we have $\mathbf{x}_i$ and an average in the front.

How does on choose h? The same question we have seen again before. Either using a rule of thumb (García-Portugués, 2013) or by maximum likelihood cross validation (5.5). Let us say $q = p - 1$, where p is the number of variables, or dimensions in the Euclidean space,

so $\mathbb{S}^{p-1} = \mathbb{S}^q$. The rule of thumb by García-Portugués (2013) is

$$h_{ROT} = \left\{ \begin{array}{ll} \left[\frac{8 \sinh^2(\hat{\kappa})}{\hat{\kappa} n[(1+4\hat{\kappa}^2)\sinh(2\hat{\kappa}) - 2\hat{\kappa}\cosh(2\hat{\kappa})]} \right]^{\frac{1}{6}}, & q = 2 \\ \left[\frac{4\pi^{\frac{1}{2}} I_{\frac{q-1}{2}}(\hat{\kappa})^2}{\hat{\kappa}^{\frac{q+1}{2}} n \left[2q I_{\frac{q+1}{2}}(2\hat{\kappa}) + (2+q)\hat{\kappa} I_{\frac{q+3}{2}}(2\hat{\kappa}) \right]} \right]^{\frac{1}{4+q}}, & q \geq 2 \end{array} \right\}$$

The following R code calculates the kernel density estimates of a directional data sample.

```
vmf.kde <- function(x, h = NULL, thumb = "none") {
  ## x is the data
  ## h is the bandwidth you want
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data
  ## thumb is either 'none' (defualt), or 'rot' (Garcia-Portugues, 2013)
  if ( !is.null(h) ) {
    if (thumb == "rot") {
      k <- vmf(x)$kappa  ## concentration parameter
      q <- p - 1
      if (q == 2)  h <- ( 8 * sinh(k)^2 / ( k * n * ( (1 + 4 * k^2) * sinh(2 * k) -  2 *
      if (q >= 3) {
        up <- 4 * pi^0.5 * besselI(k, (q - 1)/2)^2
        down <- k^( (q + 1)/2) * n * (2 * q * besselI(2 * k, (q + 1)/2) +  (2 + q) * k *
        h <- ( up/down ) ^ ( 1/(4 + q) )
      }
    } else if (thumb == "none")  h <- as.numeric( vmfkde.tune(x, low = 0.1, up = 1)[1] )
  } else h <- h

  d <- tcrossprod( x )/h^2
  cpk <- (1/h^2)^( p/2 - 1) / ( (2 * pi) ^ (p/2) * besselI(1/h^2, p/2 - 1) )
  f <- Rfast::rowmeans( exp(d) ) * cpk
  list( h = h, f = f )
}
```

The next R code chooses the value of h via maximum likelihood cross validation (5.5).

```
vmfkde.tune_2 <- function(x, h = seq(0.1, 1, by = 0.01), plot = TRUE) {
  ## x is the data
  ## h is the bandwidth grid you want
```

```
  runtime <- proc.time()

  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data
  cv <- numeric( length(h) )
  d <- tcrossprod(x)
  diag(d) <- NA  ##  we do not want to take the diagonal elements

  for ( j in 1:length(h) ) {
    A <- d/h[j]^2
    cpk <- ( (1/h[j]^2)^(p/2 - 1) ) / ( (2 * pi)^(p/2) * besselI(1/h[j]^2, p/2 - 1) )
    f <- rowSums( exp(A + log(cpk)), na.rm = T )/(n - 1)
    cv[j] <- mean(log(f))
  }

  runtime <- proc.time() - runtime

  if ( plot )  plot(h, cv, type = "l")
  list(hopt = h[which.max(cv)], cv = cv, runtime = runtime)
}
```

The next code is a bit faster, since it uses the *optimize* and not *for* loop.

```
vmfkde.tune1 <- function(x, low = 0.1, up = 1) {
  ## x is the data
  x <- as.matrix(x)  ## makes sure x is a matrix
  x <- x / sqrt( Rfast::rowsums(x^2) )  ## makes sure x is directional data
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data
  d <- tcrossprod( x )
  diag(d) <- NA  ##  we do not want to take the diagonal elements
  con <- (2 * pi)^(p/2)

   funa <- function(h) {
    A <- d/h^2
    cpk <- (1/h^2)^(p/2 - 1) / con / besselI(1/h^2, p/2 - 1)
    f <- rowSums( exp(A + log(cpk)), na.rm = TRUE )/(n - 1)
    mean(log(f))
  }
```

```
  a <- optimize(funa, c(low, up), maximum = TRUE)
  res <- c(a$maximum, a$objective)
  names(res) <- c("Optimal h", "cv")
  res
}
```

The final code is the best of all

```
vmfkde.tune <- function (x, low = 0.1, up = 1) {
  p <- dim(x)[2]
  n <- dim(x)[1]
  d <- tcrossprod(x)
  diag(d) <-  -Inf
  con <- (2 * pi)^(p/2)
  funa <- function(h) {
    A <- d/h^2
    cpk <- (1/h^2)^(p/2 - 1)/besselI(1/h^2, p/2 - 1)
    f <- colSums( exp(A) )
    n * log(cpk) + sum( log(f) )
  }
  a <- optimize(funa, c(low, up), maximum = TRUE)
  res <- c(a$maximum, a$objective/n - log(con) - log(n - 1) )
  names(res) <- c("Optimal h", "cv")
  res
}
```

10.7.4 The Rayleigh test of uniformity

The von Mises-Fisher distribution is a fundamental distribution for directional data. However, there is a simpler one, the uniform distribution on the (hyper)sphere (or circle of course). If the concentration parameter κ of the von Mises-Fisher distribution is 0, then we end up with the uniform distribution. Mardia et al. (1979) and Mardia and Jupp (2000) mention the Rayleigh test for testing the null hypothesis that $\kappa = 0$ against the alternative of $\kappa > 0$. They mention that under the null hypothesis

$$T = np\bar{R}^2 \sim \chi^2_p,$$

where n and p are the sample size and the number of dimensions and $\bar{R} = \frac{\|\sum_{i=1}^{p} \mathbf{x_i}\|}{n}$ also given in (10.4). Mardia et al. (1979, pg. 440) mentions that the case of $p = 3$ was first proved by Rayleigh (1919).

The error in the above approximation of the test statistic is of order $\mathcal{O}\left(n^{-1}\right)$. In Mardia and Jupp (2000) a better approximation can be found which reduces the error to $\mathcal{O}\left(n^{-2}\right)$ and is attributable to Jupp (2001)

$$T_m = \left(1 - \frac{1}{2n}\right) T + \frac{1}{2n\left(p+2\right)} T^2.$$

The function below offers the possibility of a parametric bootstrap calculation of the p-value, for the non modified test statistic. We remind that we must simulate from a multivariate normal with the zero vector as the mean vector and the identity as the covariance matrix. We then project the values on to the (hyper)sphere and this results into the uniform distribution on the (hyper)sphere. Thus we generate values from a uniform many times in order to do the parametric bootstrap (simulating under the null hypothesis, that of uniformity).

```
rayleigh <- function(x, modif = TRUE, B = 999) {
  ## x contains the data in Euclidean coordinates
  ## B is by default eaual to 999 bootstrap samples
  ## If B==1 then no bootstrap is performed
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data
  m <- Rfast::colsums(x)
  test <- sum( m^2 ) *p / n

  if ( modif ) {
    test <- ( 1 - 1/(2 * n) ) * test + test^2 / ( 2 * n * (p + 2) )
  }

  if (B == 1) {
    pvalue <- pchisq(test, p, lower.tail = FALSE)
    res <- c(test, pvalue)
    names(res) <- c('test', 'p-value')

  } else {
    tb <- numeric(B)
    for (i in 1:B) {
      x <- matrix( RcppZiggurat::zrnorm(n * p), ncol = p )
      x <- x / sqrt( Rfast::rowsums(x^2) )
      mb <- Rfast::colsums(x)
      tb[i] <- p * sum( mb^2 ) / n
    }
```

```
    res <- c( test, (sum(tb > test) + 1)/(B + 1) )
    names(res) <- c('test', 'Bootstrap p-value')
  }

  res

}
```

10.7.5 Test for the mean direction of a sample

The log-likelihood ratio test statistic for the null hypothesis of $\boldsymbol{\mu} = \boldsymbol{\mu}_0$ is

$$w = n \left[\hat{\kappa} \left\| \bar{\mathbf{x}} \right\| - \tilde{\kappa} \boldsymbol{\mu}_0 \bar{\mathbf{x}} - \alpha_p \left(\hat{\kappa} \right) + \alpha_p \left(\tilde{\kappa} \right) \right],$$

where

$$\alpha_p \left(\kappa \right) = \left(1 - p/2 \right) \log \left(\frac{\kappa}{2} \right) + \log \Gamma \left(\frac{p}{2} \right) + \log \left[I_{p/2-1} \left(\kappa \right) \right]. \tag{10.5}$$

Under the null hypothesis $w \sim \chi^2_{p-1}$. The next R function offers a bootstrap calibration of the test statistic. To transform the data under the null hypothesis, we use the *rotation* function we saw before.

```
meandir.test <- function(x, mu, B = 999) {
  ## x is the sample
  ## mu is the hypothesized mean direction under H0
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data
  k1 <- vmf(x)$k  ## concentration parameter under H1
  xbar <- Rfast::colmeans(x)  ## x-bar
  m1 <- xbar / sqrt( sum(xbar^2) )

  lik <- function(k, x) {
    n * (p/2 - 1) * log(k) - 0.5 * n * p * log(2 * pi) + k * sum(x %*% mu) -
      n * (log( besselI(k, p/2 - 1, expon.scaled = TRUE) ) + k)
  }

  ## log-likelihood under H0
  qa0 <- optimize(lik, c(0, 100000), x = x, maximum = TRUE)
  k0 <- qa0$maximum  ## concentration parameter under H0
  apk0 <- (1 - p/2) * log(k0/2) + lgamma(p/2) +
  log( besselI(k0, p/2 - 1, expon.scaled = TRUE) ) + k0
```

```
  apk1 <- (1 - p/2) * log(k1/2) + lgamma(p/2) +
  log( besselI(k1, p/2 - 1, expon.scaled = TRUE) ) + k1
  w <- 2 * n * (k1 * sqrt(sum(xbar^2)) - k0 * sum(mu * xbar) - apk1 + apk0)
  if (B == 1)  pvalue <- pchisq(w, p - 1, lower.tail = FALSE)

  if (B > 1) {
    A <- rotation(m1, mu)
    y <- tcrossprod(x, A)  ## bring the data under H0
    ## y has mean direction equal to mu
    wb <- numeric(B)
    for (i in 1:B) {
      nu <- sample(1:n, n, replace = TRUE)
      z <- y[nu, ]
      k1 <- vmf(z)$k  ## concentration parameter under H1
      zbar <- Rfast::colmeans(z)  ## z-bar

      qa0 <- optimize(lik, c(0, 100000), x = z, maximum = TRUE)  ## log-likelihood under
      k0 <- qa0$maximum  ## concentration parameter under H0
      apk0 <- (1 - p/2) * log(k0/2) + lgamma(p/2) +
      log( besselI(k0, p/2 - 1, expon.scaled = TRUE) ) + k0
      apk1 <- (1 - p/2) * log(k1/2) + lgamma(p/2) +
      log( besselI(k1, p/2 - 1, expon.scaled = TRUE) ) + k1
      wb[i] <- 2 * n * (k1 * sqrt(sum(zbar^2)) - k0 * sum(mu * zbar) -
      apk1 + apk0)
    }

    pvalue <- (sum(wb > w) + 1)/(B + 1)
  }
  list(mean.dir = m1, pvalue = pvalue)
}
```

10.7.6 Normalizing constant of the Bingham and the Fisher-Bingham distributions

The Fisher-Bingham distribution density is given by Kume and Wood (2005)

$$f(\mathbf{x}|\mathbf{A},\boldsymbol{\gamma}) = \frac{1}{c(\mathbf{A},\boldsymbol{\gamma})}\exp\left(-\mathbf{x}^T\mathbf{A}\mathbf{x}+\boldsymbol{\gamma}^T\mathbf{x}\right), \tag{10.6}$$

where $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{p\times p}$ and $\boldsymbol{\gamma} \in \mathbb{R}^p$ with p denoting the number of dimensions of the (hyper)sphere. We will follow their notation and without loss of generality work with $\boldsymbol{\Lambda} =$

diag $(\lambda_1, \ldots, \lambda_p)$, with $0 < \lambda_1 \leq \ldots \leq \lambda_p$, where λ_i is the i-th eigenvalue of the matrix $\mathbf{A}$. The $\mathbf{A}$ matrix is the Bingham part. The vector $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)$ is the Fisher part.

Kume and Wood (2005) derived the saddlepoint approximations to the normalizing constant of the Fisher-Bingham distribution. The Fisher and the Bingham distribution can be considered as special cases of the aforementioned distribution. Their paper is a bit technical and usually technical papers tend to be technical and not easy to understand at a glance. For this reason we will try to explain, briefly, the calculations required to derive the approximation. We will follow the same notation as in their paper for consistency and convenience to the reader purposes.

Saddlepoint approximation requires a cumulant generating function as its starting point (Butler, 2007). In this case that is given by

$$K_\theta(t) = \sum_{i=1}^{p} \left\{ -\frac{1}{2}\log(1 - t/\lambda_i) + \frac{1}{4}\frac{\gamma_i^2}{\lambda_i - t} - \frac{\gamma_i^2}{4\lambda_i} \right\} \quad (t < \lambda_1). \tag{10.7}$$

The first derivative of (10.7) is

$$K_\theta^{(1)}(t) = \sum_{i=1}^{p} \left\{ \frac{1}{2}\frac{1}{\lambda_i - t} + \frac{1}{4}\frac{\gamma_i^2}{(\lambda_i - t)^2} \right\}$$

and higher derivatives of (10.7) are given by

$$K_\theta^{(j)}(t) = \sum_{i=1}^{p} \left\{ \frac{(j-1)!}{2}\frac{1}{(\lambda_i - t)^j} + \frac{j!}{4}\frac{\gamma_i^2}{(\lambda_i - t)^{j+1}} \right\}.$$

The first order saddlepoint density approximation of $f_\theta(\alpha)$ (the f_θ evaluated at a point α) is

$$\hat{f}_{\theta,1}(\alpha) = \left[2\pi \hat{K}_\theta^{(2)}(\hat{t}) \right]^{-1/2} \exp\left(\hat{K}_\theta(\hat{t}) - \hat{t} \right), \tag{10.8}$$

where $\hat{t}$ is the unique solution in $(-\infty, \lambda_1)$ to the saddlepoint equation $\hat{K}_\theta^{(2)}(\hat{t}) = \alpha$ and in our case $\alpha = 1$ (see the paper by Kume and Wood (2005) for more information why). In fact the $\hat{t}$ has a bounded range (it is a simple form) but we will not mention it here and $\hat{t}$ can be found accurately using numerical methods, e.g. as a root solver (available in R).

The second and third order saddlepoint density approximations of $f_\theta(\alpha)$ are given by

$$\hat{f}_{\theta,2}(1) = \hat{f}_{\theta,1}(1)(1 + T) \quad \text{and} \quad \hat{f}_{\theta,3}(1) = \hat{f}_{\theta,1}(1)\exp(T) \quad \text{respectively,} \tag{10.9}$$

where $T = \frac{1}{8}\hat{\rho}_4 - \frac{5}{24}\hat{\rho}_3^2$, with $\hat{\rho}_j = \frac{K_\theta^{(j)}(\hat{t})}{\left[K_\theta^{(2)}(\hat{t})\right]^{j/2}}$.

The Fisher-Bingham normalising constant is written as

$$c\left(\boldsymbol{\lambda},\boldsymbol{\gamma}\right)=2\pi^{p/2}\left(\prod_{i=1}^{p}\lambda_i^{-1/2}\right)f_{\theta}\left(1\right)\exp\left(\frac{1}{4}\sum_{i=1}^{p}\frac{\gamma_i^2}{\lambda_i}\right), \tag{10.10}$$

where $f_{\theta}\left(1\right)$ is found in Kume and Wood (2005).

The saddlepoint approximations of the Fisher-Bingham normalizing constant (10.10) are given by

$$\hat{c}_1\left(\boldsymbol{\lambda},\boldsymbol{\gamma}\right)=2^{1/2}\pi^{(p-1)/2}\left[K_{\theta}^{(2)}\left(\hat{t}\right)\right]^{-1/2}\left[\prod_{i=1}^{p}\left(\lambda_i-\hat{t}\right)^{-1/2}\right]\exp\left(-\hat{t}+\frac{1}{4}\sum_{i=1}^{p}\frac{\gamma_i^2}{\lambda_i-\hat{t}}\right),$$

$$\hat{c}_2\left(\boldsymbol{\lambda},\boldsymbol{\gamma}\right) \;=\; \hat{c}_1\left(\boldsymbol{\lambda},\boldsymbol{\gamma}\right)\left(1+T\right) \;\text{ and }\; \hat{c}_3\left(\boldsymbol{\lambda},\boldsymbol{\gamma}\right)=\hat{c}_3\left(\boldsymbol{\lambda},\boldsymbol{\gamma}\right)\exp\left(T\right).$$

The R function below calculates the saddlepoint approximations of the normalizing constants of the Fisher, the Bingham and the Fisher-Bingham distribution. For the Bingham part it only accepts the eigenvalues of the **B** matrix. All you need to do is give it what it needs.

In Kume and Wood (2005) there is an important property which we should take into account. On page 468 of their paper they state that "*A useful practical consequence of this equivariance property is that, when using the approximation* $\hat{c}_k\left(\lambda,\gamma\right)$ *we can dispense with the restriction that the* λ_i *be strictly positive, even though, in the saddlepoint density approximation (11), the* λ_i *do need to be positive*". But what is this equivariance property they are referring to? This property states that

$$c\left(\boldsymbol{\lambda},\boldsymbol{\gamma}\right)=c\left(\boldsymbol{\lambda}+a\mathbf{1}_p,\boldsymbol{\gamma}\right)e^{a}.$$

So, in the case where one or possibly more eigenvalues of the **B** matrix are negative, if we make them all positive, by adding a scalar a, then the final saddlepoint approximation to the normalizing constant must by multiplied by the exponential of that scalar. This I would say is a property which helps things a lot. A final notice, is that the next R function calculates the logarithm of the normalizing constant.

If you are a Matlab user, then you are directed to Simon Preston's homepage. In his section *Files* you can find Matlab codes to calculate the saddlepoint approximations of the Fisher-Bingham distribution. These codes were designed for the normalizing constant of the Fisher-Bingham distributions products of spheres and Stiefel manifolds, using Monte Carlo methods as well (see Kume et al. (2013)). A main difference the reader must notice is that in Kume et al. (2013) the Bingham part in the Fisher-Bingham density does **not** have a minus sign $(-)$ as in our case (see (10.6), there is a minus sign). Simon's code uses the notation of Kume et al. (2013). Furthermore, in Simon's section *Shape analysis* the interested reader will

find Matlab codes for shape analysis.

```
fb.saddle <- function(gam, lam) {
  ## gam is the parameters of the Fisher part
  ## lam is the eigenvalues of the matrix of the Bingham part
  lam <- sort(lam)  ## sorts the eigenvalues of the Bingham part
  mina <- min(lam)
  if (mina <= 0) {
      aaa <- abs(mina) + 1
      lam <- lam + aaa ## makes all the lambdas positive and greater than zero
  }
  p <- length(gam)  ## dimensionality of the distribution
  para <- c(gam, lam)  ## the parameters of the Fisher-Bingham
  saddle.equat <- function(ta, para) {
    ## saddlepoint equation
    p <- length(para)/2
    gam <- para[1:p]
    lam <- para[ -c(1:p) ]
    f <- sum( 0.5/(lam - ta) + 0.25 * ( gam^2/(lam - ta)^2 ) ) - 1
    f
  }
  low <- lam[1] - 0.25 * p - 0.5 * sqrt(0.25 * p^2 + p * max(gam)^2)  ## lower bound
  up <- lam[1] - 0.25 - 0.5 * sqrt(0.25 + min(gam)^2)  ## not the exact upper
  ## bound but a bit higher
  ela <- uniroot(saddle.equat, c(low, up), para = para, tol = 1e-08)
  tau <- ela$root  ## tau which solves the saddlepoint equation
  ### below are the derivatives of the cummulant generating function
  kfb <- function(j, gam, lam, ta) {
    if (j == 1) {
      kd <- sum( 0.5/(lam - ta) + 0.25 * ( gam^2/(lam - ta)^2 ) )
    } else if (j > 1) {
      kd <- sum( 0.5 * factorial(j - 1)/(lam - ta)^j + 0.25 * factorial(j) *
        gam^2/(lam - ta)^(j + 1) )
    }
    kd
  }
  rho3 <- kfb(3, gam, lam, tau)/kfb(2, gam, lam, tau)^1.5
  rho4 <- kfb(4, gam, lam, tau)/kfb(2, gam, lam, tau)^2
  Ta <- rho4/8 - 5/24 * rho3^2
  c1 <- 0.5 * log(2) + 0.5 * (p - 1) * log(pi) -
```

```
        0.5 * log( kfb(2, gam, lam, tau) ) - 0.5 * sum( log(lam - tau) ) -
        tau + 0.25 * sum( gam^2/(lam - tau) )
  ## c1 <- sqrt(2) * pi^(0.5 * (p - 1) ) * kfb(2, gam, lam, tau)^(-0.5) *
  ## prod(lam - tau)^(-0.5) * exp( -tau + 0.25 * sum( gam^2/(lam - tau) ) )
  c2 <- c1 + log1p(Ta)
  c3 <- c1 + Ta
  ## the next multiplications brings the modification with the negative
  ## values in the lambdas back
  if (mina <= 0) {
    c1 <- c1 + aaa
    c2 <- c2 + aaa
    c3 <- c3 + aaa
  }
  logcon <- c(c1, c2, c3)
  names(logcon) <- c("first order", "second order", "third order")
  logcon
}
```

10.7.7 Normalizing constant of the Bingham and the Fisher-Bingham distributions using MATLAB

As we mentioned before Simon Preston's homepage contains Matlab codes for calculating the normalizing constant of the Fisher-Bingham distribution. For those who rely more on Matlab than R and for those who want to calculate the normalizing constant using Monte Carlo for example or want the normalizing constant on products of spheres and stiefel manifolds and do not know R the answer is here. Kwang-Rae Kim from the university of Nottingham helped me create a front end with Matlab. That is, implement Matlab functions in Matlab and get the answer using only R. The user needs to have a Matlab v6 or higher installed on his/her computer.

At first we need to connect R with Matlab. For this reason we must download the R package R.matlab (Bengtsson, 2014). We then save the file *FB.zip* from Simon Preston's homepage into our computer. The .zip file has regular folder inside called *FB_norm_const*. Inside *FB_norm_const* there are two folders, *spheres* and *stiefel*. We are interested in the first folder (I do not know much about stiefel manifolds). The reader who knows can do the same as the ones we describe below.

We take the folder *spheres* and save it somewhere in our computer (desktop?). You can also unzip the *FB.zip* file and do the same things.

We then load the library into R and do the following steps

1. Change the working directory of R to the folder *spheres*.

2. Type **Matlab$startServer()**
   Wait until the server is open, wait. This will create three files in the folded *spheres*. Next time you do the same work, delete them first. I do not think it affects next time, but just in case.

3. Type **matlab=Matlab()**

4. Type **isOpen=open(matlab)**

5. Type **isOpen** (the answer should be TRUE).

We are almost there, Matlab, we have connection. Open the folder *spheres* to see what's in there. We are interested in two Matlab functions *logNormConstSP* and *logNormConstMC*. The first uses saddlepoint approximation and the second uses Monte Carlo. I will show how to use the first one only (the syntax for Monte Carlo is the same apart from an extra parameter, n, the number of Monte Carlo samples) in the one sphere case only. For the case of products of spheres see the function inside. Simon explains the arguments.

The function has this name **logC = logNormConstSP(d,a,B,approxType)**. The argument **d** is the number of dimensions, the argument **a** is the vector $\boldsymbol{\gamma}$ in (10.6) and the argument **B** is the matrix $-\mathbf{A}$ in (10.6). A key thing is that in Kume et al. (2013) the Bingham part in the Fisher-Bingham density does **not** have a minus sign $(-)$ as in our case (in (10.6) there is a minus sign). Finally **approxType** takes the values **1**, **2** or **3** corresponding to the first (10.8), second and third order (10.9) saddlepoint approximations. The value **4** produces a vector with all three orders. A second key thing we must highlight is that Simon calculates the logarithm of the constant, so the final answer should be exponentiated.

Let us calculate for example the Bingham normalizing constant. This means that $\boldsymbol{\gamma} = \mathbf{0}$ and **B** is a matrix. We say that the eigenvalues of **B** are $(1, 2, 3)$. This means that Simon's Matlab code needs the negative eigenvalues. Or in general, the negative of the matrix **B** we have. Let us see this example. Type in R

```
evaluate(matlab, "logC = logNormConstSP(3, [0 0 0]', diag([-1 -2 -3]), 3);")
## Wait until the command is executed, wait.
res <- getVariable(matlab, "logC")
res
```

You should see this

```
$logC
          [,1]
[1,] 0.6595873


attr(," header ")
```

```
attr(,"header")$description
[1] "MATLAB 5.0 MAT?file, Platform: PCWIN64, Created on: Wed Feb 19 11:36:59 2014
"
attr(,"header")$version
[1] "5"
attr(,"header")$endian
[1] "little"
```

The answer is the logarithm of the third order (10.9) saddlepoint approximation to the normalizing constant of the Bingham distribution (the vector $\boldsymbol{\gamma}$ is zero). The result is the (**res$logC**). Compare this with the answer from **fb.saddle(c(0,0,0),c(1,2,3))**.

Below we summarize the steps in two R codes. At first the user must run these commands (copy and paste as they are) in order make the connection between the two programs.

```
require(R.matlab, quiet = TRUE, warn.conflicts = FALSE)
Matlab$startServer()
Sys.sleep(30)
matlab <- Matlab()
isOpen <- open(matlab)
```

Then the function one needs to use every time for calculating the Fisher-Bingham normalizing constant (using saddlepoint approximation or Monte Carlo integration) given below. The convenience of this function is that one does not need to know the Matlab syntax. Note, that the input parameters are the same as in the function *fb.saddle*. That is, put the same matrix **B** or the eigenvalues. Inside the function, I put a minus sign ($-$) to agree with Simon's code. The parameter *d* is a number or a vector of length equal to the number of spheres we have (Kume et al. (2013) calculate the normalizing constant for product of spheres, not just one sphere). If it is a number then it contains the number of dimensions of the sphere. If it is a vector, then it contains the dimensions of the spheres. Note, all the spheres in the case have the same dimensions. The parameter *a* is the Fisher part of the Fisher-Bingham distribution and the matrix **B** is the Bingham part. Do not forget to change the directory of R the folder *spheres* as we said before.

```
FB_saddle <- function(d, a, B, method = "SP") {
  ## d is a vector of length k, where k is the number of spheres
  ## if k=1 (one sphere), then d is a number showing the dimensions of the sphere
  ## if k=2, then we have two spheres and d=c(3,3) for example,
  ## meaning that we have two spheres of dimensions 3 each
  ## a is the gamma parameter, the Fisher part
  ## B is the matrix parameter, the Bingham part
  ## method can be either "SP" or "MC"
```

```
  setVariable(matlab, d = d)
  setVariable(matlab, a = a)
  setVariable(matlab, B = -B)
  if (method == "SP") {
    ## this does saddlepoint approximation
    evaluate(matlab, "logC = logNormConstSP(d, a, B, 3) ; ")
    res <- getVariable(matlab, "logC")
    result <- list(norm.const = res$logC)
  }
  if (method == "MC") {
    ## this does Monte Carlo integration
    evaluate(matlab, "[logC, se_logC] = logNormConstMC(d, a, B, 1e + 05) ; ")
    res <- getVariable(matlab, "logC")
    se.const <- getVariable(matlab, "se_logC")
    result <- list(norm.const = res$logC, se.norm.const = se.const$se.logC)
  }
  result
}
```

10.7.8 The Kent distribution on the sphere

The Kent distribution was proposed by John Kent (Kent, 1982) as a sub-model of the Fisher-Bingham distribution on the sphere. So, I will focus on the sphere only here. It's density function is given by (Kent, 1982)

$$f(\mathbf{x}) = c(\kappa, \beta)^{-1} \exp\left\{\kappa \boldsymbol{\alpha}_1^T \mathbf{x} + \beta\left[\left(\boldsymbol{\alpha}_2^T \mathbf{x}\right)^2 - \left(\boldsymbol{\alpha}_3^T \mathbf{x}\right)^2\right]\right\}, \tag{10.11}$$

where κ, β and $\mathbf{A} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_3)$ are parameters that have to be estimated. Kent (1982) mentions that the $\kappa \leq 0$ and $\beta \leq 0$ represent the concentration and the ovalness of the distribution respectively and these two parameters will be estimated via numerical maximization of the log-likelihood. The normalizing constant in (10.11) depends upon these two parameters only but its calculation is almost impossible up to now. For this reason we will approximate it using the saddlepoint approximation of Kume and Wood (2005) we saw before (see Section 10.7.6). We need to suppose though that $2\beta < \kappa$ in order for the distribution to have the correct behaviour. Note that if $\beta = 0$, then we have the von Mises-Fisher density. Finally $\mathbf{A}$ is an orthogonal matrix where $\boldsymbol{\alpha}_1$ is the mean direction or pole, $\boldsymbol{\alpha}_2$ is the major axis and $\boldsymbol{\alpha}_3$ is the minor axis.

The Fisher Bingham distribution is written as

$$f(\mathbf{x}) \propto \exp\left(\kappa \mathbf{x}^T \boldsymbol{\mu} + \mathbf{x}^T \mathbf{A} \mathbf{x}\right) \text{ or as } f(\mathbf{x}) \propto \exp\left(\kappa \mathbf{x}^T \boldsymbol{\mu} - \mathbf{x}^T \mathbf{A} \mathbf{x}\right).$$

The first form is where (10.11) comes from but the second form is used in Kent et al. (2013) and in Kume and Wood (2005). In the first case $\mathbf{A} = \text{diag}(0, \beta, -\beta)$. We will use the second case, since the normalizing constant (Section 10.7.6) utilizes the second formula. In both cases though, the normalizing constant depends upon κ and β only. The normalizing constant we saw in Section 10.7.6 requires the γ vector and the λ vector. In the second case we need to use $\gamma = (0, \kappa, 0)^T$ and $\lambda = (0, -\beta, \beta)^T$ as input values in the function *fb.saddle* we saw in Section 10.7.6. In terms of Simon's MATLAB function (see Section 10.7.7) we would specify $\gamma = (0, 0, \kappa)^T$ and $\lambda = (\beta, -\beta, 0)^T$.

So, the log-likelihood of the Kent distribution from (10.11) is

$$\ell = -n * c(\kappa, \beta) + \kappa \sum_{i=1}^{n} \boldsymbol{\alpha}_1^T \mathbf{x}_i + \beta \left[\sum_{i=1}^{n} \left(\boldsymbol{\alpha}_2^T \mathbf{x}_i \right)^2 - \sum_{i=1}^{n} \left(\boldsymbol{\alpha}_3^T \mathbf{x}_i \right)^2 \right]. \tag{10.12}$$

We will now describe the estimation the parameters of (10.11) as Kent (1982) mentions. For the orthogonal matrix $\mathbf{A}$ we will mention the moment estimation. We must choose an orthogonal matrix $\mathbf{H}$ to rotate the mean vector $\bar{\mathbf{x}} = n^{-1} \left(\sum_{i=1}^{n} \mathbf{x}_{1i}, \sum_{i=1}^{n} \mathbf{x}_{2i}, \sum_{i=1}^{n} \mathbf{x}_{3i} \right)^T$ to the north polar axis $(1, 0, 0)^T$. So, $\mathbf{H}$ can be

$$\mathbf{H} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta\cos\phi & \cos\theta\cos\phi & -\sin\phi \\ \sin\theta\sin\phi & \cos\theta\sin\phi & -\cos\phi \end{bmatrix},$$

where θ and ϕ are the polar co-ordinates of $\bar{\mathbf{x}}$. Let $\mathbf{B} = \mathbf{H}^T \mathbf{S} \mathbf{H}$, where $\mathbf{S} = n^{-1} \sum \mathbf{x}_i \mathbf{x}_i^T$. Then choose a rotation $\mathbf{K}$ about the north pole to diagonalize $\mathbf{B}_L$, where

$$\mathbf{B}_L = \begin{bmatrix} b_{22} & b_{23} \\ b_{32} & b_{33} \end{bmatrix}$$

is the lower 2×2 sub-matrix of $\mathbf{B}$, with eigenvalues $l_1 > l_2$. If we choose ψ such that $\tan(2\psi) = 2b_{23} / (b_{22} - b_{33})$, ensuring that $\| \bar{\mathbf{x}} \| > 0$ and $l_1 > l_2$ then we can take

$$\mathbf{K} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix}.$$

The moment estimate of $\mathbf{A}$ is given by $\tilde{\mathbf{A}} = \mathbf{HK}$. As for the parameters κ and β we will maximize (10.12) with respect to these two parameters. I repeat that we will use $\gamma = (0, \kappa, 0)^T$

and $\lambda = (0, -\beta, \beta)^T$ as input values in the function *fb.saddle* we saw in Section 10.7.6. The next R function calculates the **A** matrix, the κ and β and the log-likelihood and has been tested with the data that appear in Kent (1982). Some elements in the **A** matrix are slightly different (numerical errors possibly), but I do not think this is an issue.

In a recent communication I had with Professor Kent (Leeds university) he wrote this in his e-mail: *"Note that* $2|\beta|/\kappa$ *is analogous in some ways to the correlation parameter for a bivariate normal distribution. In particular, negative values are just as meaningful as positive values"*.

```
kent.mle <- function(x) {
  ## x is the data in Euclidean coordinates

  tic <- proc.time()

  n <- dim(x)[1]  ## sample size
  xbar <- Rfast::colmeans(x)  ## mean vector
  xbar <- xbar / sqrt( sum(xbar^2) ) ## mean direction
  u <- c( acos(xbar[1]), ( atan(xbar[3] / xbar[2]) + pi * I(xbar[2]<0) )
  %% (2 * pi) )
  ## u is the mean vector to latitude and longitude
  theta <- u[1]
  phi <- u[2]
  costheta <- cos(theta)
  sintheta <- sin(theta)
  cosphi <- cos(phi)
  sinphi <- sin(phi)

  H <- matrix( c(costheta, sintheta * cosphi,
       sintheta * sinphi, -sintheta, costheta * cosphi,
       costheta * sinphi, 0, -sinphi, cosphi), ncol = 3)
  S <- crossprod(x) / n
  B <- crossprod(H, S) %*% H
  psi <- 0.5 * atan(2 * B[2, 3]/(B[2, 2] - B[3, 3]))
  K <- matrix(c(1, 0, 0, 0, cos(psi), sin(psi), 0,
       -sin(psi), cos(psi)), ncol = 3)
  G <- H %*% K  ## The G matrix Kent describes, the A in our notation
  r1 <- sqrt( sum(xbar^2) )
  lam <- eigen(B[-1, -1])$values
  r2 <- lam[1] - lam[2]

  ## the next function will be used to estimate the kappa and beta
```

```
  xg1 <- sum( x %*% G[, 1] )
  xg2 <- sum( ( x %*% G[, 2] )^2 )
  xg3 <- sum( ( x %*% G[, 3])^2 )

  mle <- function(para) {
    ## maximization w.r.t. to k and b
    k <- para[1]
    b <- para[2]
    gam <- c(0, k, 0)
    lam <- c(0, -b, b)
    ckb <- fb.saddle(gam, lam)[3]
    g <-  -( -n * ckb + k * xg1 + b * ( xg2 - xg3 ) )
    g
  }

  ini <- vmf(x)$k
  ini <- c(ini, ini/2.1)  ## initial values for kappa and beta
  qa <- optim(ini, mle)
  para <- qa$par
  k <- para[1]
  b <- para[2]  ## the estimated parameters
  gam <- c(0, k, 0)
  lam <- c(0, -b, b)
  ckb <- as.numeric( fb.saddle(gam, lam)[3] )
  ## the line below calculates the log-likelihood
  l <-  -n * ckb + k * xg1 + b * ( xg2 - xg3 )
  para <- c(k, b)

  runtime <- proc.time() - tic

  names(para) <- c("kappa", "beta")
  colnames(G) <- c("mean", "major", "minor")

  list(G = G, para = para, logcon = ckb, loglik = l, runtime = runtime)

}
```

Kent (1982) gave the formula to calculate the normalising constant exactly

$$c(\kappa,\beta)=2\pi\sum_{j=1}^{\infty}\frac{\Gamma(j+0.5)}{\Gamma(j+1)}\beta^{2j}\left(\frac{\beta}{2}\right)^{-2j-0.5}I_{2j+0.5}(\kappa), \tag{10.13}$$

where $I_v(z)$ denotes the modified Bessel function of the first kind and order v calculated at z.

I did a few experiments and saw that the saddlepoint approximation Kume and Wood (2005) gives very accurate results, very very close to the true values. I am using the saddlepoint approximation though in the function *kent.mle* because it is faster than the exact calculation.

```
kent.logcon <- function(k, b, j = 100) {

   j <- 0:j
   ka <- 2 * pi * gamma(j + 0.5) / gamma(j + 1)* b^( 2 * j ) *
   ( k / 2 )^( -2 * j - 0.5 ) * besselI(k, 2 * j + 0.5)
   log( sum(ka) )

}
```

10.7.9 Fisher versus Kent distribution

Kent (1982) proposed a test statistic to test whether a von Mises-Fisher distribution on the sphere is preferable to a Kent distribution. To be honest, I did not make the test statistic. Something is wrong, I did not get it and I made a mistake, I don't know. For this reason I will describe the test as I found it in Rivest (1986).

Hypothesis test of Fisher versus Kent distribution on the sphere

1. Calculate the sample mean direction $\hat{\boldsymbol{\mu}}$ and the sample concentration parameter $\hat{\kappa}$ assuming a von Mises-Fisher model on the sphere with $\mathbf{x}$ being the sample data of sample size equal to n.

2. Calculate the orthogonal matrix

$$\hat{\mathbf{P}}=\mathbf{I}_3-\frac{\left(\mathbf{e}_1-\hat{\boldsymbol{\mu}}\right)\left(\mathbf{e}_1-\hat{\boldsymbol{\mu}}\right)^T}{1-\hat{\mu}_1},$$

   where $\mathbf{e}_1=(1,0,0)^T$ and $\hat{\mu}_1$ is the first element of the sample mean direction. Not, that $\hat{\mathbf{P}}$ is a symmetric matrix whose first column (or first row) is the sample mean direction $\hat{\boldsymbol{\mu}}$.

3. Calculate $\mathbf{z} = \hat{\mathbf{P}}\mathbf{x}$ and take $\mathbf{y}$ which consists of the last two columns of the $\mathbf{z}$ matrix $\mathbf{y} = (z_{2i}, z_{3i})$.

4. Calculate the two eigenvalues l_1 and l_2 of $\mathbf{S} = \sum_{i=1}^{n} \mathbf{y}_i \mathbf{y}_i^T$.

5. Kent's statistic is written as

$$\hat{T} = n \left(\frac{\hat{\kappa}}{2} \right)^2 \frac{I_{1/2}(\hat{\kappa})}{I_{5/2}(\hat{\kappa})} (l_1 - l_2)^2.$$

The R function presented below offers the possibility of non parametric bootstrap as well.

```
fishkent <- function(x, B = 999) {
  ## x contains the data
  ## B is by default eaual to 999 bootstrap re-samples
  ## If B==1 then no bootstrap is performed

  n <- dim(x)[1]  ## sample size
  estim <- vmf(x)
  k <- estim$k  ## the estimated concentration parameter
  ## under the H0, that the Fisher distribution is true
  mu <- estim$mu  ## the estimated mean direction under H0
  e1 <- c(1, 0, 0)
  i3 <- diag(3)
  P <- i3 -  tcrossprod(e1 - mu) / (1 - mu[1])
  y <- tcrossprod(x, P)[, 2:3]
  lam <- eigen( crossprod(y) / n, symmetric = TRUE )$values
  rat <- besselI(k, 0.5, expon.scaled = TRUE) / besselI(k, 2.5, expon.scaled = TRUE)
  Ta <- n * (k / 2)^2 * rat * (lam[1] - lam[2])^2

  if (B == 1) {
    pvalue <- pchisq(Ta, 2, lower.tail = FALSE)
    res <- c(Ta, pvalue)
    names(res) <- c('test', 'p-value')

  } else {
    Tb <- numeric(B)
    for (i in 1:B) {
      nu <- sample(1:n, n, replace = TRUE)
      z <- x[nu, ]
      estim <- vmf(z)
```

```
      k <- estim$k  ## the estimated concentration parameter
      ## under the H0, that the Fisher distribution is true
      mu <- estim$mu  ## the estimated mean direction under H0
      P <- i3 -  tcrossprod(e1 - mu) / (1 - mu[1])
      y <- tcrossprod(z, P)[, 2:3]
      lam <- eigen( crossprod(y) / n, symmetric = TRUE )$values
      rat <- besselI(k, 0.5, expon.scaled = TRUE)/besselI(k, 2.5, expon.scaled = TRUE)
      Tb[i] <- n * (k/2)^2 * rat * (lam[1] - lam[2])^2
    }

    res <- c( Ta, (sum(Tb > Ta) + 1)/(B + 1) )
    names(res) <- c('test', 'Bootstrap p-value')
  }

  res
}
```

10.8 Simulation of random values

10.8.1 Simulation from a von Mises-Fisher distribution

Wood (1994) provided a new algorithm for simulating from the von Mises-Fisher distribution. It is essentially a ejection sampling algorithm which we meet it again in Dhillon and Sra (2003). We wrote the R code presented below based on the paper by Dhillon and Sra (2003). The arguments of the algorithm are $\boldsymbol{\mu}$, k and n, the mean direction, the concentration parameter and the sample size. The algorithm given below generates vectors from the mean direction $(0, \ldots, 0, 1)$ and then using the rotation matrix (10.1) we transform the vectors so that they have the desired mean direction. This algorithm works for arbitrary q in $\mathbb{S}^q$.

Algorithm to simulate from the von Mises-Fisher distribution

1. $p = \dim(\boldsymbol{\mu})$, the dimension of the data
2. $\mathbf{ini} = (0, \ldots, 0, 1)$, the initial mean direction
3. $b = \frac{-2k + \sqrt{4k^2 + (p-1)^2}}{p-1}$
4. $x_0 = \frac{1-b}{1+b}$
5. $m = \frac{p-1}{2}$

6. $c = kx_0 + (d-1)\log\left(1 - x_0^2\right)$
7. S is a matrix with n rows and p columns
8. for $i \ \ in \ \ 1:n$
   - $t = -1000$
   - $u = 1$
   - **while** $(t - c < \log(u))$
     - Generate z from $Beta\,(m, m)$ and u from $U\,(0, 1)$
     - $w = \frac{1-(1+b)*z}{1-(1-b)*z}$
     - $t = k * w + (p-1) * \log(1 - x0 * w)$
9. Generate **v1** from $N_{p-1}\left(\mathbf{0}, \mathbf{I}_{p-1}\right)$
10. $\mathbf{v} = \frac{\mathbf{v1}}{\|\mathbf{v1}\|}$. This is a uniform $p-1$ dimensional unit vector
11. $S[i,] = \left(\sqrt{1 - w^2} * \mathbf{v}, \mathbf{w}\right)$
12. Calculate the rotation matrix **A** using (10.1) in order to rotate the initial mean direction from **ini** to $\boldsymbol{\mu}$.
13. **X**=**AS**. The **X** comes from a von Mises-Fisher distribution with concentration parameter k and mean direction $\boldsymbol{\mu}$.

The R code given below is a bit slower than the the function found in Hornik and Grün (2014) but it still sees the job through and you can see what the algorithm does.

```
rvmf <- function(n, mu, k) {
  ## n is the sample size
  ## mu is the mean direction and
  ## k is the concentration parameter
  ## n is the sample size
  d <- length(mu)  ## the dimensions
  if (k > 0) {
    mu <- mu / sqrt( sum(mu^2) )  ## the mean direction
    ini <- c(numeric(d - 1), 1)  ## mean direction is set to (0, ..., 0, 1)
    d1 <- d - 1
    v1 <- matrix( RcppZiggurat::zrnorm(n * d1), ncol = d1)
    v <- v1 / sqrt( rowsums(v1^2) )
    b <- ( -2 * k + sqrt(4 * k^2 + d1^2) ) / d1
    x0 <- (1 - b)/(1 + b)
```

```
    w <- numeric(n)
    m <- 0.5 * d1
    ca <- k * x0 + (d - 1) * log(1 - x0^2)

    for (i in 1:n) {
      ta <-  -1000
      u <- 1
      while ( ta - ca < log(u) ) {
        z <- rbeta(1, m, m)
        u <- runif(1)
        w[i] <- ( 1 - (1 + b) * z ) / ( 1 - (1 - b) * z )
        ta <- k * w[i] + d1 * log(1 - x0 * w[i])
      }
    }
    S <- cbind(v, w)
    A <- rotation(ini, mu)  ## calculate the rotation matrix
    ## in order to rotate the initial mean direction from ini to mu
    x <- tcrossprod(S, A)  ## the x has direction mu
  } else {  ## uniform distribution
    ## requires MASS if k = 0
    x1 <- matrix( RcppZiggurat::zrnorm(n * d), ncol = d )
    x <- x1 / sqrt( Rfast::rowsums(x1^2) )
  }
  x
}
```

10.8.2 Simulation from a Bingham distribution

Kent et al. (2013) proposed the angular central Gaussian (ACG) distribution (Tyler, 1987) as an envelope distribution in the rejection sampling algorithm for generating random values from a Bingham distribution. The Bingham distribution on the (hyper)sphere $\mathbb{S}^{q-1}$ is written as

$$f_{bing}(\mathbf{x}) = c_{bing} e^{\left(-\mathbf{x}^T \mathbf{A} \mathbf{x}\right)} = c_{bing} f^*_{bing}(\mathbf{x}),$$

where c_{bing} is the normalizing constant and $\mathbf{A}$ is a $q \times q$ symmetric matrix. The density of the central angular distribution is

$$f_{ACG}(\mathbf{x}) = c_{ACG} f^*_{ACG}(\mathbf{x}),$$

where where $c_{ACG} = \frac{\Gamma(q/2)}{2\pi^{q/2}} |\mathbf{\Omega}|^{-1/2}$ is the normalizing constant and $f^*_{ACG}(\mathbf{x}) = \left(\mathbf{x}^T \mathbf{\Omega} \mathbf{x}\right)^{-q/2}$.

To simulate a random value from the ACG one has to generate a random value from a multivariate normal and then normalize it such that its unit vector is 1. If $\mathbf{y} \sim N_q(0, \mathbf{\Sigma})$, then $\mathbf{x} = \frac{\mathbf{y}}{\|\mathbf{y}\|}$ follows an $ACG(\mathbf{\Omega})$ with $\mathbf{\Omega} = \mathbf{\Sigma}^{-1}$.

Before we explain the algorithm of how simulate from the Bingham distribution we will say a few tricks. First, we will obtain the eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_q$ of the symmetric matrix $\mathbf{A}$. Then subtract the smallest eigenvalue from them all and thus we have $\lambda'_1 \geq \lambda'_2 \geq \ldots \lambda'_q = 0$. Then form the diagonal matrix $\mathbf{\Lambda}' = \text{diag}\left\{\lambda'_1, \ldots, \lambda'_q\right\}$. As Fallaize and Kypraios (2014) mention, if $\mathbf{x}$ comes from a Bingham with matrix parameter $\mathbf{A}$, then $\mathbf{y} = \mathbf{xV}$ comes from a Bingham with matrix parameter $\mathbf{\Lambda}$, and this matrix comes from the spectral decomposition of $\mathbf{A} = \mathbf{V\Lambda V}^T$.

The next code simulates observations from a Bingham distribution with a diagonal matrix parameter say $\mathbf{\Lambda}'$. The input eigenvalues are the $q-1$ non zero eigenvalues λ'_i for $i = 1 \ldots, q-1$. So, if you right multiply the matrix containing the simulated values by $\mathbf{V}^T$ the transformed matrix contains the simulated values from a Bingham with a matrix parameter $\mathbf{A}$.

The constant changes only and in fact if we subtract or add the same scalar to all eigenvalues the constant is multiplied or divided respectively, by the exponential of that scalar.

One more key thing we have to highlight is that this distribution is used for modelling axial data. This is because it has the so called antipodal symmetry. That is, the direction is not important, the sign in other words is irrelevant in contrast to the von Mises or the von Mises-Fisher distribution. Thus, $f_{bing}(\mathbf{x}) = f_{bing}(-\mathbf{x})$.

The steps to describe the rejection sampling in order to simulate from a Bingham distribution are a combination of Kent et al. (2013) and of Fallaize and Kypraios (2014).

Algorithm to simulate from a Bingham distribution

1. Set $\mathbf{\Omega} = \mathbf{\Omega}(b) = \mathbf{I}_q + \frac{2}{b}\mathbf{B}$ and $M = e^{-0.5(q-b)} (q/b)^{q/2}$.
2. Draw a u from $U(0,1)$ and a $\mathbf{z}$ from $ACG(\mathbf{\Omega})$.
3. If $u < \frac{e^{\left(-\mathbf{z}^T \mathbf{A} \mathbf{z}\right)}}{M(\mathbf{z}^T \mathbf{\Omega} \mathbf{z})^{-q/2}}$ accept $\mathbf{z}$
4. Repeat steps $2-3$ until the desired number of random values is obtained.

Christopher Fallaize and Theo Kypraios from the university of Nottingham have provided the following R code for simulating from a Bingham distribution. They have set $b = 1$, even though it's not the otpimal solution but as they say it works well in practice.

```
f.rbing <- function(n, lam) {
  ## n is the sample size
```

```
  ## lam are the q - 1 non zero eigenvalues

  lam <- sort(lam, decreasing = TRUE)  ## sort the eigenvalues in desceding order
  nsamp <- 0
  X <- NULL
  lam.full <- c(lam, 0)
  qa <- length(lam.full)
  mu <- numeric(qa)
  sigacginv <- 1 + 2 * lam.full
  SigACG <- diag( 1 / ( 1 + 2 * lam.full ) )

  Ntry <- 0

  while (nsamp < n) {
    x.samp <- FALSE
    while ( !x.samp ) {
      yp <- MASS::mvrnorm(n = 1, mu = mu, Sigma = SigACG)
      y <- yp / sqrt( sum(yp^2) )
      lratio <-  - sum( y^2 * lam.full ) - qa/2 * log(qa) +
      0.5 * (qa - 1) + qa/2 * log( sum(y^2 * sigacginv ) )

      if ( log(runif(1) ) < lratio) {
        X <- c(X, y)
        x.samp <- TRUE
        nsamp <- nsamp + 1
      }
      Ntry <- Ntry + 1
    }
  }

  X <- matrix(X, byrow = TRUE, ncol = qa)
  ## the X contains the simulated values
  ## the avtry is the estimate of the M in rejection sampling
  ## 1/M is the probability of acceptance
  list(X = X, avtry = Ntry/n)

}
```

The next function is a more general than the previous one for a non diagonal symmetric matrix parameter **A** and it calls the previous function.

```
rbingham <- function(n, A) {

  p <- ncol(A)  ## dimensionality of A
  eig <- eigen(A)
  lam <- eig$values  ## eigenvalues
  V <- eig$vectors  ## eigenvectors
  lam <- lam - lam[p]
  lam <- lam[-p]

  ### f.rbing part
  lam <- sort(lam, decreasing = TRUE)  ## sort the eigenvalues in desceding order
  nsamp <- 0
  X <- NULL
  lam.full <- c(lam, 0)
  qa <- length(lam.full)
  mu <- numeric(qa)

  sigacginv <- 1 + 2 * lam.full
  SigACG <- diag( 1 / ( 1 + 2 * lam.full ) )

  Ntry <- 0

  while (nsamp < n) {
    x.samp <- FALSE
    while ( !x.samp ) {
      yp <- MASS::mvrnorm(n = 1, mu = mu, Sigma = SigACG)
      y <- yp / sqrt( sum( yp^2 ) )
      lratio <-  - sum( y^2 * lam.full ) - qa/2 * log(qa) +
        0.5 * (qa - 1) + qa/2 * log( sum(y^2 * sigacginv ) )
      if ( log(runif(1) ) < lratio) {
        X <- c(X, y)
        x.samp <- TRUE
        nsamp <- nsamp + 1
      }
      Ntry <- Ntry + 1
    }
  }

  x <- matrix(X, byrow = TRUE, ncol = qa)
```

```
  ## the x contains the simulated values
  tcrossprod(x, V) ## simulated data

}
```

10.8.3 Simulation from a Fisher-Bingham distribution

The Fisher-Bingham distribution is written as Kent et al. (2013)

$$f_{FB}\left(\mathbf{x}\right) = c_{FB} e^{\left(\kappa \mathbf{x}^T \boldsymbol{\mu} - \mathbf{x}^T \mathbf{A} \mathbf{x}\right)} = c_{FB} f_{FB}^{*}\left(\mathbf{x}\right) \tag{10.14}$$

Kent et al. (2013) mentions that the Fisher-Bingham distribution (10.6) can be bounded by a Bingham density

$$f_{FB}^{*}\left(\mathbf{x}\right) \leq e^{\left(\kappa - \mathbf{x}^T \mathbf{A}^{(1)} \mathbf{x}\right)} = e^{\kappa} e^{\left(-\mathbf{x}^T \mathbf{A}^{(1)} \mathbf{x}\right)}, \tag{10.15}$$

where $\mathbf{A}^{(1)} = \mathbf{A} + (\kappa/2)\left(\mathbf{I}_q - \boldsymbol{\mu}\boldsymbol{\mu}^T\right)$. The story now is known more or less. Initially we use the rejection sampling to generate from this Bingham distribution (see the functions *f.rbing* and *rbingham* in the previous section). Then, we use again rejection sampling to see which of them we will keep. We keep the simulated values for which the inequality (10.15) holds true.

But, initially, we simulate from a Fisher-Bingham with mean direction equal to $(0, 1, 0)$ and then we rotate the data (*rotation* function) such that the mean is where we want it to be.

The next function does something not very clever but at least fast enough. It generates 5 times the requested sample (n) from a Bingham distribution and then sees how many of them are accepted as coming from the Fisher-Bingham distribution. I assume the accepted ones will be more than n and so then it randomly selects n of them. Two rejection samplings take place and that is why I did this. Below is the old code.

```
rfb_old <- function(n, k, m, A) {
  ## n is the required sample size
  ## k is the concentration parameter, the Fisher part
  ## m is the mean vector, the Fisher part
  ## A is the symmetric matrix, the Bingham part

  m <- m / sqrt( sum(m^2) )
  m0 <- c(0, 1, 0)
  mu <- c(0, 0, 0)
  B <- rotation(m0, m)
  q <- length(m0)
  A1 <- A + k/2 * ( diag(q) - m0 %*% t(m0) )
```

```
  eig <- eigen(A1)
  lam <- eig$values
  V <- eig$vectors
  lam <- lam - lam[q]
  lam <- lam[-q]

  x1 <- matrix( 0, n, 3 )
  i <- 1

  while (i <= n) {
    x <- f.rbing(1, lam)$X  ## Chris and Theo's code
    x <- tcrossprod(x, V) ## simulated data
    u <- log( runif(1) )
    ffb <- k * x[, 2]  - sum( x %*% A * x )
    fb <- k - sum( x %*% A1 * x )
    if ( u <= c(ffb - fb) ) {
      x1[i, ] <- x
      i <- i + 1
    }
  }

  tcrossprod(x1, B) ## simulated data with the wanted mean direction
}
```

Can we make it faster? Yes we can. When calling *f.rbing(1, lam)* we can change it to simulate more than one values. If we use a *for* loop to simulate values one by one, that is slow. That is why we simulate many values and check how many of them are accepted. We then simulate more values and check how many are accepted until we sample the desired number of values.

```
rfb <- function(n, k, m, A) {
  ## n is the required sample size
  ## k is the concentration parameter, the Fisher part
  ## m is the mean vector, the Fisher part
  ## A is the symmetric matrix, the Bingham part
  m <- m / sqrt( sum(m^2) )
  m0 <- c(0, 1, 0)
  mu <- c(0, 0, 0)
  B <- rotation(m0, m)
  q <- length(m0)
```

```
  A1 <- A + k/2 * ( diag(q) - m0 %*% t(m0) )
  eig <- eigen(A1)
  lam <- eig$values
  V <- eig$vectors
  lam <- lam - lam[q]
  lam <- lam[-q]
  x <- f.rbing(n, lam, fast = TRUE)$X  ## Chris and Theo's code
  x <- tcrossprod(x, V) ## simulated data
  u <- log( runif(n) )
  ffb <- k * x[, 2]  - Rfast::rowsums( x %*% A * x )
  fb <- k - Rfast::rowsums( x %*% A1 * x )
  x1 <- x[u <= c(ffb - fb), ]
  n1 <- dim(x1)[1]

  while (n1 < n) {
    x <- f.rbing(n - n1, lam, fast = TRUE)$X  ## Chris and Theo's code
    x <- tcrossprod(x, V) ## simulated data
    u <- log( runif(n - n1) )
    ffb <- k * x[, 2]  - Rfast::rowsums( x %*% A * x )
    fb <- k - Rfast::rowsums( x %*% A1 * x )
    x1 <- rbind(x1, x[u <= c(ffb - fb), ])
    n1 <- dim(x1)[1]
  }

  tcrossprod(x1, B) ## simulated data with the wanted mean direction
}
```

If we want to simulate from a Kent distribution then we have to use the *rfb* function we saw in Section 10.8.3. The point is to suitably fix the parameter $\boldsymbol{\mu}$ and $\mathbf{A}$ of (10.14). So for a concentration parameter κ and an ovalness parameter β, we would have to specify the A matrix, the ovalness parameter basically as

```
A <- diag(c(-b, 0, b))
```

where $b > 0$ and then type in R

```
rfb(n, k, m, A)
```

Try this with some values of $\boldsymbol{\mu}$, κ and β and then use the *kent.mle* function above to see the estimates of κ and β.

10.8.4 Simulation of random values from a von Mises-Fisher mixture model

In order to simulate values from mixture model, we need the sample size, the mixing probabilities and the mean vector and concentration parameter of each population. The *rvmf* function will prove useful.

```
rmixvmf <- function(n, prob, mu, k) {
  ## n is the sample size
  ## prob is a vector with the mixing probabilities
  ## mu is a matrix with with the mean directions
  ## k is a vector with the concentration parameters
  p2 <- c( 0, cumsum(prob) )
  p <- ncol(mu)  ## dimensionality of the data
  u <- runif(n)
  g <- length(k)  ## how many clusters are there
  ina <- as.numeric( cut(u, breaks = p2) )  ## the cluster of each observation
  ina <- sort(ina)
  nu <- as.vector(table(ina))  ## frequency table of each cluster
  y <- array(dim = c(n, p, g))
  for (j in 1:g)  y[1:nu[j], , j] <- rvmf(nu[j], mu[j, ], k[j])
  x <- y[1:nu[1], , 1]
  for (j in 2:g)  x <- rbind(x, y[1:nu[j], , j])  ## simulated data
  ## data come from the first cluster, then from the second and so on
  list(id = ina, x = x)
}
```

10.9 Contour plots

10.9.1 Contour plots of the von Mises-Fisher distribution

We provide a simple function to produce contour plots of the von Mises-Fisher distribution. Georgios Pappas from the University of Nottingham made this possible. He explained the idea to me and all I had to do was write the code. The con Mises-Fisher direction needs two arguments, a mean direction ($\boldsymbol{\mu}$) and a concentration parameter (κ). Similar to other distributions, the mean direction is not really important. The shape will not change if the mean direction changes. So we only need the concentration parameter. Since this distribution is rotationally symmetric about its mean the contours will be circles. Rotational symmetry is the analogue of a multivariate normal with equal variance in all the variables and zero correlations. In other words, the covariance matrix is a scalar multiple of the identity matrix.

We rewrite the density as we saw it in (10.2), excluding the constant terms, for conve-

nience purposes.

$$f_p(\mathbf{x}; \mu, \kappa) \propto \exp\left(\kappa \boldsymbol{\mu}^T \mathbf{x}\right),$$

We need a plane tangent to the sphere exactly at the mean direction. The inner product of the a unit vector with the mean direction which appears on the exponent term of the density (10.2) is equal to an angle θ. So for points on the tangent plane we calculate this angle every time and then calculate the density (which needs only κ now). If you did not understand this ask a physicist, they do angles and know of manifolds in general.

Let us see this graphically now. See Figure 10.1 below. Suppose this is one slice of a quarter of a sphere. We have a point on the sphere (A) and want to project it onto the tangent plane. The plane is tangent to the mean direction which is the black vertical line, the segment OB. What we want to do now, is flatten the sphere (or peel off if you prefer), so that the point A touches the plane. The green line is the arc, OA, and the point A" on the plane corresponds to A on the sphere. The important feature here is that the length of OA and the length of OA" are the same. So we projected the point A on the plane in such a way that it's arc length from the mean direction remains the same on the plane. How much is this arc length? The answer is equal to θ radians, where θ is the angle formed by the two radii, OB and BA.

The other case is when we project the chord of the sphere (red line) onto the plane and in this case the point A on the sphere corresponds to point A' on the tangent plane. In this case, the length of OA and OA' are the same. I believe the colours will help you identify the relation between the point on the circle and on the tangent plane.

The mean direction is not important, but the angle between a point on the sphere and its mean direction is, and we only need the concentration parameter to define our contour plots. Similarly to the univariate case, where the relevant distance between the points and the mean is of importance only and not the mean itself and then the variance determines the kurtosis of the distribution. So, here the angle between the observations and the mean direction only is important. Thus, in the plane we take lots of points and we calculate the angles from the mean direction every time. The concentration parameter is what affect what we see.

In this case, the von Mises-Fisher distribution, the contour plots will always be circles, because this distribution is the analogue of an isotropic multivariate normal (no correlation and all variances equal). The higher the concentration parameter κ is, the more gathered the circles are, and so on. Let us highlight that we peeled off the sphere here (i.e. used the green line in Figure 10.1).

```
vmf.contour <- function(k) {
  ## k is the concentration parameter
```

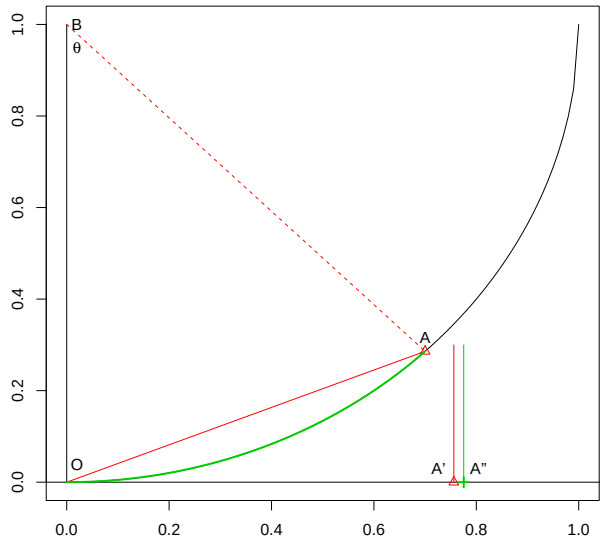

Figure 10.1: A slice of a quarter of the sphere along with a chord and an arc. The red and green lines indicate the projection of the point on the sphere onto the tangent plane.

```
  rho <- pi/2  ## radius of the circular disc
  x <- seq(-rho, rho, by = 0.01)
  n <- length(x)
  mat <- matrix(rep(x^2, n), ncol = n)
  z <- mat + t(mat)
  theta <- sqrt(z)
  ind <- ( theta < rho )  ## checks if x^2+y^2 < rho^2
  ind[ !ind ] <- NA

  xa <- 0.5 * log(k) + k * cos(theta) - 1.5 * log(2 * pi) -
  log( besselI(k, 0.5, expon.scaled = TRUE) ) - k
  mat <- exp(xa) * ind
  contour(x, x, mat)

}
```

10.9.2 Contour plots of the Kent distribution

The Kent distribution as we saw it in (10.11) has the following formula on the sphere

$$f(\mathbf{x}) = c(\kappa, \beta)^{-1} \exp\left\{\kappa \boldsymbol{\alpha}_1^T \mathbf{x} + \beta\left[\left(\boldsymbol{\alpha}_2^T \mathbf{x}\right)^2 - \left(\boldsymbol{\alpha}_3^T \mathbf{x}\right)^2\right]\right\},$$

The parameters κ and β are the two arguments necessary for the construction of the contour plots, since as we said in the case of the von Mises-Fisher distribution, the mean direction is not important, but the angle between it and the points is. As for the two other terms in the exponential, they are also expressed in terms of angles (see also Kent 1982). Let us only say that in this case we used the projection described using the red line in Figure 10.1.

We will mention two more things, first, that this function requires (whenever the Kent distribution is involved actually) the *fb.saddle* function and secondly, note that when $\kappa > \beta$ the distribution is unimodal as Kent (1982) mentions. If the opposite is true, then the distribution is bimodal and has some connections with the Wood distribution Wood (1982).

```
kent.contour <- function(k, b) {
  ## k is the concentration parameter
  ## b is the ovalness parameter
  ## b must be less than k/2

  gam <- c(0, k, 0)
  lam <- c(0, -b, b)
  con <- fb.saddle(gam, lam)[3]
  rho <- sqrt(2)
  x <- seq(-rho, rho, by = 0.01)
  n <- length(x)
  mat1 <- matrix(rep(x^2, n), ncol = n)
  mat2 <- t(mat1)
  z <- sqrt( mat1 + mat2 )
  ind <- ( z^2 < rho^2 )  ## checks if x^2+y^2 < rho^2

  ind[ !ind ] <- NA
  theta <- 2 * asin(0.5 * z)
  xa <- k * cos(theta) + b * (mat1 - mat2) - con
  mat <- exp(xa) * ind
  contour(x, x, mat)

}
```

10.9.3 Contour plots of the Kent distribution fitted to spherical data

This function is different from the previous one. Suppose you have data, a matrix with two columns, the first is latitude and the second is the longitude (you can also have a matrix with three columns and it is transformed into the spherical coordinates internally). We apply the

euclid function to transform the data into Euclidean coordinates, then fit a Kent distribution (*kent.mle* function) and estimate the desired parameters and axes. Then for a grid of points, latitude and longitude we calculate the density at these points and then plot the contour and the points. Note, that this is not the plot of the Lambert projected data. We do the same thing in the contour plot of a von Mises-Fisher kernel and in the contour plot of mixtures of von Mises-Fisher distributions.

```
kent.datacontour <- function(x) {
  ## u contains the data in latitude and longitude
  ## the first column is the latitude and the
  ## second column is the longitude
  ## if u are eucliean coordinates turn them into
  ## latitude and longitude
  dm <- dim(x)
  if ( dm[2] == 3 ) {
    u <- euclid.inv(x)
  } else if ( dm[2] == 2 ) {
    u <- x
    x <- euclid(x) ## Euclidean coordinates used by Kent (1982)
  }

  n <- dm[1]  ## sample size
  a <- kent.mle(x) ## MLE estimation of the Kent distribution
  G <- a$G ## G matrix, the mean direction and the major-minor axes
  k <- a$para[1] ## kappa, concentration parameter
  b <- a$para[2] ## beta, ovalness

  gam <- c(0, k, 0)
  lam <- c(0, -b, b)
  ckb <- fb.saddle(gam, lam)[3] ## logarithm of the normalising constant
  n <- 100
  x1 <- seq(min(u[, 1]) - 5, max(u[, 1]) + 5, length = n)  ## latitude
  x2 <- seq(min(u[, 2]) - 5, max(u[, 2]) + 5, length = n)  ## longitude
  mat <- matrix(nrow = n, ncol = n)

  for (i in 1:n) {
    for (j in 1:n) {
      y <- euclid( c(x1[i], x2[j]) )
      can <-  -ckb + k * y %*% G[, 1] + b * (y %*% G[, 2])^2 -
        b * (y %*% G[, 3])^2
```

```
      if ( abs(exp( can) ) < Inf ) {
        mat[i, j] <- exp(can)
      } else  mat[i, j] <- NA
    }
  }

  contour(x1, x2, mat, nlevels = 10, col = 2, xlab = "Latitude",
          ylab = "Longitude")
  points(u[, 1], u[, 2])

}
```

10.9.4 Contour plots of a von Mises-Fisher kernel density estimate on the sphere

We have seen how to construct the kernel density estimate of spherical data using a von Mises-Fisher kernel (Section 10.7.3). Given that we have a matrix with two columns, latitude and longitude, the goal is to plot these two columns and also see some kernel contour plots. The idea is simple, at first we transform the data into the euclidean coordinates (see *euclid* function about this) and then we choose the bandwidth parameter h either using maximum likelihood cross validation or by the rule of thumb suggested by García-Portugués (2013). Once we decide on the value of h, we need a grid of points at which we calculate the kernel density estimate using the sample. Finally, the ready built-in function in R, *contour* shows the contour plots. So, unlike the two previous functions, where the contour plots appear with no data, the following R code plots the data and shows their kernel contour plots.

```
vmf.kerncontour <- function(u, thumb = "none") {
  ## u contains the data in latitude and longitude
  ## the first column is the latitude and the
  ## second column is the longitude
  ## thumb is either 'none' (defualt), or 'rot' (Garcia-Portugues, 2013)
  n <- dim(u)[1]  ## sample size
  x <- euclid(u)

  if (thumb == "none") {
    h <- as.numeric( vmfkde.tune_2(x, low = 0.1, up = 1)[1] )
  }

  if (thumb == "rot") {
    k <- vmf(x)$kappa
    h <- ( (8 * sinh(k)^2) / (k * n * ( (1 + 4 * k^2) * sinh(2 * k) -
```

```
    2 * k * cosh(2 * k)) ) )^(1/6)
  }

  n <- 100  ## n1 and n2 specify the number of points taken at each axis
  x1 <- seq(min(u[, 1]) - 5, max(u[, 1]) + 5, length = n)  ## latitude
  x2 <- seq(min(u[, 2]) - 5, max(u[, 2]) + 5, length = n)  ## longitude
  cpk <- 1 / (  ( h^2)^0.5 *(2 * pi)^1.5 * besselI(1/h^2, 0.5) )
  mat <- matrix(nrow = n, ncol = n)

  for (i in 1:n) {
    for (j in 1:n) {
      y <- euclid( c(x1[i], x2[j]) )
      a <- as.vector( tcrossprod(x, y / h^2) )
      can <- mean( exp(a + log(cpk)) )
      if (abs(can) < Inf) {
        mat[i, j] <- can
     } else  mat[i, j] <- NA
    }
  }
  contour(x1, x2, mat, nlevels = 10, col = 2, xlab = "Latitude",
  ylab = "Longitude")
  points(u[, 1], u[, 2])
}
```

10.9.5 Contour plots of a von Mises-Fisher mixture model on the sphere

The next function produces contour plots of the von mises-Fisher mixture model for spherical data. The data must be in two columns, latitude and longitude respectively and we also need an object carrying the results of the mixture model.

```
mixvmf.contour <- function(u, mod) {
  ## u contains the data in latitude and longitude the first column is
  ## the latitude and the second column is the longitude
  ## mod is a mix.vmf object
  n <- dim(u)[1]  ## sample size
  n1 <- 100
  x1 <- seq(min(u[, 1]) - 5, max(u[, 1]) + 5, length = n1)  ## latitude
  x2 <- seq(min(u[, 2]) - 5, max(u[, 2]) + 5, length = n1)  ## longitude
  mat <- matrix(nrow = n1, ncol = n1)
  mu <- mod$param[, 1:3]  ## mean directions
```

```
  tmu <- t(mu)
  k <- mod$param[, 4]  ## concentration parameters
  p <- mod$param[, 5]  ## mixing probabilities
  g <- length(p)  ## how many clusters
  lika <- con <- numeric(g)
  for (l in 1:g) {
  con[l] <- 0.5 * log(k[l]) - 1.5 * log(2 * pi) - log(besselI(k[l], 0.5,
  expon.scaled = TRUE)) - k[l]
  }
  for (i in 1:n1) {
    for (j in 1:n2) {
      #x <- c( cos(x1[i]) * cos(x2[j]), cos(x1[i]) * sin(x2[j]), sin(x2[j]) )
      x <- euclid( c(x1[i], x2[j]) )
      lika <- con + k * ( x %*% tmu )
      can <- sum( p * exp(lika) )
      if (abs(can) < Inf) {
        mat[i, j] <- can
      } else  mat[i, j] <- NA
    }
  }
  contour(x1, x2, mat, nlevels = 8, col = 4, xlab = "Latitude",
  ylab = "Longitude")
  points(u[, 1], u[, 2])
}
```

10.10 Discriminant analysis for (hyper-)spherical (and circular) data

10.10.1 Discriminant analysis using the von Mises-Fisher distribution

There are not many papers on discriminant analysis. We will use the von Mises-Fisher distribution to perform this analysis (Morris and Laycock, 1974) similarly to the multivariate (or univariate) normal in $\mathbb{R}^p$. The idea is simple. For each group we estimate the mean vector and the concentration parameter and then the density of an observation is calculated for each group. The group for which the density has the highest value is the group to which the observation is allocated. We saw the form of the von Mises-Fisher density in (10.2). To avoid any computational overflow stemming from the Bessel function we will use the logarithm of the density and that will be the discriminant score

$$\delta_i = \frac{p}{2}\log\kappa_i + \kappa_i \mathbf{z}^T \boldsymbol{\mu}_i - \frac{1}{2}\log(2\pi) - \log\left[I_{p/2-1}(\kappa_i)\right],$$

for $i = 1, \ldots, g$, where g is the number of groups, κ_i and $\boldsymbol{\mu}_i$ are the concentration parameter and mean direction of the i-th group and $\mathbf{z}$ is an observation in $\mathbb{S}^{p-1}$. At first we have to see how well the method does. For this we have created the next R function to estimate the error via cross validation.

```
vmf.da <- function(x, ina, fraction = 0.2, R = 1000, seed = FALSE) {
  ## x is the data set
  ## ina is the group indicator variable
  ## fraction denotes the percentage of the sample to be used as the test sample
  ## R is the number of cross validations

  runtime <- proc.time()
  p <- dim(x)[2]  ## p is the dimensionality of the data
  per <- numeric(R)
  n <- dim(x)[1]  ## sample size
  ina <- as.numeric(ina)
  frac <- round(fraction * n)
  g <- max(ina)  ## how many groups are there
  mesi <- matrix(nrow = g, ncol = p)
  k <- numeric(g)
  ## if seed==TRUE then the results will always be the same
  if ( seed )  set.seed(1234567)

  for (i in 1:R) {
    mat <- matrix(nrow = frac, ncol = g)
    est <- numeric(frac)
    nu <- sample(1:n, frac)
    test <- x[nu, ]
    id <- ina[-nu]
    train <- x[-nu, ]
    for (j in 1:g) {
      da <- vmf(train[id == j, ])  ## estimates the parameters of the vMF
      mesi[j, ] <- da$mu  ## mean direction of each group
      k[j] <- da$kappa ## concentration of each group
    }
    for (j in 1:g) {
      mat[, j] <- (p/2 - 1) * log(k[j]) + k[j] * test %*% mesi[j, ] - 0.5 *
      p * log(2 * pi) - log( besselI(k[j], p/2 - 1, expon.scaled = TRUE) ) - k[j]
    }
    est <- Rfast::rowMaxs(mat)
```

```
    per[i] <- sum( est == ina[nu] ) / frac
  }

  percent <- mean(per)
  s1 <- sd(per)
  s2 <- sqrt(percent * (1 - percent)/R)
  conf1 <- c(percent - 1.96 * s1, percent + 1.96 * s1)  ## 1st way of a CI
  conf2 <- c(percent - 1.96 * s2, percent + 1.96 * s2)  ## 2nd way of a CI

  ## next we check if the confidence limits exceeds the allowed limits
  if (conf1[2] > 1) conf1[2] <- 1
  if (conf1[1] < 0) conf1[1] <- 0
  if (conf2[2] > 1) conf2[2] <- 1
  if (conf2[1] < 0) conf2[1] <- 0

  conf3 <- quantile(per, probs = c(0.025, 0.975))  ## 3rd way of a CI
  ci <- rbind(conf1, conf2, conf3)
  runtime <- proc.time() - runtime
  colnames(ci) <- c("2.5%", "97.5%")
  rownames(ci) <- c("standard", "binomial", "empirical")
  percent <- c(percent, s1, s2)
  names(percent) <- c('percent', 'sd1', 'sd2')
  list(percent = percent, ci = ci, runtime = runtime)
}
```

For prediction purposes the next R function is to be used.

```
vmfda.pred <- function(xnew, x, ina) {
  ## xnew is the new observation(s)
  ## x is the data set
  ## ina is the group indicator variable
  xnew <- as.matrix(xnew)
  if (ncol(xnew) == 1)  xnew <- t(xnew)
  xnew <- xnew / sqrt( Rfast::rowsums(xnew ^ 2) )

  p <- dim(x)[2]  ## dimensonality of the data
  ina <- as.numeric(ina)
  g <- max(ina)
  mesi <- matrix(nrow = g, ncol = p)
  k <- numeric(g)
```

```
  nu <- nrow(xnew)
  mat <- matrix(nrow = nu, ncol = g)
  est <- numeric(nu)

  for (j in 1:g) {
    da <- vmf(x[ina == j, ])  ## estimates the parameters of the vMF
    mesi[j, ] <- da$mu  ## mean direction
    k[j] <- da$k  ## concentration
  }

  for (j in 1:g) {
    mat[, j] <- (p/2 - 1) * log(k[j]) + k[j] * xnew %*% mesi[j, ] - 0.5 * p *
    log(2 * pi) - log( besselI(k[j], p/2 - 1, expon.scaled = TRUE) ) - k[j]
  }

  Rfast::rowMaxs(mat)
}
```

10.10.2 Discriminant analysis using the k-NN algorithm

We will use the angular distance we saw in compositional data (8.39) for the k-NN algorithm. The angular distance between $\mathbf{x}$ and $\mathbf{y} \in \mathbb{S}^{d-1}$ is defined as

$$D(\mathbf{x}, \mathbf{y}) = \cos^{-1}\left(\mathbf{x}^T\mathbf{y}\right).$$

The function below is used to allocate new observations to some known groups for a given number of nearest neighbours.

```
dirknn <- function(x, xnew, k = 5, ina, type = "S", mesos = TRUE) {
  ## x is the matrix containing the data
  ## xnew is the new data
  ## k is the number of neighbours to use
  ## ina indicates the groups, numerical variable
  ## type is either 'S' or 'NS'. Should the standard k-NN be use or not
  ## if mesos is TRUE, then the arithmetic mean distange of the k nearest
  ## points will be used.
  ## If not, then the harmonic mean will be used. Both of these apply for
  ## the non-standard algorithm, that is when type='NS'

  xnew <- matrix(xnew, ncol = dim(x)[2])  ## makes sure xnew is a matrix
```

```
  n <- dim(x)[1]  ## sample size
  ina <- as.numeric(ina) ## makes sure ina is numeric
  nc <- max(ina)  ## The number of groups
  nu <- nrow(xnew)
  apo <- tcrossprod(x, xnew)
  apo <- acos(apo)
  g <- numeric(nu)
  ta <- matrix(nrow = nu, ncol = nc)

  if (type == "NS") {
    ## Non Standard algorithm
    for (m in 1:nc) {
      dista <- apo[ina == m, ]
      dista <- Rfast::sort_mat(dista)
      if ( mesos ) {
        ta[, m] <- Rfast::colmeans( dista[1:k, ] )
      } else  ta[, m] <- k / Rfast::colsums( 1 / dista[1:k, ] )
    }
    g <- Rfast::rowMins(ta)

  } else {
    ## Standard algorithm
    for (l in 1:nu) {
      xa <- cbind(ina, apo[, l])
      qan <- xa[order(xa[, 2]), ]
      sa <- qan[1:k, 1]
      tab <- table(sa)
      g[l] <- as.integer( names(tab)[ which.max(tab) ] )
    }
  }

  g
}
```

In order to select, or choose, the number of nearest neighbours, we apply an m-fold cross validation and estimate the bias using the TT estimate of bias.

```
dirknn.tune <- function(z, M = 10, A = 5, ina, type = "S",
                        mesos = TRUE, mat = NULL) {
  ## x is the matrix containing the data
```

```
## M is the number of folds, set to 10 by default
## A is the maximum number of neighbours to use
## ina indicates the groups, numerical variable
## type is either 'S' or 'NS'. Should the standard k-NN be use or not
## if mesos is TRUE, then the arithmetic mean distange of the k nearest
## points will be used.
## If not, then the harmonic mean will be used. Both of these apply for
## the non-standard algorithm, that is when type='NS'

runtime <- proc.time()
n <- dim(z)[1]  ## sample size
ina <- as.numeric(ina)
if ( A >= min( table(ina) ) )   A <- min(table(ina)) - 3  ## The maximum
## number  of nearest neighbours to use
ina <- as.numeric(ina) ## makes sure ina is numeric
ng <- max(ina)  ## The number of groups

if ( is.null(mat) ) {
  nu <- sample(1:n, min( n, round(n / M) * M ) )
  ## It may be the case this new nu is not exactly the same
  ## as the one specified by the user
  ## to a matrix a warning message should appear
  suppressWarnings()
  mat <- matrix( nu, ncol = M )
} else  mat <- mat

M <- dim(mat)[2]
per <- matrix(nrow = M, ncol = A - 1)
rmat <- dim(mat)[1]

dis <- tcrossprod( z )
diag(dis) <- 1
dis[ dis > 1 ] <- 1
dis <- acos(dis)

## The k-NN algorith is calculated M times. For every repetition a
## fold is chosen and its observations are classified
for (vim in 1:M) {
```

```
id <- as.vector( ina[ mat[, vim] ] )  ## groups of test sample
ina2 <- as.vector( ina[ -mat[, vim] ] )   ## groups of training sample
aba <- as.vector( mat[, vim] )
aba <- aba[aba > 0]
apo <- dis[-aba, aba]
ta <- matrix(nrow = rmat, ncol = ng)

if (type == "NS") {
  ## Non Standard algorithm
  for ( j in 1:c(A - 1) ) {
    knn <- j + 1
    for (l in 1:ng) {
      dista <- apo[ina2 == l, ]
      dista <- Rfast::sort_mat(dista)
      if ( mesos ) {
        ta[, l] <- Rfast::colmeans( dista[1:knn, ] )
      } else {
        ta[, l] <- knn / Rfast::colsums( 1 / dista[1:knn, ] )
      }
    }
    g <- Rfast::rowMins(ta)
    per[vim, j] <- sum( g == id ) / rmat
  }

} else {
  ## Standard algorithm
  g <- numeric( rmat )
  for ( j in 1:c(A - 1) ) {
    knn <- j + 1
    for (k in 1:rmat) {
      xa <- cbind(ina2, apo[, k])
      qan <- xa[order(xa[, 2]), ]
      sa <- qan[1:knn, 1]
      tab <- table(sa)
      g[k] <- as.integer( names(tab)[ which.max(tab) ] )
    }
    per[vim, j] <- sum( g == id ) / rmat
  }
}
```

```
  }

  ela <- Rfast::colmeans(per)
  bias <- per[ , which.max(ela)] - Rfast::rowMaxs(per, value = TRUE)
  estb <- mean( bias )  ## TT estimate of bias
  runtime <- proc.time() - runtime
  names(ela) <- paste("k=", 2:A, sep = "")
  plot(2:A, ela, type = "b", xlab = "k nearest neighbours",
       pch = 9, ylab = "Estimated percentage of correct classification")
  percent <- c( max(ela) + estb)
  names(percent) <- c("Bias corrected estimated percentage")

  list( per = ela, percent = percent, runtime = runtime )
}
```

10.11 Model based clustering using mixtures of von Mises-Fisher distributions

This Section is about model based clustering and we will see how to use the EM algorithm for this purpose, simulate random data from a mixture model, choose the number of components using BIC and finally plot the contours of any model in the spherical case. We remind the reader that there is already a package called movMF written by Hornik and Grün (2014) for this purpose.

10.11.1 Fitting a mixture model

The mixture model comprising of D components is written as

$$h\left(\mathbf{x}|\boldsymbol{\Theta}\right) = \sum_{i=1}^{D} \pi_i f_i\left(\mathbf{x}|\boldsymbol{\theta}_i\right),$$

where $\boldsymbol{\Theta} = \left(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_D\right)$ and $\boldsymbol{\theta}_i = \left(\kappa_i, \boldsymbol{\mu}_i\right)$ and $\mathbf{x} \in \mathbb{S}^p$. The $\pi_i s$ are the mixing probabilities, need to be estimated also. I will describe the EM algorithm briefly, because I am not an expert, for this example.

The EM stands for *Expectation* and *Maximization*, the two steps of the algorithm. The key idea behind this algorithm is to perform likelihood maximization or parameter estimation when some information is missing. In our case, the missing information is the mixture probabilities, how many populations are there and which are their mixing probabilities from which the data were generated. The E step comes here, it calculates an expected value for this missing information. Then, with this knowledge, we can maximize the objective

function and estimate its parameters.

The t-th step of the algorithm is briefly described below

E step. Estimate the probability of each observation belonging to a component by

$$p_{ij}^{t} = \frac{\pi_i^{t-1} f_i\left(\mathbf{x}|\boldsymbol{\theta}_i\right)}{\sum_{m=1}^{D} \pi_m^{t-1} f_m\left(\mathbf{x}|\boldsymbol{\theta}_m\right)}$$

M step. Update the parameters

$$\hat{\pi}_i^t = \frac{1}{n}\sum_{j=1}^{n} p_{ij}^t, \;\; \hat{\boldsymbol{\mu}}_i^t = \frac{\sum_{j=1}^{n} p_{ij}^t \mathbf{x}_j}{\left\| \sum_{j=1}^{n} p_{ij}^t \mathbf{x}_j \right\|} \;\text{ and }\; A_p\left(\hat{\kappa}^t\right) = \frac{I_{p/2}\left(\hat{\kappa}\right)}{I_{p/2-1}\left(\hat{\kappa}\right)} = \frac{\| \sum_{j=1}^{n} p_{ij}^t \mathbf{x}_j \|}{\sum_{j=1}^{n} p_{ij}^t}.$$

In order to solve the equation and obtain κ, the reader is referred back to (10.3), the two-step truncated Newton-Raphson solution given by Sra (2012).

Step 3. Repeat the E and M steps until the log-likelihood does not increase any more.

We need some initial values to start with. For this reason, similarly to Hornik and Grün (2014) we will start with a K-mean clustering. Hornik and Grün (2014) suggests a spherical K-means algorithm but we did the classical K-means algorithm for Euclidean data. So, take the predicted memberships from the output of the algorithm and calculate the $\hat{\pi}_i^0 s$. Then proceed to the M step and calculate $\boldsymbol{\mu}_i^0$ and $\hat{\kappa}_i^0$.

```
mix.vmf <- function(x, g) {
  ## x contains the data
  ## g is the number of clusters
  p <- dim(x)[2]  ## dimensionality of the data
  n <- dim(x)[1]  ## sample size of the data
  lik <- NULL

  lika <- matrix(nrow = n, ncol = g)
  pij <- matrix(nrow = n, ncol = g)
  ka <- numeric(g)

  Apk <- function(p, k) {
    besselI(k, p/2, expon.scaled = TRUE) / besselI(k, p/2 - 1, expon.scaled = TRUE)
  }

  runtime <- proc.time()
  ## Step 1
```

```
l <- 1
mesa <- array(dim = c(g, p, 50))
crit <- numeric(50)
cl <- matrix(nrow = n, ncol = 50)

for (vim in 1:30) {
  ini <- kmeans(x, g)  ## initially a k-means for starting values
  mesa[, , vim] <- ini$centers
  cl[, vim] <- ini$cluster
  crit[vim] <- ini$betweenss/ini$totss
}

epi <- which.max(crit)
w <- as.vector( table(cl[, epi]) )

if ( min(w) <= 3 ) {
  mess <- paste( "Too many clusters to fit for this data. Try one less" )
  res <- list(mess = mess, loglik = NA)

} else {

  w <- as.vector( table(cl[, epi]) )/n  #'# initial weights
  m1 <- mesa[, , epi]
  Rk <- sqrt( Rfast::rowsums(m1^2) )  ## mean resultant lengths of the initial clusters
  mat <- m1/Rk  ## initial mean directions

  for (j in 1:g) {
    R <- Rk[j]
    k <- numeric(4)
    i <- 1
    k[i] <- R * (p - R^2)/(1 - R^2)
    i <- 2
    apk <- Apk(p, k[i - 1])
    k[i] <- k[i - 1] - ( apk - R)/( 1 - apk^2 - (p - 1)/k[i - 1] * apk )
    while (abs(k[i] - k[i - 1]) > 1e-07) {
      i <- i + 1
      apk <- Apk(p, k[i - 1])
      k[i] <- k[i - 1] - (apk - R)/( 1 - apk^2 - (p - 1)/k[i - 1] * apk )
    }
```

```
  ka[j] <- k[i] ## initial concentration parameters
  lika[, j] <- (p/2 - 1) * log(ka[j]) - 0.5 * p * log(2 * pi) -
    log(besselI(ka[j], p/2 - 1, expon.scaled = TRUE)) - ka[j] +
    ka[j] * (x %*% mat[j, ])
}

wlika <- w * exp(lika)
rswlika <- Rfast::rowsums(wlika)
lik[1] <- sum( log( rswlika ) )  ## initial log-likelihood

l <- 2
## Step 2
pij <- wlika / rswlika  ## weights at step 2
w <- Rfast::colmeans(pij) ## weights for step 2

for (j in 1:g) {
  m1 <- Rfast::colsums(pij[, j] * x)
  mat[j, ] <- m1 / sqrt( sum(m1^2) )  ## mean directions at step 2
  R <- sqrt( sum(m1^2) ) / sum( pij[, j] )  ## mean resultant lengths at step 2
  k <- numeric(4)
  i <- 1
  k[i] <- R * (p - R^2)/(1 - R^2)
  i <- 2
  apk <- Apk(p, k[i - 1])
  k[i] <- k[i - 1] - ( apk - R)/( 1 - apk^2 - (p - 1)/k[i - 1] * apk )

  while (abs(k[i] - k[i - 1]) > 1e-07) {
    i <- i + 1
    apk <- Apk(p, k[i - 1])
    k[i] <- k[i - 1] - (apk - R)/( 1 - apk^2 - (p - 1)/k[i - 1] * apk )
  }
  ka[j] <- k[i]
  lika[, j] <- (p/2 - 1) * log(ka[j]) - 0.5 * p * log(2 * pi) -
    log(besselI(ka[j], p/2 - 1, expon.scaled = TRUE) ) - ka[j] +
    ka[j] * (x %*% mat[j, ])
}

wexplika <- w * exp( lika)
lik[2] <- sum( log( Rfast::rowsums( wexplika ) ) )  ## log-likelihood at step 2
```

```
## Step 3 and beyond
while ( lik[l] - lik[l - 1] > 1e-05 ) {
  l <- l + 1

  pij <- wexplika / Rfast::rowsums( wexplika )  ## weights
  w <- Rfast::colmeans(pij)

  for (j in 1:g) {
    m1 <- Rfast::colsums(pij[, j] * x)
    mat[j, ] <- m1 / sqrt( sum(m1^2) )  ## mean directions at step l
    R <- sqrt( sum(m1^2) ) / sum(pij[, j])  ## mean resultant lengths at step l
    k <- numeric(4)
    i <- 1
    k[i] <- R * (p - R^2)/(1 - R^2)
    i <- 2
    apk <- Apk(p, k[i - 1])
    k[i] <- k[i - 1] - (apk - R)/( 1 - apk^2 - (p - 1)/k[i - 1] * apk )

    while (abs(k[i] - k[i - 1]) > 1e-07) {
      i <- i + 1
      apk <- Apk(p, k[i - 1])
      k[i] <- k[i - 1] - (apk - R)/( 1 - apk^2 - (p - 1)/k[i - 1] * apk )
    }

    ka[j] <- k[i]
    lika[, j] <- (p/2 - 1) * log(ka[j]) - 0.5 * p * log(2 * pi) -
      log(besselI(ka[j], p/2 - 1, expon.scaled = TRUE) ) - ka[j] +
      ka[j] * (x %*% mat[j, ])
  }

  wexplika <- w * exp( lika)
  lik[l] <- sum( log( Rfast::rowsums( wexplika ) ) )
}  ## log-likelihood at step l

ta <- Rfast::rowMaxs(pij)  ## estimated cluster of each observation
param <- cbind( mat, ka, table(ta)/n )
runtime <- proc.time() - runtime
colnames(param) <- c( paste("mu", 1:p, sep = ""), 'kappa', 'probs' )
```

```
    rownames(param) <- paste("Cluster", 1:g, sep = " ")

    res <- list(param = param, loglik = lik[l], pred = ta, runtime = runtime)

  }

  res

}
```

10.11.2 Choosing the number of components of the mixture model

A good method to choose how many components one wants is via the BIC.

```
bic.mixvmf <- function(x, A = 3) {
  ## x contains the data
  ## A is the maximum number of clusters, set to 3 by default

  runtime <- proc.time()
  n <- dim(x)[1]  ## sample size of the data
  p <- dim(x)[2]  ## dimensionality of the data
  bic <- 1:A
  mod <- vmf(x)
  bic[1] <-  - 2 * mod$loglik+ p * log(n)  ## BIC assuming one cluster

  for (vim in 2:A) {
    a <- mix.vmf(x, vim)  ## model based clustering for some possible clusters
    bic[vim] <-  -2 * a$loglik + ( (vim - 1) + vim * p ) * log(n)
  }  ## BIC for a range of different clusters

  runtime <- proc.time() - runtime

  names(bic) <- 1:A
  ina <- rep(1, A)
  ina[which.min(bic)] <- 2  ## chosen number of clusters will
  ## appear with red on the plot

  plot(1:A, bic, pch = 10, col = ina, xlab = "Number of components",
  ylab = "BIC values", type = "b")
  list(bic = bic, runtime = runtime)
```

```
}
```

10.12 Lambert's equal area projection

In order to visualize better spherical data (we are on $\mathbb{S}^2$) it's not sufficient to plot in a scatter diagram the latitude versus the longitude because of the spherical nature of the data. For this reason we should project the sphere on the tangent plane and then plot the projected points. This is the way maps are made, by using an azimuthial projection, to preserve distances. A good choice is the Lambert's (azimuthial) equal area projection. We will try to explain what it is, but if you did not understand then see Fisher et al. (1987) who explains graphically this one and some other projections. Figure 10.1 presented above shows the difference.

Suppose we have points on the sphere, denoted by θ (latitude) and ϕ (longitude). Following Kent (1982) we will project the points on the (half) sphere down to the tangent plane inside a spherical disk with radius 2

$$z1 = \rho \cos \phi, \;\; z2 = \rho \sin \phi, \tag{10.16}$$

where $\rho = 2 \sin \theta/2$. In our case, the radius is one, but if you multiply by 2 then the radius becomes 2. So this projection corresponds to the red line in Figure 10.1.

At first, let us say something. We must rotate the data so that their mean direction is the north pole (for convenience reasons) and then spread, open, expand the north hemisphere so that it becomes flat (or project the points on the tangent to the north pole plane). So starting from two sets of points (latitude and longitude) we move on to the sphere (Euclidean coordinates), then find their mean direction, rotate the data such that their mean direction is the north pole, go back to the latitude and longitude and then apply (10.16). For the next two functions we need the functions *euclid*, *rotation*, *vmf* and *euclid.inv*.

```
lambert <- function(y) {
  ## y contains the data in degrees, latitude and longitude
  u <- euclid(y)  ## transform the data into euclidean coordinates
  m <- Rfast::colmeans(u)
  m <- m / sqrt(sum( m^2) )  ## the mean direction
  b <- c(0, 0, 1)  ## the north pole
  H <- rotation(m, b)  ## the rotation matrix
  u1 <- tcrossprod(u, H)  ## rotating the data so that their mean
  ## direction is the north pole
  u2 <- euclid.inv(u1)  ## bring the data into degrees again
  u2 <- pi * u2 / 180  ## from degrees to radians
  theta <- u2[, 1]
```

```
  phi <- u2[, 2]
  rho <- 2 * sin(theta / 2)  ## radius of the disk is sqrt(2)
  z1 <- rho * cos(phi)  ## x coordinate
  z2 <- rho * sin(phi)  ## y coordinate
  cbind(z1, z2)  ## the Lambert equal area projected data on the disk
}
```

The inverse of the Lambert projection is given by the next R function. For this one we need to have the original mean direction towards which we will bring the back onto the sphere. We reverse the projection of the data onto the sphere and then rotate them from the north pole to the given mean direction. Then we transform them into latitude and longitude.

```
lambert.inv <- function(z, mu) {
  ## z contains the Lambert equal area projected data
  ## mu is the initial mean direction to which we will
  ## rotate the data after bringing them on the sphere

  z <- as.matrix(z)
  long <- ( atan(z[, 2]/z[, 1]) + pi * I(z[, 1] < 0) ) %% (2 * pi)
  lat <- 2 * asin( 0.5 * sqrt( Rfast::rowsums(z^2) ) )
  u <- cbind(lat, long)  ## the data on the sphere in radians
  u <- u * 180/pi  ## from radians to degrees
  y <- euclid(u)  ## the data in euclidean coordinates

  ## their mean direction is not exactly the north pole
  b <- c(0, 0, 1)  ## the north pole from which we will rotate the data
  mu <- mu / sqrt( sum(mu^2) )  ## make sure that mu is a unit vector
  H <- rotation(b, mu)  ## rotate the data so that their mean direction is mu
  u <- tcrossprod(y, H)
  euclid.inv(u)

}
```

Log of changes from version to version

After a massive public demand (one e-mail basically) I was suggested to add a log a log of the changes in different versions or any changes I make. I started from version 3.9 as I do not remember the previous changes.

99. 9/6/2022. Version 9.8. Update of the functions *kern.reg* and *kernreg.tune*. Bug fix of *james* function.

98. 1/6/2021. Version 9.7. I corrected a bug in the *choose.pc* and made it significantly faster. I corrected a mistake in the functions *kern.reg* and *kernreg.tune*.

97. 29/2/2020. Version 9.6. I made the functions *pcr* and *pcr.tune* faster.

96. 2/8/2019. Version 9.5. Minor changes in the code of the functions. Addition of the fast permutation p-value for the correlation. I also updated the code in some functions, taken from my R packages.

95. 26/3/2019. Version 9.4. Addition of a code to faster simulate from the Bingham and related distributions.

94. 25/2/2019. Version 9.3. Small and minor corrections. Change of my e-mail address.

93. 9/4/2018. Version 9.2. Time optimisation of a few functions. Added some more functions. Correction of a bug in *fisher.da*. The function *mle.lnorm* is now called *mvlnorm.mle*.

92. 20/12/2016. Version 9.1. I made some functions faster, namely the *rmvnorm*, *rmvt*, *rmvlaplace* and *diri.nr(x, type = 2)*. A typo in the function *ppca* is now corrected and this function has been renamed into *ppca.choose* because the function *ppca* is now added. In general, many improvements in terms of time have been carried out for many functions.

91. 14/10/2016. Version 9. I made many functions faster, even up to 10 time faster, and added an error stopping case in the function *mix.vmf*. If in the first step there are 3 or less observations in a cluster, the process will stop. Addition of a function which produces a permutation based p-value for the hypothesis testing of correlations between a vector and many variables. Correction of a mistake in the function *rot.matrix*.

90. 28/5/2016. Version 8.9. Addition of the median direction for spherical and hyperspherical data. Correction of a mistake in the function *circ.summary*. I think Mardia and Jupp (2000) had a mistake or a misunderstanding of the lower and upper quantiles of a distribution. I made the function *rda.tune*. Of course, it can still go faster, but this will happen later. I have to change some other cross-validation functions as well. The point is to avoid unnecessary calculations and the big thing is to avoid doing the same

calculations multiple times. During my PhD this function was so badly written that instead of minutes (now) it was taking days. I vectorised the function *mle.lnorm* (now is faster) and made many functions a bit faster. Time is measured in many functions now. Correction of a mistake in the p-value calculation in the function *sarabai*. I fixed a minor mistake in the MNV variation of the James test (function *james*). I added two functions for faster calculation of the covariance and the correlation matrices. I made the function *sq.correl* faster, doing less calculations. The matrix exponential is now a general function for any full rank square matrix. The function *sarabai* is now about 6 times faster due to a more efficient use of linear algebra. I also made some more functions a bit faster. Two more, faster functions for the spatial median regression have been added.

89. 15/2/2016. Version 8.8. Addition of a newly suggested method for outlier detection for high-dimensional data. If you use the *predict.compknn* or the *compknn.tune* functions with the angular, the Hellinger or the taxicab distance it is now much faster. The two *for* loops used in each distance are gone. I fixed some problems with the functions *alfa.reg*, *alfareg.tune* and their dependant functions as well. I added a few stuff in the Wishart distribution and the MLE for the Dirichlet-multinomial. The discriminant analysis for multinomial data now offers now the product of Poisson distributions and the Dirichlet-multinomial distribution. Addition of the k-NN algorithm for circular and (hyper-)spherical data. I fixed a bug in the functions *knn.reg*, *kern.reg* and *ridge.reg*. I also changed the functions *knnreg.tune*, *kernreg.tune*, *pcr.tune*, *glmpcr.tune* and *ridge.tune*. An m-fold cross validation is now implemented for all of them. I corrected a typo in the function *alfaknn.tune*. Addition of the function *comp.den* which is a rather generic function for estimating distribution parameters in compositional data. Addition of the function *fast.alfa* as a very fast alternative to the function *profile*.

88. 11/12/2015. Version 8.7. Addition of the α-regression and the ESOV regression for compositional data. The plot in the function multivreg is now optional. An argument is added (TRUE or FALSE). This affects the function *comp.reg*, where there no plot will appear by default. A slight change in the functions *mrda*, *pcr*, *pcr.tune*, *spml.reg* and *spher.reg*. *mvreg.cat* and some functions for regression with compositional data as responses offer bagging (bootstrap aggregation). I added the α-regression, when the response variable is compositional data. I changed the function for the calculation of the spatial median using an iterative algorithm which is much much faster than when using an optimiser such as *nlm*.

87. 18/10/2015. Version 8.6. Many changes have taken place and in particular debugging of most of the functions. Addition of the parallel computing option in the *knn.tune* function. The functions *knn.tune*, *ridge.tune* and *mkde.tune* allow now parallel computation. The *compknn.tune* is being used by *compknntune.par* which allows for parallel

computation. A slight presentation change in the *compknn.tune*. A function to calculate the matrix of partial correlations has been added. Correction of a typo in the *ridge.multivreg* function. Addition of an argument in the function *cov.gen*. Change of the functions *comp.compreg*, textitcompcompreg.tune and *comp.reg*. Robust regression is not supported now. I changed the output of the function *multivreg* and I think it is better now. This affected the function *comp.reg* which was also changed. Correction of a typo in the functions *kl.compreg* and *ols.compreg*. Going from *optim* to *nlm* I forgot to change everything. Change of some names, from *knn.tune* to *knnreg.tune*, from *pred.knn* to *knn.reg*, from *kern.tune* to *kernreg.tune* and from *ridge.multivreg* to *ridge.reg*. A change in the function *profile*, I removed some unnecessary parentheses. The function *vmkde.tune* includes now an option for parallel computation. I made the function *rda.pred* shorter by a couple of line and a lot faster. When you have many observations and many variables the time differences are clear. I removed a coupe of lines from the function *pred.multinomda*. The function *expm* had a silly mistake in the final line, instead of *expA* there was *expaA*. Change of the title from *Multivariate statistical functions in R* to *Multivariate data analysis in R, a collection of functions*. Some small changes to the *vmfda.pred* and *rda.pred* functions. The function *rda.tune* offers parallel computation now. I added confidence intervals of the coefficients in the function *multivreg*. A small change in the function *ridge.tune*. Some presentation and style changes in the function *ridge.reg*. A small change of the output style in the function *profile*. Minor changes in the functions *mkde.tune*, *mkde* and *multivt*. The function *kern.reg* had a bug which is now fixed and I changed its format. I believe I made it more flexible now. The function *kernreg.tune* has also changed a bit, one line was removed and another one was slightly changed. Minor changes in the functions *knnreg.tune* and *knn.reg*. Regularised discriminant analysis and the k-NN algorithm for compositional data using the α-transformation have been added. The α in the denominator of the α-distance (8.20) is now with an absolute value. I never wrote it, even in my PhD thesis, because I knew it had to positive. Correction of a typo in the function *pcr.tune*. Some presentation changes in the functions *compknn.tune* and *compknntune.par*. Addition of the principal components regression when the independent variables are compositional data by employing the α-transformation. In the ridge regression the predictor variables are now being standardised first and in the tuning of λ, the default intervals are smaller now but with a smaller step as well. A small change in the output of the functions *ridge.reg*, *hotel2T2* and *james* has now taken place. Addition of the function *ridge.plot*. Many functions have been made either a bit faster or super faster than before. I removed some redundant lines, vectorised them, or used faster functions. A small change in the output of the functions *boot.correl*, *lamconf*, *profile* and *rvonmises*. I made the function *mix.vmf* a bit shorter. In addition, the output style is changed. The function *bic.mixvmf* is now shorter. The *boot.correl* had a stupid mistake

with the p-value, but now it is corrected. Addition of the function *glm.pcr* which does principal components regression for binary and count data. Addition of the functions *pcr.plot* and *glmpcr.plot* for visualizing the principal components regression. Change of the function *ridge.reg*. Bootstrap standard errors are now being provided only. Some subsubsections are now included in the subsections. Change of the output style in the functions *sym.test*, *fishkent*, *fb.saddle* (and of the functions that call this function), *vmf.da*, *vmfda.pred rayleigh*, *vmf.kde*, *spher.cor*, *circlin.cor*, *hotel2T2*. Addition of the function *corr.mat*. Panagiotis Tzirakis helped me vectorize some of the contour plots functions. A change in the function *james*. The estimate of the common mean in the bootstrap case is the most appropriate one now. I changed the function *alfa*. The output is now a list, so all the functions involving this had to change slightly. The benefit of this is that the function *profile* is much faster now, such as twice as fast. In addition, the function *adist* changed its name into *alfadist* since there is an already built-in function in R called *adist*. I added a vectorised version of the function *boot.correl*and a vectorised version of *bp.gof*. I vectorised the *kern.reg* for only when the polynomial degree equals zero. Nevertheless it is faster now. I vectorised the *knn.reg* as well. Addition of the permutation based hypothesis testing for zero correlation (*permcor*). Correction of the function *cov.gen* (very bad mistake) and change its name into *rwishart*. Addition of the functions *james.hotel* and *maovjames.hotel* showing numerically the relationship between the James tests and Hotelling's one sample T^2 test. I robustified the initial values of the *spatmed.reg* function (spatial median regression). I added a second and faster function (without parallel) to tune the bandwidth parameter in the multivariate kernel density estimation. I will keep both of them, but change them at some point to make them even faster. Both use 2 *for* loops which slows down things. A slight change in the *correl* function. I changed the function *rinvdir* to *rdir*. There is an option to specify whether Dirichlet or inverted Dirichlet values will be generated. The function *bckfold.da* is now shorter. Finally I changed Fisher's linear discriminant analysis. In addition, a function to estimate is performance is now added. Addition of the function for multivariate linear regression allowing for mixed predictors. Correction of some typos in the function *spml.reg*. Addition of the function $vmfkde.tune_2$ as a faster alternative to *vmfkde.tune*. A change in the *vmkde.tune* took place. A small improvement of the functions *rvmf*, *Arotation*, *vmf.kerncontour* and *mixvmf.contour*. A re-ordering of some subsections took place in Section 10. Addition of the Newton-Raphson algorithm for estimating the parameters of a Dirichlet distribution. A small change in the function *corr.mat*. A small change in the *circlin.cor*, many variables can be considered now.

86. 14/7/2015. Version 8.5. Addition of the classification for compositional data using the α-transformation Section. At the moment, only the power transformation for discriminant analysis using the the k-NN algorithm has been added. Removal of the ro-

bust multivariate regression section (*rob.multivreg* function). I will do this again later. Change of the order of some subsections in the compositional data Section. Correction of a silly mistake in the *diri.reg* function. I was returning the logarithm of the ϕ parameter. I fixed it now. The output of the *diri.reg* and *diri.reg2* functions is now corrected. Addition of some tips for faster computations in the beginning. Addition of the E-M algorithm explanation in the circular regression Section.

85. 25/6/2015. Version 8.4. Addition of the α-transformation, and its associated distance and mean vector, for compositional data. The profile log-likelihood of α as a way of choosing its values was added as well. Some structural and presentation related changes in the Section of compositional data have taken place. I changed a bit the *dirireg* function. In order for the ϕ parameter to be strictly positive I added an exponential term inside instead of the logical function. I took the *sq.correl* function regression Section and put it in the correlation Section. Addition of the partial correlation coefficient. Addition of two brackets in a line of multinom.da function. In the description of the algorithm for simulating data from a von Mises-Fisher distribution I had a not so clear description of a *while* and *for* functions. Giorgos Borboudakis from the Foundation of Research and Technology (FORTH) and member of the MXM group pointed it out to me and I think now I made it a bit more clear now.

84. 27/5/2015. Version 8.3. Model based clustering for compositional data is now added, fitting the model, contour plots and random values generation. Correction of a silly mistake in the *helm* function. I have a different version and I did not spot that there was a mistake here. If you find any more mistakes, please e-mail them to me. Correction of a silly mistake in the *rmixvmf* function.

83. 18/5/2015. Version 8.2. Addition of the discrimination for multinomial data. Keep formatting of more functions. Correction of some functions, some lines were outside the paper size. Update of the *rfb* function and correction of a typo in the text following. It was *fb.sim*, now it is *rfb*. I changed the *diri.reg*, *diri.reg2*, *ols.compreg*, *kl.compreg* functions. I changed the *optim* function to *nlm*. This optimiser is faster. I changed the name of the variables in the *spher.reg* function. From u and v we have now x and y respectively. Change of the *vmf* function. At first, the *tol* is a bit smaller, from 10^{-8} it is now set to 10^{-7}. In addition, I have just became aware of the overflow of the *besselI* function when the concentration parameter (κ) is more than $100,000$. So, if the approximation is more than $100,000$, the function will return that approximation.

82. 3/5/2015. Version 8.1. Addition of the Moore-Penrose pseudo-inverse matrix. I updated the two Sections where I mention the contour plots of the skew normal distribution, for Euclidean and compositional data. Update of the *diri.est* function. I changed a bit the type "prec". I corrected a silly mistake in the *mkde* function.

81. 24/4/2015. Version 8.0. Addition of the contour plots for the Kent distribution fitted to some data. Correction of a typo in the *mixvmf.contour* function. This ", was forgotten in the penultimate line. I changed the *fb.sim* function. Now it is a bit shorter and correct. The rotation matrix to get the preferred mean direction was missing and its new name is *rfb*. Update of the *kent.mle* function. The estimation is more accurate now. Change of the order in the (hyper)-spherical data Section. I made the *Arotation* shorter by one line.

80. 23/4/2015. Version 7.9. Correction of a typo in Equation (10.4). The sum goes up to n, not p. Addition of the model based clustering for mixtures of von Mises-Fisher distributions. Functions for choosing the number of components (*bic.mixvmf*), random values simulation (*rmixvmf*) and contour plots (*mixvmf.contour*) are added. Correction of the previous correction, the square root in the metric for covariance matrices was not necessary, since the square distance was mentioned in the formula of the metric. Following Marco Maier's advice I changed the *rep(0,R)* to *numeric(R)*, the *apply(x,2,mean)* to *colMeans(x)* and the *t(x)%*%x* to *crossprod(x)*. In addition, the quadratic form of the normal and other expressions elsewhere have been now replaced with the squared Mahalanobis distance. I formatted some more functions and updated some more as well. Update of the *comp.compreg* and *compcompreg.tune* functions.

79. 21/4/2015. Version 7.8. Correction of a typo in the metric for covariance matrices Section. The square root was missing from the formula. The function had it.

78. 16/4/2015. Version 7.8. Formatting of some more functions. Addition of an influential diagnostic in the simple Dirichlet regression. A small change in the *kl.compreg* function. I increased the tolerance value from 1^{-5} to 1^{-4}. Correction of a typo in the *comp.reg* and *spatmed.reg* functions. A small change in the output of the *diri.reg* and *diri.reg2* functions. Correction of a silly mistake in the plot of the *pcr.tune* function. Correction of a mistake in the *lnorm.contours*, *invdir.contours* and *kern.contours* functions. From the transition to the new version of the functions, these mistakes occurred. Correction of a sily mistake in the *rob.multivreg* function. Addition of two functions to perform regression analysis for compositional data when the independent variables are also compositional data. The idea is to use principal component regression.

77. 9/4/2015. Version 7.7. Formatting of some more functions. Update of the *vmf.kerncontour* function, an unnecessary line was removed.

76. 4/4/2015. Version 7.7. Format of some functions. The target is that all will change in time. This is due to Marco Maier who suggested me to do this. He introduced the R package *formatR* to me, which formats my functions one by one, so the whole process will take some time. Addition of a metric for covariance matrices. Addition

of the Hotelling's T^2 test for repeated measures ANOVA. Change of the *diri.contour*, *norm.contour*, *t.contour* and *skewnorm.contour* functions. Now, the option to make the data appear exists. Change of the output style in the *f*, *hotel2T2* and *james* functions. Correction of a typo in the *het.circaov* and *circ.cor2* functions. Correction of a typo in the *comp.spatmed* function. It worked for me before, that is why I had not spotted it before.

75. 25/3/2015. Version 7.6. Addition of the inverted Dirichlet distribution, MLE of its parameters (*invdir.est*), random values generation (*rinvdir*) and contours plots in the case of two dimensional data (*invdir.contours*). Change of the presentation style of the Section about *Distributions*.

74. 19/3/2015. Version 7.5. Addition of the bivariate Poisson distribution Section. MLE of the parameters and a goodness of fit test are included. Contour plots of this distribution are also provided. MLE of the parameters of a multivariate log-normal and contour plots of the bivariate distribution are also provided. Correction of a minor typo in the multivariate Laplace distribution. The R function is not changed. A correction of a typo in the *fb.saddle* function. The smallest eigenvalue in Fallaize and Kypraios (2014) can be equal to zero, but in Kume and Wood (2005) it has to be greater than zero. Since this function is based upon the second paper I had to correct it. Some rewording took place there also.

73. 12/3/2015. Version 7.4. Addition of the von Mises-Fisher kernel contour plots for spherical data. Change of the output style in the *kfold.da*, *bckfold.da*, *lamconf*, *ternary* and *vmf.da* functions.

72. 1/3/2015. Version 7.3. Addition of the kernel density estimation for circular, spherical and hyper-spherical data. Some word changes in the projected bivariate normal regression of angular data. Change of the *spher.reg* function. If the 3×3 matrix is not a rotation matrix, i.e. its determinant is not $+1$, a modification is made to make it 1. Correction of a typo in the *knn.tune*, *kern.tune* and *ridge.tune* functions. Addition of two lines in the APL-trimmed mean Section.

71. 23/2/2015. Version 7.2. Addition of the spatial sign covariance matrix and some rewording in the spatial median Section. Change of the names *rand.mvnorm* and *rand.mvt* to *rmvnorm* and *rmvt* respectively. The *cov.gen* was also slightly changed, because the *rand.mvnorm* was used. Correction of the *kern.reg* function in the Laplacian kernel. Addition of a multivariate Laplace distribution. Random values generation and moments estimation of its parameters. A new Section, called *Distributions*, was created.

70. 20/2/2015. Version 7.1. Correction of a typo in the functions *norm.contour*, *t.contour* and *skewnorm.contour*. Instead of *type* the argument in the function was *iso*. Update

of the *mkde* function. Two rules of thumb are now added. Update of the *kern.contour* and *apl.mean* functions as well. Correction of a typo in the *best.aplmean* function and some small changes in the wording of the Section about kernel regression. Addition of the contour plots of a kernel density estimate for compositional data when there are three components (function *comp.kerncontour*). Update of the *spatmed.reg* function to include standard errors of the beta parameters. Correction of the *kern.reg* function. The Laplacian kernel is now correct. Some rewording in that Section took place as well.

69. 17/2/2015. Version 7.0. Addition of the multivariate kernel density estimation and its contour plot for a 2-dimensional dataset. The *robust statistical analyses* subsection became a Section termed *Robust statistics*. The spatial and spatial median regression are now included in that Section. The APL-trimmed mean is added to that Section as well.

68. 16/2/2015. Version 6.9. Change of the names *dirireg2* and *diri.reg2* into *dirireg* and *diri.reg* respectively. Increase of the maximum iterations in the functions where optimization is needed. Addition of the Dirichlet regression where the precision parameter ϕ is linked with the same covariates as the compositional data. Correction of a typo inside the *spat.med* function.

67. 7/2/2015. Version 6.8. Addition of the multinomial logit regression for compositional data.

66. 5/2/2015. Version 6.7. Change of a typo in the *cov.gen* function and in general update of the function and of the wording in that Section.

65. 2/2/2015. Version 6.7. Addition of another test for testing the equality of concentration parameters of 2 or more samples in the circular and spherical data cases only. Correction of a typographical mistake in *het.circaov* function. Expansion of the *comp.reg* function to allow for more types of regression. Addition of the *rvonmises* function for simulating random values from the the von Mises distribution using the *rvmf* function.

64. 29/1/2015. Version 6.6. The log-likelihood ratio test for the hypothesis of the concentration parameters is deleted. The appropriate ones as described in Mardia and Jupp (2000) will be added in the next versions.

63. 26/1/2015. Version 6.5. Addition of the *meandir.test* function for testing hypothesis about the mean direction of a single sample.

62. 25/1/205. Version 6.5. Addition of the analysis of variance Section for two or more circular samples. Hypothesis testing for the equality of the concentration parameters are included as well. ANOVA for hyper-spherical data is also added but no hypothesis

testing for the equality of concentration parameters. The mean resultant length is given as an output in the *vmf* function. The option to plot circular data is now added in the *circ.summary* function. Some bits about the von Mises distributions are also added. The density of the von Mises is removed from the circular regression. A modified test statistic of the Rayleigh test of uniformity is also added. A presentational change has taken place.

61. 20/1/2014. Version 6.4. Inclusion of the harmonic mean in the k-NN regression as an option and correction of some typographical errors.

60. 25/12/2014. Version 6.4. The *sq.correl* function is now added. This gives a multivariate analogue of the coefficient of determination in the univariate regression. A typographical mistake in the multivariate regression is now corrected, p is the number of independent variables.

59. 5/12/2014. Version 6.3. The *multivt2* function for the estimation of the parameters of the multivariate t distribution is now added.

58. 3/12/2014. Version 6.2. The *multivt* function for the estimation of the parameters of the multivariate t distribution is now updated.

57. 26/11/2014. Version 6.2. A high dimensional two sample mean vector hypothesis testing procedure is now added, function *sarabai*. In addition, the *cov.gen* is a bit changed, corrected I would say.

56. 7/11/2014. Version 6.1. Estimation of the Dirichlet parameters takes place in one function now, called *diri.est*. I combined the functions *diri.mle*, *diri.phi* and *diri.ent* into one function. The uses has to specify which estimating procedure he wants.

55. 1/11/2014. Version 6.1 The multivariate standardization functions became one function, now called *multiv.stand*. The first principal component can now be added in the ternary diagram (function *ternary*) should the user wishes to see it. The Kuulback-Leibler divergence and the Bhattacharyya distance, between two Dirichlet distributions became one function now.

54. 31/10/2014. Version 6.1. Addition of the James test for testing the equality of more than 2 mean vectors without assuming equality of the covariance matrices (MANOVA without homoscedasticity). Minor changes in the functions *multivreg*, *rob.multivreg* and *comp.reg*. The argument for the betas in the list became *beta* instead of *Beta*.

53. 24/10/2014. Version 6.0. Multivariate ridge regression has been added. A way for generating covariance matrices was also added and the two functions in the Dirichlet regression were updated. Some minor typos were corrected.

52. 13/10/2014. Version 5.9. Addition of the spatial median and of the spatial median regression. Addition of the spatial median for compositional data as well.

51. 8/9/2014. Version 5.8. After a break we return with corrections in the functions *lambert* and *lambert.inv*. The mistake was not very serious, in the sense that the plot will not change much, the relevant distances will change only. But even so, it was not the correct transformation.

50. 28/7/2014. Version 5.8. Changes in the *euclid* and *euclid.inv* functions. The transformations inside the functions was not in accordance with what is described on the text. Some typos in the spherical-spherical regression description are now corrected.

49. 25/7/2014. Version 5.8. Typographical changes in the circular summary and in the projected bivariate normal sections. The codes are OK, but the descriptions had typos.

48. 2/7/2014. Version 5.8. A structural change and a correction in the *diri.reg* function and name change only of *multivnorm* to *rand.mvnorm*. Increase of the the highest number of degrees of freedom parameter in the *multivt* function and correction of a silly typographical mistake in the *rand.mvnorm* function. Addition of the *rand.mvt* for simulating random values from a multivariate t distribution. Also a small change in the order of some Sections. For some reason the *rand.mvnorm* would put the data in a matrix with 4 columns. So the result would always be a 4 dimensional normal. I corrected it now.

47. 29/6/2014. Version 5.7. A change in the *rda.pred* function. I made it faster by rearranging some lines internally. The function is the same. I also added the scores to appear as outputs.

46. 26/6/2014. Version 5.7. Some morphological changes and addition of the Dirichlet regression for compositional data. Addition of the forward search algorithm and the contour plots of the von Mises-Fisher and Kent distributions. Georgios Pappas' help with the contours made them possible to appear in this document.

45. 25/6/2014. Version 5.6. Addition of the model selection process in discriminant analysis.

44. 23/6/2014. Version 5.5. A slight change in the *ternary* function, addition of a graphical option. Changes in the Dirichlet estimation, I made them proper functions now. Change in the *multivreg* function. There was a problem if there was one independent variable with no name. I fixed a problem with the *rob.multivreg* function also. A minor mistake fixed in the functions *vmf.da* and *vmfda.pred* which did not affect the outcome. A constant term was wrong. The *spher.reg* function has become a bit broader now. Compositional regression is now added.

43. 16/6/2014. version 5.4. Fixation of a silly mistake in the *rbingham* function. The mistake was in the second line of the code.

42. 13/6/2014. Version 5.4. Addition of the variance of the concentration parameter κ in the *vmf* function.

41. 13/6/2014. Version 5.4. I fixed a mistake in the *circ.summary* function.

40. 13/6/2014. Version 5.4. I fixed some mistakes in the functions *circ.cor1*, *circ.cor2*, *circlin.cor*, *spher.cor*. The problem was that I was not drawing bootstrap re-samples under the null hypothesis. So I removed the bootstrap. the same was true for the *rayleigh* function. But in this function, I can generate samples under the null hypothesis. For this purpose, parametric bootstrap is now implemented. In addition, the function *circ.summary* changed and follows the directions of Mardia and Jupp (2000). A confidence interval for the mean angle is also included now.

39. 11/6/2014. Version 5.4. Theo Kypraios spotted a mistake in the *rbingham* function which has now been corrected.

38. 5/6/2014. Version 5.4. Addition of the test of Fisher versus Kent distribution on the sphere. Some presentation changes occurred in the MLE of the von Mises-Fisher distribution section.

37. 4/6/2014: Version 5.3. Addition of the Rayleigh test of uniformity. Slight changes in the *kent.mle* function regarding the presentation of the results.

36. 12/5/2014: Version 5.2. Some words added about estimating the concentration parameter in the von Mises-Fisher distribution.

35. 9/5/2014: Version 5.2. Editing of the Section about the simulation from a Bingham distribution. More information is added to make it clearer and a new function is used to simulate from a Bingham with any symmetric matrix parameter. A reordering of some sections took place and also the addition of a function to simulate from a Fisher-Bingham distribution and the Kent distribution on the sphere.

34. 8/5/2014: Version 5.1. Editing of the explanation of the function *FB.saddle*. I believe I made some points more clear.

33. 7/5/2014: Version 5.1. Correction of a space mistake in the *vmfda.pred* function. A line was not visible in the .pdf file. Correction of am mistake in the *vmf* function. The log-likelihood was wrong.

32. 3/5/2014: Version 5.1 Addition of the parameter estimation in the Kent distribution plus corrections of some typographical mistakes.

31. 10/4/2014: Version 5.0. Addition of the calculation of the log-likelihood value in the von Mises-Fisher distribution and correction of typographical errors.

30. 2/4/2014: Version 5.0. Addition of the (hyper)spherical-(hyper)spherical correlation and of the discriminant analysis for directional data using the von Mises-Fisher distribution. Whenever the *set.seed* option appeared we made some modifications also. That is, in the functions *knn.tune*, *kern.tune*, *pcr.tune* and *rda.tune*. addition of the seed option in the functions *kfold.da* and *bckfold.da*. The function *fb.saddle* is slightly changed. Now the logarithm of the Fisher-Bingham normalizing constant is calculated. This change happened to avoid computational overflow when the constant takes high values.

29. 31/3/2014: Version 4.9 Some minor changes in the functions *knn.tune* and *kern.tune*.

28. 29/3/2014: Version 4.9. Addition of the Lambert's equal are projection of the sphere onto a tangent plane. Change in the regularised discriminant analysis function. Cross validation for tuning of its parameters is now available.

27. 26/3/2014: Version 4.8. Fix of a silly mistake in the functions *knn.tune* and *pred.knn*.

26. 24/3/2014: Version 4.8. A minor correction in the function *multivreg*. A minor also change related to its presentation words. Addition of the function *rob.multivreg* which performs robust multivariate regression. Some presentation changes throughout the document also.

25. 23/3/2014: Version 4.7. Minor change in the k-NN regression. Now it accepts either Euclidean or Manhattan distance. Morphological change in the function *correl* and change of some words in the relevant section.

24. 21/3/2014: Version 4.7. Fix of a stupid mistake in the function *vmf*. The mean direction was wrongly calculated. Interchange between the sum and the square root.

23. 21/3/2014: Version 4.7. Removal of the function for Fisher type regression for angular response variables.

22. 20/3:2014: Version 4.7. Addition of the option to set seed in the functions *knn.tune*, *kern.tune* and *pcr.tune* (previously known as *pcr.fold*). This allows to compare the MSPE between these three different methods.

21. 20/3/2014: Version 4.7. Change in the functions *kfold.da* and *bckfold.da*. Correction of the confidence limits if they happen to go outside 0 or 1. In the *bckfold.da* I made sure that the same test samples are always used for the values of the power parameter λ. In this way the estimated percentage of correct classification is comparable in a fair way. Change of the title also. A similar change took place for the function *knn.tune*, so

that the MSPE for every value of the bandwidth parameter h is based on the same test samples. This change was also made in the function *pcr.fold* as well. Actually in the *pcr.fold* this was already happening but now the user can obtain the test samples used. The k-NN and kernel regressions accept univariate dependent variables now.

20. 18/3/2014: Version 4.6. Correction of a foolish mistake in the functions textiteuclid and *euclid.inv*. It did not handle correctly vectors and data which were not in matrix class.

19. 17/3/2014: Version 4.6. Fix of a problem with negative eigenvalues in the Fisher-Bingham normalizing constant.

18. 13/3/2014: Version 4.6. Addition of a second type correlation coefficient for pairs of angular variables. The new function is *circ.cor2*. The old function is now called circ.cor1 and a couple of typograhical mistakes inside it are now corrected. A change in the functions *vm.reg* and *spml.reg*. The calculation of the pseudo-R^2 changed. A change in the function *circ.summary* also. Minor typographical changes and removal of a few lines in the function*den.contours* which do not affect the function at all.

17. 12/3/2014: Version 4.5. Fixation of a possible problem with the column names in the multivariate regression (function *multivreg*). Small changes in the function itself as well.

16. 12/3/2014: Version 4.5. Fixation of a typographical error in the description of the algorithm for simulating random values from a von Mises-Fisher distribution and changing the functions *euclid* and *euclid.inv* to include the case of vectors, not only matrices.

15. 10/3/2014: Version 4.5. Addition of the circular-linear correlation coefficient. Addition of the bootstrap calculation of the p-value in the circular correlation. Fixation of a typographical error in the function *circ.summary*.

14. 8/3/2014: Version 4.4 Addition of the Box-Cox transformation for discriminant analysis. Expansion of the multivariate regression function *multivreg*. Some morphological changes also.

13. 7/3/2014: Version 4.3. Addition of the L_1 metric kernel in the kernel regression and change of the names of the two kernel regression functions. Addition of some words as well.

12. 6/3/2014: Version 4.2. Addition of one line for the column names in the functions *euclid* and *euclid.inv*. Morphological changes in the Section of discrimination and minor changes in the function *kfold.da*. Removal of the command *library(MASS)* from *multivt* and *den.contours*.

11. 4/3/2014: Version 4.2. Addition of a function to generate from a multivariate normal distribution. A change in the Nadaraya-Watson case of the kernel regression function. A change in the variance of the coefficients in the principal component regression function. Addition of some words in the standardization section and in the hypothesis testing for a zero correlation coefficient.

10. 1/3/2014: Version 4.1. Fixation of an error in the function *poly.tune*.

9. 27/2/2014: Version 4.1. Addition of a couple of things in the Fisher-Bingham normalizing constant section.

8. 19/2/2014: Version 4.1. Addition of the calculation of the Fisher-Bingham normalizing constant by connecting R to Matlab. Kwang-Rae Kim helped a lot with this one. Also a few changes in the introduction of the section about directional data.

7. 17/2/2014: Version 4.0. Correction in the *poly.reg* function (kernel regression). Some changes also in the introduction.

6. 16/2/2014: Version 4.0. Correction in the function pcr.fold (Cross validation for principal component regression). Instead of BIC I use now MSPE and a correction on the centering of the dependent variable.

5. 14/2/2014: Version 4.0. Updated version with some typos corrected.

4. 14/2/2014: Version 4.0. Word changes in the Fisher-Bingham normalizing constant and addition of one line in the function (*lam=sort(lam)*) and inclusion of this log of changes.

3. 13/2/2014: Version 4.0. Change of the *poly.tune* function. The cross-validation for the choice of the common bandwidth h is implemented by diving the sample to test and training sets many times. Improved cross validation. A change in the function *poly.reg* also.

2. 12/2/2014: Version 4.0. Addition of the *Fisher-Bingham normalizing constant*.

1. 11/2/2014: Version 3.9. Change of the Bingham random value simulation function with the function given by Christopher Fallaize and Theo Kypraios.

References

Abramowitz, M. and Stegun, I. (1970). *Handbook of mathematical functions*. New York: Dover Publishing Inc.

Agostinelli, C. and Lund, U. (2011). *R package `circular`: Circular Statistics*. R package version 0.4-3.

Agresti, A. (2002). *Categorical data analysis, 2nd edition*. New Jersey: John Wiley & Sons.

Aguilera, A. M., Escabias, M., and Valderrama, M. J. (2006). Using principal components for estimating logistic regression with high-dimensional multicollinear data. *Computational Statistics & Data Analysis*, 50(8):1905–1924.

Aitchison, J. (1983). Principal component analysis of compositional data. *Biometrika*, 70(1):57–65.

Aitchison, J. (1989). Measures of location of compositional data sets. *Mathematical Geology*, 21(7):787–790.

Aitchison, J. (2003). *The Statistical Analysis of Compositional Data*. New Jersey: (Reprinted by) The Blackburn Press.

Aliferis, C. F., Statnikov, A., Tsamardinos, I., Mani, S., and Koutsoukos, X. D. (2010). Local causal and markov blanket induction for causal discovery and feature selection for classification part i: Algorithms and empirical evaluation. *The Journal of Machine Learning Research*, 11:171–234.

Amaral, G. J. A., Dryden, I. L., and Wood, A. T. A. (2007). Pivotal bootstrap methods for k-sample problems in directional statistics and shape analysis. *Journal of the American Statistical Association*, 102(478):695–707.

Anderson, T. W. (2003). *An introduction to multivariate statistical analysis (3rd edition)*. New Jersey: John Wiley & Sons.

Atkinson, A. C., Riani, M., and Cerioli, A. (2004). *Exploring multivariate data with the forward search*. Springer.

Azzalini, A. (2005). The skew-normal distribution and related multivariate families*. *Scandinavian Journal of Statistics*, 32(2):159–188.

Azzalini, A. (2011). *R package `sn`: The skew-normal and skew-t distributions (version 0.4-17)*. Università di Padova, Italia.

Azzalini, A. and Valle, A. D. (1996). The multivariate skew-normal distribution. *Biometrika*, 83(4):715–726.

Bai, Z. D. and Saranadasa, H. (1996). Effect of high dimension: by an example of a two sample problem. *Statistica Sinica*, 6(2):311–329.

Baxter, M. J. (2001). Statistical modelling of artefact compositional data. *Archaeometry*, 43(1):131–147.

Baxter, M. J., Beardah, C. C., Cool, H. E. M., and Jackson, C. M. (2005). Compositional data analysis of some alkaline glasses. *Mathematical Geology*, 37(2):183–196.

Bengtsson, H. (2014). *R.matlab: Read and write of MAT files together with R-to-MATLAB connectivity*. R package version 2.2.3.

Boyles, R. A. (1997). Using the chi-square statistic to monitor compositional process data. *Journal of Applied Statistics*, 24(5):589–602.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

Breusch, T. S., Robertson, J. C., and Welsh, A. H. (1997). The emperor's new clothes: a critique of the multivariate t regression model. *Statistica Neerlandica*, 51(3):269–286.

Brown, P. J. and Zidek, J. V. (1980). Adaptive multivariate ridge regression. *The Annals of Statistics*, 8(1):64–74.

Browne, R. P., ElSherbiny, A., and McNicholas, P. D. (2015). *mixture: Mixture Models for Clustering and Classification*. R package version 1.4.

Butler, R. W. (2007). *Saddlepoint approximations with applications*. Cambridge University Press.

Cabrera, J. and Watson, G. (1990). On a spherical median related distribution. *Communications in Statistics-Theory and Methods*, 19(6):1973–1986.

Casella, G. and Berger, R. L. (2002). *Statistical inference*. Duxbury Pacific Grove, CA.

Chakraborty, B. (2003). On multivariate quantile regression. *Journal of statistical planning and inference*, 110(1):109–132.

Chang, T. (1986). Spherical regression. *The Annals of Statistics*, 14(3):907–924.

Chen, S. X., Qin, Y.-L., et al. (2010). A two-sample test for high-dimensional data with applications to gene-set testing. *The Annals of Statistics*, 38(2):808–835.

Davison, A. C. and Hinkley, D. V. (1997). *Bootstrap methods and their application*. Cambridge university press.

Dhillon, I. S. and Sra, S. (2003). Modeling data using directional distributions. Technical report, Technical Report TR-03-06, Department of Computer Sciences, The University of Texas at Austin.

Efron, B. and Tibshirani, R. J. (1993). *An introduction to the bootstrap*. Chapman & Hall/CRC.

Egozcue, J. J., Pawlowsky-Glahn, V., Mateu-Figueras, G., and Barceló-Vidal, C. (2003). Isometric logratio transformations for compositional data analysis. *Mathematical Geology*, 35(3):279–300.

Eltoft, T., Kim, T., and Lee, T.-W. (2006). On the multivariate laplace distribution. *Signal Processing Letters, IEEE*, 13(5):300–303.

Emerson, S. (2009). *Small sample performance and calibration of the Empirical Likelihood method*. PhD thesis, Stanford university.

Endres, D. M. and Schindelin, J. E. (2003). A new metric for probability distributions. *Information Theory, IEEE Transactions on*, 49(7):1858–1860.

Everitt, B. (2005). *An R and S-PLUS companion to multivariate analysis*. London: Springer Verlag.

Fallaize, C. J. and Kypraios, T. (2014). Exact bayesian inference for the bingham distribution. *arXiv preprint arXiv:1401.2894*.

Fieller, E. C., Hartley, H. O., and Pearson, E. S. (1957). Tests for rank correlation coefficients. i. *Biometrika*, 44(3/4):470–481.

Fieller, E. C. and Pearson, E. S. (1957). Tests for rank correlation coefficients: Ii. *Biometrika*, 48(1/2):29–40.

Filzmoser, P. (2005). Identification of multivariate outliers: a performance study. *Austrian Journal of Statistics*, 34(2):127–138.

Filzmoser, P. and Gschwandtner, M. (2014). *mvoutlier: Multivariate outlier detection based on robust methods*. R package version 2.0.4.

Fisher, N. (1985). Spherical medians. *Journal of the Royal Statistical Society. Series B (Methodological)*, 47(2):342–348.

Fisher, N. I. (1995). *Statistical analysis of circular data*. Cambridge University Press.

Fisher, N. I. and Lee, A. J. (1992). Regression models for an angular response. *Biometrics*, pages 665–677.

Fisher, N. I., Lewis, T., and Embleton, B. J. J. (1987). *Statistical analysis of spherical data*. Cambridge University Press.

Förstner, W. and Moonen, B. (2003). A metric for covariance matrices. In *Geodesy-The Challenge of the 3rd Millennium*, pages 299–309. Springer.

Fraiman, R. and Meloche, J. (1999). Multivariate l-estimation. *Test*, 8(2):255–317.

Fraley, C. and Raftery, A. E. (2002). Model-based clustering, discriminant analysis and density estimation. *Journal of the American Statistical Association*, 97:611–631.

Fraley, C., Raftery, A. E., Murphy, T. B., and Scrucca, L. (2012). *mclust Version 4 for R: Normal Mixture Modeling for Model-Based Clustering, Classification, and Density Estimation*.

García-Portugués, E. (2013). Exact risk improvement of bandwidth selectors for kernel density estimation with directional data. *Electronic Journal of Statistics*, 7:1655–1685.

Gervini, D. (2003). A robust and efficient adaptive reweighted estimator of multivariate location and scatter. *Journal of Multivariate Analysis*, 84(1):116–144.

Gini, C. and Galvani, L. (1929). Di talune estensioni dei concetti di media ai caratteri qualitativi. *Metron*, 8(1-2):3–209.

Goulet, V., Dutang, C., Maechler, M., Firth, D., Shapira, M., and Stadelmann, M. (2013). *expm: Matrix exponential*. R package version 0.99-0.

Greenacre, M. (2009). Power transformations in correspondence analysis. *Computational Statistics & Data Analysis*, 53(8):3107–3116.

Greenacre, M. (2011). Measuring subcompositional incoherence. *Mathematical Geosciences*, 43(6):681–693.

Gregory, K. (2014). *highD2pop: Two-Sample Tests for Equality of Means in High Dimension*. R package version 1.0.

Guidoum, A. C. (2015). *kedd: Kernel estimator and bandwidth selection for density and its derivatives.* R package version 1.0.2.

Habbema, J. D. F., Hermans, J., and Van den Broek, K. (1974). A stepwise discrimination analysis program using density estimation. In *Compstat 1974: Proceedings in Computational Statistics, Vienna*.

Hadi, A. S. and Ling, R. F. (1998). Some cautionary notes on the use of principal components regression. *The American Statistician*, 52(1):15–19.

Haldane, J. B. S. (1948). Note on the median of a multivariate distribution. *Biometrika*, 35(3-4):414–417.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The elements of statistical learning: data mining, inference, and prediction*. Springer, Berlin.

Hijazi, R. H. (2006). Residuals and diagnostics in dirichlet regression. *ASA Proceedings of the General Methodology Section*, pages 1190–1196.

Hornik, K. and Grün, B. (2014). movMF: An R package for fitting mixtures of von mises-fisher distributions. *Journal of Statistical Software*, 58(10):1–31.

James, G. S. (1954). Tests of linear hypotheses in univariate and multivariate analysis when the ratios of the population variances are unknown. *Biometrika*, 41(1/2):19–43.

Jammalamadaka, R. S. and Sengupta, A. (2001). *Topics in circular statistics*. World Scientific.

Jammalamadaka, S. R. and Sarma, Y. R. (1988). A correlation coefficient for angular variables. *Statistical Theory and Data Analysis*, 2:349–364.

Johnson, R. A. and Wichern, D. V. (2007). *Applied multivariate statistical analysis*. Pearson Prentice Hall, New Jersey.

Jolliffe, I. T. (2005). *Principal component analysis*. New York: Springer-Verlag.

Jupp, P. E. (2001). Modifications of the rayleigh and bingham tests for uniformity of directions. *Journal of Multivariate Analysis*, 77(2):1–20.

Kärkkäinen, T. and Äyrämö, S. (2005). On computation of spatial median for robust data mining. *Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems, EUROGEN, Munich*.

Karlis, D. (2003). An em algorithm for multivariate poisson distribution and related models. *Journal of Applied Statistics*, 30(1):63–77.

Karlis, D. and Meligkotsidou, L. (2005). Multivariate poisson regression with covariance structure. *Statistics and Computing*, 15(4):255–265.

Karlis, D. and Ntzoufras, I. (2003). Analysis of sports data by using bivariate poisson models. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 52(3):381–393.

Kawamura, K. (1984). Direct calculation of maximum likelihood estimator for the bivariate poisson distribution. *Kodai mathematical journal*, 7(2):211–221.

Kent, J. T. (1982). The fisher-bingham distribution on the sphere. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 71–80.

Kent, J. T., Ganeiber, A. M., and Mardia, K. V. (2013). A new method to simulate the bingham and related distributions in directional data analysis with applications. *arXiv preprint arXiv:1310.8110*.

Kim, J. and Scott, C. D. (2012). Robust kernel density estimation. *The Journal of Machine Learning Research*, 13(1):2529–2565.

Kocherlakota, S. and Kocherlakota, K. (1998). *Bivariate discrete distributions*. Wiley Online Library.

Koenker, R. (2015). *quantreg: Quantile Regression*. R package version 5.18.

Krishnamoorthy, K. and Xia, Y. (2006). On selecting tests for equality of two normal mean vectors. *Multivariate Behavioral Research*, 41(4):533–548.

Krishnamoorthy, K. and Yu, J. (2004). Modified Nel and Van der Merwe test for the multivariate Behrens-Fisher problem. *Statistics & Probability Letters*, 66(2):161–169.

Kuhn, H. W. (1973). A note on fermat's problem. *Mathematical programming*, 4(1):98–107.

Kullback, S. (1997). *Information theory and statistics*. New York: Dover Publications.

Kume, A., Preston, S., and Wood, A. T. (2013). Saddlepoint approximations for the normalizing constant of fisher–bingham distributions on products of spheres and stiefel manifolds. *Biometrika*, 100(4):971–984.

Kume, A. and Wood, A. T. A. (2005). Saddlepoint approximations for the bingham and fisher–bingham normalising constants. *Biometrika*, 92(2):465–476.

Kwangil, R., Changliang, Z., Zhaojun, Wang, and Guosheng, Y. (2015). Outlier detection for high-dimensional data. *Biometrika*, 102(3):589–599.

Lagani, V., Kortas, G., and Tsamardinos, I. (2013). Biomarker signature identification in "omics" data with multi-class outcome. *Computational and structural biotechnology journal*, 6(7):1–7.

Lancaster, H. O. (1965). The helmert matrices. *American Mathematical Monthly*, 72(1):4–12.

Lange, K. L., Little, R. J., and Taylor, J. M. (1989). Robust statistical modeling using the t distribution. *Journal of the American Statistical Association*, 84(408):881–896.

Le, H. and Small, C. G. (1999). Multidimensional scaling of simplex shapes. *Pattern Recognition*, 32(9):1601–1613.

Loukas, S. and Kemp, C. (1986). The index of dispersion test for the bivariate poisson distribution. *Biometrics*, pages 941–948.

Lund, U. (1999). Least circular distance regression for directional data. *Journal of Applied Statistics*, 26(6):723–733.

Lund, U. and Agostinelli, C. (2012). *CircStats: Circular Statistics, from "Topics in circular Statistics" (2001)*. R package version 0.2-4.

Mackenzie, J. K. (1957). The estimation of an orientation relationship. *Acta Crystallographica*, 10(1):61–62.

Maier, M. J. (2011). *DirichletReg: Dirichlet Regression in R*. R package version 0.3-0.

Maier, M. J. (2014). Dirichletreg: Dirichlet regression for compositional data in r. Technical report, WU Vienna University of Economics and Business.

Mardia, K. V. and Jupp, P. E. (2000). *Directional statistics*. Chicester: John Wiley & Sons.

Mardia, K. V., Kent, J. T., and Bibby, J. M. (1979). *Multivariate Analysis*. London: Academic Press.

Mardia, K. V. and Mardia, K. V. (1972). *Statistics of directional data*. Academic Press London.

Mavridis, D. and Moustaki, I. (2008). Detecting outliers in factor analysis using the forward search algorithm. *Multivariate behavioral research*, 43(3):453–475.

Minka, T. (2000). Estimating a dirichlet distribution. Technical report, Technical report, MIT.

Minka, T. P. (2001). Automatic choice of dimensionality for pca. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 598–604. MIT Press.

Moler, C. and Van Loan, C. (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49.

Moore, E. H. (1920). Abstract. *Bulletin of the American Mathematical Society*, 26(394-395):38.

Morris, J. E. and Laycock, P. (1974). Discriminant analysis of directional data. *Biometrika*, 61(2):335–341.

Möttönen, J., Nordhausen, K., Oja, H., et al. (2010). Asymptotic theory of the spatial median. In *Nonparametrics and Robustness in Modern Statistical Inference and Time Series Analysis: A Festschrift in honor of Professor Jana Jurečková*, pages 182–193. Institute of Mathematical Statistics.

Murteira, J. M. R. and Ramalho, J. J. S. (2014). Regression analysis of multivariate fractional data. *Econometric Reviews*, To appear.

Nadarajah, S. and Kotz, S. (2008). Estimation methods for the multivariate t distribution. *Acta Applicandae Mathematicae*, 102(1):99–118.

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142.

Nelder, J. and Mead, R. (1965). A simplex algorithm for function minimization. *Computer Journal*, 7(4):308–313.

Neto, E. C. (2015). Speeding up non-parametric bootstrap computations for statistics based on sample moments in small/moderate sample size applications. *PloS ONE*, 10(6):To appear.

Ng, K. W., Tian, G. L., and Tang, M. L. (2011). *Dirichlet and Related Distributions: Theory, Methods and Applications*, volume 889. Chichester: John Wiley & sons.

Nychka, D., Furrer, R., and Sain, S. (2015). *fields: Tools for Spatial Data*. R package version 8.2-1.

Oliveira, M., Crujeiras, R. M., and Rodríguez-Casal, A. (2013). *NPCirc: Nonparametric Circular Methods*. R package version 2.0.0.

Opgen-Rhein, R. and Strimmer, K. (2006). Inferring gene dependency networks from genomic longitudinal data: a functional data approach. *Revstat*, 4(1):53–65.

Österreicher, F. and Vajda, I. (2003). A new class of metric divergences on probability spaces and its applicability in statistics. *Annals of the Institute of Statistical Mathematics*, 55(3):639–653.

Owen, A. B. (2001). *Empirical likelihood*. CRC press, Boca Raton.

Papadakis, M., Tsagris, M., Dimitriadis, M., Fafalios, S., Tsamardinos, I., Fasiolo, M., Borboudakis, G., Burkardt, J., Zou, C., and Lakiotaki, K. (2019). *Rfast: Fast R Functions*. R package version 1.9.4.

Pawlowsky Glahn, V., Egozcue, J. J., and Tolosana Delgado, R. (2007). *Lecture notes on compositional data analysis*.

Penrose, R. (1956). On best approximate solutions of linear matrix equations. In *Mathematical Proceedings of the Cambridge Philosophical Society*, pages 17–19. Cambridge Univ Press.

Pewsey, A., Neuhäuser, M., and Ruxton, G. D. (2013). *Circular Statistics in R*. Oxford University Press.

Presnell, B., Morrison, S. P., and Littell, R. C. (1998). Projected multivariate linear models for directional data. *Journal of the American Statistical Association*, 93(443):1068–1077.

Rajan, J. and Rayner, P. (1997). Model order selection for the singular value decomposition and the discrete karhunen–loeve transform using a bayesian approach. *IEE Proceedings-Vision, Image and Signal Processing*, 144(2):116–123.

Rauber, T. W., Braunb, T., and Berns, K. (2008). Probabilistic distance measures of the dirichlet and beta distributions. *Pattern Recognition*, 41:637–645.

Rayleigh, L. (1919). On the problem of random vibrations, and of random flights in one, two, or three dimensions. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 37(220):321–347.

Rayner, J. C., Thas, O., and Best, D. J. (2009). *Smooth tests of goodness of fit: using R*. John Wiley & Sons.

Rivest, L.-P. (1986). Modified kent's statistics for testing goodness of fit for the fisher distribution in small concentrated samples. *Statistics & probability letters*, 4(1):1–4.

Robert, P. W. (1976). On the choice of smoothing parameters for parzen estimators of probability density functions. *IEEE Transactions on Computers*.

Rossi, P. (2015). *bayesm: Bayesian Inference for Marketing/Micro-Econometrics*. R package version 3.0-2.

Scealy, J. and Welsh, A. (2014). Fitting kent models to compositional data with small concentration. *Statistics and Computing*, 24(2):165–179.

Scealy, J. L. and Welsh, A. H. (2011a). Properties of a square root transformation regression model. In *Proceedings of the 4rth Compositional Data Analysis Workshop, Girona, Spain*.

Scealy, J. L. and Welsh, A. H. (2011b). Regression for compositional data by using distributions defined on the hypersphere. *Journal of the Royal Statistical Society. Series B*, 73(3):351–375.

Schaefer, J., Opgen-Rhein, R., and Strimmer, K. (2007). corpcor: efficient estimation of covariance and (partial) correlation. r package version 1.4. 7.

Schnute, J. T. and Haigh, R. (2007). Compositional analysis of catch curve data, with an application to sebastes maliger. *ICES Journal of Marine Science*, 64:218–233.

Shabalin, A. A. (2012). Matrix eqtl: ultra fast eqtl analysis via large matrix operations. *Bioinformatics*, 28(10):1353–1358.

Sharp, W. (2006). The graph median–a stable alternative measure of central tendency for compositional data sets. *Mathematical geology*, 38(2):221–229.

Silverman, B. W. (1986). *Density estimation for statistics and data analysis*, volume 26. New York: CRC press.

Sra, S. (2012). A short note on parameter approximation for von mises-fisher distributions: and a fast implementation of i s (x). *Computational Statistics*, 27(1):177–190.

Statnikov, A., Aliferis, C. F., Tsamardinos, I., Hardin, D., and Levy, S. (2005). A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis. *Bioinformatics*, 21(5):631–643.

Stephens, M. A. (1972). Multri-sample tests for the von mises distribution. Technical report, Technical Report 190, Department of Statistics, Stanford University (130, 135).

Stephens, M. A. (1979). Vector correlation. *Biometrika*, 66(1):41–48.

Stephens, M. A. (1982). Use of the von mises distribution to analyse continuous proportions. *Biometrika*, 69(1):197–203.

Tarmast, G. (2001). Multivariate log-normal distribution. *Proceedings of 53rd Session of International Statistical Institute*.

Taylor, C. C. (2008). Automatic bandwidth selection for circular density estimation. *Computational Statistics & Data Analysis*, 52(7):3493–3500.

Tibshirani, R. J. and Tibshirani, R. (2009). A bias correction for the minimum error rate in cross-validation. *The Annals of Applied Statistics*, 3(1):822–829.

Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622.

Todorov, V. and Filzmoser, P. (2010). Robust statistic for the one-way manova. *Computational Statistics & Data Analysis*, 54(1):37–48.

Tsagris, M. (2014). The k-nn algorithm for compositional data: a revised approach with and without zero values present. *Journal of Data Science*, 12(3):519–534.

Tsagris, M. (2015a). A novel, divergence based, regression for compositional data. In *Proceedings of the 28th Panhellenic Statistics Conference, Athens, Greece*.

Tsagris, M. (2015b). Regression analysis with compositional data containing zero values. *Chilean Journal of Statistics*, 6(2):47–57.

Tsagris, M., Athineou, G., and Sajib, A. (2016a). *Directional: A collection of R functions for directional data analysis*. R package version 2.1.

Tsagris, M., Elmatzoglou, I., and Frangos, C. C. (2012). The assessment of performance of correlation estimates in discrete bivariate distributions using bootstrap methodology. *Communications in Statistics-Theory and Methods*, 41(1):138–152.

Tsagris, M. and G., A. (2016). *Compositional: A collection of R functions for compositional data analysis*. R package version 1.5.

Tsagris, M. and Papadakis, M. (2018). Taking r to its limits: 70+ tips. *PeerJ PrePrints*, page e26605v1.

Tsagris, M., Preston, S., and Wood, A. T. (2016b). Improved classification for compositional data using the *alpha*-transformation. *Journal of Classification*, To appear.

Tsagris, M. T., Preston, S., and Wood, A. T. A. (2011). A data-based power transformation for compositional data. In *Proceedings of the 4rth Compositional Data Analysis Workshop, Girona, Spain*.

Tsamardinos, I., Rakhshani, A., and Lagani, V. (2014). Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization. In *Artificial Intelligence: Methods and Applications*, pages 1–14. Springer.

Tsybakov, A. B. (2009). *Introduction to nonparametric estimation*. Springer.

Tyler, D. E. (1987). Statistical analysis for the angular central gaussian distribution on the sphere. *Biometrika*, 74(3):579–589.

Van Den Boogaart, K. G. and Tolosana-Delgado, R. (2013). *Analyzing Compositional Data with R*. Springer.

Varadhan, R. and Gilbert, P. (2009). BB: An R package for solving a large system of nonlinear equations and for optimizing a high-dimensional nonlinear objective function. *Journal of Statistical Software*, 32(4):1–26.

Wand, M. M. P. and Jones, M. M. C. (1995). *Kernel smoothing*. Crc Press.

Watson, G. S. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, 26(4):359–372.

Watson, G. S. (1983a). Large sample theory of the langevin distribution. *Journal of Statistical Planning and Inference*, 8(1):245–256.

Watson, G. S. (1983b). *Statistics on spheres*. Wiley New York.

Watson, G. S. and Nguyen, H. (1985). A Confidence Region in a Ternary Diagram from Point Counts. *Mathematical Geology*, 17(2):209–213.

Wood, A. (1982). A bimodal distribution on the sphere. *Applied Statistics*, 31(1):52–58.

Wood, A. T. A. (1994). Simulation of the von mises fisher distribution. *Communications in statistics-simulation and computation*, 23(1):157–164.

Woronow, A. (1997). The elusive benefits of logratios. In *Proceedings of the 3rd Annual Conference of the International Association for Mathematical Geology, Barcelona, Spain*.

Yee, T. W. (2010). The VGAM package for categorical data analysis. *Journal of Statistical Software*, 32(10):1–34.

Zhao, J. and Jiang, Q. (2006). Probabilistic pca for t distributions. *Neurocomputing*, 69(16):2217–2226.