Supplementary Material

Mandible shape reflects ecotypic divergence and population dynamics in an Arctic carnivore: the case of the Icelandic Arctic fox.

Anna Bára Másdóttir[1,2], Snæbjörn Pálsson[2], Nicolas Lecomte[3], Bruce J. McAdam[1], Ester Rut Unnsteinsdóttir[1].

Authors and Affiliations

1) Natural Science Institute of Iceland, 210 Garðabær, Iceland.

2) Institute of Life and Environmental Sciences, University of Iceland, 102 Reykjavík, Iceland.

3) Canada Research Chair in Polar and Boreal Ecology and Centre d'Études Nordiques, Département de Biologie, Université de Moncton, Moncton, New Brunswick, Canada E1A 3E9.

Corresponding author: Anna Bára Másdóttir, anna.b.masdottir@natt.is / annab@hi.is.

The following code originates from the shapeR package, as we use some of its internal functions for our modified version used to detect mandible outlines and generate the Fourier coefficients. As shapeR was originally developed to analyse fish otoliths, references to otoliths appear throughout the code.

```
#####################################
##  OTOLITH SHAPE ANALYSIS
##      Morphological functions
##      by Lisa Anne Libungan
##      and Snæbjörn Pálsson
##
##      modified by Anna Bára Másdóttir
##      and Bruce J McAdam
##
#####################################



.shapeR.check.outline.list = function(object)
{
```

```
  if( length(object@outline.list) == 0 || (length(object@outline.list) ==1  &&
length(object@outline.list[[1]]) ==0 ))
    stop("No outlines. Need to run detect.outline?")
}

.shapeR.check.shape.coefs = function(object)
{
  if( length(object@shape.coef.raw) == 0 )
    stop("No shape coefficients. Need to run getShapeCoefficients?")
}

.shapeR.check.shape.coefs.std = function(object)
{
  if( length(object@wavelet.coef.std) == 0 || length(object@fourier.coef.std) == 0)
    stop("Coefficients have not been standardized. Need to run stdCoefs?")
}

.shapeR.rbind.fill.matrix = function (...)
{
```

```
matrices <- list(...)
if (length(matrices) == 0)
  return()
if (is.list(matrices[[1]]) && !is.matrix(matrices[[1]])) {
  matrices <- matrices[[1]]
}
tmp <- unlist(lapply(matrices, is.factor))
if (any(tmp)) {
  stop("Input ", paste(which(tmp), collapse = ", "), " is a factor and ",
       "needs to be converted first to either numeric or character.")
}
matrices[] <- lapply(matrices, as.matrix)
lcols <- lapply(matrices, function(x) dimnames(x)[[2]])
cols <- unique(unlist(lcols))
rows <- unlist(lapply(matrices, nrow))
nrows <- sum(rows)
output <- matrix(NA, nrow = nrows, ncol = length(cols))
colnames(output) <- cols
```

```
  pos <- matrix(c(cumsum(rows) - rows + 1, rows), ncol = 2)
  for (i in seq_along(rows)) {
    rng <- seq(pos[i, 1], length = pos[i, 2])
    output[rng, lcols[[i]]] <- matrices[[i]]
  }
  output
}
```

```
#####################################################################
########
#
# Based on Springer, New York.
# Error in the function in the book, wrote a new function
# See: https://sites.google.com/site/raduiovita/morphometrics-software
# f5.10. this function has been slightly modified
#
#####################################################################
#######
```

```
.shapeR.NEF<-function(M, n=dim(M)[1]/2,start=F)

{

  ef<-.shapeR.efourier(M,n)

  A1<-ef$an[1]; B1<-ef$bn[1]

  C1<-ef$cn[1]; D1<-ef$dn[1]

  theta<-(0.5*atan(2*(A1*B1+C1*D1)/(A1^2+C1^2-B1^2-D1^2)))%%pi



  phaseshift<-matrix(c(cos(theta),sin(theta),-sin(theta),cos(theta)),2,2)

  M2<-matrix(c(A1,C1,B1,D1),2,2)%*%phaseshift

  v<-apply(M2^2,2, sum)

  if (v[1]<v[2]){theta<-theta+pi/2}

  theta<-(theta+pi/2)%%pi-pi/2



  Aa<-A1*cos(theta)+B1*sin(theta)

  Cc<-C1*cos(theta)+D1*sin(theta)

  scale<-sqrt(Aa^2+Cc^2)

  psi<-atan(Cc/Aa)
```

```
  size<-(1/scale)
  rotation<-matrix(c(cos(psi),-sin(psi),sin(psi),cos(psi)),2,2)
  A<-B<-C<-D<-numeric(n)
  if (start){theta<-0}
  for (i in 1:n){
    mat<-
size*rotation%*%matrix(c(ef$an[i],ef$cn[i],ef$bn[i],ef$dn[i]),2,2)%*%matrix(c(cos(i*thet
a),sin(i*theta),-sin(i*theta),cos(i*theta)),2,2) #this resizes the first harmonic so its major
axis is equal to 1; this is the classical way to standardize (normalize) EF coefficients, but
may not be appropriate
    #mat<-
rotation%*%matrix(c(ef$an[i],ef$cn[i],ef$bn[i],ef$dn[i]),2,2)%*%matrix(c(cos(i*theta),sin
(i*theta),-sin(i*theta),cos(i*theta)),2,2) #without the size correction, just the rotation
    A[i]<-mat[1,1]
    B[i]<-mat[1,2]
    C[i]<-mat[2,1]
    D[i]<-mat[2,2]}
  list(A=A,B=B,C=C,D=D,size=scale,theta=theta,psi=psi,ao=ef$ao,co=ef$co)
}
```

```
########### Elliptic Fourier ######################


#Based on Claude, J. 2008. Morphometrics with R. Springer, New York.

.shapeR.efourier<-function(M, n=dim(M)[1]/2)

{

  p<-dim(M)[1]

  Dx<-M[,1]-M[c(p,(1:p-1)),1]

  Dy<-M[,2]-M[c(p,(1:p-1)),2]

  Dt<-sqrt(Dx^2+Dy^2)

  # Error in this function. Same coordinate, side by side, divided by zero

  # Fixed with this

  Dt[which(Dt==0)] = 1

  t1<-cumsum(Dt)

  t1m1<-c(0, t1[-p])

  T<-sum(Dt)

  an<-bn<-cn<-dn<-numeric(n)

  for (i in 1:n){

    an[i]<- (T/(2*pi^2*i^2))*sum((Dx/Dt)*
```

```
                        (cos(2*i*pi*t1/T)-cos(2*pi*i*t1m1/T)))
    bn[i]<- (T/(2*pi^2*i^2))*sum((Dx/Dt)*
                        (sin(2*i*pi*t1/T)-sin(2*pi*i*t1m1/T)))
    cn[i]<- (T/(2*pi^2*i^2))*sum((Dy/Dt)*
                        (cos(2*i*pi*t1/T)-cos(2*pi*i*t1m1/T)))
    dn[i]<- (T/(2*pi^2*i^2))*sum((Dy/Dt)*
                        (sin(2*i*pi*t1/T)-sin(2*pi*i*t1m1/T)))
  }
  ao<-2*sum(M[,1]*Dt/T)
  co<-2*sum(M[,2]*Dt/T)
  list(ao=ao,co=co,an=an,bn=bn,cn=cn,dn=dn)
}

#As in Claude, J. 2008. Morphometrics with R. Springer, New York.
.shapeR.iefourier<-function(an,bn,cn,dn,k,n,ao=0,co=0)
{
  theta<-seq(0,2*pi, length=n+1)[-(n+1)]
  harmx <- matrix (NA, k, n)
```

```
  harmy <- matrix (NA, k, n)
  for (i in 1:k){
    harmx[i,]<-an[i]*cos(i*theta)+bn[i]*sin(i*theta)
    harmy[i,]<-cn[i]*cos(i*theta)+dn[i]*sin(i*theta)
  }
  x<-(ao/2) + apply(harmx, 2, sum)
  y<-(co/2) + apply(harmy, 2, sum)
  list(x=x, y=y)
}
```

```
.shapeR.inverse.wavelet =
function(coefs,m.o,plotDetail=T,return.polar=F,num.points=512*2,n.levels=5)
{
  wwd = wd(rep(0,num.points))

  for(level in 0:(log2(num.points)-1))
    wwd = putD(wwd,level=level,v=rep(0,2^level))
```

```
  wwd = putD(wwd,level=0,v=coefs[1])

  wwd = putD(wwd,level=1,v=coefs[2:3])



  for(level in 2:n.levels)

    wwd = putD(wwd,level=level,v=coefs[(sum(2^(c(1:(level-1)))):sum(2^(c(1:level))))[-1]+1])



  rad=wr(wwd)+m.o

  angle = seq(-pi,pi,by=2*pi/num.points)[-1]



  if(return.polar==F)

    list(X=rad*cos(angle),Y=rad*sin(angle))

  else

    list(angle=angle,radii=rad)

}




.shapeR.resizeImage = function(im, size.ratio) {

  # function to resize an image
```

```
  # im = input image, w.out = target width, h.out = target height

  # Bonus: this works with non-square image scaling.



  # initial width/height

  w.in = nrow(im)

  h.in = ncol(im)



  w.out = floor(w.in*size.ratio)

  h.out = floor(h.in*size.ratio)

  # Create empty matrix

  im.out = matrix(rep(0,w.out*h.out), nrow =w.out, ncol=h.out )




  # Do resizing -- select appropriate indices

  im.out <- im[ floor(size.ratio* 1:w.out), floor(size.ratio* 1:h.out)]



  return(im.out)

}
```

```
.shapeR.find.outline=function(skra,threshold=.15,mouse.click=F,main=NA,display.imag
es=F)
{
  #M<- read.pnm(skra) # skra innan ""
  M<- readJPEG(skra) # skra innan ""
  if(length(dim(M))>2){
    M<- suppressWarnings(pixmapRGB(M[,,1:3]))
    M<- as(M, "pixmapGrey")
  }
  else{
    M<- suppressWarnings(pixmapGrey(M))
  }
  M@grey[which(M@grey<=threshold)]<-0
  M@grey[which(M@grey>threshold)]<-1

  if(mouse.click== T || display.images==T)
    plot(M,main=main)
```

```
  start=list(x=NA,y=NA)
  if(mouse.click==T)
  {
    # uses coordinates to start, need to select one spot
    start<-locator(1)
  }
  else{
    start$x=M@size[2]/2
    start$y=M@size[1]/2
  }

  Rc<-.shapeR.Conte(c(round(start$x),round(start$y)),M@grey)
  if(mouse.click == T || display.images==T)
    lines(Rc$X, Rc$Y, lwd=2,col="red")
  return(Rc)
}

# From Claude, J. 2008. Morphometrics with R. Springer, New York.
```

```
.shapeR.Conte=function(x, imagematrix)

{

 I<-imagematrix

 x<-rev(x)

 x[1]<-dim(I)[1]-x[1]

 while (abs(I[x[1],x[2]]-I[x[1],(x[2]-1)])<0.1)

 {x[2]<-x[2]-1;

 #if(x[2]<1)

 #  simpleError("Problem in detecting outline when collecting coordinates.")

 }

 a<-1

 M<-matrix(c(0,-1,-1,-1,0,1,1,1,1,1,0,-1,-1,-1,0,1),2,8,byrow=T)

 M<-cbind(M[,8],M,M[,1])

 X<-0; Y<-0;

 x1<-x[1]; x2<-x[2]

 SS<-NA; S<-6



 while ((any(c(X[a],Y[a])!=c(x1,x2) ) | length(X)<3))
```

```
{


 if (abs(I[x[1]+M[1,S+1],x[2]+M[2,S+1]]-I[x[1],x[2]])<0.1)
 {
  a<-a+1;X[a]<-x[1];Y[a]<-x[2];x<-x+M[,S+1]
  SS[a]<-S+1; S<-(S+7)%%8
 }
 else if (abs(I[x[1]+M[1,S+2],x[2]+M[2,S+2]]
          -I[x[1],x[2]])<0.1)
 {
  a<-a+1;X[a]<-x[1];Y[a]<-x[2];x<-x+M[,S+2]
  SS[a]<-S+2; S<-(S+7)%%8
 }
 else if (abs(I[x[1]+M[1,(S+3)],x[2]+M[2,(S+3)]]
          -I[x[1],x[2]])<0.1)
 {
  a<-a+1;X[a]<-x[1];Y[a]<-x[2];x<-x+M[,(S+3)]
  SS[a]<-S+3; S<-(S+7)%%8
```

```
    }
    else S<-(S+1)%%8

    if(a>(dim(I)[1]+dim(I)[2])*1e2)
    {
      X[a]=x1
      Y[a]=x2
    }
  }
  list(X=(Y[-1]), Y=((dim(I)[1]-X))[-1])
}

#################################################################
#
#  Function to get the maximum length, height, area etc.
#  Use gpc.poly to get perimeter
#
#################################################################
```

```
.shapeR.rotate.svd=function(outline)
{
  M=cbind(outline$X,outline$Y)
  Ma=M%*%svd(var(M))$u
  Ma[,1]=-Ma[,1]
  return(Ma)
}

.shapeR.otolith.image.parameters=function(x)
{
  xbil=max(diff(range(x[,1])))
  ybil=max(diff(range(x[,2])))
  area = .shapeR.area(x)
  perimeter=.shapeR.fourier(x,16)$perimeter
  return(list(width=xbil,height=ybil,area=area,perimeter=perimeter))
}

.shapeR.area = function(m){
```

```
 n = dim(m)[1]



 x1 = m[,1]

 y1 = m[,2]



 x2 = c(m[-1,1],m[1,1])

 y2 = c(m[-1,2],m[1,2])



 tarea = abs(sum((x2-x1)*(y2+y1))/2)



 return(tarea)

}




.shapeR.get.otolith.image.parameters = function(outline){

  x = .shapeR.rotate.svd(outline)

  .shapeR.otolith.image.parameters(x)

}
```

```
.shapeR.fourier=function(M,n)

{

  p<-dim(M)[1]

  an<-numeric(n)

  bn<-numeric(n)

  tangvect<-M-rbind(M[p,],M[-p,])

  perim<-sum(sqrt(apply((tangvect)^2, 1, sum)))

  v0<-(M[1,]-M[p,])

  tet1<-Arg(complex(real=tangvect[,1],

                    imaginary = tangvect [,2]))

  tet0<-tet1[1]

  t1<-(seq(0, 2*pi, length=(p+1)))[1:p]

  phi<-(tet1-tet0-t1)%%(2*pi)

  ao<- 2* sum(phi)/p

  for (i in 1:n){

    an[i]<- (2/p) * sum( phi * cos (i*t1))
```

```
    bn[i]<- (2/p) * sum( phi * sin (i*t1))
  }
  list(ao=ao, an=an, bn=bn, phi=phi, t=t1,perimeter=perim, thetao=tet0)
}

#Claude, J. 2008. Morphometrics with R. Springer, New York., page 53
.shapeR.regularradius<-function(Rx, Ry, n)
{
  le<-length(Rx)
  M<-matrix(c(Rx, Ry), le,2)
  M1<-matrix(c(Rx-mean(Rx), Ry-mean(Ry)), le,2)
  V1<-complex(real=M1[,1], imaginary=M1[,2])
  M2<-matrix(c(Arg(V1), Mod(V1)), le,2)
  V2<-NA
  #The following code finds the indices of the nearest pixel on the outline using the
angul
  #increment.
  for (i in 0:(n-1))
  {
```

```
  V2[i+1]<-which.max((cos(M2[,1]-2*i*pi/n)))
 }
 V2<-sort(V2)

 ord = order(M2[V2,1])

list("pixindices"=V2[ord],"radii"=M2[V2[ord],2],"coord"=M1[V2[ord],],"angle"=M2[V2[ord
],1])
}

####### Center images, find centroid
.shapeR.centroid=function(outline)
{
 M=outline
 M$X=outline$X-mean(outline$X)
 M$Y=outline$Y-mean(outline$Y)
 #plot(utl2$X,utl2$Y,type="l")
 return(M)
```

```
}



###########

#LLEONART (2000)

.shapeR.standard.coef=function(y.in,in.class,x,std.type="mean",is.na.classes,p.crit)

{

  if(is.na.classes)

    in.class = rep("1",length(x))

  #y=y.in+abs(y.in)+.1

  y=y.in+ifelse(min(y.in)<0,-min(y.in),0)+.1

  unique(in.class)->class

  mean.x=tapply(x,factor(in.class),mean)

  nc = length(class)

  if(std.type=="mean")

    mean.x = rep(mean(mean.x),nc)

  if(std.type=="min")

    mean.x = rep(min(mean.x),nc)
```

```
if(std.type=="max")
  mean.x = rep(max(mean.x),nc)



lm.y=list()
for(i in 1:nc) lm.y[[i]]=lm(log(y[in.class==class[i]])~log(x[in.class==class[i]]))
#Default regression set as 0, i.e. no standardization.
b=rep(0,nc)
for(i in 1:nc)
{
  p.value.b = summary(lm.y[[i]])$coefficients["log(x[in.class == class[i]])","Pr(>|t|)"]
  #If p value from regression is less than 0.05, we keep the regression coefficient
  if(is.finite(p.value.b) &&  p.value.b<p.crit){
    #print(paste(class[i],"p.value=",p.value.b,"b=",lm.y[[i]]$coefficients[2]))
    b[i] = lm.y[[i]]$coefficients[2]
  }
}
```

```
  yy=y
  for(i in 1:nc)
  {
    ind = in.class==class[i]
    yy[ind]=y[ind]*(mean.x[i]/x[ind])^b[i]
  }
  return(yy)



}
```

```
.shapeR.coef.standardize.f =
function(coef,classes,std.by,std.type="mean",is.na.classes=F,p.crit=0.05,bonferroni=TR
UE,...)
{

  coef.standard = matrix(NA,nrow=dim(coef)[1],ncol=dim(coef)[2])
```

```
k=1
if(bonferroni)
  k = dim(coef)[2]


r.y = c()
for(i in 1:ncol(coef))
{
  #ANCOVA
  if(is.na.classes){
    t.ancova = aov(coef[,i]~std.by)
    if(summary(t.ancova)[[1]]["std.by","Pr(>F)"]<p.crit/k)
    {
      r.y = c(r.y,i)
    }
  }
  else{
    t.ancova = aov(coef[,i]~classes*std.by)
```

```
    if(summary(t.ancova)[[1]]["classes:std.by","Pr(>F)"]<p.crit/k)
    {
      r.y = c(r.y,i)
    }
  }

coef.standard[,i]= .shapeR.standard.coef(coef[,i],classes,std.by,std.type,is.na.classes,p.crit)
 }

 if(is.null(r.y))
 {
  message("No coefficients removed")
  return(list(coef.standard=coef.standard,r.y))
 }

 message("Removed coefficients: ",appendLF=F)
 message(paste(r.y,collapse=","))
```

```
  return(list(coef.standard=coef.standard[,-r.y],r.y=r.y))


}


.shapeR.variation.among = function(in.classes,coef)
{
  classes = factor(in.classes)


  mean.var=function(x) mean(tapply(x,classes,var))
  mean.var.among=function(x) var(tapply(x,classes,mean))


  a = length(levels(classes))
  na = tapply(coef[,1],classes,length)


  n0 = 1/(a-1)*(dim(coef)[1] - sum(na^2)/sum(na))


  var.within=apply(coef,2,mean.var)
```

```
  var.among=n0*apply(coef,2,mean.var.among)

  add.var.component=(var.among-var.within)/n0

  return(add.var.component/(add.var.component+var.within))
}

#####################################################################
######
#
# Based on Claude, J. 2008. Morphometrics with R. Springer, New York.
#
#####################################################################
#######
.shapeR.smoothout<-function(M.in, n)
{
  M = cbind(M.in$X,M.in$Y)
  p<-dim(M)[1]
  a<-0
```

```
  while (a<=n)
  {
    a<-a+1
    Ms<-rbind(M[p,],M[-p,])
    Mi<-rbind(M[-1,],M[1,])
    M<-M/2+Ms/4+Mi/4
  }
  return(data.frame(X=M[,1],Y=M[,2]))
}
```

The following code is the procedure to remove the teeth of the mandibles to eliminate the variance they brought.

```
# The procedure to identify tooth start and end and replace them with
# a straight line to eliminate variance in teeth presence/absence

library(dplyr)
library(ggplot2)
library(jpeg)

cat("Loading data...\n")
load("shapeData.rdata")
cat("loaded\n")

width <- 5472
height <- 3648

ids <- levels(outlines_pixels$id)
```

```
# FIRST TIME YOUR RUN THIS CODE, toothTable SHOULD BE EMPTY

#toothTable <- data.frame()

# IF YOU ALREADY HAVE A toothTable, LOAD IT HERE

#load("toothTable.rdata")




# should really randomise the order

for(i in sample(ids)[1]){

  if(!(i %in% toothTable$id)){

    thisPic <- subset(outlines_pixels, id==i)

    jpg <- readJPEG(file.path("Original", paste0(i, ".jpg")))



    accepted <- F

    while(!accepted){

      windows() # open graphics device (r studio doesn't work properly!)

      plot(thisPic$x, thisPic$y, asp=1)

      rasterImage(jpg, xleft=0, xright=width, ybottom=0, ytop=height)
```

```
  lines(thisPic$x, thisPic$y, col="red")
  cat("Click start of teeth (behind front teeth)\n")
  click1 <- identify(thisPic$x, thisPic$y, n=1, plot=F)
  points(thisPic$x[click1], thisPic$y[click1], col="green", pch=19)
  cat("Click end of teeth (behind M3)\n")
  click2 <- identify(thisPic$x, thisPic$y, n=1, plot=F)
  x1 <- thisPic$x[click1]
  x2 <- thisPic$x[click2]
  y1 <- thisPic$y[click1]
  y2 <- thisPic$y[click2]
  lines(c(x1, x2), c(y1, y2), col="red")
  response <- readline(prompt="accept? (y/n)")
  accepted <- response=="y"
}
print("accepted")
dev.off()
toothTable <- rbind(toothTable,
                    data.frame(id=i,
```

```
                        start = click2,
                        end=click1,
                        x1 = x2, y1=y2, x2=x1, y2=y1))

  }
}

save(toothTable, file="toothTable.rdata")

# Now actually edit the images
removeTeeth <- function(i){
  thisPic <- subset(outlines_pixels, id==i)
  start <- toothTable$start[toothTable$id==i]
  end <- toothTable$end[toothTable$id==i]
  x1 <- toothTable$x1[toothTable$id==i]
  x2 <- toothTable$x2[toothTable$id==i]
  y1 <- toothTable$y1[toothTable$id==i]
  y2 <- toothTable$y2[toothTable$id==i]
```

```
straightline <- lm(c(y1, y2) ~ c(x1, x2))

insertX <- x2:x1

insertY <- coef(straightline)[1] + coef(straightline)[2] * insertX

newX <- c(thisPic$x[1:(start-1)],

          insertX,

          thisPic$x[(end+1):nrow(thisPic)])



newY <- c(thisPic$y[1:(start-1)],

          insertY,

          thisPic$y[(end+1):nrow(thisPic)])

newPic <- data.frame(id=i,

              x = newX,

              y = newY)



plot(newPic$x, newPic$y)

jpg <- readJPEG(file.path("Original", paste0(i, ".jpg")))

rasterImage(jpg, xleft=0, xright=width, ybottom=0, ytop=height)

lines(newPic$x, newPic$y, col="red")
```

```
  plot(newPic$x, newPic$y, asp=1, main=i)

  return(thisPic)



}



toothlessImages <- data.frame()



for(i in toothTable$id){

  newImage <- removeTeeth(i)

  toothlessImages <- rbind(toothlessImages, newImage)

}
```

The following is the modified code from the shapeR function to generate the Fourier coefficients in this study.

```
# The shapeR library requires files to be in a multi-level directory structure

# with metadata in a CSV file and builds a complex object based around this structure.

# Unfortunately this does not work well with other libraries such as

# dplyr and if metadata is obtained from other sources such as an SQL database.

#

# This script is based on shapeR code, and uses its internal functions,

# but builds data into data frames and matrices without embedding them in

# a shapeR object.

# The code runs for all images in a single folder, and assumes that filename

# is an id that can be later linked to other data e.g. using dplyr::left_join()



library(dplyr)

library(ggplot2)

library(jpeg)

library(pixmap)
```

```
# Trying to reduce reliance on this, but use it for some functions

source("shapeR_functions.R")




# SET UP FOLDER ETC. ##############################




folderName <- normalizePath("YourFolderName/")

metadataName <- "Metadata.csv"



# parameters relating to image size and quality

cal <- 395 # pixels per cm on all images

threshold <- 0.5 # grey threshold for outline detection



# parameters relating to fourier transform

nFDs <- 32 # Maximum number of harmonics
```

```
# Various graphs are plotted when running this code

# this tells us the maximum number of images to include

maxToPlot <- 12



# DETECT OUTLINE ################################



# I have put all the (fixed) images in the same folder

fnames = list.files(folderName, pattern="\\.(jpg|jpeg|png)",ignore.case=T)



getOutlinesFromFiles <-

  function(fnames){



    strip.fnames = gsub(".(jpg|jpeg|png)","",fnames,ignore.case=T)



    outlines_pixels <- data.frame()



    for(i in 1:length(fnames)){
```

```
fname <- fnames[i]

id <- strip.fnames[i]

cat("Reading outline from ", fname, "(", i, "out of", length(fnames), ")\n")

Rc = as.data.frame(.shapeR.find.outline(file.path(folderName, fname),

                    threshold=threshold,

                    main=fname,

                    mouse.click=F,

                    display.images=F)) %>%

  rename(x=X, y=Y)



Rc$id <- id



# THERE IS A PROBLEM HERE, IN THAT SOME OF THE IMAGES

# HAVE DIFFERENT START POINTS

# THE FOLLOWING CODE ENSURES THAT THE START POINT IS ALWAYS

# THE LEFTMOST (LOWEST X-COORD) POINT ON THE OUTLINE

leftmost <- which.min(Rc$x)

Rc <- rbind(Rc[leftmost:nrow(Rc),], # from leftmost to end
```

```
                Rc[1:(leftmost-1),]) # from current beginning to leftmost


      outlines_pixels <- rbind(outlines_pixels, Rc)


    }


    outlines_pixels <- outlines_pixels %>%
      #mutate(id = factor(as.character(outlines_pixels$id))) %>%
      select(id, x, y)


    return(outlines_pixels)


}


outlines_pixels <- getOutlinesFromFiles(fnames)


cat("Plotting images\n")
```

```
# Want to check it identifies appropriate start points
firstPoints <- outlines_pixels %>%
  group_by(id) %>%
  filter(row_number()==1) %>%
  ungroup

idsToPlot <- unique(outlines_pixels$id)
if(length(idsToPlot) > maxToPlot){
  idsToPlot <- sample(idsToPlot, maxToPlot)
}

print(ggplot(subset(outlines_pixels, id %in% idsToPlot), aes(x,y, colour=id)) +
        geom_path() +
        geom_point(data=subset(firstPoints, id %in% idsToPlot)) +
        coord_fixed() +
        facet_wrap(vars(id)) +
        ggtitle("Original outlines, pixel scale"))
```

```
# MODIFY THE IMAGES TO REMOVE TEETH ############

load("toothTable.rdata")

removeTeeth <- function(outlines_pixels, toothTable){

  # Can only do this for images that have outlines and tooth data
  ids <- intersect(unique(outlines_pixels$id), toothTable$id)

  toothlessOutlines <- data.frame()

  for(i in ids){
    cat("Removing teeth from", i, "\n")
    thisPic <- subset(outlines_pixels, id==i)
    start <- toothTable$start[toothTable$id==i]
    end <- toothTable$end[toothTable$id==i]
    x1 <- toothTable$x1[toothTable$id==i]
```

```
x2 <- toothTable$x2[toothTable$id==i]

y1 <- toothTable$y1[toothTable$id==i]

y2 <- toothTable$y2[toothTable$id==i]

straightline <- lm(c(y1, y2) ~ c(x1, x2))

insertX <- x2:x1

insertY <- coef(straightline)[1] + coef(straightline)[2] * insertX

newX <- c(thisPic$x[1:(start-1)],

        insertX,

        thisPic$x[(end+1):nrow(thisPic)])



newY <- c(thisPic$y[1:(start-1)],

        insertY,

        thisPic$y[(end+1):nrow(thisPic)])

newPic <- data.frame(id=i,

                x = newX,

                y = newY)



#plot(newPic$x, newPic$y)
```

```
    #jpg <- readJPEG(file.path("Original", paste0(i, ".jpg")))
    #rasterImage(jpg, xleft=0, xright=width, ybottom=0, ytop=height)
    #lines(newPic$x, newPic$y, col="red")
    #plot(newPic$x, newPic$y, asp=1, main=i)

    toothlessOutlines <- rbind(toothlessOutlines, newPic)

  }
  return(toothlessOutlines)

}

outlines_toothless_pixels <- removeTeeth(outlines_pixels, toothTable)

print(ggplot(subset(outlines_toothless_pixels, id %in% idsToPlot), aes(x,y, colour=id)) +
        geom_path() +
        geom_point(data=subset(firstPoints, id %in% idsToPlot)) +
        coord_fixed() +
```

```
        facet_wrap(vars(id)) +
        ggtitle("toothless outlines, pixel scale"))


# RESCALE IMAGES TO CM ############################


outlines_cm <- outlines_pixels %>%
  mutate(x = x/cal, y = y/cal)


outlines_toothless_cm <- outlines_toothless_pixels %>%
  mutate(x = x/cal, y = y/cal)


print(ggplot(subset(outlines_cm, id %in% idsToPlot), aes(x,y, colour=id)) +
        geom_path() +
        geom_path(data = subset(outlines_toothless_cm, id %in% idsToPlot), alpha=0.5) +
        coord_fixed() +
        facet_wrap(vars(id)) +
        ggtitle("comparision of original and toothless outlines, cm scale"))
```

```
# GENERATE FOURIER COEFFICIENTS ###################


# column names
# these have _ characters to make it easier to extract the
# FD number and a, b, c, d
fourier.coef.raw.names = c(paste("FD",1:nFDs,'a',sep="_"),
                           paste("FD",1:nFDs,'b',sep="_"),
                           paste("FD",1:nFDs,'c',sep="_"),
                           paste("FD",1:nFDs,'d',sep="_"))


# We will produced 2 sets of fourier coefficients
# the first matches the original image scale (cm)
# the second is rescaled (this is for analysis)
# This also returns a modified version of the images
# shifted to the same center as the fourier
generateCourierCoefs <- function(outlines_cm){


    fourier.coef.raw.cm = matrix(NA,nrow=0,ncol= nFDs*4)
```

```
fourier.coef.raw <- matrix(NA,nrow=0,ncol= nFDs*4)

outlines_shifted <- data.frame()



#Set column names fofourier.coef.raw.cmr 32 Fourier harmonics

colnames(fourier.coef.raw.cm) = fourier.coef.raw.names

colnames(fourier.coef.raw) = fourier.coef.raw.names



ids <- unique(outlines_cm$id)



for(i in ids){



  cat("Doing fourier coefficients for", i, "\n")



  utlina = filter(outlines_cm, id==i) %>% dplyr::select(x, y)



  #Fourier coeffs

  # This does not rescale or rotate (cm scale)

  efourier <- .shapeR.efourier(utlina, n=nFDs)
```

```
# this one rescales and rotates

N=.shapeR.NEF(utlina,n=nFDs)

  # n is a parameter in the NEF function.

  # Gives information about how many harmonics is needed.

  # Possible to set as 32, 64, or skip it.



valuesCM <- matrix(c(efourier$an, efourier$bn, efourier$cn, efourier$dn), nrow=1)

colnames(valuesCM) <- colnames(fourier.coef.raw.cm)

rownames(valuesCM) = i

fourier.coef.raw.cm <- rbind(fourier.coef.raw.cm, valuesCM)

ao <- efourier$ao

co <- efourier$co

#cat(ao, co, "\n")

# Now use the ao and co to shift the cm outline data to match the fds

outlines_shifted <-

  rbind(outlines_shifted,

        utlina %>%

    mutate(id = i,
```

```
            x = x-ao/2,

            y = y-co/2) )


    valuesRe = matrix(c(N$A,N$B,N$C,N$D),nrow=1)

    colnames(valuesRe) = colnames(fourier.coef.raw)

    rownames(valuesRe) = i

    fourier.coef.raw = rbind(fourier.coef.raw,valuesRe)


  }


  rownames(fourier.coef.raw) <- ids

  rownames(fourier.coef.raw.cm) <- ids


  return(list(fourier.coef.raw.cm = fourier.coef.raw.cm,

            fourier.coef.raw = fourier.coef.raw,

            outlines_shifted = outlines_shifted))

}
```

```
result <- generateCourierCoefs(outlines_cm)

cat("Now doing fourier on toothless images\n")

resultToothless <- generateCourierCoefs(outlines_toothless_cm)



outlines_cm <- result$outlines_shifted

fourier.coef.raw.cm <- result$fourier.coef.raw.cm

fourier.coef.raw <- result$fourier.coef.raw



outlines_toothless_cm <- resultToothless$outlines_shifted

fourier.coef.toothless.raw.cm <- resultToothless$fourier.coef.raw.cm

fourier.coef.toothless.raw <- resultToothless$fourier.coef.raw



rm(result, resultToothless) # tidy up



# sometimes it is useful to have separate rows

# for each FD (e.g. for dplyr and ggplot)

makeFDLongData <- function(coefs){

  coefs %>%
```

```
    as.data.frame %>%
    mutate(id=row.names(fourier.coef.raw)) %>%
    tidyr::pivot_longer(
      cols=starts_with("FD"),
      names_to = c("ignore", "fd", "abcd"),
      names_sep = "_") %>%
  select(-ignore) %>%
  mutate(fd = as.numeric(fd))
}


print(
  fourier.coef.raw %>%
    makeFDLongData() %>%
    filter(id %in% idsToPlot) %>%
    ggplot(aes(fd, value, colour=id)) +
        geom_point() +
        facet_wrap(vars(abcd), ncol=1) +
    ggtitle("FD values, original images")
```

```
  )


print(
  fourier.coef.toothless.raw %>%
    makeFDLongData() %>%
    filter(id %in% idsToPlot) %>%
    ggplot(aes(fd, value, colour=id)) +
    geom_point() +
    facet_wrap(vars(abcd), ncol=1)+
    ggtitle("FD values, toothless images")
)



# something goes wrong with the shifting in the toothless images
# the value of ao generated by .shapeR.efourier() is
# too small by about 2cm
# the following code is a bit of a hack that builds a correction
correction <-
```

```
  outlines_cm %>%
  group_by(id) %>%
  summarise(minXOriginal = min(x)) %>%
  left_join(outlines_toothless_cm %>%
              group_by(id) %>%
              summarise(minXToothless = min(x))) %>%
  mutate(xCorrection = minXToothless - minXOriginal)


outlines_toothless_cm <-
  outlines_toothless_cm %>%
  left_join(correction) %>%
  mutate(x = x - xCorrection) %>%
  select(id, x, y)



# diagnostic plots
source("reconstruct_from_fourier.r")
```

```
print(
  reconstructFromFouriers(fourier.coef.raw.cm[idsToPlot,,drop=F]) %>%
    ggplot(aes(x, y)) +
    geom_path(colour="black") +
    geom_path(data=subset(outlines_cm, id %in% idsToPlot), colour="red") +
    coord_fixed() +
    facet_wrap(vars(id)) +
    ggtitle("Reconstruct original outline from 32FDs")
)


print(
  reconstructFromFouriers(fourier.coef.toothless.raw.cm[idsToPlot,,drop=F]) %>%
    ggplot(aes(x, y)) +
    geom_path(colour="black") +
    geom_path(data=subset(outlines_toothless_cm, id %in% idsToPlot), colour="red") +
    geom_path(data=subset(outlines_cm, id %in% idsToPlot), colour="red", alpha=0.25) +
    coord_fixed() +
    facet_wrap(vars(id)) +
```

```
    ggtitle("Reconstruct toothless outline from 32FDs")

)




# BASIC SHAPE COEFS (area, length, width, perimeter) ###############

cat("Getting basic shape coefs (area, length width, perimeter\n")

getShapeCoefficients <- function(outlines){

    sp.coef <- data.frame(
      id = character(0),
      area = numeric(0),
      height = numeric(0),
      width = numeric(0),
      perimeter = numeric(0))

    for(i in unique(outlines$id)){
```

```
    utlina = filter(outlines, id==i)



    width <- diff(range(utlina$x))

    height <- diff(range(utlina$y))

    area <- .shapeR.area(cbind(utlina$x,utlina$y))

    perimeter =.shapeR.fourier(cbind(utlina$x,utlina$y),16)$perimeter

    sp.coef = rbind(sp.coef,

                data.frame(id = i,

                        area = area ,

                        height = height ,

                        width = width ,

                        perimeter = perimeter ))



  }

  return(sp.coef)



}
```

```
# images have been rescaled to cm so all values are cm or cm2

sp.coef.cm <- getShapeCoefficients(outlines_cm)



head(sp.coef.cm)



# and toothless

sp.coef.toothless.cm <- getShapeCoefficients(outlines_toothless_cm)



head(sp.coef.toothless.cm)









# STANDARDISE THE FOURIER COEFICIENTS ####################

# This is the set that should be used for analysis

# they are grouped by 1, 2, 3, 4, 5, ... instead of all 'a', all 'b' etc.

# 1a, 1b, 1c are not included as they are the same (-1, 0, 0) for all images

# WARNING: unlike shapeR, we keep all FDs.  If you only want to analyse a few

# e.g. 12 FDs then use matrix subscripting to extract these
```

```
# e.g. coef12 <- fourier.coef[,1:48]

# use colnames(fourier.coef) to check how many you need



cat("Standardising fourier coefficients\n")



indicesFD <- rep(0:(nFDs-1), each=4)

indicesABCD <- rep(seq(1,nFDs*4,by=nFDs), times=nFDs)

indicesReordered <- indicesFD+indicesABCD

droppedIndices <- indicesReordered[1:3]

indicesReordered <- indicesReordered[-c(1, 2, 3)] # drop the first 3

# Find the names of these columns (used when we need to restore)

droppedColumnNames <- dimnames(fourier.coef.raw)[[2]][droppedIndices]

rm(indicesFD, indicesABCD) # not needed



# THESE ARE THE ONES TO ANALYSE IN CAPSCALE ETC.

fourier.coef <- fourier.coef.raw[,indicesReordered]

fourier.coef.toothless <- fourier.coef.toothless.raw[,indicesReordered]
```

```
# SAVE THE DATA ###############################

cat("Saving data as", file.path(folderName, "shapeData.rdata"), "\n")

save(outlines_cm, outlines_pixels, # original outlines, different scales

    outlines_toothless_cm, outlines_toothless_pixels, # toothless

    fourier.coef.raw.cm, fourier.coef.raw, fourier.coef, # with teeth

    fourier.coef.toothless.raw.cm, fourier.coef.toothless.raw, fourier.coef.toothless, #
with teeth

    sp.coef.cm, sp.coef.toothless.cm,

    nFDs, droppedColumnNames, fourier.coef.raw.names,

    file=file.path(folderName, "shapeData.rdata"))



cat("Finished.\n")




# DIAGNOSTIC PLOTS ################################



# reconstruct from the standardised

print(
```

```
  reconstructFromFouriers(fourier.coef.raw[1,,drop=F]) %>%
    ggplot(aes(x, y, colour=id)) + geom_path() + coord_fixed() +
    ggtitle("example of a single reconstructed image")
)

print(
  reconstructFromFouriers(fourier.coef.raw[idsToPlot,,drop=F]) %>%
    ggplot(aes(x, y, colour=id)) + geom_path() + coord_fixed() +
    ggtitle("example of multiple reconstructed images")
)

print(
  reconstructFromFouriers(fourier.coef[1,, drop=F]) %>%
    ggplot(aes(x, y)) + geom_path() + coord_fixed() +
    ggtitle("example of reconstructed image from different FD set")
)

print(
```

```
  reconstructFromFouriers(fourier.coef[idsToPlot,, drop=F]) %>%
    ggplot(aes(x, y, colour=id)) + geom_path() + coord_fixed()+
    ggtitle("example of multiple reconstructed image from different FD set")
)

# reconstruct from the cm scale, and compare with original
print(
  reconstructFromFouriers(fourier.coef.raw.cm[1,,drop=F]) %>%
    ggplot(aes(x, y, colour=id)) +
      geom_path() +
      geom_path(data=subset(outlines_cm, id==row.names(fourier.coef.raw.cm)[1]),
lty=2)+
      coord_fixed() +
    ggtitle("example of single FD on cm scale against original outline")
)

# reconstruct from the cm scale, and compare with original
print(
  reconstructFromFouriers(fourier.coef.toothless.raw.cm[1,,drop=F]) %>%
```

```
  ggplot(aes(x, y, col=id)) +
  geom_path() +
  geom_path(data=subset(outlines_toothless_cm,
id==row.names(fourier.coef.toothless.raw.cm)[1]), lty=2)+
  coord_fixed()+
  ggtitle("example of single toothless FD on cm scale against original outline")
)

# Sequence of different FDs against the original outline
testKs <- c(1, 2, 3, 4, 8, 12, 24, 32)
outlinesForK <- data.frame()
for(k in testKs){
  outline <- reconstructFromFouriers(fourier.coef.raw.cm[idsToPlot[1],,drop=F], k=k)
  outline$k <- k
  outlinesForK <- rbind(outlinesForK, outline)
}

print(
  outlinesForK %>%
```

```
  ggplot(aes(x, y)) +
  geom_path() +
  geom_path(data=subset(outlines_cm, id==idsToPlot[1]), col="red")+
  coord_fixed() +
  facet_wrap(vars(k)) +
  ggtitle(paste0("Example of different numbers of FDs (", idsToPlot[1], ")"))
)

allMetadata <- read.csv(file.path(folderName, metadataName)) %>%
  mutate(picname = as.character(picname))

# This extracts metadata for the images in fourier.coef
# and in the correct order
imageMetadata <-
  data.frame(picname = row.names(fourier.coef)) %>%
  left_join(allMetadata)

popPlot <-
```

```
  averageFDs(fourier.coef, imageMetadata$pop) %>%
  reconstructFromFouriers() %>%
  rename(pop=id) %>%
  ggplot(aes(x, y, colour=pop)) + geom_path() + coord_fixed() +
  ggtitle("average FDs for pops")
print(popPlot)



imageMetadataToothless <-
  data.frame(picname = row.names(fourier.coef.toothless)) %>%
  left_join(allMetadata)


toothlessPopPlot <-
  averageFDs(fourier.coef.toothless, imageMetadataToothless$pop) %>%
  reconstructFromFouriers() %>%
  rename(group=id) %>%
  ggplot(aes(x, y, colour=group)) + geom_path() + coord_fixed()+
```

```
  ggtitle("average toothless FDs for pops")

print(toothlessPopPlot)
```

The following code is to reconstruct the outlines so ggplot can be used to plot them.

```
# The following code will reconstruct outlines from fouriers
# so that you can plot using ggplot
# it will work with either the raw coefficients (by cm or scaled)
# or the set of 125 (rather than 128 coefficients) that are used in analysis
# see examples below...

source("shapeR_functions.R")

nFDs <- 32

restoreCoefsToRaw <- function(fourier.coef){
  # To turn the coef matrix that is missing 1a, 1b and 1c, we add those
  # columns back in.  Values area always -1, 0, 0
  extra <- matrix(data=c(-1, 0, 0), byrow=T,
                  nrow=nrow(fourier.coef),
                  ncol=3 )
```

```
  dimnames(extra) <- list(
    as.vector(dimnames(fourier.coef)[[1]]),
    droppedColumnNames)

  # add the extra columns, and arrange in same order as fourier.coef.raw
  restored <- cbind(extra, fourier.coef)[,fourier.coef.raw.names,drop=F]

  return(restored)
}

# This will reconstruct multiple outlines from each row in
# a matrix of coefficients
# It will work with either coefficients in order 1a,1b,1c... or 1d,2a,2b,2c...
reconstructFromFouriers <-
  function(coef.f, k=32, n=1024){

    # work out if we have all the coefficients or 1a, 1b, 1c are missing
    if(ncol(coef.f) < nFDs*4){
```

```
  coef.f <- restoreCoefsToRaw(coef.f)
}

result <- data.frame()

for(i in 1:nrow(coef.f)){
  # extract the a, b, c, d coefs
  aCoefs <- coef.f[i,1:32]
  bCoefs <- coef.f[i,33:64]
  cCoefs <- coef.f[i,65:96]
  dCoefs <- coef.f[i,97:128]
  iw.utl <- as.data.frame(.shapeR.iefourier(aCoefs, bCoefs, cCoefs, dCoefs,
                                    k=k, n=n)) %>%
    mutate(id = row.names(coef.f)[i]) %>%
    select(id, x, y)

  result <- rbind(result, iw.utl)
```

```
    }

    return(result)

  }



# To average FDs from a group

averageFDs <- function(coef.f, group){

  result <- coef.f %>%

    as.data.frame() %>%

    mutate(group=factor(group)) %>%

    group_by(group) %>%

    summarise_all(mean) %>%

    select(-group) %>%

    as.matrix(dimnames=list(levels(factor(group)), colnames(coef.f)), rownames.force=T)

  # it does not do the row names correctly

  dimnames(result) <- list(levels(factor(group)), colnames(coef.f))

  return(result)

}
```