Multivariate statistical functions in R

Michail T. Tsagris
mtsagris@yahoo.gr

College of engineering and technology, American university of the middle east, Egaila, Kuwait

Version 6.1

Athens, Nottingham and Abu Halifa (Kuwait)
31 October 2014

Contents

A short introduction

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. 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. The functions have been tested using example data sets found at the references.

As I update the versions I 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, along with corrections or any comments, 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 even this version needs a bit polishing and in some cases more explanation of the algorithms. These will be done (I hope so) in time.

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.

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.

1 Mean vectors

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

1.1 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

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

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 (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).

```
hotel1T2=function(x,M,a=0.05,R=999) {
## x is the data set
## M is the hypothesised mean
## a is the significance level, set by default to 0.05 and
## R is the number of bootstrap replicates set by default to 999
x=as.matrix(x)
m=colMeans(x)  ## sample mean vector
s=cov(x)  ## sample covariance matrix
n=nrow(x)  ## sample size
p=ncol(x)  ## dimensionality of the data
d=m-M   ## difference between the sample mean and the null hypothesis mean
df1=p   ## degrees of freedom of the numerator of the F distribution
df2=n-p  ## degrees of freedom of the numerator of the F distribution
test=as.vector( (n*(n-p))/((n-1)*p)*d%*%solve(s)%*%d ) ## test statistic
if (R==1) {
pvalue=1-pf(test,p,n-p)  ## p-value of the test statistic
crit=qf(1-a,df1,df2)  ## critival value of the F disitrubition
result=list(m=m,test=test,df1=df1,df2=df2,critical=crit,p.value=pvalue) }
if (R>1) { ## bootstrap calibration
```

```
t=rep(0,R)
m=matrix(rep(colMeans(x),n),nrow=n,byrow=TRUE)  ## sample mean vector
M=matrix(rep(M,n),nrow=n,byrow=TRUE)  ## mean vector under the null hypothesis
y=x-m+M  ## brings the data under the null hypothesis, i.e. mean vector equal to M
for (i in 1:R) {
b=sample(1:n,n,replace=TRUE)
s1=cov(y[b,])
d1=colMeans(y[b,])-M[1,]
t[i]=(n*(n-p))/((n-1)*p)*d1%*%solve(s1)%*%d1  }
pvalue=( sum(t>test)+1 )/(R+1)  ## bootstrap p-value
hist(t,xlab="bootstrapped test statistic",main=" ")
abline(v=test,lty=2,lwd=2) ## The dotted vertical line is the test statistic value
result=list(p.value=pvalue)  }
result  }
```

1.2 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:

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

Under H_0 the statistic F given by

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

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).

```
hotel2T2=function(x1,x2,a=0.05,R=999) {
## x1 and x2 are the multivariate samples
## a is the significance level, set by default to 0.05
```

```
## R is the number of bootstrap replicates set by default to 999
x1=as.matrix(x1)
x2=as.matrix(x2)
p=ncol(x1)  ## dimensionality of the data
n1=nrow(x1)  ## size of the first sample
n2=nrow(x2)  ## size of the second sample
n=n1+n2  ## total sample size
xbar1=apply(x1,2,mean)  ## sample mean vector of the first sample
xbar2=apply(x2,2,mean)  ## sample mean vector of the second sample
dbar=xbar2-xbar1 ## difference of the two mean vectors
v=((n1-1)*var(x1)+(n2-1)*var(x2))/(n-2)  ## 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=1-pf(test,p,n-p-1)  ## p-value of the test statistic
result=list(test=test,critical=crit,p.value=pvalue,df1=p,df2=n-p-1) }
if (R>1) { ## bootstrap calibration
z=rbind(x1,x2)  ## the two samples combined in one
mc=matrix(rep(colMeans(z),n),nrow=n,byrow=TRUE)  ## the combined sample mean vector
m1=matrix(rep(colMeans(x1),n1),nrow=n1,byrow=TRUE)  ## first mean vector
m2=matrix(rep(colMeans(x2),n2),nrow=n2,byrow=TRUE)  ## second 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
y1=x1-m1+mc[1:n1,]
y2=x2-m2+mc[1:n2,]
t=rep(0,R)
for (i in 1:R) {
b1=sample(1:n1,n1,replace=TRUE)
b2=sample(1:n2,n2,replace=TRUE)
yb1=apply(y1[b1,],2,mean)  ## sample mean vector of the first sample
yb2=apply(y2[b2,],2,mean)  ## sample mean vector of the second sample
db=yb2-yb1 ## difference of the two mean vectors
vb=((n1-1)*var(y1[b1,])+(n2-1)*var(y2[b2,]))/(n-2)  ## pooled covariance matrix
t2=(n1*n2*db%*%solve(vb)%*%db)/n
t[i]=as.vector( ((n-p-1)*t2)/((n-2)*p) )  }
pvalue=( sum(t>test)+1 )/ (R+1)
hist(t,xlab="bootstrapped test statistic",main=" ")
abline(v=test,lty=2,lwd=2) ## The dotted vertical line is the test statistic value
```

```
result=list(p.value=pvalue) }
result }
```

1.3 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:

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

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

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

where

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

The modified Nel and van der Merwe (1986) test is based on the quadratic form

$$T_{mnv}^2 = \left(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2\right)^T \hat{\mathbf{\Sigma}}^{-1} \left(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2\right), \tag{1.1}$$

where $\hat{\mathbf{\Sigma}} = \text{Cov}(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2) = \frac{\mathbf{S}_1}{n_1} + \frac{\mathbf{S}_2}{n_2}$.
It is shown in Krishnamoorthy and Yu (2004) that $T_{mnv}^2 \sim \frac{\nu p}{\nu - p + 1} F_{p,\nu-p+1}$ approximately, where

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

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) {
```

```
## y1 and y2 are the two samples
## a is the significance level, set by default to 0.05
## 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 re-samples are set by default
y1=as.matrix(y1)
y2=as.matrix(y2)
p=ncol(y1)  ## dimensionality of the data
n1=nrow(y1)  ## size of the first sample
n2=nrow(y2)  ## size of the second sample
n=n1+n2  ## the toal sample size
ybar1=apply(y1,2,mean)  ## sample mean vector of the first sample
ybar2=apply(y2,2,mean)  ## sample mean vector of the second sample
dbar=ybar2-ybar1  ## difference of the two mean vectors
A1=cov(y1)/n1 ; A2=cov(y2)/n2
V=A1+A2   ## covariance matrix of the difference
test=as.numeric(dbar%*%solve(V)%*%dbar)
b1=solve(V)%*%A1
b2=solve(V)%*%A2
if (R==1) { ## James test
A=1+(1/(2*p))*( (sum(diag(b1)))^2/(n1-1)+(sum(diag(b2)))^2/(n2-1) )
B=(1/(p*(p+2)))*( sum(diag(b1%*%b1))/(n1-1)+sum(diag(b2%*%b2))/(n2-1)+
0.5*((sum(diag(b1)))^2/(n1-1)+(sum(diag(b2)))^2/(n2-1)) )
x2=qchisq(1-a,p)
delta=(A+B*x2)
twoha=x2*delta ## corrected critical value of the chi-square distribution
pvalue=1-pchisq(test/delta,p)  ## p-value of the test statistic
result=list(test=test,correction=delta,corrected.critical.value=twoha,p.value=pvalue) }
if (R==2)  { ## MNV test
low=( sum(diag(b1%*%b1))+sum(diag(b1))^2 )/n1+( sum(diag(b2%*%b2))+sum(diag(b2))^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=1-pf(test,p,v-p+1)  ## p-value of the test statistic
result=list(test=test,critical=crit,df1=p,df2=v-p-1,p.value=pvalue) }
if (R>2) { ## bootstrap calibration
z=rbind(y1,y2)  ## the two samples combined in one
mc=matrix(rep(colMeans(z),n),nrow=n,byrow=TRUE)  ## the combined sample mean vector
```

```
m1=matrix(rep(colMeans(y1),n1),nrow=n1,byrow=TRUE)  ## first mean vector
m2=matrix(rep(colMeans(y2),n2),nrow=n2,byrow=TRUE)  ## second 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
x1=y1-m1+mc[1:n1,]
x2=y2-m2+mc[1:n2,]
t=rep(0,R)
for (i in 1:R) {
b1=sample(1:n1,n1,replace=TRUE)
b2=sample(1:n2,n2,replace=TRUE)
xbar1=apply(x1[b1,],2,mean)  ## sample mean vector of the first sample
xbar2=apply(x2[b2,],2,mean)  ## sample mean vector of the second sample
db=xbar2-xbar1  ## difference of the two mean vectors
A1=cov(x1[b1,])/n1 ; A2=cov(x2[b2,])/n2
V=A1+A2   ## covariance matrix of the difference
t[i]=as.numeric(db%*%solve(V)%*%db) }
pvalue=( sum(t>test)+1 )/ (R+1)
hist(t,xlab="bootstrapped test statistic",main=" ")
abline(v=test,lty=2,lwd=2) ## The dotted vertical line is the test statistic value
result=list(p.value=pvalue) }
result }
```

1.4 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

$$J = \sum_{i=1}^{k} \left(\bar{\mathbf{x}}_i - \bar{\mathbf{X}}\right)^T \mathbf{W}_i \left(\bar{\mathbf{x}}_i - \bar{\mathbf{X}}\right), \tag{1.2}$$

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 (1.2) with a $\chi^2_{r,1-\alpha}$, where $r = p\,(k-1)$ 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 (1.2) with a corrected χ^2 distribution instead. Let A and B be

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

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
x=as.matrix(x) ## makes sure x is a matrix
ina=as.numeric(ina) ## the group indicator variable
ni=as.vector(table(ina)) ## the group sample sizes
k=max(ina) ## the number of groups
p=ncol(x) ## the dimensionality
n=nrow(x) ## the total sample size
## the objects below will be used later
me=mi=W=matrix(nrow=k,ncol=p)
t=rep(0,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) {
mi[i,]=colMeans(x[ina==i,])
wi[,,i]=solve( var(x[ina==i,])/ni[i] )
me[i,]=mi[i,]%*%wi[,,i] }
W=apply(wi,1:2,sum)
ma=apply(me,2,sum)
mesi=ma%*%solve(W) ## common mean vector
for (i in 1:k) t[i]=(mi[i,]-mesi)%*%wi[,,i]%*%t(mi[i,]-mesi)
test=sum(t) ## the test statistic
r=p*(k-1)
t1=t2=numeric(k)
for (i in 1:k){
exa1=diag(p)-solve(W)%*%wi[,,i]
```

```
exa2=exa1%*%exa1
t1[i]=sum(diag(exa1))
t2[i]=sum(diag(exa2)) }
A=1+1/(2*r)*sum(t1^2/(ni-1))
B=1/(r*(r+2))*sum( t2/(ni-1)+t1^2/(2*(ni-1)) )
x2=qchisq(1-a,r)
delta=(A+B*x2)
twoha=x2*delta ## corrected critical value of the chi-square distribution
pvalue=1-pchisq(test/delta,r)  ## p-value of the test statistic
result=list(test=test,correction=delta,corrected.critical.value=twoha,p.value=pvalue)
result }
```

2 Covariance matrices

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

2.1 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

$$-2\log\lambda = ntr\left\{\boldsymbol{\Sigma}_0^{-1}\mathbf{S}\right\} - n\log\left|\boldsymbol{\Sigma}_0^{-1}\mathbf{S}\right| - np, \tag{2.1}$$

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 (2.1) 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 significance level set by default to 0.05
x=as.matrix(x)
Sigma=as.matrix(Sigma)
p=ncol(x)  ## dimensionality of the data
n=nrow(x)  ## total sample size
S=cov(x)  ## sample covariance matrix
## the next 2 lines construct the test statistic
mesa=solve(Sigma)%*%S
test=n*sum(diag(mesa))-n*log(det(mesa))-n*p
df=0.5*p*(p+1)  ## the degrees of freedom of the chi-square distribution
pvalue=1-pchisq(test,df)  ## p-value of the test statistic
crit=qchisq(1-a,df)  ## critical value of the chi-square distribution
list(test=test,degres=df,p.value=pvalue,critical=crit) }
```

2.2 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: $H_0 : \mathbf{\Sigma}_1 = ... = \mathbf{\Sigma}_k$. The algorithms are taken from Aitchison, 2003, pg. 155 and Mardia et al., 1979, pg. 140.

2.2.1 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

$$-2log\lambda = n \log |\mathbf{S}| - \sum_{i=1}^{k} n_i \log |\mathbf{S_i}| = \sum_{i=1}^{k} n_i \log \left| \mathbf{S}_i^{-1} \mathbf{S} \right|, \tag{2.2}$$

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 significance level, set to 0.05 by default
x=as.matrix(x)
p=ncol(x)  ## dimension of the data set
n=nrow(x)  ## total sample size
k=max(ina)  ## number of groups
nu=rep(0,k)  ## the sample size of each group will be stored later
pame=rep(0,k)
## the next 2 "for" functions separate the k groups and extract the
## covariance matrix of each group
## the way is not the best but it works
nu=as.vector(table(ina))
mat=mat1=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) {
mat[,,i]=((nu[i]-1)/nu[i])*cov(x[ina==i,])
mat1[,,i]=(nu[i]-1)*cov(x[ina==i,]) }
Sp=apply(mat1,1:2,sum)/n
for (i in 1:k) pame[i]=det(solve(mat[,,i])%*%Sp)
test=sum(nu*log(pame))  ## test statistic
```

```
df=0.5*p*(p+1)*(k-1)  ## degrees of freedom of the asymptotic chi-square
pvalue=1-pchisq(test,df)  ## p-value of the test statistic
crit=qchisq(1-a,df)  ## critical value of the chi-square distribution
list(test=test,degrees=df,critical=crit,p.value=pvalue) }
```

2.2.2 Box's M test

According to Mardia et al., 1979, pg. 140, it may be argued that if n_i is small, then (2.2) 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 (2.2). 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 significance level, set to 0.05 by default
x=as.matrix(x)
p=ncol(x)  ## dimension of the data set
n=nrow(x)  ## total sample size
k=max(ina)  ## number of groups
nu=rep(0,k)  ## the sample size of each group will be stored here later
pame=rep(0,k)  ## the determinant of each covariance will be stored here
## the next "for" function calculates the covariance matrix of each group
nu=as.vector(table(ina))
mat=mat1=array(dim=c(p,p,k))
```

```
for (i in 1:k) {
mat[,,i]=cov(x[ina==i,])
pame[i]=det(mat[,,i]) ## the detemirnant of each covariance matrix
mat1[,,i]=(nu[i]-1)*cov(x[ina==i,])  }
## the next 2 lines calculate the pooled covariance matrix
Sp=apply(mat1,1:2,sum)
Sp=Sp/(n-k)
for (i in 1: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)
df=0.5*p*(p+1)*(k-1)  ## degrees of freedom of the chi-square distribution
pvalue=1-pchisq(test,df) ## p-value of the test statistic
crit=qchisq(1-a,df)  ## critical value of the chi-square distribution
list(M.test=test,degrees=df,critical=crit,p.value=pvalue) }
```

3 Regression, correlation and discriminant analysis

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

3.1 Correlation

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

Fisher's transformation for the correlation coefficient is defined as

$$\hat{z} = \frac{1}{2} \log \frac{1+r}{1-r} \tag{3.1}$$

with inverse equal to

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

The estimated standard error of (3.1) is $\frac{1-r^2}{\sqrt{n-3}}$ (Efron and Tibshirani, 1993). 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 (3.1), assuming asymptotic normality of (3.1) and performs hypothesis testing for the true (any, non only zero) value of the correlation.

```
correl=function(y,x,a=0.05,rho=0) {
## y and x are the two variables
## a is the significance level
## rho is the hypothesised correlation
y=as.vector(y)
x=as.vector(x)
n=length(x)
r=cor(y,x) ## the correlation value
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-r^2)/sqrt(n-3) ## standard error for Fisher's transformation of Ho
test=(zh1-zh0)/se  ### test statistic
pvalue=2*(1-pnorm(abs(test))) ## p-value
zL=zh1-qnorm(1-a/2)*se ; zH=zh1+qnorm(1-a/2)*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)=c('lower','uper')
list(correlation=r,p.value=pvalue,CI=CI) }
```

3.1.2 Non-parametric bootstrap 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 (3.1). 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
x=as.matrix(x)
s=cov(x) ; n=nrow(x)
lam=eigen(s)$values
vec=eigen(s)$vectors
A=vec%*%diag(sqrt(lam))%*%t(vec)
z=x%*%solve(A) ## This makes the correlation matrix equal to
## the identity matrix, thus rho=0
t=rep(0,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=T)
y=z[nu,] ; rb=cor(y)[2]
t[i]=0.5*log((1+rb)/(1-rb)) }
pvalue=(sum(t>test)+1)/(B+1) ## bootstrap p-value
hist(t,xlab="bootstrapped test statistic",main=" ")
abline(v=test,lty=2,lwd=2) ## The dotted vertical line is the test statistic value
list(test=test,p.value=pvalue)  }
```

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.

3.1.3 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 (3.1) 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) {
## r1 and r2 are the two correlation coefficients
## n1 and n2 are the two sample sizes
z1=0.5*log((1+r1)/(1-r1))  ## Fisher's transformation
z2=0.5*log((1+r2)/(1-r2))  ## Fisher's transformation
test=(z1-z2)/sqrt(1/(n1-3)+1/(n2-3))  ## test statistic
pvalue=2*(1-pnorm(abs(test)))  ## p-value calculation
list(test=test,p.value=pvalue) }
```

3.2 Regression

3.2.1 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 $\hat{\mathbf{\Sigma}}_{\mathbf{e}} = \frac{1}{n-p}\mathbf{Y}^T\mathbf{P}\mathbf{Y}$ with $\mathbf{P} = \mathbf{I}_n - \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T$ is the error covariance matrix. The sample size is denoted by n, p indicates the number of independent variables + 1 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

$$DE_i = \sqrt{\hat{\mathbf{e}}_i^T\hat{\mathbf{\Sigma}}_{\mathbf{e}}^{-1}\hat{\mathbf{e}}_i^T} \text{ and } DX_i = \sqrt{\left(\mathbf{X}_i - \hat{\boldsymbol{\mu}}_{\mathbf{X}}\right)^T\hat{\mathbf{\Sigma}}_{\mathbf{XX}}^{-1}\left(\mathbf{X}_i - \hat{\boldsymbol{\mu}}_{\mathbf{X}}\right)} \tag{3.2}$$

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) {
## y is the dependent variable and is expected to be a matrix
## if y is a vector then the classical univariate regression
## is performed
## x contains the independent variable(s)
y=as.matrix(y)
x=as.matrix(x)
n=nrow(y) ## sample size
d=ncol(y) ## dimensionality of y
p=ncol(x) ## dimensionality of x
X=cbind(1,x)  ## the design matrix
beta=solve(t(X)%*%X)%*%t(X)%*%y ## the parameters
P=diag(n)-X%*%solve(t(X)%*%X)%*%t(X)
s=(1/(n-p-1))*t(y)%*%P%*%y
sxx=cov(x) ## covariance of the independent variables
res=y-X%*%beta ## residuals
dres=sqrt(diag(res%*%solve(s)%*%t(res))) ## Mahalanobis distances of the residuals
mx=matrix(rep(colMeans(x),n),byrow=T,ncol=p)
dx=sqrt(diag((x-mx)%*%solve(sxx)%*%t(x-mx)))
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')
```

```
crit.res=sqrt(qchisq(0.975,d))
crit.x=sqrt(qchisq(0.975,p))
abline(h=crit.res)
abline(v=crit.x)
resid.outliers=which(dres>crit.res)
x.leverage=which(dx>crit.x)
out.and.lever=which(dx>crit.x & dres>crit.res)
S=kronecker(solve(t(X)%*%X),s) ## covariance of the parameters
sigma=t(matrix(sqrt(diag(S)),ncol=p+1))
fitted=X%*%beta ## fitted values
colnames(fitted)=colnames(y)
colnames(sigma)=colnames(beta)=colnames(y)
rownames(beta)=rownames(sigma)=c('Intercept',paste('x', 1:p, sep=''))
list(beta=beta,Std.errors=sigma,resid.outliers=resid.outliers,
x.leverage=x.leverage,out.and.lever=out.and.lever,fitted=fitted) }
```

3.2.2 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 neighbours. The dependent variable can be either univariate or multivariate. A cross validation algorithm to choose the value of k is described below and after that the relevant code is given below.

1. First standardize the independent variables so that they are all in the same scale.
2. Choose a value of k.
3. Remove a number of pairs of vectors $(\mathbf{y}^*, \mathbf{x}^*)$ from the sample. The $\mathbf{x}^*$ is the test sample and the remaining $\mathbf{x}$ observations form the training sample. Say you remove 20% of them and so the test sample size is equal to ν.
4. Find the k closest neighbours of the test set $\mathbf{x}^*$ from the training sample. That is, calculate the distances, of all the remaining observations, from $\mathbf{x}^*$ and take the k observations with the smallest distance. I ma using the Euclidean distance but this can change by the user.
5. Calculate the average of the corresponding training dependent values $\mathbf{y}$. This is the estimated value $\hat{\mathbf{y}}_i^*$ of the observed $\mathbf{y}_i^*$ for $i = 1, \ldots, \nu$.
6. Calculate the sum of the squared distances $\sum_{i=1}^{\nu} \left(\hat{\mathbf{y}}_i^* - \mathbf{y}_i^*\right)^2$.

7. Repeat steps 3-6 for all observations and take the average of the squared distances from step 6. This is the mean predicted squared error (MSPE).

8. Repeat steps 3-7 for many values of k, say from 2 to 10 (the maximum number of nearest neighbours depends upon the sample size of course) and choose the value of k which minimizes the MSPE.

The function *knn.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 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 *kern.tune* and *pcr.tune*. Thus, the MSPE for all three methods is directly comparable.

```
knn.tune=function(y,x,fraction=0.20,R=1000,A=10,type='euclidean',seed=FALSE){
## y is the multivariate (or univariate) dependent variable
## x containts the independent variables(s)
## fraction is the percentage of data to be used for testing purposes
## the remaining data belong to the training set
## it is assumed that the training set contains at least 11 observations
## R is the number of cross validations to be performed
## A is the highest number of nearest neighbours
## type is for the distance, Euclidan or Manhattan distance
y=as.matrix(y)
x=as.matrix(x)
stand=function(x) (x-mean(x))/sd(x)
X=apply(x,2,stand) ## standardize the independent variable(s)
n=nrow(y)
ind=1:n
crit=matrix(nrow=R,ncol=A-1)
num=round(fraction*n) ## test set sample size
if (type=='euclidean') {
apostasi=as.matrix(dist(x,diag=T,upper=T,method='euclidean')) }
if (type=='manhattan') {
apostasi=as.matrix(dist(x,diag=T,upper=T,method='manhattan')) }
## apostasi is an nxn matrix containing
## the distances of each observation from the other based on the
## independent variable(s)
## Euclidean distance is set by default
deigma=matrix(nrow=R,ncol=num)
## 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 seed==TRUE then the results will always be the same
if (seed==TRUE) set.seed(1234567)
for (vim in 1:R) {
est=matrix(nrow=num,ncol=ncol(y))
deigma[vim,]=sample(1:n,num)
test=y[deigma[vim,],] ## the test set dependent variables
aba=as.vector(deigma[vim,])
index=ind[-deigma[vim,]]
for (j in 1:c(A-1)) {
knn=j+1
apo=apostasi[aba,-aba]
## apo contains the distances of the test set from the others
for (k in 1:num) {
dis=cbind(apo[k,],index)
dis=dis[order(dis[,1]),] ## sorts the distances
yb=y[dis[1:knn,2],]
yb=as.matrix(yb)
est[k,]=colMeans(yb) }
crit[vim,j]=mean((test-est)^2) } }
mspe=colMeans(crit)
plot(2:knn,mspe,xlab='Nearest neighbours',ylab='MSPE',type='b')
chosen=which.min(mspe)+1
names(mspe)=2:knn
list(k=chosen,mspe=mspe)
```

The next code performs k-NN multivariate regression for a given value of k

```
pred.knn=function(xnew,y,x,k,type='euclidean'){
## xnew is the new observation
## y is the multivariate dependent variable
## x contains the independent variable(s)
## k is the number of nearest neighbours to use
## type is for the distance, Euclidan or Manhattan distance
y=as.matrix(y)
x=as.matrix(x)
n=nrow(y)
xnew=as.matrix(xnew)
```

```
if (type=='euclidean') {
apostasi=as.matrix(dist(rbind(xnew,x),diag=T,upper=T,method='euclidean')) }
if (type=='manhattan') {
apostasi=as.matrix(dist(rbind(xnew,x),diag=T,upper=T,method='manhattan')) }
nu=nrow(xnew)
est=matrix(nrow=nu,ncol=ncol(y))
for (i in 1:nu) {
dis=cbind(apostasi[i,-c(1:nu)],1:n)
dis=dis[order(dis[,1]),] ## sorts the distances
yb=y[dis[1:k,2],]
yb=as.matrix(yb)
est[i,]=colMeans(yb) }
est }
```

3.2.3 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 by Tsybakov (2009).

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

$$f(\hat{x};h) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right). \tag{3.3}$$

An example of a kernel function is the standard normal. Thus, (3.3) can be written as

$$f(\hat{x};h) = \frac{1}{nh\sqrt{2\pi}}\sum_{i=1}^{n} e^{-\frac{-(X_i-x)^2}{2h^2}}. \tag{3.4}$$

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

$$f(\hat{x};h) = \frac{c}{nh}\sum_{i=1}^{n} e^{-\frac{|X_i-x|}{h}}, \tag{3.5}$$

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

$$\hat{m}(x, p, h) = \mathbf{e}_1^T \left[\mathbf{X}^T (\mathbf{x}, p) \mathbf{W}_\mathbf{x} \mathbf{X} (\mathbf{x}, p) \right]^{-1} \mathbf{X}^T (\mathbf{x}, p) \mathbf{W}_\mathbf{x} \mathbf{Y}. \tag{3.6}$$

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

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

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.

If $p = 0$ then we end up with the Nadaraya-Watson estimator (Nadaraya, 1964) and (Watson, 1964) and in this case (3.6) 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 $\sum_{i=1}^n K\left(\frac{\mathbf{X}_i - \mathbf{x}}{h}\right) = 0$.

Finally $\mathbf{e}_1$ is a $(p+1) \times 1$ vector whose first element is 1 and all other elements are zero. Let us go and see (3.6) without $\mathbf{e}_1^T$. The resulting matrix is of $(1+p) \times q$. We want the first row of this matrix and that is why we use the $\mathbf{e}_1$ vector.

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 $\mathbf{H} = h\mathbf{I}_q$, where $\mathbf{I}_q$ is the identity matrix. Thus the kernel functions (3.4) and (3.5) are written

as

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

respectively, where $\|\mathbf{x}-\mathbf{y}\|^p = \sum_{i=1}^{d} |x_i - y_i|^p$ is the L_p norm and d here stands for the dimensionality of the data. 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. The next code performs local polynomial regression for a given polynomial. It estimates the value of the dependent variable (univariate or multivariate) based on measurements from the independent variable(s).

```
kern.reg=function(x,Y,X,h,r=0,type='gauss') {
## Y is the multivariate (or univariate) dependent variable
## X contains the independent variable(s)
## x is a specific X value
## 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 taxi
Y=as.matrix(Y)
X=as.matrix(X)
x=as.matrix(x)
p=ncol(X)
n=nrow(Y)
me=colMeans(X)
m=matrix(rep(me,n),byrow=T,ncol=p)
s=apply(X,2,sd)
X=(X-m)/s ## standardize the independent variable(s)
x=(x-me)/s ## standardize the x values also
x=matrix(rep(x,n),byrow=T,nrow=n)
if (type=='gauss') z=diag( exp(-0.5*(X-x)%*%t(X-x)/h^2) )
if (type=='taxi')  z=rowSums(abs(x-X))/h
if (r==0) {
mhx=colSums(z*Y)/sum(z)
if (sum(z)==0) mhx=0 }
if (r>0) {
W=diag(z)
if (r==1) Z=X-x
if (r>1) {
```

```
p=ncol(X)
Z=array(dim=c(n,p,r))
for (j in 1:r) Z[,,j]=(X-x)^j
Z=matrix(Z,ncol=r*p) }
X1=cbind(1,Z)
be=solve(t(X1)%*%W%*%X1)%*%t(X1)%*%W%*%Y
mhx=be[1,] }
mhx }
```

3.2.4 Choosing the bandwidth in kernel regression in a very simple way

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.

```
kern.tune=function(Y,X,h,r=0,fraction=0.20,R=1000,type='gauss',seed=FALSE) {
## Y is the multivariate (or univariate) dependent variable
## X contains the independent variables
## h is the bandwidth
## r is the degree of the local polynomial, usually 0 or 1
## 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
## type denotes the type of kernel to be used, gauss or taxi
Y=as.matrix(Y)
X=as.matrix(X)
n=nrow(Y)
msp=matrix(ncol=length(h),nrow=R)
k=round(fraction*n) ## test sample size
deigma=matrix(nrow=R,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 seed==TRUE then the results will always be the same
if (seed==TRUE) set.seed(1234567)
for (vim in 1:R) deigma[vim,]=sample(1:n,k)
for (j in 1:length(h)) {
for (i in 1:R) {
mhx=ytest=as.matrix(Y[deigma[i,],])
xtest=as.matrix(X[deigma[i,],])
ytrain=as.matrix(Y[-deigma[i,],])
xtrain=as.matrix(X[-deigma[i,],])
mhx=matrix(nrow=k,ncol=ncol(Y))
for (l in 1:k) mhx[l,]=kern.reg(xtest[l,],ytrain,xtrain,h[j],r,type)
msp[i,j]=mean((ytest-mhx)^2) } }
mspe=colMeans(msp)
names(mspe)=h
plot(h,mspe,type='l',xlab='Bandwidth (h)',ylab='MSPE',
main=paste(r,'order local polynomial',sep=' '))
list(hopt=h[which.min(mspe)], min.mspe=min(mspe),mspe=mspe) }
```

3.2.5 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 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 Jolliffe (2005) 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.

```
pcr=function(y,x,k) {
## y is the univariate dependent variable
## x contains the independent variables
## k shows the number of components to keep
x=as.matrix(x)
y=as.vector(y)
stand=function(x) (x-mean(x))/sd(x)
m=mean(y)
y=y-m ## standardize the dependent variable
x=apply(x,2,stand) ## standardize the independent variables
n=nrow(x) ; p=ncol(x)
eigen=eigen(t(x)%*%x) ## eigen analysis of the design matrix
values=eigen$values ## eigenvalues
per=values/sum(values) ## proportion of each eigenvalue
per2=cumsum(per) ## cumulative proportion of each eigenvalue
vec=eigen$vectors ## eigenvectors, or principal components
z=x%*%vec ## PCA scores
sigma=deviance(lm(y~x))/(n-p-1) ## estimated variance
b=vec[,1:k]%*%solve(t(z[,1:k])%*%z[,1:k])%*%t(z[,1:k])%*%y ## PCA based coefficients
yhat=as.vector( m+x%*%b ) ## fitted values for each PCA model
```

```
mse=sum((y+m-yhat)^2)/(n-k) ## mean squared error of the PCA model
r2=1-(n-1)/(n-k-1)*(1-(cor(y+m,yhat))^2) ## adjusted R squared for the PCA model
va=sigma*vec[,1:k]%*%solve(t(z[,1:k])%*%z[,1:k])%*%t(vec[,1:k]) ## covariance matrix
## of the parameters of the PCA model
vara=sqrt(diag(va)) ## standard errors of the coeffcients of the PCA model
param=cbind(b,vara)
colnames(param)=c('beta','std.error')
list(fitted=yhat,parameters=param,mse=mse,adj.rsq=round(r2,3)) }
```

3.2.6 Choosing the number of components in principal component regression

In the previous Section 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 cross validation.

1. Split the data into two sets , the training (large fraction of the sample) and the test (small fraction of the data).

2. Perform principal component regression analysis using the training set.

3. Estimate the values of the dependent variable using the test set and thus calculate the mean prediction error, the mean of the squared difference between the observed and the estimated values.

4. Repeat steps 1-3 R (say 1000) times and average all mean prediction errors when 1, 2 or all p principal components have been used.

5. Choose the number of principal components with the minimum mean prediction error

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,fraction=0.20,R=1000,seed=FALSE){
## y is the univariate dependent variable
## 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
```

```
x=as.matrix(x)
y=as.vector(y)
stand=function(x) (x-mean(x))/sd(x)
x=apply(x,2,stand) ## standardize the independent variables
n=nrow(x) ; p=ncol(x)
k=round(fraction*n) ## test sample size
deigma=matrix(nrow=R,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
me=mean(y)
y=y-me ## center the dependent variable
crit=matrix(nrow=R,ncol=p)
## if seed==TRUE then the results will always be the same
if (seed==TRUE) set.seed(1234567)
for (vim in 1:R) deigma[vim,]=sample(1:n,k)
for (i in 1:R) {
yhat=ytest=y[deigma[i,]]
xtest=x[deigma[i,],]
ytrain=y[-deigma[i,]]
xtrain=x[-deigma[i,],]
eigen=eigen(t(xtrain)%*%xtrain)
vec=eigen$vectors ## eigenvectors, or principal components
z=xtrain%*%vec ## PCA scores
b=vec
for (m in 1:p){
b[,m]=vec[,1:m]%*%solve(t(z[,1:m])%*%z[,1:m])%*%t(z[,1:m])%*%ytrain
yhat=as.vector( xtest%*%b[,m] )
crit[i,m]=mean((ytest-yhat)^2) } } ## mean squared error of prediction
mspe=colMeans(crit)
names(mspe)=paste('PC',1:p)
plot(mspe,type='b',ylab='MSPE values',
xlab='Number of principal components')
list(mspe=mspe,optimal=which.min(mspe)) }
```

3.2.7 The spatial median and spatial median regression

The so called spatial median, is the vector $\boldsymbol{\delta}$ which minimizes the sum $\sum_{i=1}^{n} \parallel \mathbf{y}_i - \boldsymbol{\delta} \parallel$ (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).

The function below calculates the spatial median.

```
spat.med=function(x){
## contains the data
x=as.matrix(x) ; p=ncol(x) ## dimensionality of the data
medi=function(me,x) sum( sqrt(rowSums((x-me)^2)) ) ## function to calculate the median
## the we use optim to obtain the spatial median
qa=optim(rnorm(p),medi,x=x,control=list(maxit=20000))
qa=optim(qa$par,medi,x=x,control=list(maxit=20000))
qa=optim(qa$par,medi,x=x,control=list(maxit=20000))
median=qa$par
median }
```

If we substitute the spatial median $\boldsymbol{\delta}$ 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

$$\sum_{i=1}^{n} \parallel \mathbf{y}_i - \mathbf{B}\mathbf{x}_i \parallel .$$

```
spatmed.reg=function(y,x){
## y contains the dependent variables
## x contains the independent variable(s)
x=cbind(1,x) ## add the constant term
y=as.matrix(y) ; x=as.matrix(x)
p=ncol(x) ; d=ncol(y) ## dimensionality of x and y
z=list(y=y,x=x)
## medi is the function to peform median regression
medi=function(beta,z) {
y=z$y ; x=z$x
p=ncol(x)
be=matrix(beta,nrow=p)
est=x%*%be
sum( sqrt(rowSums((y-est)^2)) ) }
## we use optim to obtain the beta coefficients
qa=optim(rnorm(p*d),medi,z=z,control=list(maxit=20000))
qa=optim(qa$par,medi,z=z,control=list(maxit=20000))
qa=optim(qa$par,medi,z=z,control=list(maxit=20000))
```

```
beta=matrix(qa$par,ncol=ncol(y))
list(beta=beta,fitted=x%*%beta) }
```

3.2.8 Multivariate 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

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

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 **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 (**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**

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

The next R function performs ridge regression for a given value of λ.

```
ridge.multivreg=function(y,x,lambda) {
## y is the dependent variable and is expected to be a matrix
## if y is a vector then the classical univariate regression
## is performed
## x contains the independent variable(s)
## lambda is the ridge regularization parameter
## if lambda=0, the classical multivariate regression is implemented
```

```
y=as.matrix(y)
x=as.matrix(x)
n=nrow(y) ## sample size
d=ncol(y) ## dimensionality of y
p=ncol(x) ## dimensionality of x
my=matrix(rep(colMeans(y),n),ncol=d,byrow=T)
mx=matrix(rep(colMeans(x),n),ncol=d,byrow=T)
yy=y-my ## center the dependent variables
xx=x-mx ## center the independent variables
beta=solve(t(xx)%*%xx+lambda*diag(p))%*%t(xx)%*%yy ## the parameters
P=diag(n)-xx%*%solve(t(xx)%*%xx)%*%t(xx)
s=(1/(n-p))*t(yy)%*%P%*%yy
S=kronecker(solve(t(xx)%*%xx+lambda*diag(p)),s) ## covariance of the parameters
sigma=t(matrix(sqrt(diag(S)),ncol=p)) ## standard errors of the parameters
fitted=my+xx%*%beta ## fitted values
colnames(fitted)=colnames(y)
colnames(sigma)=colnames(beta)=colnames(y)
rownames(beta)=rownames(sigma)=paste('x', 1:p, sep='')
list(beta=beta,Std.errors=sigma,fitted=fitted) }
```

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 the principal component, the k-nn and the kernel regression implemented before.

```
ridge.tune=function(y,x,lambda=seq(0,5,by=0.1),fraction=0.20,R=1000,seed=FALSE){
## y is the dependent variable and is expected to be a matrix
## if y is a vector then the classical univariate regression
## is performed
## x contains the independent variable(s)
## lambda is the ridge regularization parameter
## if lambda=0, the classical multivariate regression is implemented
## 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 repetitions
y=as.matrix(y)
x=as.matrix(x)
n=nrow(y) ## sample size
k=round(fraction*n) ## test sample size
deigma=matrix(nrow=R,ncol=k)
```

```
crit=matrix(nrow=R,ncol=length(lambda))
## if seed==TRUE then the results will always be the same
if (seed==TRUE) set.seed(1234567)
## deigma will contain the positions of the test set
## this is stored but not showed to the user but can be
## access it though by running the commands outside this function
for (vim in 1:R) deigma[vim,]=sample(1:n,k)
for (i in 1:R) {
ytest=yhat=y[deigma[i,],]
xtest=x[deigma[i,],]
ytrain=y[-deigma[i,],]
xtrain=x[-deigma[i,],]
for (j in 1:length(lambda)) {
mod=ridge.multivreg(ytrain,xtrain,lambda[j])
my=matrix(rep(colMeans(ytest),k),nrow=k,byrow=T)
mx=matrix(rep(colMeans(xtest),k),nrow=k,byrow=T)
est=my+(xtest-mx)%*%mod$beta ## fitted values
crit[i,j]=mean(diag(t(ytest-est)%*%(ytest-est))) } }
mspe=colMeans(crit)
plot(lambda,mspe,type='b',xlab=expression(paste(lambda,' values')),
ylab='Mean squared error of prediction')
list(chosen=lambda[which.min(mspe)],mspe=mspe) }
```

3.3 Discriminant analysis

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

3.3.1 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.

```
fisher=function(z,group) {
## z contains the data
## group denotes the groups
k=max(group) ; n=nrow(z)
d=ncol(z) ; pred=rep(0,n)
for (j in 1:n) {
x=z[-j,] ; ina=group[-j]
xbar=colMeans(x)
S=array(dim=c(ncol(x),ncol(x),k))
B1=array(dim=c(ncol(x),ncol(x),k))
mat=matrix(rep(0,d*k),nrow=d,ncol=k)
for (i in 1:k) {
S[,,i]=nrow(x[ina==i,])*cov(x[ina==i,])
B1[,,i]=nrow(x[ina==i,])*( (colMeans(x[ina==i,])-xbar)%*%t(colMeans(x[ina==i,])-xbar) )
mat[,i]=colMeans(x[ina==i,]) }
W=apply(S,1:2,sum)  ## The within sum of squares
B=apply(B1,1:2,sum)  ## The between sum of squares
M=solve(W)%*%B
lambda=as.vector(eigen(M)$vectors[,1])  ## Fisher's discriminant function
w=matrix(z[j,],d,1)
like=rep(0,k)
for (m in 1:k) {
like[m]=abs(lambda%*%w-lambda%*%mat[,m]) }
pred[j]=which.min(like) }  ## 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. 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.

3.3.2 k-fold 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 k-fold cross validation. Thus it performs k-fold cross validation for linear or discriminant analysis. The user specifies the value of k and then the function removes k values (test sample) at random. It performs

discriminant analysis for the remaining $n - k$ 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 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.

```
kfold.da=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
x=as.matrix(x)
p=numeric(R) ; n=nrow(x)
ina=as.factor(ina)
k=round(fraction*n) ## test sample size
## if seed==TRUE then the results will always be the same
if (seed==TRUE) set.seed(1234567)
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)
g=predict(dok,test)$class
p[i]=sum(diag(table(g,ina[nu])))/k }
if (method=='qda') {
dok=qda(train,id)
g=predict(dok,test)$class
p[i]=sum(diag(table(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 way of a confidence interval
conf2=c(per-1.96*s2,per+1.96*s2) ## 2nd way of a confidence 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 way of a confidence interval
list(percentage=per,conf.int1=conf1,conf.int2=conf2,conf.int3=conf3) }
```

3.3.3 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=ncol(x) ; 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=500,method='qda',seed=TRUE)$percentage }
per[1]=max(est)
id=which.max(est)
z=cbind(z,x[,id])
z1=x[,-id]
## STEP 2
est=numeric(p-1)
for (j in 1:length(est)) {
z2=z1[,j]
est[j]=kfold.da(cbind(z,z2),ina,fraction=0.2,R=500,method='qda',seed=TRUE)$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=500,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) }
```

3.3.4 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,seed=FALSE) {
## x is the matrix with 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
## quad denotes whether lda or qda is to be used
## lambda is the range of values for the Box-Cox transformation
x=as.matrix(x)
B=length(lambda)
percent=numeric(B)
conf1=conf2=conf3=matrix(nrow=B,ncol=2)
n=nrow(x)
k=round(fraction*n) ## test sample size
mat=matrix(nrow=R,ncol=k)
## if seed==TRUE then the results will always be the same
if (seed==TRUE) set.seed(1234567)
for (j in 1:R) mat[j,]=sample(1:n,k) ## choosing random test samples
## for every lambda the same test samples are used
for (i in 1:B) {
## the next two lines are 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)
per=numeric(R)
for (l in 1:R) {
train=y[-mat[l,],] ; test=y[mat[l,],]
```

```
id=ina[-mat[l,]] ; ida=ina[mat[l,]]
if (method=='lda') { ## LDA is to be used
dok=lda(train,id)
g=predict(dok,test)$class
per[l]=sum(diag(table(g,ida)))/k  }
if (method=='qda') { ## QDA is to be used
dok=qda(train,id)
g=predict(dok,test)$class
per[l]=sum(diag(table(g,ida)))/k } }
percent[i]=mean(per) ## mean estimated percentage of correct classification
s1=sd(per) ; s2=sqrt(percent[i]*(1-percent[i])/R)
conf1[i,]=c(percent[i]-1.96*s1,percent[i]+1.96*s1) ## 1st way of a confidence interval
conf2[i,]=c(percent[i]-1.96*s2,percent[i]+1.96*s2) ## 2nd way of a confidence interval
## next we check if the confidence limits exceeds the allowed limits.
if (conf1[i,2]>1) conf1[i,2]=1
if (conf1[i,1]<0) conf1[i,1]=0
if (conf2[i,2]>1) conf2[i,2]=1
if (conf2[i,1]<0) conf2[i,1]=0
conf3[i,]=quantile(per,probs=c(0.025,0.975)) }  ## 3rd way of a confidence interval
names(percent)=lambda
rownames(conf1)=rownames(conf2)=rownames(conf3)=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 way confidence limits also
list(percentage=percent,conf.int1=conf1,conf.int2=conf2,conf.int3=conf3) }
```

3.3.5 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

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

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

```
rda.pred=function(xnew,x,ina,gam=1,del=0) {
## xnew is the new observation
## 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
x=as.matrix(x) ; n=nrow(x) ; D=ncol(x)
xnew=as.matrix(xnew,ncol=D)
```

```
nu=nrow(xnew) ## number of the new observations
ina=as.numeric(ina) ; nc=max(ina)
ng=as.vector(table(ina)/n)
est=numeric(nu)
prob=matrix(nrow=nu,ncol=nc)
Tska=Ska=sk=s=array(dim=c(D,D,nc))
t=matrix(nrow=n,ncol=nc)
ng=rep(0,nc) ; mesos=matrix(nrow=nc,ncol=D)
for (m in 1:nc) {
ng[m]=nrow(x[ina==m,])
s[,,m]=(ng[m]-1)*cov(x[ina==m,])
sk[,,m]=cov(x[ina==m,])
mesos[m,]=colMeans(x[ina==m,]) }
for (i in 1:nu) {
z=as.matrix(xnew[i,],ncol=D)
if (ncol(z)!=D) z=t(z)
Sp=apply(s,1:2,sum)/(sum(ng)-nc)
sp=rep(mean(diag(Sp)),D)
Sa=gam*Sp+(1-gam)*diag(sp)
for (m in 1:nc) {
Ska[,,m]=del*sk[,,m]+(1-del)*Sa
Tska[,,m]=solve(Ska[,,m]) }
for (j in 1:nc) {
t[i,j]=log(ng[j]/sum(ng))-0.5*log(det(2*pi*Ska[,,j]))-
0.5*(z-mesos[j,])%*%Tska[,,j]%*%t(z-mesos[j,]) }
est[i]=which.max(t[i,])
prob[i,]=exp(t[i,])/sum(exp(t[i,])) } ## the probability of classification
list(est.group=est,probability=prob,scores=t) }
```

3.3.6 Tuning the γ and δ parameters in regularised discriminant analysis

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.

```
rda.tune=function(x,ina,fraction=0.2,R=1000,gam=seq(0,1,by=0.1),del=seq(0,1,by=0.1),
seed=FALSE){
## x contains 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
```

```
## gam is between pooled covariance and diagonal
## gam*Spooled+(1-gam)*diagonal
## del is between QDA and LDA
## del*QDa+(1-del)*LDA
x=as.matrix(x)
ina=as.numeric(ina)
n=nrow(x)  ## total sample size
k=round(fraction*n) ## test sample size
mat=matrix(nrow=R,ncol=k)
group=array(dim=c(length(gam),length(del),R))
s1=s2=s3=matrix(nrow=length(gam),ncol=length(del))
## if seed==TRUE then the results will always be the same
if (seed==TRUE) set.seed(1234567)
for (j in 1:R) mat[j,]=sample(1:n,k) ## choosing random test samples
for (vim in 1:R) {
test=x[mat[vim,],]  ## test sample
id=ina[mat[vim,]]   ## groups of test sample
train=x[-mat[vim,],] ## training sample
ida=ina[-mat[vim,]]  ## groups of training sample
for (k1 in 1:length(gam)) {
for (k2 in 1:length(del)) {
g=rda.pred(test,train,ida,gam[k1],del[k2])$est.group
group[k1,k2,vim]=sum(g==id)/k } } }
percent=apply(group,1:2,mean)
su=apply(group,1:2,sd)
dimnames(percent)=dimnames(su)=list(gamma=gam,delta=del)
list(percent=percent,stand.error=su) }
```

3.4 Robust statistical analyses

3.4.1 Robust multivariate regression

Rousseeuw et al. (2004) proposed a robust multivariate regression which is based on robust estimation of the joint location and scatter of the explanatory and response variables. A preprint of their paper is available form ResearchGate. This means that we can also use this function when we have univariate or multivariate dependent and independent variables. We will now assume that both dependent and independent variables are multivariate (the function accepts univariate variables also). The parameters of the joint multivariate normal distribution for $\mathbf{Y}$ (the d-dimensional dependent variable) and $\mathbf{X}$ (the p-dimensional independent

variable only, the first column is NOT the vector of 1_s) are as follows

$$\boldsymbol{\mu} = (\boldsymbol{\mu}_{\mathbf{Y}}, \boldsymbol{\mu}_{\mathbf{X}})^T \text{ and } \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{\mathbf{YY}} & \boldsymbol{\Sigma}_{\mathbf{XY}} \\ \boldsymbol{\Sigma}_{\mathbf{YX}} & \boldsymbol{\Sigma}_{\mathbf{XX}} \end{pmatrix}$$

Rousseeuw et al. (2004) assumes that the linear model is written as $\mathbf{Y} = \boldsymbol{\alpha} + \mathbf{XB} + \mathbf{e}$. Then, the classical least squares estimators for $\mathbf{B}$, $\boldsymbol{\alpha}$ and $\hat{\boldsymbol{\Sigma}}$ are given by

$$\begin{aligned} \hat{\mathbf{B}} &= \hat{\boldsymbol{\Sigma}}_{\mathbf{XX}}^{-1}\hat{\boldsymbol{\Sigma}}_{\mathbf{YX}}, \\ \hat{\boldsymbol{\alpha}} &= \hat{\boldsymbol{\mu}}_{\mathbf{Y}} - \hat{\boldsymbol{\mu}}_{\mathbf{X}}^T\hat{\mathbf{B}} \text{ and} \\ \hat{\boldsymbol{\Sigma}}_{\mathbf{e}} &= \boldsymbol{\Sigma}_{\mathbf{YY}} - \hat{\mathbf{B}}^T\boldsymbol{\Sigma}_{\mathbf{XX}}\hat{\mathbf{B}}. \end{aligned}$$

The same formulas appear in Johnson and Wichern (2002). The only thing we have to do now is to use a formula to calculate robust estimates of the location vector $\boldsymbol{\mu}$ and the scatter matrix $\boldsymbol{\Sigma}$. The answer is the function *cov.rob* from the library *MASS*. It offers two ways, the first one is the *MCD* estimator and the second the *MVE* estimator of the robust scatter and location. Both of these methods require a fraction of the sample, and the optimal fraction is around 50% (it maximises the breakdown point). Note that the function *cov.rob* requires at least 18 observations.

The *MCD* (Minimum Covariance Determinant) estimator tries to find the fraction of the data for which the determinant of their covariance matrix is minimized. So it tries to find a subset whose observations are very close to one another, they are as concentrated as possible. When this subset is identified, its mean vector and covariance matrix are calculated and that's what we need. The *MVE* (Minimum Volume Ellipsoid) on the other hand tries to do something similar. It searches for the subset whose observations form an ellipsoidal object with volume as small as possible.

When we have the robust estimates for the joint mean vector and covariance matrix we can get the robust estimates of $\mathbf{B}$, $\boldsymbol{\alpha}$ and $\hat{\boldsymbol{\Sigma}}$ and also calculate the robust Mahalanobis distances of the independent variables and the residuals (3.2). The function *rob.multivreg* given below does almost the same things as *multivreg* apart from the standard errors. These are not produced.

```
rob.multivreg=function(y,x,method='mcd',quan=0.5){
## y and x are either univariate or multivariate variables
## methods can either be 'mcd' or 'mve'
## quan is the fraction of the data to be used. The optimal value is set to 0.5
## quan might have to change. The subset size for the robust estimates must be at
## least equal to 18.
library(MASS)
y=as.matrix(y)
```

```
x=as.matrix(x)
z=cbind(y,x)
n=nrow(z) ## sample size
d=ncol(y) ## dimensionality of y
p=ncol(x) ## dimensionality of x
robust=cov.rob(z,method=method,quantile.used=floor(quan*n)) ## robust estimates
rob.s=robust$cov
rob.sxx=rob.s[-c(1:d),-c(1:d)] ## robust covariance of the x, Sxx
rob.be=rob.s[1:d,-c(1:d)]%*%solve(rob.sxx) ## estimated betas
rob.my=robust$center[1:d] ## mean vector of y
rob.mx=robust$center[-c(1:d)] ## mean vector of x
rob.b0=rob.my-rob.mx%*%t(rob.be) ## estimated b0
rob.beta=rbind(rob.b0,t(rob.be)) ## robust betas
rob.se=cov(y)-rob.be%*%rob.sxx%*%t(rob.be) ## Error covariance
rob.res=y-cbind(1,x)%*%rob.beta ## residuals
## below are the Mahalanobis distances of the residuals
rob.dres=sqrt(diag(rob.res%*%solve(rob.se)%*%t(rob.res)))
rob.mx=matrix(rep(rob.mx,n),byrow=T)
rob.dx=sqrt(diag((x-rob.mx)%*%solve(rob.sxx)%*%t(x-rob.mx)))
crit.res=sqrt(qchisq(0.975,d))
crit.x=sqrt(qchisq(0.975,p))
plot(rob.dx,rob.dres,xlim=c(0,max(rob.dx)+0.5),ylim=c(0,max(rob.dres)+0.5),
xlab='Robust Mahalanobis distance of x',ylab='Robust Mahalanobis distance of residuals')
abline(h=crit.res)
abline(v=crit.x)
rob.residoutliers=which(rob.dres>crit.res)
rob.xleverage=which(rob.dx>crit.x)
rob.outandlever=which(rob.dx>crit.x & rob.dres>crit.res)
colnames(rob.beta)=colnames(y)
rownames(rob.beta)=c('Intercept',paste('x', 1:p, sep=''))
rob.fitted=cbind(1,x)%*%rob.beta ## robust fitted values
list(beta.rob=rob.beta,rob.residoutliers=rob.residoutliers,
rob.xleverage=rob.xleverage,rob.outandlever=rob.outandlever,rob.fitted=rob.fitted) }
```

3.4.2 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 well.

3.4.3 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 observa-

tions 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){
## z contains the data
## quan is the percentage of the sample size to be used as the initial subset
## as the initial subset
z=as.matrix(z)
n=nrow(z) ## sample size
p=ncol(z) ## dimensionality
arxi=quan*n ## initial subset size
if (arxi< 0.5*p*(p+1)+1 ) arxi=0.5*p*(p+1)+1
z=cbind(1:n,z) ## this will us identify the sequence of entrance
## n the final sample we will see the order of entrance
Xmcd=cov.mcd(z[,-1],5000) ## searches amongst 5000 subsets for the best
dist=mahalanobis(z[,-1], Xmcd$center, Xmcd$cov)
names(dist)=1:n
dist=sort(dist)
b=as.integer(names(dist[1:arxi]))
ini=z[b,] ## initial subset
z3=z[-b,] ##
vim=nrow(z3) ## steps of the FS
dis=numeric(vim)
for (j in 1:c(vim-1)) {
d=numeric(nrow(z3))
for (i in 1:nrow(z3)) {
d[i]=mahalanobis(z3[i,-1],colMeans(ini[,-1]),var(ini[,-1])) }
a=which.min(d)
dis[j]=min(d)
ini=rbind(ini,z3[a,])
```

```
z3=z3[-a,] }
z3=matrix(z3,ncol=length(z3))
ini=rbind(ini,z3)
dis[vim]=mahalanobis(z3[,-1],colMeans(ini[1:(n-1),-1]),var(ini[1:(n-1),-1]))
nama=ini[,1] ; ini=ini[,-1]
plot(dis,type='l')
MD=c(dist[1:arxi],dis)
names(MD)=nama
list(order=nama,MD=MD) }
```

4 Some other multivariate functions

In this section we show some other functions for multivariate data, such as standardization, a simple normality test and some other functions.

4.1 Distributional related functions

4.1.1 Standardization I

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 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 inverse of $\boldsymbol{\Sigma}$ and its square root can be written as

$$\boldsymbol{\Sigma}^{-1} = \mathbf{V}\text{diag}\left(\lambda_1^{-1}, \ldots, \lambda_p^{-1}\right)\mathbf{V}^T \text{ and } \boldsymbol{\Sigma}^{-1/2} = \mathbf{V}\text{diag}\left(\lambda_1^{-1/2}, \ldots, \lambda_p^{-1/2}\right)\mathbf{V}^T \text{ respectively.}$$

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.

```
multivzscore=function(x) {
## x contains the data
x=as.matrix(x)
n=nrow(x) ; s=cov(x) ; p=ncol(x)
m=matrix(rep(colMeans(x),n),byrow=TRUE,ncol=p)
lam=eigen(s)$values
vec=eigen(s)$vectors
```

```
B=vec%*%diag(1/sqrt(lam))%*%t(vec)
z=(x-m)%*%B
z }
```

4.1.2 Standardization II

This standardization is simply centering 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 $i = 1, ..., p$. An alternative robust way is to use the median and the median absolute deviation instead.

```
zscore.1=function(x) (x-mean(x))/sd(x)
zscore.2=function(x) (x-median(x))/(median(abs(x-median(x))))
```

Then the *apply* command will do the rest.

4.1.3 Generating 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 $N(0, 1)$.
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 not be of full rank. The algorithm will still work, since we do not calculate the inverse of a zero eigenvalue. Th8us zero eigenvalues are allowed.

```
rand.mvnorm=function(n,mu,sigma){
## n is the sample size
## mu is the mean vector
## sigma is the covariance matrix
```

```
## sigma does not have to be of full rank
p=length(mu)
x=matrix(rnorm(n*p),ncol=p)
m=matrix(rep(mu,n),byrow=TRUE,ncol=p)
lam=eigen(sigma)$values
vec=eigen(sigma)$vectors
B=vec%*%diag(sqrt(lam))%*%t(vec)
z=x%*%B+m
z }
```

4.1.4 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)
0.5*( sum(diag(solve(s2)%*%s1))+(m2-m1)%*%solve(s2)%*%t(t(m2-m1))-
log(det(s1)/det(s2))-length(m1) ) }
```

4.1.5 Generation of covariance matrices

I have written a simple code to generate covariance matrices based on the Wishart distribution. If $\mathbf{X}_i\ N_p\left(\mathbf{0},\boldsymbol{\Sigma}\right)$, then $\mathbf{A}=\sum_{i=1}^{n}\mathbf{X}_i\mathbf{X}_i^T$ follows a p-variate Wishart distribution with parameters $n\boldsymbol{\Sigma}$ and $n\mathbf{W}\left(\boldsymbol{\Sigma},n\right)$ (Anderson, 2003). The algorithm to generate covariance matrices from a Wishart distribution with expected value equal to $\boldsymbol{\Sigma}$ is

1. Generate say 1000 random values $\mathbf{X}_i$ from a $N_p\left(\mathbf{0},\boldsymbol{\Sigma}\right)$. Note, n must be greater than p. So, if you have more dimensions than 1000, change this number.
2. Store in an array the matrices $\mathbf{X}_i\mathbf{X}_i^T$.
3. Take the average of these 1000 matrices.

The function is a bit (???) slow, so if any reader thinks or knows of a faster way, please send me a message.

```
cov.gen=function(n,Sigma){
p=ncol(Sigma) ## dimension of Sigma
sim=array(dim=c(p,p,n))
for (j in 1:n) {
A=array(dim=c(p,p,1000))
for (i in 1:1000) {
x=mvrnorm(1000,rep(0,p),Sigma) ## generate multivariate normal values
A[,,i]=t(x)%*%x/1000 } ## generate Wishart values and divide by 1000
sim[,,j]=apply(A,1:2,mean) } ## take the average of the Wishart values
sim }
```

4.1.6 Multivariate t distribution

The density of the multivariate t distribution is

$$f_d(\mathbf{y}) = \frac{\Gamma\left(\frac{\nu+d}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)|\pi\nu\Sigma|^{1/2}\left[1+\frac{1}{\nu}(\mathbf{y}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{y}-\boldsymbol{\mu})\right]^{\frac{\nu+d}{2}}}, \tag{4.1}$$

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

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

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 mean 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.

```
multivt=function(y) {
## the next mvt function is for the appropriate
## degrees of freedom
## y contains the data
mvt=function(y,v) {
a=cov.trob(y,nu=v)
se=a$cov ; n=nrow(y) ; p=ncol(y)
```

```
me=as.vector(a$center)
me=matrix(rep(me,n),byrow=T,ncol=p)
f= n*lgamma((v+p)/2)-n*lgamma(v/2)-0.5*n*p*log(pi*v)-0.5*n*log(det(se))-
0.5*(v+p)*sum(log(diag(1+(y-me)%*%solve(se)%*%t(y-me)/v)))
f }
b=optimize(mvt,c(0.9,20000),y=y,maximum=T)
df=b$maximum ; loglik=b$objective
## df is the optimal degrees of freedom
result=cov.trob(y,nu=df) ## the center and covariance matrix
## will be calculated based on the optimal degrees of freedom
list(center=result$center,covariance=result$cov,degrees.of.freedom=df,log.lik=loglik) }
```

4.1.7 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

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

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

```
rand.mvt=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(rnorm(n*p),ncol=p)
w=sqrt(v/rchisq(n,v))
m=matrix(rep(mu,n),byrow=TRUE,ncol=p)
lam=eigen(sigma)$values
vec=eigen(sigma)$vectors
B=vec%*%diag(sqrt(lam))%*%t(vec)
z=w*x%*%B+m
z }
```

4.1.8 Contour plot of the bivariate normal, t and skew normal distribution

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. If we wish to use the skew normal distribution we will have to use the R package *sn* written by Azzalini (2011). 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.

```
den.contours=function(x,type='normal') {
## x is a bivariate dataset
## type can be either 'normal', 't' or 'skewnorm'
x=as.matrix(x)
## the user must make sure he/she has bivariate data. If the data are not bivariate
## the function will not work
## the default distibution in normal, but there are other options, such as
## t and skew normal
m=colMeans(x) ## mean vector
s=cov(x) ## covariance matrix
n1=100
n2=100 ## n1 and n2 specify the number of points taken at each axis
## if for example the y axis is longer than the x axis, then you might
## want to change n2.
sa=solve(s) ## inverse of the covariance matrix
con=1/sqrt(det(2*pi*s))  ## normalizng constant of the multivariate normal
x1=seq(min(x[,1])-1,max(x[,1])+1,length=n1)
x2=seq(min(x[,2])-1,max(x[,2])+1,length=n2)
mat=matrix(nrow=n1,ncol=n2)
for (i in 1:n1) {
for (j in 1:n2) {
can=con*exp(-0.5*(c(x1[i]-m[1],x2[j]-m[2])%*%sa%*%c(x1[i]-m[1],x2[j]-m[2])))
if (abs(can)<Inf) mat[i,j]=can else mat[i,j]=NA } }
## 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])
param=list(mesos=colMeans(x),covariance=var(x))
if (type=='t') {
x=as.matrix(x)
## 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$degrees.of.freedom
st=solve(s)
x1=seq(min(x[,1])-1,max(x[,1])+1,length=n1)
x2=seq(min(x[,2])-1,max(x[,2])+1,length=n2)
mat=matrix(nrow=n1,ncol=n2)
for (i in 1:n1) {
for (j in 1:n2) {
ca=lgamma((v+2)/2)-lgamma(v/2)-0.5*log(det(pi*v*s))-
0.5*(v+2)*(log( 1+(c(x1[i]-m[1],x2[j]-m[2])%*%st%*%c(x1[i]-m[1],x2[j]-m[2]))/v  ))
can=exp(ca)
if (abs(can)<Inf) mat[i,j]=can else mat[i,j]=NA  } }
## we did this to avoid any issues with high numbers
contour(x1,x2,mat,nlevels=10,col=2,xlab=colnames(x)[1],ylab=colnames(x)[2])
points(x[,1],x[,2])
param=list(center=m,scatter=s,df=v) }
if (type=='skewnorm') {
x=as.matrix(x)
library(sn)
para=msn.mle(y=x)$dp
x1=seq(min(x[,1])-1,max(x[,1])+1,length=n1)
x2=seq(min(x[,2])-1,max(x[,2])+1,length=n2)
mat=matrix(nrow=n1,ncol=n2)
for (i in 1:n1) {
for (j in 1:n2) {
y=c(x1[i],x2[j])
can=dmsn(y,dp=para)
if (abs(can)<Inf) mat[i,j]=can else mat[i,j]=NA   } }
contour(x1,x2,mat,nlevels=10,col=2,xlab=colnames(x)[1],ylab=colnames(x)[2])
points(x[,1],x[,2])
param=para }
param }
```

4.2 Matrix related functions

4.2.1 Choosing the number of principal components using SVD

We will start by explaining what is 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 (Eigenvalues and Eigenvectors, what Principal Component Analysis does). Suppose we have a $n \times p$ matrix $\mathbf{X}$. Then using SVD we can write the matrix as

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T, \tag{4.2}$$

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 $\mathbf{X}^T\mathbf{X}$). We remind that the maximum rank of an $n \times p$ matrix is equal to $\min\{n, p\}$. Using (4.2), 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 (4.2). 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 Jolliffe, 2005). The code in R is given below

```
choose.pc=function(x) {
## x contains the data
center=function(x) x-mean(x)
x=apply(x,2,center) ## center the matrix
A=svd(x) ## SVD of the centred matrix
u=A$u ; d=A$d ; v=A$v ; p=length(d)
press=rep(0,p)
for (i in 1:p) {
y=x
for (j in 1:ncol(x)) {
z=as.matrix(x[,1:i])
for (k in 1:i) z[,k]=u[,k]*d[k]*t(v[j,k]) ## reconstruction using m eigenvectors
y[,j]=rowSums(z) }
press[i]=sqrt(sum((y-x)^2)) } ## calculation of the PRESS values
plot(press,type='b',xlab='Number of components',ylab='Error')
list(press=press) }
```

4.2.2 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

$$\alpha = \left(\lambda_1^2+...+\lambda_k^2\right) / \left(\lambda_1^2+...+\lambda_p^2\right)$$

and

$$tr\boldsymbol{\Sigma}^2 = \lambda_1^2 + ... + \lambda_p^2$$

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=999) {
## 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
x=as.matrix(x)
n=nrow(x) ; p=ncol(x)
lam=eigen(cov(x))$values  ## eigenvalues of the covariance matrix
psi=sum(lam[1:k])/sum(lam)  ## percentage retained by the first k components
if (B==1) {
trasu=sum(lam)
trasu2=sum(lam^2)
alpha=sum( (lam^2)[1:k] )/trasu2
t2=( (2*trasu2)*(psi^2-2*alpha*psi+alpha) )/( (n-1)*(trasu^2) )
low=psi-qnorm(1-a/2)*sqrt(t2)
up=psi+qnorm(1-a/2)*sqrt(t2)
result=list(psi=psi,lower.limit=low,upper.limit=up) }
if (B>1) { ## bootstrap version
t=rep(0,B)
for (i in 1:B) {
b=sample(1:n,n,replace=TRUE)
lam=eigen(cov(x[b,]))$values
t[i]=sum(lam[1:k])/sum(lam) }
t=sort(t)
low1=psi-qnorm(1-a/2)*sd(t)
up1=psi+qnorm(1-a/2)*sd(t)
quan=quantile(t,probs=c(a/2,1-a/2))
hist(t)
abline(v=psi,lty=2,lwd=2)
abline(v=mean(t),lty=1,lwd=3)
```

```
legend(low1,B/10,cex=0.8,c("psi","bootstrap psi"),lty=c(2,1),lwd=c(2,3))
result=list(psi=psi,psi.boot=mean(t),est.bias=mean(t)-psi,normal.low=low1,
normal.up=up1,percetile.low=quan[1],percentile.up=quan[2]) }
result }
```

4.2.3 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 $\mathbf{1_D^T}$, 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

$$\mathbf{H} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & \dots & \dots & 0 \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & -\frac{2}{\sqrt{6}} & 0 & \dots & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{1}{\sqrt{i(i+1)}} & \dots & \frac{1}{\sqrt{i(i+1)}} & -\frac{i}{\sqrt{i(i+1)}} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{dD}} & \dots & \dots & \dots & \dots & \frac{1}{\sqrt{dD}} & -\frac{dD}{\sqrt{dD}} \end{pmatrix} \tag{4.3}$$

The R-code for the Helmert sub-matrix is

```
helm=function(n) {
h=matrix(rep(0,n^2),nrow=n,ncol=n)
h[1,]=1/sqrt(n)
for (i in 2:n) {
for (j in 1:i-1)
h[i,j]=1/sqrt(i*(i-1))
h[i,j+1]=-sum(h[i,]) }
h=h[c(2:n),]
h }
```

What we have to do now is go to 4.4 and instead of F put the Helmert matrix

```
ginv2=function(A) {
d=ncol(A)
F=helm(d)
inv=t(F)%*%solve(F%*%A%*%t(F))%*%F
inv }
```

We can compare the results from these two methods with the results that are produced from the package corpcor (Schaefer et al., 2007). The disadvantage of these two alternative ways is that they require the rank of the square matrix to be equal to its dimensions minus 1 and as the dimensions grow large they will not work properly.

4.2.4 A pseudoinverse matrix

We will give a very simple way to evaluate a pseudoinverse 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.

$$\mathbf{F} = [\mathbf{I}_d : -\mathbf{j}_d]. \tag{4.4}$$

This is simply the identity matrix with one extra column to the right with all elements equal to -1. Then the pseudoinverse Γ^- is equal to:

$$\Gamma^- = \mathbf{F}^T \left(\mathbf{F}\mathbf{\Gamma}\mathbf{F}^T \right)^{-1} \mathbf{F}$$

```
ginv1=function(A) {
d=ncol(A)-1
F=cbind( matrix(diag(d),ncol=d),matrix(rep(-1,d),ncol=1) )
inv=t(F)%*%solve(F%*%A%*%t(F))%*%F
inv }
```

4.2.5 Exponential of a symmetric 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

$$e^{\mathbf{A}} = \mathbf{V}\text{diag}\left(e^{\lambda_1}, \ldots, e^{\lambda_p} \right) \mathbf{V}^{-1},$$

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). The R code is given below

```
expm=function(A){
## A has to be a symmetric matrix
## the next function checks that A is symmetric
if (all(t(A)-A!=0))
expA=paste('A is not symmetric')
if (all(t(A)-A==0)) {
a=eigen(A)
expA=a$vectors%*%diag(exp(a$values))%*%t(a$vectors) }
expA }
```

5 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).

In mathematical terms, we can define the relevant sample space as

$$\mathbb{S}^d = \left\{ (x_1, ..., x_D) | x_i \geq 0, \sum_{i=1}^{D} x_i = 1 \right\}, \tag{5.1}$$

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.

5.1 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:

$$P = \begin{bmatrix} 0 & 1 & 0.5 \\ 0 & 0 & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{5.2}$$

The columns of (5.2) 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 (5.2) is given by

$$\mathbf{y} = (y_1, y_2) = \left(x_2 + \frac{x_3}{2}, \frac{x_3\sqrt{3}}{2} \right) \tag{5.3}$$

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

$$\boldsymbol{\mu}_0 = \left(\frac{g_1}{g_1 + \ldots + g_D}, \ldots, \frac{g_D}{g_1 + \ldots + g_D} \right), \tag{5.4}$$

where

$$g_i = \prod_{j=1}^{n} x_{ij}^{1/n},\ i = 1, \ldots, D.$$

The simple arithmetic mean is defined as

$$\boldsymbol{\mu}_1 = \left(\frac{1}{n}\sum_{j=1}^{n} x_{1j}, \ldots, \frac{1}{n}\sum_{j=1}^{n} x_{Dj} \right) \tag{5.5}$$

```
ternary=function(x,means=TRUE){
## x contains the data
x=as.matrix(x,ncol=3)
x=x/rowSums(x)
nam=colnames(x)
n=nrow(x) ; ina=rep(1,n)
## 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=colMeans(x)
x=rbind(x,m1,m2)
for (i in 1:n) { ## the next for function checks for zeros.
if (x[i,1]==0 | x[i,2]==0 | x[i,3]==0) ina[i]=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==TRUE) { ## should the mean 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') }
title(main=NULL)
list(closed.geometric=m1,arithmetic=m2) }
```

5.2 The spatial median for compositional data

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. We saw this function in Section 3.2.7. The only addition now is the additive log-ratio transformation used in compositional data.

```
comp.spatmed=function(x){
## x contains the data
x=as.matrix(x)
D=ncol(x) ## dimensionality of the data
y=log(x[,-D]/x[,D]) ## the additive log-ratio transformation
delta=spat.med(y)
exp(delta)/(1+exp(delta)) }
```

5.3 The Dirichlet distribution

The Dirichlet distribution is a distribution whose support is the simplex (5.1). The density of the Dirichlet distribution is the following

$$f\left(x_1,\ldots,x_D;\alpha_1,\ldots,\alpha_D\right)=\frac{1}{\mathbf{B}\left(\boldsymbol{\alpha}\right)}\prod_{i=1}^{D}x_i^{\alpha_i-1} \tag{5.6}$$

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 parameter of the Dirichlet distribution.

5.3.1 Estimating the parameters of the Dirichlet via the log-likelihood

The log-likelihood of the Dirichlet has the following form:

$$l = n\log\Gamma\left(\sum_{i=1}^{D}\alpha_i\right) - n\sum_{i=1}^{D}\log\Gamma(\alpha_i) + \sum_{j=1}^{n}\sum_{i=1}^{D}(\alpha_i - 1)\log x_{ij}$$

The function in R is the following:

```
diri.mle=function(x) {
## x is the compositional data
loglik=function(param,x=x) {
n=nrow(x) ## the sample size
x=x/rowSums(x)
-( n*lgamma(sum(param))-n*sum(lgamma(param))+sum(log(x)%*%(param-1)) ) }
da=optim(runif(ncol(x),0,20),loglik,x=x,control=list(maxit=2000))
da=optim(da$par,loglik,x=x,control=list(maxit=2000))
da=optim(da$par,loglik,x=x,control=list(maxit=2000),hessian=T)
list(param=da$par,std=sqrt(diag(solve(da$hessian))),loglik=-da$value) }
```

Thus after generating values from a Dirichlet distribution, we can use the "optim" function to maximize the log-likelihood. The argument "hessian=T" 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 form of the Dirichlet density is via the parameter ϕ:

$$f(\mathbf{x}) = \frac{\Gamma\left(\sum_{i=1}^{D}\phi a_i^*\right)}{\prod_{i=1}^{D}\Gamma\left(\phi a_i^*\right)}\prod_{i=1}^{D}x_i^{\phi a_i^* - 1}, \tag{5.7}$$

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

$$\ell = n \log \Gamma(\phi) - \sum_{j=1}^{n} \sum_{i=1}^{D} \log \Gamma(\phi a_i^*) + \sum_{j=1}^{n} \sum_{i=1}^{D} (\phi a_i^* - 1) \log x_{ij}, \tag{5.8}$$

The function for this parametrization (no covariates here) is given below

```
diriphi.mle=function(x) {
## x is the compositional data
diri=function(param,x=x) {
x=x/rowSums(x) ## makes sure the data are compositional
n=nrow(x) ## the sample size
phi=param[1] ; b=c(1-sum(param[-1]),param[-1])
if ( all(b>0) & all(b<1) & phi>0) {
f=-( n*lgamma(phi)-n*sum(lgamma(phi*b))+sum(log(x)%*%(phi*b-1)) )  }
else f=100000
f }
da=optim(c(10,colMeans(x)[-1]),diri,x=x,control=list(maxit=2000))
da=optim(da$par,diri,x=x,control=list(maxit=2000))
da=optim(da$par,diri,x=x,control=list(maxit=2000),hessian=T)
phi=da$par[1] ; a=da$par[-1]
list(phi=phi,a=a,std=sqrt(diag(solve(da$hessian))),b=phi*c(1-sum(a),a),
loglik=-da$value) }
```

5.3.2 Estimating the parameters of the Dirichlet distribution through entropy

We will make use of the following relationship

$$E[\log X_i] = \psi(\alpha_i) - \psi(\alpha_0), \tag{5.9}$$

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 5.9. 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).

```
diri.ent=function(x){
## x is the compositional data
library(BB)
x=as.matrix(x)
x=x/rowSums(x) ## makes sure x is compositional data
n=nrow(x) ## sample size
sir=function(param) {
f=rep(0,length(param))
ma=colMeans(log(x))
for (i in 1:length(f)) {
f[i]=ma[i]-digamma(param[i])+digamma(sum(param)) }
f }
da=BBsolve(runif(ncol(x),0,20),sir,control=list(maxit=2000,tol=1e-10))
param=da$par
lik=n*lgamma(sum(param))-n*sum(lgamma(param))+sum(log(x)%*%(param-1))
list(param=param,loglik=lik) }
```

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).

5.3.3 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=nrow(x) ## the sample size
D=ncol(x) ## the dimensionality of the data
loglik=function(param,x)-( n*lgamma(sum(param))-n*sum(lgamma(param))+
sum(log(x)%*%(param-1)) )
sym=function(a,x) n*lgamma(D*a)-n*D*lgamma(a)+sum(log(x)*(a-1))
t0=optimize(sym,c(0,1000),x=x,maximum=TRUE)
t1=optim(colMeans(x)*10,loglik,x=x,control=list(maxit=2000))
t1=optim(t1$par,loglik,x=x,control=list(maxit=2000))
t1=optim(t1$par,loglik,x=x,control=list(maxit=2000))
a1=t1$par ; a0=t0$maximum
h1=-as.numeric(t1$value) ; h0=as.numeric(t0$objective)
test=2*(h1-h0)
p.value=1-pchisq(test,D-1)
list(estimated.parameters=a1,one.parameter=a0,log.lik1=h1,log.lik0=h0,
```

```
df=D-1,test=test,p.value=p.value) }
```

5.3.4 Kullback-Leibler divergence 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 from here (Dirichlet KL-divergence). It is a technical report written Daniel Beale from the university of Bath. This divergence is equal to

$$KL\left(D_1\left(a\right) \parallel D_2\left(b\right)\right) = \sum_{i=1}^{D}\left(a_i - b_i\right)\left[\Psi\left(a_i\right) - \Psi\left(a_0\right)\right] + \sum_{i=1}^{D}\log\frac{\Gamma\left(b_i\right)}{\Gamma\left(a_i\right)} + \log\frac{\Gamma\left(a_0\right)}{\Gamma\left(b_0\right)},$$

where $a_0 = \sum_{i=1}^{D} a_i$, $b_0 = \sum_{i=1}^{D} b_i$ and $\Psi\left(.\right)$ is the digamma function.

```
KL=function(a,b) {
## KL-Divergence between Dir(a) and Dir(b)
a0=sum(a) ; b0=sum(b)
f=sum( (a-b)*(digamma(a)-digamma(a0)) )+sum(lgamma(b)-lgamma(a))+
lgamma(a0)-lgamma(b0)
f }
```

5.3.5 Bhattacharyya distance between two Dirichlet distributions

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

$$\begin{aligned} J_B\left(D_1\left(a\right), D_2\left(b\right)\right) &= \log\Gamma\left(\sum_{i=1}^{D}\frac{a_i+b_i}{2}\right) + \frac{1}{2}\sum_{i=1}^{D}\left[\log\Gamma\left(a_i\right) + \log\Gamma\left(b_i\right)\right] \\ &- \sum_{i=1}^{D}\log\Gamma\left(\frac{a_i+b_i}{2}\right) - \frac{1}{2}\left[\log\Gamma\left(\sum_{i=1}^{D}a_i\right) + \log\Gamma\left(\sum_{i=1}^{D}b_i\right)\right] \end{aligned} \tag{5.10}$$

The code to calculate (5.10) is given below

```
Bhatt=function(a,b) {
## Bhattacharyya distance between Dir(a) and Dir(b)
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 }
```

5.4 Contour plot of distributions on $\mathbb{S}^2$

In section 5.1 we showed how construct a ternary plot by making use of a matrix (5.2). 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 4.1.8.

5.4.1 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 (5.1)). If it lies, then it comes from a composition. To find the composition we need to work out the opposite of (5.3). The coordinates of a compositional vector in $\mathbb{R}^2$ taken from (5.3) 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

$$\left\{ \begin{array}{rl} x_3 = & \frac{2y_2}{\sqrt{3}} \\ x_2 = & y_1 - \frac{y_2}{\sqrt{3}} \\ x_1 = & 1 - x_2 - x_3 \end{array} \right\}$$

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 5.1 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 (5.2). 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 (5.6). 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. The code is given below.

```
diri.contour=function(a,n=100) {
```

```
## the a is the vector of parameters
## n is the number of points of each axis used
x1=seq(0.001,0.999,length=n) # coordinats of x
x2=seq(0.001,sqrt(3)/2-0.0001,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] < sqrt(3)*x1[i]) { ## This checks if the point lies 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])/sqrt(3)
w2=x1[i]-x2[j]/sqrt(3)
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)       {
if (x2[j]<sqrt(3)-sqrt(3)*x1[i]) { ## This checks if the point lies 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])/sqrt(3)
w2=x1[i]-x2[j]/sqrt(3)
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 plot
b1=c(1/2,0,1,1/2)
b2=c(sqrt(3)/2,0,0,sqrt(3)/2)
b=cbind(b1,b2)
## the next line draws the triangle in the two dimensions
points(b[,1],b[,2],type="l",xlab=" ",ylab=" ") }
```

5.4.2 Log-ratio transformations

The Dirichlet distribution (5.6) 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. Prior to the codes for the contour plot, 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

$$\mathbf{y} = \left(\log \frac{x_1}{x_D}, \ldots, \log \frac{x_d}{x_D} \right), \tag{5.11}$$

where x_D indicates the last component (any other component can play the role of the common divisor). Another log-ratio transformation also suggested by Aitchison (2003) was the centred log-ratio transformation

$$\mathbf{y} = \left(\log \frac{x_1}{g(\mathbf{x})}, \ldots, \log \frac{x_d}{g(\mathbf{x})} \right), \text{ and } y_D = -\sum_{k=1}^{d} y_k, \text{ where} \tag{5.12}$$

where $g(\mathbf{x}) = \prod_{j=1}^{D} x_j^{1/D}$ is the geometric mean of the compositional vector. The additive log-ratio transformation maps the data from $\mathbb{S}^d$ to $\mathbb{R}^d$, in contrast to the centred log-ratio transformation 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 by the Helmert sub-matrix (4.3) the result is the isometric log-ratio transformation (Egozcue et al., 2003) which maps the data from $\mathbb{Q}^d$ onto $\mathbb{R}^d$.

$$\mathbf{z} = \mathbf{H}\mathbf{y} \tag{5.13}$$

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 (5.12) and (5.13) 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 (5.11) or the isometric log-

ratio transformation (5.13) and in the transformed data fit a multivariate distribution defined in $\mathbb{R}^d$.

5.4.3 Contour plot of the normal distribution in $\mathbb{S}^2$

The density of the multivariate normal is

$$f(\mathbf{y}) = \frac{e^{-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{y}-\boldsymbol{\mu})}}{|2\pi\boldsymbol{\Sigma}|^{1/2}} \tag{5.14}$$

We will repeat Section 5.4.1 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 (5.14).

```
norm.contour=function(m,s,type='alr',n=100) {
## m is the mean vector of the normal
## s is the covariance matrix
## the iso parameter determines whether the additive or the isometric
## log-ratio transformation will be used. If iso='alr' (the default) the additive
## log-ratio transformation is used. If iso='ilr', the isometric log-ratio is used
## n is the number of points of each axis used
x1=seq(0.001,0.999,length=n)
x2=seq(0.001,sqrt(3)/2-0.0001,length=n)
mat=matrix(nrow=n,ncol=n)
down=((2*pi)^(-2))*(det(s)^(-0.5))
st=solve(s)
for (i in 1:c(n/2) ) {
for (j in 1:n) {
if (x2[j] < sqrt(3)*x1[i]) { ## This checks if the point lies inside the triangle
## The next 4 lines calculate the composition
w3=(2*x2[j])/sqrt(3)
w2=x1[i]-x2[j]/sqrt(3)
w1=1-w2-w3
w=c(w1,w2,w3)
if (type=='alr') y=log(w[-3]/w[3]) ## additive log-ratio transformation
if (type=='ilr') {
y=log(w)-mean(log(w))
y=as.vector( y%*%t(helm(3)) ) }  ## isometric log-ratio transformation
can=down*exp(-0.5*(c(y[1]-m[1],y[2]-m[2])%*%st%*%c(y[1]-m[1],y[2]-m[2])))
if (abs(can)<Inf) mat[i,j]=can else mat[i,j]=NA }  } }
```

```
for (i in c(n/2+1):n) {
for (j in  1:n)       {
if (x2[j]<sqrt(3)-sqrt(3)*x1[i]) { ## This checks if the point lies inside the triangle
## The next 4 lines calculate the composition
w3=(2*x2[j])/sqrt(3)
w2=x1[i]-x2[j]/sqrt(3)
w1=1-w2-w3
w=c(w1,w2,w3)
if (type=='alr') y=log(w[-3]/w[3]) ## additive log-ratio transformation
if (type=='ilr') {
y=log(w)-mean(log(w))
y=as.vector( y%*%t(helm(3)) ) }  ## isometric log-ratio transformation
can=down*exp(-0.5*(c(y[1]-m[1],y[2]-m[2])%*%st%*%c(y[1]-m[1],y[2]-m[2])))
if (abs(can)<Inf) mat[i,j]=can else mat[i,j]=NA }  } }
contour(x1,x2,mat,col=3)
b1=c(1/2,0,1,1/2)
b2=c(sqrt(3)/2,0,0,sqrt(3)/2)
b=cbind(b1,b2)
points(b[,1],b[,2],type="l",xlab=" ",ylab=" ") }
```

5.4.4 Contour plot of the multivariate t distribution in $\mathbb{S}^2$

The density of the multivariate t distribution is given in (4.1). After applying the additive log-ratio (5.11) or the isometric log-ratio transformation (5.13) to the compositional data we can estimate the parameters of the multivariate t distribution via numerical optimization. In Section 4.1.6 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.

```
t.contour=function(v,m,s,iso='alr',n=100) {
## v is the degrees of freedom
## m is the location parameter
## s is the scatter parameter
## the iso parameter determines whether the additive or the isometric
## log-ratio transformation will be used. If iso='alr' (the default) the additive
## log-ratio transformation is used. If iso='ilr', the isometric log-ratio is used
## n is the number of points of each axis used
x1=seq(0.001,0.999,length=n)
x2=seq(0.001,sqrt(3)/2-0.0001,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] < sqrt(3)*x1[i]) { ## This checks if the point lies inside the triangle
## The next 4 lines calculate the composition
w3=(2*x2[j])/sqrt(3)
w2=x1[i]-x2[j]/sqrt(3)
w1=1-w2-w3
w=c(w1,w2,w3)
if (type=='alr') y=log(w[-3]/w[3]) ## additive log-ratio transformation
if (type=='ilr') {
y=log(w)-mean(log(w))
y=as.vector( y%*%t(helm(3)) ) }  ## isometric log-ratio transformation
ca=lgamma((v+p)/2)-lgamma(v/2)-0.5*log(det(pi*v*s))-
0.5*(v+p)*(log( 1+(c(y[1]-m[1],y[2]-m[2])%*%st%*%c(y[1]-m[1],y[2]-m[2]))/v  ))
can=exp(ca)
if (abs(can)<Inf) mat[i,j]=can else mat[i,j]=NA } } }
for (i in c(n/2+1):n) {
for (j in  1:n)       {
if (x2[j]<sqrt(3)-sqrt(3)*x1[i]) { ## This checks if the point lies inside the triangle
## The next 4 lines calculate the composition
w3=(2*x2[j])/sqrt(3)
w2=x1[i]-x2[j]/sqrt(3)
w1=1-w2-w3
w=c(w1,w2,w3)
if (type=='alr') y=log(w[-3]/w[3]) ## additive log-ratio transformation
if (type=='ilr') {
y=log(w)-mean(log(w))
y=as.vector( y%*%t(helm(3)) ) }  ## isometric log-ratio transformation
ca=lgamma((v+p)/2)-lgamma(v/2)-0.5*log(det(pi*v*s))-
0.5*(v+p)*(log( 1+(c(y[1]-m[1],y[2]-m[2])%*%st%*%c(y[1]-m[1],y[2]-m[2]))/v  ))
can=exp(ca)
if (abs(can)<Inf) mat[i,j]=can else mat[i,j]=NA }  } }
contour(x1,x2,mat,col=3)
b1=c(1/2,0,1,1/2)
b2=c(sqrt(3)/2,0,0,sqrt(3)/2)
b=cbind(b1,b2)
points(b[,1],b[,2],type="l",xlab=" ",ylab=" ") }
```

5.4.5 Contour plot of the skew-normal distribution in $\mathbb{S}^2$

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

$$f_d(\mathbf{y}) = \frac{2}{\left|2\pi\boldsymbol{\Omega}^{1/2}\right|} e^{-\frac{1}{2}(\mathbf{y}-\boldsymbol{\xi})\boldsymbol{\Omega}^{-1}(\mathbf{y}-\boldsymbol{\xi})^T} \Phi\left[\boldsymbol{\theta}^T\boldsymbol{\omega}^{-1}(\mathbf{y}-\boldsymbol{\xi})\right], \tag{5.15}$$

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

$$\boldsymbol{\theta}^T = \left(\frac{\delta_1}{\sqrt{1-\delta_1^2}}, \ldots, \frac{\delta_p}{\sqrt{1-\delta_d^2}}\right)^T.$$

The vector $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_d)^T$ contains the skewness related parameters of the variables and each $\delta_i \in (-1, 1)$, whereas each θ_i spans over all $\mathbf{R}$. If $\theta = \mathbf{0}$, 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 (5.15), 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.

In order to fit the skew-normal distribution (5.15) to a compositional dataset we first apply either the additive log-ratio (5.11) or the isometric log-ratio transformation (5.13). Using the transformed data we need to estimate the parameters of the skew-normal distribution. Azzalini (2011) has created an R package, called *sn* which fits the skew-normal distribution.

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

$$E(\mathbf{y}) = \boldsymbol{\xi} - (2/\pi)^{1/2}\boldsymbol{\delta} \text{ and } Var(\mathbf{y}) = \boldsymbol{\Omega} - \frac{2}{\pi}\boldsymbol{\delta}\boldsymbol{\delta}^T.$$

The code to produce a contour plot for the bivariate skew-normal distribution on the simplex is given below.

```
skewnorm.contour=function(ksi,omega,alpha,iso='alr',n=100) {
## ksi is the parameter ksi
## omega is the omega parameter
## alpha is the alpha parameter
## the iso parameter determines whether the additive or the isometric
## log-ratio transformation will be used. If iso='alr' (the default) the additive
```

```
## log-ratio transformation is used. If iso='ilr', the isometric log-ratio is used
## n is the number of points of each axis used
x1=seq(0.001,0.999,length=n)
x2=seq(0.001,sqrt(3)/2-0.0001,length=n)
mat=matrix(nrow=n,ncol=n)
for (i in 1:c(n/2) ) {
for (j in 1:n) {
if (x2[j] < sqrt(3)*x1[i]) { ## This checks if the point lies inside the triangle
## The next 4 lines calculate the composition
w3=(2*x2[j])/sqrt(3)
w2=x1[i]-x2[j]/sqrt(3)
w1=1-w2-w3
w=c(w1,w2,w3)
if (type=='alr') y=log(w[-3]/w[3]) ## additive log-ratio transformation
if (type=='ilr') {
y=log(w)-mean(log(w))
y=as.vector( y%*%t(helm(3)) ) }  ## isometric log-ratio transformation
can=dmsn(y,xi=ksi,Omega=omega,alpha=alpha)
if (abs(can)<Inf) mat[i,j]=can else mat[i,j]=NA }  } }
for (i in c(n/2+1):n) {
for (j in  1:n)       {
if (x2[j]<sqrt(3)-sqrt(3)*x1[i]) { ## This if whether the point lies inside the triangle
## The next 4 lines calculate the composition
w3=(2*x2[j])/sqrt(3)
w2=x1[i]-x2[j]/sqrt(3)
w1=1-w2-w3
w=c(w1,w2,w3)
if (type=='alr') y=log(w[-3]/w[3]) ## additive log-ratio transformation
if (type=='ilr') {
y=log(w)-mean(log(w))
y=as.vector( y%*%t(helm(3)) ) }  ## isometric log-ratio transformation
can=can=dmsn(y,xi=ksi,Omega=omega,alpha=alpha)
if (abs(can)<Inf) mat[i,j]=can else mat[i,j]=NA }  } }
contour(x1,x2,mat,col=3)
b1=c(1/2,0,1,1/2)
b2=c(sqrt(3)/2,0,0,sqrt(3)/2)
b=cbind(b1,b2)
points(b[,1],b[,2],type="l",xlab=" ",ylab=" ") }
```

5.5 Regression for compositional data

5.5.1 Regression using the additive log-ratio transformation

The additive log-ratio transformation (5.11) will be used for the implementation of regression for compositional data. we could of course use the isometric log-ratio transformation (5.13) 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 3.2 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.

$$\log\left(\frac{y_i}{y_D}\right) = \mathbf{x}^T\boldsymbol{\beta}_i \Leftrightarrow \log y_i = \log y_D + \mathbf{x}^T\boldsymbol{\beta}_i, \;\; i = 1, \ldots, d \tag{5.16}$$

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 (5.16) 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.

```
comp.reg=function(y,x,rob=FALSE){
## y is dependent variable, the compositional data
## x is the independent variable(s)
y=as.matrix(y)
y=y/rowSums(y) ## makes sure y is compositional data
x=as.matrix(x)
z=log(y[,-1]/y[,1]) ## alr transformation with the first component being the base
mod=multivreg(z,x) ## multivariate regression
beta=mod$beta
std.errors=mod$Std.errors
est1=mod$fitted
est2=cbind(1,exp(est1))
est=est2/rowSums(est2)
if (rob==TRUE) {
mod=rob.multivreg(z,x,method='mcd',quan=0.5)
beta=mod$beta.rob
std.errors=NULL
est1=mod$rob.fitted
est2=cbind(1,exp(est1))
est=est2/rowSums(est2) }
list(beta=beta,std.errors=std.errors,fitted=est) }
```

5.5.2 Dirichlet regression

An alternative method for regression is to use the Dirichlet distribution (5.6) and (5.7). The second form though (5.7) is more convenient and the estimated parameters have the same interpretation as in the additive logistic regression (5.16).

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. We do not offer this option here and I know it's not the best thing to do. The reason why this function is here is for learning purposes mainly.

The Dirichlet density (the same as in (5.7)) 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

$$\begin{aligned} a_1^* &= \frac{1}{\sum_{j=1}^{D} e^{\mathbf{x}^T \boldsymbol{\beta}_j}} \\ a_i^* &= \frac{e^{\mathbf{x}^T \boldsymbol{\beta}_i}}{\sum_{j=1}^{D} e^{\mathbf{x}^T \boldsymbol{\beta}_j}} \quad \text{for } i = 2, ..., D. \end{aligned}$$

So, the the corresponding log-likelihood (a function of the $\beta_i s$) 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 contains the log likelihood to be maximised, as a function of the regression parameters.

```
dirireg2=function(param,z=z) {
## param contains the parameter values
## z contains the compositional data and independent variable(s)
phi=param[1] ; para=param[-1]
## a small check against negative values of phi
if (phi<0) l=10000
if (phi>0) {
ya=z$ya ; xa=z$xa
## ya is the compositional data and xa the independent variable(s)
n=nrow(ya) ; d=ncol(ya)-1 ## sample size and dimensionality of  the simplex
be=matrix(para,ncol=d) ## puts the beta parameters in a matrix
mu1=cbind(1,exp(xa%*%be))
ma=mu1/rowSums(mu1) ## the fitted values
l=-( n*lgamma(phi)-sum(lgamma(phi*ma))+sum(diag(log(ya)%*%t(phi*ma-1))) )  }
```

```
## l is the log-likelihood
l }
```

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(ya,xa) {
## ya is the compositional data
ya=as.matrix(ya) ; n=nrow(ya)
ya=ya/rowSums(ya) ; xa=as.matrix(cbind(1,xa))
## the line above makes sure ya is compositional data and
## then the unit vector is added to the desing matrix
d=ncol(ya)-1 ; z=list(ya=ya,xa=xa) ## dimensionality of the simplex
rla=log(ya[,-1]/ya[,1]) ## additive log-ratio transformation
ini=solve(t(xa)%*%xa)%*%t(xa)%*%rla  ## initial values based on the logistic normal
## the next lines optimize the dirireg2 function and estimate the parameter values
el=NULL
qa=optim(c(20,as.vector(t(ini))),dirireg2,z=z,control=list(maxit=4000))
el[1]=-qa$value
qa=optim(qa$par,dirireg2,z=z,control=list(maxit=4000))
el[2]=-qa$value
vim=2
while (el[vim]-el[vim-1]>0.0001) { ## the tolerance value can of course change
vim=vim+1
qa=optim(qa$par,dirireg2,z=z,control=list(maxit=4000))
el[vim]=-qa$value }
qa=optim(qa$par,dirireg2,z=z,control=list(maxit=4000),hessian=T)
phi=qa$par[1] ; para=qa$par[-1] ## the estimated parameter values
beta=matrix(para,ncol=d) ## the matrix of the betas
colnames(beta)=colnames(ya[,-1]) ## names of the matrix of betas
mu1=cbind(1,exp(xa%*%beta))
ma=mu1/rowSums(mu1) ## fitted values
s=sqrt(diag(solve(qa$hessian))) ## std of the estimated betas
std.phi=s[1] ## std of the estimated phi
S=matrix(s[-1],ncol=d) ## matrix of the std of the estimated betas
colnames(S)=colnames(ya[,-1])
V=solve(qa$hessian) ## covariance matrix of the parameters
list(loglik=-qa$value,param=ncol(xa)*d+1,phi=phi,std.phi=std.phi,
beta=t(beta),std.errors=t(S),Cov=V,fitted=ma) }
```

5.5.3 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 (2013), 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

$$\sum_{i=1}^{n} \mathbf{u}_i^T \mathbf{u}_i, \text{ where } \mathbf{u}_i = \mathbf{y}_i - \mathbf{G}_i(\mathbf{B}) \text{ and}$$

$$\mathbf{G}_i(\mathbf{B}) = \left(\frac{1}{\sum_{j=1}^{D} e^{\mathbf{x}_i^T \boldsymbol{\beta}_j}}, \frac{e^{\mathbf{x}_i^T \boldsymbol{\beta}_2}}{\sum_{j=1}^{D} e^{\mathbf{x}_i^T \boldsymbol{\beta}_j}}, \ldots, \frac{e^{\mathbf{x}_i^T \boldsymbol{\beta}_d}}{\sum_{j=1}^{D} e^{\mathbf{x}_i^T \boldsymbol{\beta}_j}} \right),$$

with $\mathbf{y}_i \in \mathbb{S}^d$ and $d = D - 1$, where D denotes the number of components.

The 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=1000){
## 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
y=as.matrix(y)
y=y/rowSums(y) ## makes sure y is compositional data
x=as.matrix(cbind(1,x))
d=ncol(y)-1 ## dimensionality of the simplex
n=nrow(y) ## sample size
z=list(y=y,x=x)
reg=function(para,z){
y=z$y ; x=z$x
d=ncol(y)-1
be=matrix(para,byrow=T,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 estimated betas
ini=as.vector(t(coef(lm(y[,-1]~x[,-1])))) ## initial values
qa=optim(ini,reg,z=z,control=list(maxit=4000))
qa=optim(qa$par,reg,z=z,control=list(maxit=4000))
```

```
qa=optim(qa$par,reg,z=z,control=list(maxit=4000))
qa=optim(qa$par,reg,z=z,control=list(maxit=4000))
beta=matrix(qa$par,byrow=T,ncol=d)
mu1=cbind(1,exp(x%*%beta))
mu=mu1/rowSums(mu1)
std.errors=NULL
if (B>1){
betaboot=matrix(nrow=B,ncol=length(ini))
for (i in 1:B){
ida=sample(1:n,n,replace=T)
yb=y[ida,] ; xb=x[ida,]
zb=list(y=yb,x=xb)
ini=as.vector(t(coef(lm(yb[,-1]~xb[,-1])))) ## initial values
qa=optim(ini,reg,z=zb,control=list(maxit=4000))
qa=optim(qa$par,reg,z=zb,control=list(maxit=4000))
qa=optim(qa$par,reg,z=zb,control=list(maxit=4000))
qa=optim(qa$par,reg,z=zb,control=list(maxit=4000))
betaboot[i,]=qa$par }
s=apply(betaboot,2,sd)
std.errors=matrix(s,byrow=T,ncol=d) }
list(beta=beta,std.errors=std.errors,fitted=mu) }
```

6 Directional 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 Grn (2012) for mixtures of von Mises-Fisher distribution (circular, spherical or hyper-spherical).

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.

6.1 Circular statistics

At first we start with the circular data analysis, that is, data defined on the circle. Thus their space is denoted by $\mathbb{S}^1$.

6.1.1 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

$$\bar{C} = \frac{1}{n}\sum_{i=1}^{n} \cos u_i \text{ and } \bar{S} = \frac{1}{n}\sum_{i=1}^{n} \sin u_i.$$

The sample circular mean,or mean direction is given by (Mardia and Jupp, 2000)

$$\bar{\theta} = \left\{ \begin{array}{ll} \tan^{-1}\left(\bar{S}/\bar{C}\right) & \text{if } \bar{C} > 0 \\ \tan^{-1}\left(\bar{S}/\bar{C}\right) + \pi & \text{if } \bar{C} < 0 \end{array} \right\}$$

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 $(-2\log \bar{R})^{1/2}$ (Mardia and Jupp, 2000). Let us now construct a $(1-\alpha)$ % confidence interval for the true mean angle μ. We can distinguish, two cases

- $\bar{R} \leq 2/3$

$$\bar{\theta} \pm \cos^{-1}\left\{\left[\frac{2n\left(2R^2 - n\chi^2_{1,1-\alpha}\right)}{R^2\left(4n - \chi^2_{1,1-\alpha}\right)}\right]^{1/2}\right\}$$

- $\bar{R} > 2/3$

$$\bar{\theta} \pm \cos^{-1}\left\{\frac{\left[n^2 - \left(n^2 - R^2\right)\exp\left(\chi^2_{1,1-\alpha}/n\right)\right]^{1/2}}{R}\right\}$$

The R code with these summary measures is given below.

```
circ.summary=function(u,rads=F) {
## us is an angular variable
n=length(u) # sample size
if (rads==F) u=u*pi/180 ## if the data are in degrees we transform them into rads
## we transform them into Euclidean coordinates
## mesos contains the sample mean direction
C=mean(cos(u)) ; S=mean(sin(u))
Rbar=sqrt(C^2+S^2)
if (C>0) mesos=atan(S/C)
if (C<0) 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(2*pi)-n*(log(besselI(k,0,expon.scaled=T))+k)
kappa=optimize(lik,c(0,10000),maximum=TRUE)$maximum ## estimated concentration (kappa)
R=n*Rbar
if (Rbar<2/3) {
fact=sqrt( 2*n*(2*R^2-n*qchisq(0.95,1))/(R^2*(4*n-qchisq(0.95,1))) )
ci=c(mesos-acos(fact),mesos+acos(fact)) }
if (Rbar>2/3) {
```

```
fact=sqrt( n^2-(n^2-R^2)*exp(qchisq(0.95,1)/n) )/R
ci=c(mesos-acos(fact),mesos+acos(fact)) }
if (rads==F) {
mesos=mesos*180/pi
ci=ci*180/pi }
list(mesos=mesos,confint=ci,MRL=MRL,circvariance=circv,circstd=circs) }
```

6.1.2 Circular-circular correlation I

Jammalamadaka and R. (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

$$r_c = \frac{\sum_{i=1}^{n} \sin(\alpha_i - \bar{\alpha}) \sin(\beta_i - \bar{\beta})}{\sqrt{\sum_{i=1}^{n} \sin^2(\alpha_i - \bar{\alpha}) \sum_{i=1}^{n} \sin^2(\beta_i - \bar{\beta})}}, \tag{6.1}$$

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 suitbale 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

$$\sqrt{n}\sqrt{\frac{\hat{\lambda}_{02}\hat{\lambda}_{20}}{\hat{\lambda}_{22}}} r_c \sim N(0,1),$$

where

$$\hat{\lambda}_{ij} = \frac{1}{n}\sum_{i=1}^{n} \sin^i(\alpha_i - \bar{\alpha}) \sin^j(\beta_i - \bar{\beta}).$$

This is an asymptotic normality based test and below we provide the relevant R code.

```
circ.cor1=function(a,b,rads=F) {
## a and b are angular data in degrees or rads,
## by default they are in degrees
n=length(a) # sample size
deg2rad=function(deg) deg*pi/180  ## from degrees to rads
## if the data are in degrees we transform them into rads
if (rads==F) {
a=deg2rad(a)
b=deg2rad(b) }
## We calculate the mean of each vector
m1=circ.summary(a)$mesos
```

```
m2=circ.summary(b)$mesos
up=sum(sin(a-m1)*sin(b-m2))
down=sqrt( sum(sin(a-m1)^2)*sum(sin(b-m2)^2) )
rho=up/down  ## circular correlation
lam22=mean( (sin(a-m1))^2*(sin(b-m2))^2 )
lam02=mean( (sin(b-m2))^2 )
lam20=mean( (sin(a-m1))^2 )
zrho=sqrt(n)*sqrt(lam02*lam20/lam22)*rho
p.value=2*(1-pnorm(abs(zrho)))
list(rho=rho,p.value=p.value) }
```

6.1.3 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

$$r^2 = \frac{\left(r_{cc}^2 + r_{cs}^2 + r_{sc}^2 + r_{ss}^2\right) + 2\left(r_{cc}r_{ss} + r_{cs}r_{sc}\right)r_1 r_2 - 2\left(r_{cc}r_{cs} + r_{sc}r_{ss}\right)r_2 - 2\left(r_{cc}r_{sc} + r_{cs}r_{ss}\right)r_1}{\left(1 - r_1^2\right)\left(1 - r_2^2\right)} \quad (6.2)$$

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=F) {
## theta and phi are angular data in degrees or rads,
## by default they are in degrees
n=length(theta) # sample size
deg2rad=function(deg) deg*pi/180  ## from degrees to rads
## if the data are in degrees we transform them into rads
if (rads==F) {
theta=deg2rad(theta)
phi=deg2rad(phi) }
rcc=cor(cos(theta),cos(phi))
rcs=cor(cos(theta),sin(phi))
rss=cor(sin(theta),sin(phi))
rsc=cor(sin(theta),cos(phi))
r1=cor(cos(theta),sin(theta))
r2=cor(cos(phi),sin(phi))
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
p.value=1-pchisq(test,4)
list(rho=rho,p.value=p.value) }
```

6.1.4 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

$$\frac{(n-3)\, R^2_{x\theta}}{1 - R^2_{x\theta}} \sim F_{2,n-3}.$$

Since the F distribution is asymptotic we use non parametric bootstrap to calculate the p-value as well. In the following R function bootstrap is implemented by default.

```
circlin.cor=function(x,theta,rads=F) {
## x is euclidean variable
## theta is a angular variable in degrees by default
n=length(x) ## sample size
deg2rad=function(deg) deg*pi/180  ## from degrees to rads
if (rads==F) theta=deg2rad(theta)
rxc=cor(x,cos(theta)) ## x and theta correlation
rxs=cor(x,sin(theta)) ## x and sin(theta) correlation
rcs=cor(cos(theta),sin(theta)) ## 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
p.value=1-pf(Ft,2,n-3)
list(R.squared=R2xt,p.value=p.value) }
```

6.1.5 Regression for circular or angular 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 is

$$f(\theta) = \frac{e^{\kappa \cos(\theta - \mu)}}{2\pi I_0(\kappa)}, \tag{6.3}$$

where $I_0(\kappa)$ denotes the modified Bessel function of the first kind and order 0 calculated at κ. The variable θ takes any values at an interval of length 2π, μ is a real number and κ is strictly positive.

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 direction 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 (6.3). 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.

6.1.6 Projected bivariate normal for circular regression

Presnell et al. (1998) used the projected multivariate normal (Watson, 1983) to perform circular regression. The density of the projected normal in the circular case can be written as

$$f(\theta) = \frac{1}{2\pi} e^{-\frac{\gamma^2}{2}} \left[1 + \frac{\gamma \cos(\theta - \omega)\, \Phi(\gamma \cos(\theta - \omega))}{\phi(\gamma \cos(\theta - \omega))}\right], \tag{6.4}$$

where θ represents the angle, ω is the mean direction and $Phi(.)$ and ϕ) are the standard normal probability and density function respectively. Following Presnell et al. (1998) we will substitute $\gamma \cos(\theta - \omega)$ by $\mathbf{u}^T \boldsymbol{\mu}$ in the above density (6.4) and we will write its associated log-likelihood as

$$\ell(\mathbf{B}) = -\frac{1}{2} \sum_{i=1}^{n} \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + \sum_{i=1}^{n} \log\left[1 + \frac{\mathbf{u}^T \boldsymbol{\mu} \Phi(\mathbf{u}^T \boldsymbol{\mu})}{\phi(\mathbf{u}^T \boldsymbol{\mu})}\right] - n \log(2\pi),$$

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

$$\theta_i = \left[\tan^{-1}\left(\frac{\boldsymbol{\beta}_2^T \mathbf{x}_i}{\boldsymbol{\beta}_1^T \mathbf{x}_i} \right) + \pi I\left(\boldsymbol{\beta}_1^T \mathbf{x}_i < 0 \right) \right] mod 2\pi,$$

where I is the indicator function.

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 (6.1) between the observed and the estimated angles and then square it. 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.

```
spml.reg=function(y,x,rads=T){
## y is the angular depndent variable
## x contains the idnependent variable(s)
deg2rad=function(deg) deg*pi/180  ## from degrees to radians
if (rads==F) y=deg2rad(y) ## if the data are in degrees we transform them into radians
u=cbind(cos(y),sin(y)) ## bring the data onto the circle
x=cbind(1,x)
z=list(u=u,x=x)
spml=function(para,z) {
u=z$u ; x=z$x
n=nrow(u)
beta=matrix(para,ncol=2)
mu=x%*%beta
t=diag(u%*%t(mu))
l=-( -0.5*sum(diag(mu%*%t(mu)))+sum(log( 1+t*pnorm(t)/dnorm(t) ))-n*log(2*pi) )
l }
para=as.vector(coef(lm(u~x[,-1]))) ## starting values
qa=optim(para,spml,z=z,control=list(maxit=2000))
```

```
qa=optim(qa$par,spml,z=z,control=list(maxit=2000))
qa=optim(qa$par,spml,z=z,control=list(maxit=2000),hessian=T)
log.lik=qa$value
param=qa$par
beta=matrix(param,ncol=2)
se=matrix(sqrt(diag(solve(qa$hessian))),ncol=2)
colnames(beta)=colnames(se)=c('cosinus','sinus')
rownames(beta)=rownames(se)=c('Intercept',colnames(x)[-1])
mu1=x%*%beta
fitted=( atan(mu1[,2]/mu1[,1])+pi*I(mu1[,1]<0) )%%(2*pi)
rsq=as.numeric(circ.cor1(theta,fitted,rads=T,R=1)$rho)^2 ## pseudo-R squared
## the fitted values are in radians
## use the next function to turn them from radians to degrees
## rad2deg=function(rad) rad*180/pi
if (ncol(x)==2) {
plot(x[,-1],y)
points(x[,-1],fitted,col=3,pch=2) }
list(fitted=fitted,parameters=beta,std.errors=se,pseudo.r2=rsq,log.lik=-log.lik) }
```

6.2 (Hyper)spherical statistics

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.

6.2.1 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

$$\mathbf{u} = (x, y, z) = [\cos(lat) * \cos(long), \cos(lat) * \sin(long), \sin(lat)]$$

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.

```
euclid=function(u) {
## u is a matrix of two columns
## the first column is the lattitude and the second the longitude
u=as.matrix(u)
if (ncol(u)==1) u=t(u)
u=pi*u/180 # from degrees to radians
U=cbind(cos(u[,1])*cos(u[,2]),cos(u[,1])*sin(u[,2]),sin(u[,1]))
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(asin(U[,3]),( atan(U[,2]/U[,1])+pi*I(U[,1]<0) )%%(2*pi))
u=u*180/pi
colnames(u)=c('Lat','Long')
## u is a matrix of two columns
## the first column is the lattitude and the second the longitude in degrees
u }
```

6.2.2 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

$$\mathbf{Q} = \mathbf{I}_p + \sin(\alpha)\,\mathbf{A} + [\cos(\alpha) - 1]\left(\mathbf{a}\mathbf{a}^T + \mathbf{c}\mathbf{c}^T\right) \tag{6.5}$$

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)
c=a-b*(a%*%t(t(b)))
c=c/sqrt(sum(c^2))
A=t(t(b))%*%c-t(t(c))%*%b
theta=acos(sum(a*b))
diag(p)+sin(theta)*A+(cos(theta)-1)*(t(t(b))%*%b+t(t(c))%*%c) }
```

6.2.3 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
## lattitude and the second is the longitude
## theta is the angle of rotation
if (rads==TRUE) {
lat=ksi[1]
long=ksi[2]
the=theta  }
```

```
deg2rad=function(deg) deg*pi/180  ## from degrees to radians
if (rads==FALSE) {
lat=deg2rad(ksi[1])
long=deg2rad(ksi[2])
the=deg2rad(theta) }
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
tr=sum(diag(A))
rad2deg=function(rad) rad*180/pi ## from radians to degrees
rad=acos(0.5*(tr-1))
degrees=rad2deg(rad)
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(rad2deg(asin(ksi[3])), rad2deg(atan2(ksi[2],ksi[1])))
axis=matrix(axis,ncol=2)
colnames(axis)=c('latitude','longitude')
list(angle=degrees,axis=axis) }
```

6.2.4 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 $\mathbf{V}$ was derived from $\mathbf{U}$ via a rotation matrix $\mathbf{A}$ (so $\mathbf{A}$ belongs to SO(3))

$$\mathbf{V} = \mathbf{AU}.$$

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

$$\mathbf{UV}^T = \mathbf{O}_1 \mathbf{\Lambda} \mathbf{O}_2^T,$$

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 $\mathbf{U}$ is of full rank (3 in our case), the determinant of $\mathbf{UV}^T$ is nonzero with probability 1 and in this case $\mathbf{A}$ is uniquely estimated (Chang, 1986)

$$\hat{\mathbf{A}} = \mathbf{O}_2 \mathbf{O}_1^T$$

The R code is given below.

```
spher.reg=function(u,v,euclidean=FALSE,rads=FALSE) {
## u is the independent variable
## v is the dependent variable
## The first row of both matrices is the lattitude
## and the second is the longitude
u=as.matrix(u)
v=as.matrix(v)
n=nrow(u) ## sample size
if (euclidean==FALSE) {
if (rads==FALSE) {
u=pi*u/180 ## from degrees to radians
v=pi*v/180 } ## from degrees to radians
## the first row of both matrices is the lattitude and the second is the longitude
## the next two rows transform the data to Euclidean coordinates
U=cbind(cos(u[,1])*cos(u[,2]),cos(u[,1])*sin(u[,2]),sin(u[,1]))
V=cbind(cos(v[,1])*cos(v[,2]),cos(v[,1])*sin(v[,2]),sin(v[,1])) }
if (euclidean==TRUE) U=u ; V=v
UV=t(U)%*%V/n
b=svd(UV) ## SVD of the UV matrix
A=b$v%*%t(b$u)
```

```
est=U%*%t(A)
list(A=A,est=est) }
```

Since $\hat{\mathbf{A}}$ is a rotation matrix, we can then use the function we saw in the previous section (6.2.3) to calculate the rotation axis and the angle of rotation.

6.2.5 (Hyper)spherical correlation

Suppose we have two variables $\mathbf{X} \in \mathrm{S}^{p-1}$ and $\mathbf{Y} \in \mathrm{S}^{q-1}$ 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 (6.2) generalizes to

$$r^2 = \mathrm{tr}\left(\mathbf{S}_{xx}^{-1}\mathbf{S}_{xy}\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx}\right),$$

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 vaiables
x=as.matrix(x)
y=as.matrix(y)
stand=function(x) x-mean(x)
p=ncol(x) ; q=ncol(y) ## dimension of each of these two variables
x=apply(x,2,stand) ## subtract the mean
y=apply(y,2,stand) ## subtract the mean
n=nrow(x) ## sample size
s11=(t(x)%*%x)/n
s12=(t(x)%*%y)/n
s21=t(s12)/n
s22=(t(y)%*%y)/n
rsq=sum(diag(solve(s11)%*%s12%*%solve(s22)%*%s21))
test=n*rsq
pval=1-pchisq(test,p*q)
list(rsq=rsq,p.value=pval) }
```

6.2.6 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 if given by

$$f_p(\mathbf{x}; \mu, \kappa) = C_p(\kappa) \exp\left(\kappa \boldsymbol{\mu}^T \mathbf{x}\right), \tag{6.6}$$

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).

$$\hat{\kappa}^{(t)} = \hat{\kappa}^{(t-1)} - \frac{A_p\left(\hat{\kappa}^{(t-1)}\right) - \bar{R}}{1 - \left[A_p\left(\hat{\kappa}^{(t-1)}\right)\right]^2 - \frac{p-1}{\hat{\kappa}^{(t-1)}} A_p\left(\hat{\kappa}^{(t-1)}\right)}, \tag{6.7}$$

where

$$A_p\left(\hat{\kappa}^{(t-1)}\right) = \frac{I_{p/2}(\hat{\kappa})}{I_{p/2-1}(\hat{\kappa})} = \frac{\| \sum_{i=1}^{p} \mathbf{x_i} \|}{n} = \bar{R}, \tag{6.8}$$

and $I_p(\hat{\kappa})$ is the modified Bessel function of the first kind (see Abramowitz and Stegun (1970)). Similarly to Sra (2012) we will set $\hat{\kappa}^{(0)} = \frac{\bar{R}(p-\bar{R}^2)}{1-\bar{R}^2}$ to (6.7). The variance of $\hat{\kappa}$ is given by (Mardia and Jupp, 2000)

$$var(\hat{\kappa}) = \left[n\left(1 - \frac{A_p(\hat{\kappa})}{\hat{\kappa}} - A_p(\hat{\kappa})^2\right)\right]^{-1}$$

The modified Bessel function in R gives us the option to scale it exponentially. This means, that it calculates this quantity instead $I_p(\hat{\kappa}) exp^{-\hat{\kappa}}$. 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-7){
## x contains the data
## tol specifies the tolerance value for convergence
## when estimating the concentration parameter
x=as.matrix(x)
x=x/sqrt(rowSums(x^2))
p=ncol(x) ## dimensionality of the data
n=nrow(x) ## sample size length
p=ncol(x) ; n=nrow(x)
Apk=function(p,k) besselI(k,p/2,expon.scaled=T)/besselI(k,p/2-1,expon.scaled=T)
m1=colSums(x)
R=sqrt(sum(m1^2))/n
m=m1/(n*R)
k=numeric(4)
i=1
k[i]=R*(p-R^2)/(1-R^2)
i=2
k[i]=k[i-1]-(Apk(p,k[i-1])-R)/(1-Apk(p,k[i-1])^2-(p-1)/k[i-1]*Apk(p,k[i-1]))
while (abs(k[i]-k[i-1])>tol) {
i=i+1
k[i]=k[i-1]-(Apk(p,k[i-1])-R)/(1-Apk(p,k[i-1])^2-(p-1)/k[i-1]*Apk(p,k[i-1]))  }
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=T))+k)+
k*sum(x%*%m)
vark=1/( n*(1-Apk(p,k)/k-Apk(p,k)^2) )
list(mu=m,kappa=k,vark=vark,loglik=loglik) }
```

Alternatively and perhaps easier, if you want to estimate the concentration parameter κ you can solve the equation (6.8) 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.

6.2.7 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

$$np\bar{R}^2 \sim \chi^2_p,$$

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 (6.8). Mardia et al. (1979, pg. 440) mentions that the case of $p = 3$ was first proved by Rayleigh (1919).

The function below offers the possibility of a parametric bootstrap calculation of the p-value. 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,B=999) {
## x contais the data in Euclidean coordinates
## B is by default equal to 999 bootstrap samples
## If B==1 then no bootstrap is performed
n=nrow(x) ## sample size
p=ncol(x) ## dimensionality
m=colSums(x)
R=sqrt(sum(m^2))/n ## the R bar
T=n*p*R^2
if (B==1) p.value=1-pchisq(T,p)
if (B>1) {
Tb=numeric(B)
for (i in 1:B) {
x=matrix(rnorm(p*n),ncol=p)
x=x/sqrt(rowSums(x^2))
mb=colSums(x)
Rb=sqrt(sum(mb^2))/n
Tb[i]=n*p*Rb^2 }
p.value=(sum(Tb>T)+1)/(B+1) }
p.value }
```

6.2.8 Discriminant analysis for (hyper)spherical (and circular) data 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 (6.6). 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
x=as.matrix(x) ; p=ncol(x) ## p is the dimensionality of the data
per=numeric(R) ; n=nrow(x)
ina=as.numeric(ina)
frac=round(fraction*n)
g=max(ina)
mesi=matrix(nrow=g,ncol=p)
k=numeric(g)
## if seed==TRUE then the results will always be the same
if (seed==TRUE) 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 von Mises-Fisher
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]*test%*%mesi[j,]-0.5*p*log(2*pi)-
log(besselI(k[j],p/2-1,expon.scaled=T))-k[j]  }
for (l in 1:frac) est[l]=which.max(mat[l,])
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 confidence interval
conf2=c(percent-1.96*s2,percent+1.96*s2) ## 2nd way of a confidence 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(per,probs=c(0.025,0.975)) ## 3rd way of a confidence interval
list(percentage=percent,sd1=s1,sd2=s2,conf.int1=conf1,conf.int2=conf2,conf.int3=conf3) }
```

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
x=as.matrix(x)
xnew=as.matrix(xnew)
if (ncol(xnew)==1) xnew=t(xnew)
p=ncol(x) ## p is the 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[id==j,]) ## estimates the parameters of the von Mises-Fisher
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=T))-k[j]  }
for (l in 1:nu) est[l]=which.max(mat[l,])
list(est.group=est) }
```

6.2.9 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 (6.5) 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 $\mathbf{A}$ using (6.5) 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 Grn (2013) 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
mu=mu/sqrt(sum(mu^2)) ## the mean direction
d=length(mu) ## the dimensions
## k is the concentration parameter
## n is the sample size
ini=c(rep(0,d-1),1) ## the mean direction is set to (0,...,0,1)
b=( -2*k+sqrt(4*k^2+(d-1)^2) )/(d-1)
x0=(1-b)/(1+b)
S=matrix(nrow=n,ncol=d)
m=0.5*(d-1)
c=k*x0+(d-1)*log(1-x0^2)
for (i in 1:n) {
t=-1000
u=1
while (t-c<log(u)) {
z=rbeta(1,m,m)
u=runif(1)
w=(1-(1+b)*z)/(1-(1-b)*z)
t=k*w+(d-1)*log(1-x0*w) }
v1=mvrnorm(1,c(rep(0,(d-1))),diag(d-1))
v=v1/sqrt(sum(v1^2))
S[i,]=c(sqrt(1-w^2)*v,w) }
A=rotation(ini,mu) ## calculate the rotation matrix
## in order to rotate the initial mean direction from ini to mu
x=S%*%t(A) ## the x has direction mu
x }
```

6.2.10 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

$$f_{bing}\left(\mathbf{x}\right) = c_{bing} e^{\left(-\mathbf{x}^T \mathbf{A} \mathbf{x}\right)} = c_{bing} f_{bing}^{*}\left(\mathbf{x}\right),$$

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

$$f_{ACG}\left(\mathbf{x}\right) = c_{ACG} f_{ACG}^{*}\left(\mathbf{x}\right),$$

where where $c_{ACG} = \frac{\Gamma(q/2)}{2\pi^{q/2}} \left|\mathbf{\Omega}\right|^{-1/2}$ is the normalizing constant and $f_{ACG}^{*}\left(\mathbf{x}\right) = \left(\mathbf{x}^T \mathbf{\Omega} \mathbf{x}\right)^{-q/2}$.

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 $\mathbf{y} \sim N_q\left(0, \mathbf{\Sigma}\right)$, then $\mathbf{x} = \frac{\mathbf{y}}{\|\mathbf{y}\|}$ follows an $ACG\left(\mathbf{\Omega}\right)$ 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 $\mathbf{y} = \mathbf{x}\mathbf{V}$ comes from a Bingham with matrix parameter $\mathbf{\Lambda}$, and this matrix comes from the spectral decomposition of $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$.

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, $f_{bing}\left(\mathbf{x}\right) = f_{bing}\left(-\mathbf{x}\right)$.

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 $\boldsymbol{\Omega} = \boldsymbol{\Omega}(b) = \mathbf{I}_q + \frac{2}{b}\mathbf{B}$ and $M = e^{-0.5(q-b)}(q/b)^{q/2}$.

2. Draw a u from $U(0,1)$ and a $\mathbf{z}$ from $ACG(\boldsymbol{\Omega})$.

3. If $u < \frac{e^{\left(-\mathbf{z}^T\mathbf{A}\mathbf{z}\right)}}{M(\mathbf{z}^T\boldsymbol{\Omega}\mathbf{z})^{-q/2}}$ 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){
## lam contains the q-1 non negative eigenvalues
lam=sort(lam,decreasing=TRUE) ## sort the eigenvalues in desceding order
nsamp=0
X=NULL
lam.full=c(lam,0)
q=length(lam.full)
A=diag(lam.full)
SigACG.inv=diag(q)+2*A
SigACG=solve(SigACG.inv)
Ntry=0
while(nsamp < n) {
x.samp=FALSE
while(x.samp==FALSE) {
yp=mvrnorm(n=1,mu=rep(0,q),Sig=SigACG)
y=yp/sqrt(t(yp)%*%yp)
lratio=-t(y)%*%A%*%y -q/2*log(q) + 0.5*(q-1) + q/2*log(t(y)%*%SigACG.inv%*%y)
if(log(runif(1)) < lratio) {
X=c(X,y)
x.samp=TRUE
nsamp = nsamp+1    }
Ntry=Ntry+1  }   }
if(n>1) X=matrix(X,byrow=T,ncol=q)
## 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 $\mathbf{A}$ and it calls the previous function.

```
rbingham=function(n,A){
p=ncol(A) ## dimensionality of A
lam=eigen(A)$values ## eigenvalues
V=eigen(A)$vectors ## eigenvectors
lam=lam-lam[p]
lam=lam[-p]
x=f.rbing(n,lam)$X ## Chris and Theo's code
y=x%*%t(V) ## simulated data
y }
```

6.2.11 Simulation from a Fisher-Bingham distribution

The Fisher-Bingham distribution is written as Kent et al. (2013)

$$f_{FB}\left(\mathbf{x}\right) = c_{FB} e^{\left(\kappa \mathbf{x}^T \boldsymbol{\mu} - \mathbf{x}^T \mathbf{A} \mathbf{x}\right)} = c_{FB} f_{FB}^{*}\left(\mathbf{x}\right) \tag{6.9}$$

Kent et al. (2013) mentions that the Fisher-Bingham distribution (6.11) can be bounded by a Bingham density

$$f_{FB}^{*}\left(\mathbf{x}\right) \leq e^{\left(\kappa - \mathbf{x}^T \mathbf{A}^{(1)} \mathbf{x}\right)} = e^{\kappa} e^{\left(-\mathbf{x}^T \mathbf{A}^{(1)} \mathbf{x}\right)}, \tag{6.10}$$

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 (6.10) holds true.

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.

```
fb.sim=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
q=length(m)
A1=A+k/2*( diag(q)-m%*%t(m) )
lam=eigen(A1)$values
V=eigen(A1)$vectors
lam=lam-lam[q]
```

```
lam=lam[-q]
x=f.rbing(5*n,lam)$X
x=x%*%t(V)
u=log(runif(5*n))
ffb=k*x%*%m-diag(x%*%A%*%t(x))
fb=k-diag(x%*%A1%*%t(x))
ina=1:c(5*n)
keep=ina[u<=c(ffb-fb)]
ind=sample(keep,n)
y=x[ind,]
y }
```

6.2.12 Normalizing constant of the Bingham and the Fisher-Bingham distributions

The Fisher-Bingham distribution density is given by Kume and Wood (2005)

$$f\left(\mathbf{x}|\mathbf{A},\boldsymbol{\gamma}\right)=\frac{1}{c\left(\mathbf{A},\boldsymbol{\gamma}\right)}\exp\left(-\mathbf{x}^{T}\mathbf{A}\mathbf{x}+\boldsymbol{\gamma}^{T}\mathbf{x}\right), \tag{6.11}$$

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 $\boldsymbol{\Lambda}=\text{diag}\left(\lambda_1,\ldots,\lambda_p\right)$, 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 $\boldsymbol{\gamma}=\left(\gamma_1,\ldots,\gamma_p\right)$ 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

$$K_{\theta}\left(t\right)=\sum_{i=1}^{p}\left\{-\frac{1}{2}\log\left(1-t/\lambda_i\right)+\frac{1}{4}\frac{\gamma_i^2}{\lambda_i-t}-\frac{\gamma_i^2}{4\lambda_i}\right\}\quad\left(t<\lambda_1\right). \tag{6.12}$$

The first derivative of (6.12) is

$$K_{\theta}^{(1)}\left(t\right)=\sum_{i=1}^{p}\left\{\frac{1}{2}\frac{1}{\lambda_i-t}+\frac{1}{4}\frac{\gamma_i^2}{\left(\lambda_i-t\right)^2}\right\}$$

and higher derivatives of (6.12) 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

$$\hat{f}_{\theta,1}(\alpha) = \left[2\pi \hat{K}_\theta^{(2)}(\hat{t}) \right]^{-1/2} \exp\left(\hat{K}_\theta(\hat{t}) - \hat{t} \right), \tag{6.13}$$

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

$$\hat{f}_{\theta,2}(1) = \hat{f}_{\theta,1}(1)(1+T) \quad \text{and} \quad \hat{f}_{\theta,3}(1) = \hat{f}_{\theta,1}(1)\exp(T) \quad \text{respectively,} \tag{6.14}$$

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

$$c(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = 2\pi^{p/2} \left(\prod_{i=1}^{p} \lambda_i^{-1/2} \right) f_\theta(1) \exp\left(\frac{1}{4} \sum_{i=1}^{p} \frac{\gamma_i^2}{\lambda_i} \right), \tag{6.15}$$

where $f_\theta(1)$ is found in Kume and Wood (2005).

The saddlepoint approximations of the Fisher-Bingham normalizing constant (6.15) are given by

$$\hat{c}_1(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = 2^{1/2}\pi^{(p-1)/2} \left[K_\theta^{(2)}(\hat{t}) \right]^{-1/2} \left[\prod_{i=1}^{p} (\lambda_i - \hat{t})^{-1/2} \right] \exp\left(-\hat{t} + \frac{1}{4} \sum_{i=1}^{p} \frac{\gamma_i^2}{\lambda_i - \hat{t}} \right),$$

$$\hat{c}_2(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = \hat{c}_1(\boldsymbol{\lambda}, \boldsymbol{\gamma})(1+T) \quad \text{and} \quad \hat{c}_3(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = \hat{c}_3(\boldsymbol{\lambda}, \boldsymbol{\gamma})\exp(T).$$

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 $\hat{c}_k(\lambda, \gamma)$ 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

$$c(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = c(\boldsymbol{\lambda} + a\mathbf{1}_p, \boldsymbol{\gamma}) e^a.$$

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.

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 Simon calculates the logarithm of the constant and in Kume et al. (2013) the Bingham part in the Fisher-Bingham density does **not** have a minus sign $(-)$ as in our case (see (6.11), there is a minus sign). 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) lam=lam+2*abs(mina) ## make the lambdas positive
p=length(gam) ## dimensionality of the distribution
para=c(gam,lam) ### the parameters of the Fisher-Bingham
saddle.equat=function(t,para) { ## saddlepoint equation
p=length(para)/2
gam=para[1:p] ; lam=para[-c(1:p)]
f=sum( 0.5/(lam-t)+ 0.25*(gam^2/(lam-t)^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 cumulant generating function
kfb=function(j,gam,lam,t) {
if (j==1) kd=sum( 0.5/(lam-t)+ 0.25*(gam^2/(lam-t)^2) )
```

```
if (j>1) kd=sum( 0.5*factorial(j-1)/(lam-t)^j+0.25*factorial(j)*gam^2/(lam-t)^(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
T=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+log(1+T)
c3=c1+T
## the next multiplications brings the modification with the negative
## values in the lambdas back
if (mina<0) {
c1=c1+2*abs(mina)
c2=c2+2*abs(mina)
c3=c3+2*abs(mina) }
list(c1=c1,c2=c2,c3=c3) }
```

6.2.13 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 (6.11) and the argument **B** is the matrix $-\mathbf{A}$ in (6.11). 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 (6.11) there is a minus sign). Finally **approxType** takes the values **1**, **2** or **3** corresponding to the first (6.13), second and third order (6.14) 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 $\mathbf{B}$ is a matrix. We say that the eigenvalues of $\mathbf{B}$ are $(1, 2, 3)$. This means that Simon's Matlab code needs the negative eigenvalues. Or in general, the negative of the matrix $\mathbf{B}$ 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

The answer is the logarithm of the third order (6.14) saddlepoint approximation to the normalizing constant of the Bingham distribution (the vector $\boldsymbol{\gamma}$ is zero). We exponentiate the result (**exp(res$logC)**) and we get the answer. Compare this answer with the answer from the previous R function **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.

```
$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"
```

```
library(R.matlab)
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=exp(res$logC))  }
if (method=="MC") { ## this does Monte Carlo integration
evaluate(matlab,"[logC, se_logC] = logNormConstMC(d,a,B,1e+05);")
res=getVariable(matlab,"logC")
se.const=getVariable(matlab,"se_logC")
result=list(norm.const=exp(res$logC),se.norm.const=se.const$se.logC) }
result }
```

6.2.14 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)

$$f(\mathbf{x}) = c(\kappa,\beta)^{-1} \exp\left\{\kappa \boldsymbol{\alpha}_1^T \mathbf{x} + \beta\left[\left(\boldsymbol{\alpha}_2^T \mathbf{x}\right)^2 - \left(\boldsymbol{\alpha}_3^T \mathbf{x}\right)^2\right]\right\}, \tag{6.16}$$

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 $\kappa \geq 0$ 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 (6.16) 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 6.2.12). 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 (6.16) 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 6.2.12) utilizes the second formula. In both cases though, the normalizing constant depends upon κ and β only. The normalizing constant we saw in Section 6.2.12 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 6.2.12. In terms of Simon's MATLAB function (see Section 6.2.13) we would specify

$\gamma = (0, 0, \kappa)^T$ and $\lambda = (\beta, -\beta, 0)^T$.

So, the log-likelihood of the Kent distribution from (6.16) is

$$\ell = -n * c(\kappa, \beta) + \kappa \sum_{i=1}^{n} \boldsymbol{\alpha}_1^T \mathbf{x}_i + \beta \left[\sum_{i=1}^{n} \left(\boldsymbol{\alpha}_2^T \mathbf{x}_i \right)^2 - \sum_{i=1}^{n} \left(\boldsymbol{\alpha}_3^T \mathbf{x}_i \right)^2 \right]. \tag{6.17}$$

We will now describe the estimation the parameters of (6.16) 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 submatrix 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 (6.17) 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 6.2.12. The next R function calculates the $\mathbf{A}$ 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 $\mathbf{A}$ matrix are slightly different, but I do not think this is an issue.

```
kent.mle=function(x) {
## x is the data in Euclidean coordinates
n=nrow(x) ## sample size
xbar=colMeans(x) ## mean vector
degs=euclid.inv(xbar) ## bring the mean vector to latitude and longitude
u=pi*degs/180 # from degrees to radians
theta=u[1] ; phi=u[2]
```

```
H=matrix(c(cos(theta),sin(theta)*cos(phi),sin(theta)*sin(phi),
-sin(theta),cos(theta)*cos(phi),cos(theta)*sin(phi),0,-sin(phi),cos(phi)),ncol=3)
S=(1/n)*t(x)%*%x
B=t(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])$values
r2=lam[1]-lam[2]
## the next function will be used to estimate the kappa and beta
mle=function(para){
## maximization w.r.t. to k and b
k=para[1] ; b=para[2]
gam=c(0,k,0)
lam=c(0,-b,b)
ckb=fb.saddle(gam,lam)$c3
g=-( -n*ckb+k*sum(x%*%G[,1])+b*sum((x%*%G[,2])^2)-b*sum((x%*%G[,3])^2)  )
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=fb.saddle(gam,lam)$c3
## the line below calculates the log-likelihood.
l=-n*ckb+k*sum(x%*%G[,1])+b*sum((x%*%G[,2])^2)-b*sum((x%*%G[,3])^2)
para=c(k,b) ; names(para)=c('kappa','beta')
colnames(G)=c('mean','major','minor')
list(G=G,param=para,loglik=l) }
```

If we want to simulate from a Kent distribution then we have to use the *fb.sim* function we saw in Section 6.2.11. The point is to suitably fix the parameter $\boldsymbol{\mu}$ and $\mathbf{A}$ of (6.9). So for a concentration parameter κ and an ovalness parameter β, we would have to specify

```
m=c(0,1,0)
A=diag(c(-b,0,b))
```

and then type in R

```
fb.sim(n,k,m,A)
```

Try this with some values of κ and β and then use the *kent.mle* function above to see the estimates of κ and β.

6.2.15 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

$$\hat{T} = n\left(\frac{\hat{\kappa}}{2}\right)^2 \frac{I_{1/2}\left(\hat{\kappa}\right)}{I_{5/2}\left(\hat{\kappa}\right)}\left(l_1 - l_2\right)^2.$$

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=nrow(x) ## 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)
P=diag(3)-(e1-mu)%*%t(e1-mu)/(1-mu[1])
y=(x%*%P)[,2:3]
lam=eigen((1/n)*(t(y)%*%y))$values
rat=besselI(k,0.5,expon.scaled=T)/besselI(k,2.5,expon.scaled=T)
T=n*(k/2)^2*rat*(lam[1]-lam[2])^2
if (B==1) p.value=1-pchisq(T,2)
if (B>1) {
Tb=numeric(B)
for (i in 1:B) {
nu=sample(1:n,n,replace=T)
estim=vmf(x[nu,])
k=estim$k ## the estimated concentration parameter
## under the H0, that the Fisher distribution is true
mu=estim$mu ## the estimated mean direction under H0
e1=c(1,0,0)
P=diag(3)-(e1-mu)%*%t(e1-mu)/(1-mu[1])
y=(x[nu,]%*%P)[,2:3]
lam=eigen((1/n)*(t(y)%*%y))$values
rat=besselI(k,0.5,expon.scaled=T)/besselI(k,2.5,expon.scaled=T)
Tb[i]=n*(k/2)^2*rat*(lam[1]-lam[2])^2 }
p.value=(sum(Tb>T)+1)/(B+1) }
p.value }
```

6.2.16 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 (6.6), excluding the constant terms, for convenience

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 (6.6) 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 6.1 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 rads, 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.

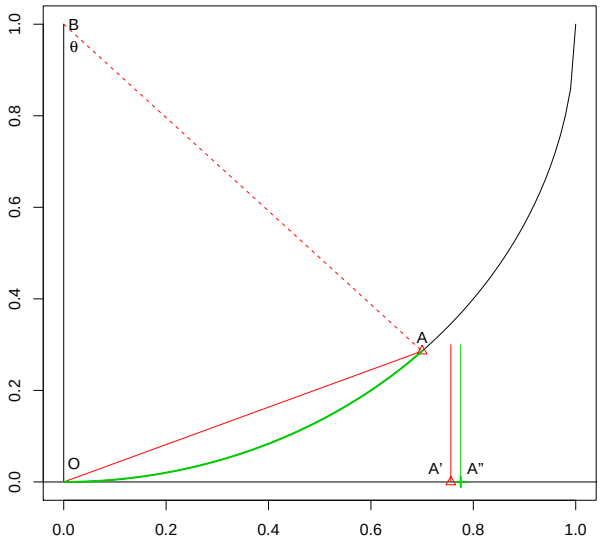

Figure 6.1: A slice of a quarter of the sphere along with a chord and an arc. The red and green lines indicate the projection of the point on the sphere onto the tangent plane.

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 6.1).

```
vmf.contour=function(k) {
## k is the concentration parameter
rho=pi/2 ## radius of the circular disc
x=y=seq(-rho,rho,by=0.01) ; n=length(x)
mat=matrix(nrow=n,ncol=n)
for (i in 1:n) {
for (j in 1:n) {
z=c(x[i],y[j])
if ( sum(z^2)<rho^2 ) {
theta=sqrt(sum(z^2))
xa=0.5*log(k)+k*cos(theta)-1.5*log(2*pi)-log(besselI(k,0.5,expon.scaled=T))-k
mat[i,j]=exp(xa) } } }
contour(x,y,mat) }
```

6.2.17 Contour plots of the Kent distribution

The Kent distribution as we saw it in (6.16) 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 6.1.

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){
gam=c(0,k,0) ; lam=c(0,b,-b)
con=fb.saddle(gam,lam)$c3
rho=sqrt(2)
x=y=seq(-rho,rho,by=0.01) ; n=length(x)
## radius of the circular disc is 1
mat=matrix(nrow=n,ncol=n)
for (i in 1:n) {
for (j in 1:n) {
z=c(x[i],y[j])
if ( sum(z^2)<rho^2 ) {
theta=2*asin( 0.5*sqrt(sum(z^2)) )
g2x=x[i] ; g3x=y[j]
xa=k*cos(theta)+b*(g2x^2-g3x^2)-con
mat[i,j]=exp(xa) } } }
contour(x,y,mat) }
```

6.2.18 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 6.1 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

$$z1 = \rho \cos \phi, \;\; z2 = \rho \sin \phi, \tag{6.18}$$

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 6.1.

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 (6.18). For the next two functions we need the functions *euclid*, *rotation*, *vmf* and *euclid.inv*.

```
lambert=function(y) {
## y contains the data in degrees, lattitude and longitude
u=euclid(y) ## transform the data into euclidean coordinates
m=colMeans(u)
m=m/sqrt(sum(m^2)) ## the mean direction
b=c(0,0,1) ## the north pole towards which we will rotate the data
H=rotation(m,b) ## the rotation matrix
u1=u%*%t(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 rads
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
z=cbind(z1,z2) ## the Lambert equal area projected data on the disk
z  }
```

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 un-project 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)
if (ncol(z)==1) z=t(z)
long=( atan(z[,2]/z[,1])+pi*I(z[,1]<0) )%%(2*pi)
lat=2*asin(0.5*sqrt(rowSums(z^2)))
u=cbind(lat,long) ## the data on the sphere in rads
u=u*180/pi ## from rads 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 their mean direction is mu
u=y%*%t(H)
u=euclid.inv(u)
u }
```

Log of changes from version to version

After a massive public demand (just 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 (I do not remember the previous changes).

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 centring 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 (version 0.4-3)*. CA: Department of Environmental Sciences, Informatics and Statistics, Ca' Foscari University, Venice, Italy. UL: Department of Statistics, California Polytechnic State University, San Luis Obispo, California, USA.

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.

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. (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.

Bengtsson, H. (2014). *R.matlab: Read and write of MAT files together with R-to-MATLAB connectivity*. R package version 2.2.3.

Brown, P. J. and Zidek, J. V. (1980). Adaptive multivariate ridge regression. *The Annals of Statistics*, 8(1):64–74.

Butler, R. W. (2007). *Saddlepoint approximations with applications*. Cambridge University Press.

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.

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.

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*.

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. 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.

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.

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.

Hornik, K. and Grn, B. (2012). *movMF: Mixtures of von Mises-Fisher Distributions*. R package version 0.1-0.

Hornik, K. and Grn, B. (2013). *movMF: Mixtures of von Mises-Fisher Distributions*. R package version 0.1-2.

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, S. R. and R., S. Y. (1988). A correlation coe cient for angular variables. *Statistical Theory and Data Analysis*, 2:349–364.

Jammalamadaka, S. R. and Sengupta, A. (2001). *Topics in circular statistics*. World Scientific.

Johnson, R. A. and Wichern, D. W. (2002). *Applied multivariate statistical analysis*. New Jersey: Prentice Hall.

Jolliffe, I. T. (2005). *Principal component analysis*. New York: Springer-Verlag.

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.

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.

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.

Lancaster, H. O. (1965). The helmert matrices. *American Mathematical Monthly*, 72(1):4–12.

Le, H. and Small, C. G. (1999). Multidimensional scaling of simplex shapes. *Pattern Recognition*, 32(9):1601–1613.

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.

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.

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. (2013). Regression analysis of multivariate fractional data. *Econometric Reviews*, To appear.

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142.

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.

Oliveira, M., Crujeiras, R. M., and Rodrguez-Casal, A. (2013). *NPCirc: Nonparametric Circular Methods*. R package version 2.0.0.

Pawlowsky Glahn, V., Egozcue, J. J., and Tolosana Delgado, R. (2007). *Lecture notes on compositional data analysis*.

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.

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.

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.

Rousseeuw, P. J., Van Aelst, S., Van Driessen, K., and Gulló, J. A. (2004). Robust multivariate regression. *Technometrics*, 46(3).

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.

Sharp, W. (2006). The graph median–a stable alternative measure of central tendency for compositional data sets. *Mathematical geology*, 38(2):221–229.

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.

Stephens, M. A. (1979). Vector correlation. *Biometrika*, 66(1):41–48.

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*, pages 359–372.

Watson, G. S. (1983). *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.

Yee, T. W. (2010). The VGAM package for categorical data analysis. *Journal of Statistical Software*, 32(10):1–34.