Elliptical Fourier Analysis

Ryan N. Felice

Loading Packages and Scripts

```
library(geomorph)
library(Momocs)
```

```
source('./utility_functions/MorphometricExtraction_Functions.r')
source('./utility_functions/MorphoFiles_Function.r')
source('./utility_functions/OutlineAnalysis_Functions.r')
```

Read data

Let's load in all the data for all the belemnite hooks. Note that we are reading in a single .nts file. A nice feature of the .nts file format for landmark data is that all data is stored in a single file, with one specimen per row. We are using the `readland.nts` function from the geomorph package to read the data.

```
Belemnite.Full<-readland.nts("./Data/Belemnite_SmoothedOutline.nts")
dim(Belemnite.Full)
```

```
[1]  70   2 174
```

Using the function `dim`, we can see have 70 outline points in two dimensions (x and y) and 174 observations. Actually what we have is 87 specimens that were each measured twice to help us examine user error/measurement error. The next step is to separate the two batches. The simple way to do that is that we know that the first 87 specimens are replicate 1 and the second 87 specimens are replicate two, so we *could* simply do the following:

```
Belemnite.R1<-Belemnite.Full[,,1:87]
Belemnite.R2<-Belemnite.Full[,,88:174]
```

However, there is a better way: we can use the names of the specimens to sort them. This protects us from mistakes, especially when we have many hundreds or even thousands of specimens. It is always a good idea to name your files in a clear and methodical way. In this case we have added R1 or R2 to each file name to indicate which replicate they are associated with. Now we can programmatically separate the two replicates

```
Specimens<-dimnames(Belemnite.Full)[[3]]

Belemnite.R1<-Belemnite.Full[,,str_detect(Specimens, "R1")]
Belemnite.R2<-Belemnite.Full[,,str_detect(Specimens, "R2")]
```

EFA

Now we are ready to carry out the normalized elliptical fourier analysis, decomposing out 70 semilandmarks for each specimen into harmomics. How can we do this? Ordinarily we would use the `efourier` function in the Momocs package. The problem is that Momocs is currently in a state of flux, with some functions being phased out and new ones replacing them, and so not everything in Momocs works together very well. Thus, we will use Momocs for plotting but not for the initial Fourier and inverse Fourier analyses. Instead we will use custom code written by Manuel F. G. Weinkauf.

The `NEF` function will translate our landmarks into harmonics. For this example, we will specify that we want to describe each outline with 15 harmonics

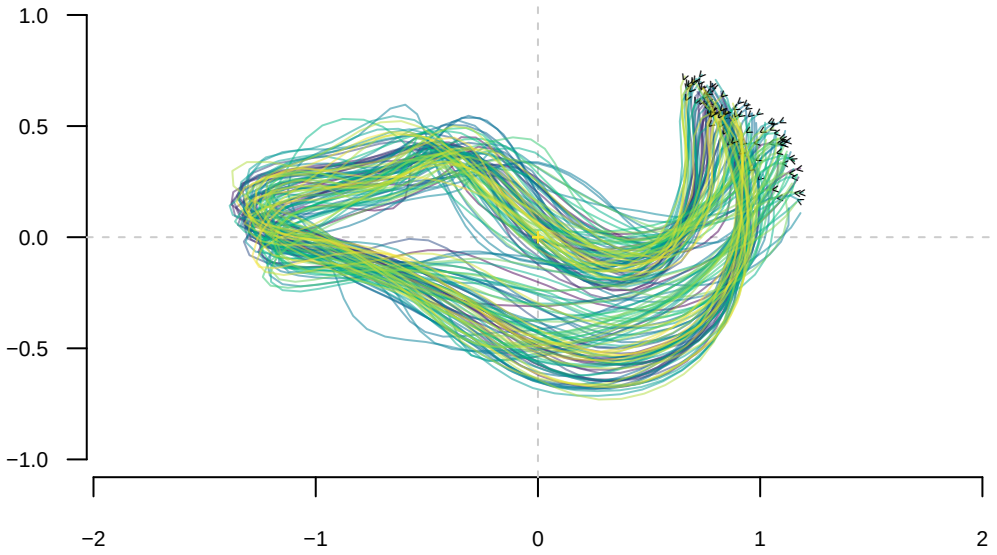
```
#make an empty list to hold our Fourier analysis results
EFA<-list()
#create a vector of specimen names which are the
Spec.Names<-strsplit(dimnames(Belemnite.R1)[[3]], split=".", fixed=TRUE)
for (i in 1:dim(Belemnite.R1)[3]) {
  EFA[[i]]<-NEF(Belemnite.R1[,,i], Harmonics=15)
  names(EFA)[i]<-Spec.Names[[i]][1]
}
```

Examine the output, an object called EFA. What does it contain?

Now we can assess how well the EFA describes the data. To do this, we will use the inverse Fourier function (`iefourier`) to convert the harmonics back in to landmarks.

```
#Reconstruct outlines
Recon.Outlines<-list()
for (i in 1:length(EFA)) {
  Coords<-iefourier(an=EFA[[i]]$A, bn=EFA[[i]]$B, cn=EFA[[i]]$C, dn=EFA[[i]]$D,
                    Harmonics=15, Points=70)
  Recon.Outlines[[i]]<-matrix(c(Coords$x, Coords$y), length(Coords$x), 2)
  names(Recon.Outlines)[[i]]<-names(EFA)[[i]]
}
#Out is a function from the Momocs package that turns a matrix of coordindates
#in a closed outline into an object of class Coo, which is useful for
Recon.Out<-Out(Recon.Outlines)
```

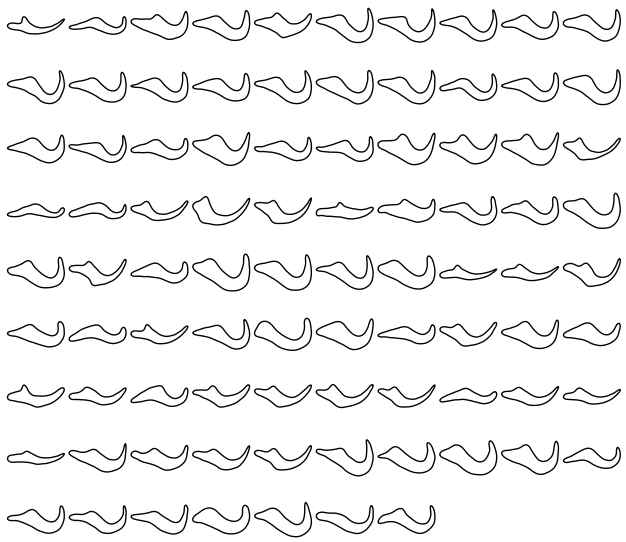
```
#Plot outlines on top of each other
Col.vec<-hcl.colors(n=length(Recon.Out$coo), palette="viridis", alpha=0.5)
coo_plot(Recon.Out$coo[[1]], border=Col.vec[1],
         xlim=c(-1.5, 1.5),
         ylim=c(-1, 1))
for (i in 2:length(Recon.Out$coo)) {
  coo_draw((Recon.Out$coo[[i]]), border=Col.vec[i])
}
```



Looks pretty good! notice that as part of the normalized EFA algorithm the specimens have been rotated, positioned, and scaled.

Another cool way to look at our data is to use the `mosaic` function in Momocs to plot all the specimens at the same time

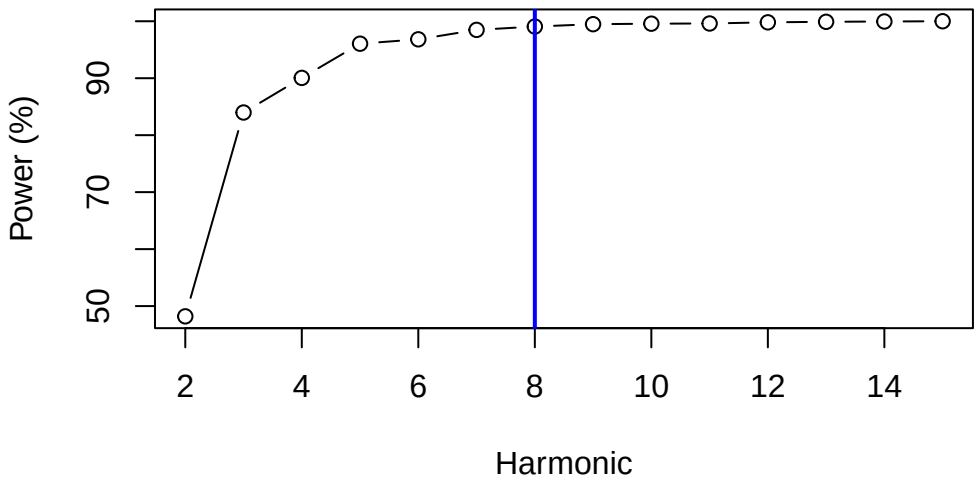
```
mosaic(Recon.Out)
```



Calculate Power

We have used 15 harmonics and each harmonic has 4 parameters, so we have 60 variables per specimen right now.

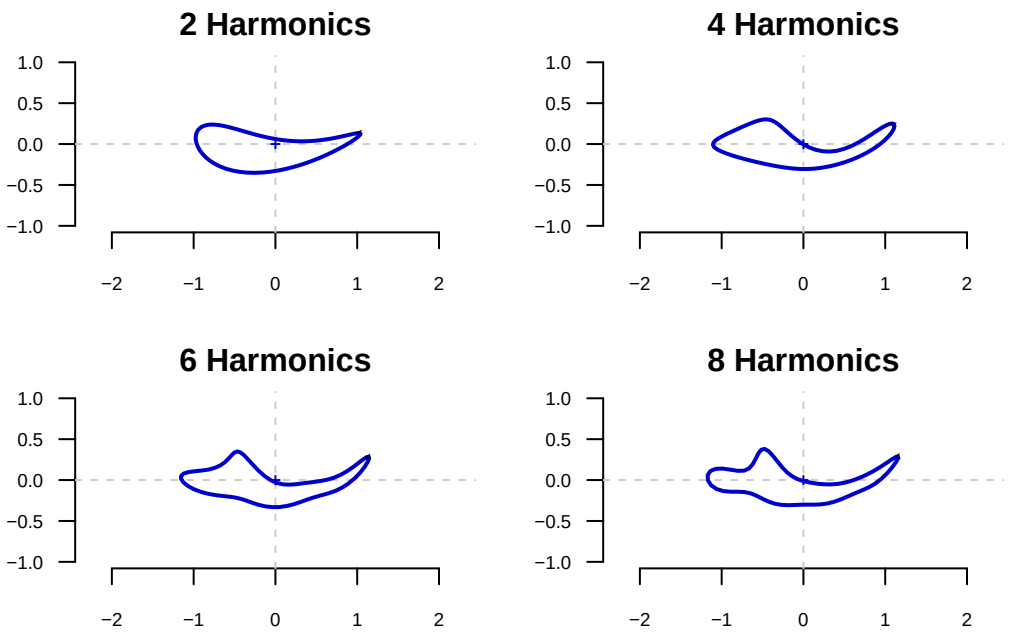
```
#Calculate harmonic powers
Harm.Power<-Power(an=EFA[[1]]$A, bn=EFA[[1]]$B, cn=EFA[[1]]$C, dn=EFA[[1]]$D,
                  first=FALSE, last=14)
plot(x=2:(length(Harm.Power)+1), y=Harm.Power*100, xlab="Harmonic", ylab="Power (%)",
     type="b")
abline(v=min(which(Harm.Power>=0.99))+1, col="blue", lwd=2)
```



Okay, so it looks like 8 harmonics give us 99% of original shape. Lets have a look at that

```
Recon.Outlines<-list()
for (i in 1:4) {
  harms_temp<-c(2,4,6,8)[i]
  Coords<-iefourier(an=EFA[[1]]$A, bn=EFA[[1]]$B, cn=EFA[[1]]$C, dn=EFA[[1]]$D,Harmonics=harm
  Recon.Outlines[[i]]<-matrix(c(Coords$x, Coords$y), length(Coords$x), 2)
}
harm_recons<-Out(Recon.Outlines)

#plot
layout(matrix(1:4, 2, 2, byrow=TRUE))
coo_plot(harm_recons$coo[[1]], border="mediumblue", xlim=c(-1.2, 1.2),
         ylim=c(-1, 1), lwd = 2, main = "2 Harmonics")
coo_plot(harm_recons$coo[[2]], border="mediumblue", xlim=c(-1.2, 1.2),
         ylim=c(-1, 1), lwd = 2, main = "4 Harmonics")
coo_plot(harm_recons$coo[[3]], border="mediumblue", xlim=c(-1.2, 1.2),
         ylim=c(-1, 1), lwd = 2, main = "6 Harmonics")
coo_plot(harm_recons$coo[[4]], border="mediumblue", xlim=c(-1.2, 1.2),
         ylim=c(-1, 1), lwd = 2, main = "8 Harmonics")
```



OK that is great and all but we have actually only assessed power for the first specimen.

```
Harm.Power <- matrix(NA, length(EFA), 14)
Cutoff <- vector(mode="numeric", length=length(EFA))
for (i in 1:length(EFA)) {
  Harm.Power[i,]<-Power(an=EFA[[i]]$A,
                        bn=EFA[[i]]$B,
                        cn=EFA[[i]]$C,
                        dn=EFA[[i]]$D,
                        first=FALSE,
                        last=14)
  Cutoff[i]<-min(which(Harm.Power[i,]>=0.99))+1
}
Harm.Power<-Harm.Power*100
SD<-apply(Harm.Power, MARGIN=2, FUN=sd)
MEAN<-apply(Harm.Power, MARGIN=2, FUN=mean)
plot(x=2:(ncol(Harm.Power)+1),
     y=MEAN,
     xlab="Harmonic",
     ylab="Power (%)",
     type="b")
for (i in 1:length(MEAN)) {
  if (SD[i]>0) {
```

```
    arrows(x0=i+1, y0=MEAN[i],
           x1=i+1, y1=MEAN[i]+SD[i],
           length=0.1, angle=90,
           col="grey50")
    arrows(x0=i+1, y0=MEAN[i],
           x1=i+1, y1=MEAN[i]-SD[i],
           length=0.1, angle=90, col="grey50")
  }
}
abline(h=99, col="blue", lwd=2)
```

How many harmonics should we use?