Geometric morphometrics

Outline data preparation

Manuel F. G. Weinkauf

26–27 August 2022

Contents

1 Introduction

In this exercise sheet, you will learn different methods how to prepare outline data for geometric morphometric analyses. For this exercise, we will start using a dataset of outlines for 87 belemnite armhooks (Fig. 1).

Each outline was extracted twice for a total set of 174 outlines. The outlines are named as 'O_nn.Rmm', where 'nn' is a running ID number for the belemnite hook and 'mm' is either '1' or '2' (i.e. replication 1 or 2, respectively). As an example, 'O_28.R1' are the outline coordinates of belemnite hook ID 28 from the first replication of outline extraction; 'O_28.R2' are the outline coordinates from the same hook from the second outline data extraction replication.

The outline data consist of 70 equidistant semi-landmarks. The raw version of outlines is available in 'Belemnite_RawOutline.nts'. A version of the same outlines with one smoothing iteration applied is available at 'Belemnite_SmoothedOutline.nts'.

For more information on the data, you are referred to

Hoffmann, R., Weinkauf, M. F. G., and Fuchs, D. (2017) Grasping the shape of belemnoid arm hooks—a quantitative approach. *Paleobiology* 43 (2): 304–20. doi: 10.1017/pab.2016.44

2 Setting up the R session

For outline analyses, we will mainly use the R-package 'Momocs'. This package is in constant development and the only R-package with significant outline analysis functionality. It includes plenty of functions, which can be overwhelming, and unfortunately the documentation is lacking details in some places and it still has quite some bugs. But overall, it is growing quickly and is very useful for both outline and landmark analyses.

On top of that, we will use the package 'hangler' to have a look at another form of outline anlysis that is more recent than elliptic Fourier analysis and touted by some workers in the field to be far superior (although this depends on whom you ask). This is the fast Fourier transform.

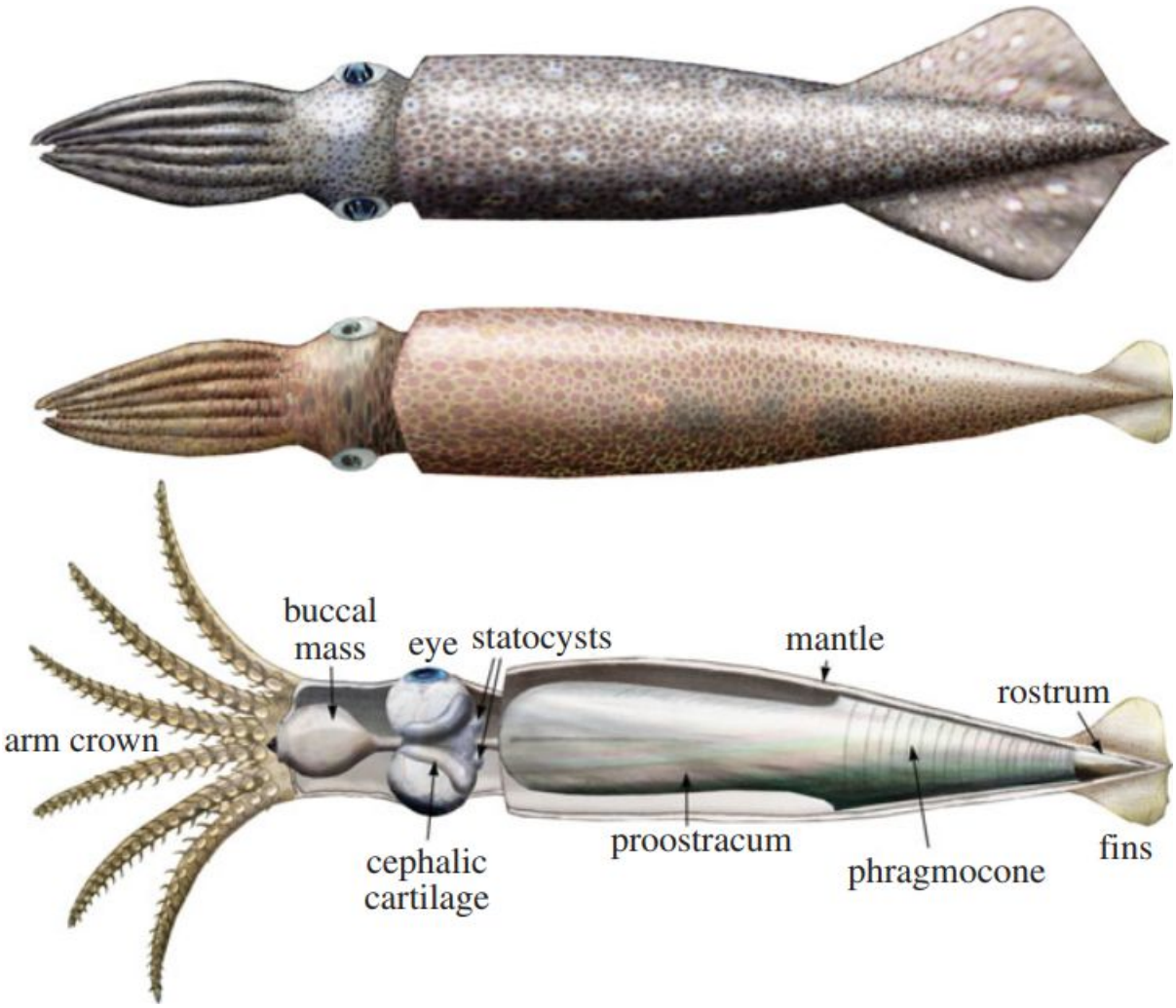

Figure 1: Depiction of a belemnite animal with armhooks visible in the lowermost picture. The armhooks in belemnites served the same purpose as suckers in modern coleoidea. From Klug et al. (2016) Adaptations to squid-style high-speed swimming in Jurassic belemnitids. *Biol. Lett.* 12: 20150877

Beside, we will use the 'geomorph' package and some of my own code included in the 'MorphoFiles_Function.r' and 'OutlineAnalysis_Functions.r' source files.

```
setwd("C:/R_Data/Erlangen_Morphometrics/Session2_OutlineDataPreparation")
library(geomorph)
library(Momocs)
library(hangler)
library(stringr)
source("MorphoFiles_Function.r")
source("OutlineAnalysis_Functions.r")
```

3 Reading the dataset

For reading data from an .nts-file, you can either use the 'readland.nts()'-function from the package 'geomorph' or the 'Read.NTS()' function from 'MorphoFiles_Function.r'. Both do exactly the same thing.

```
Belemnite.Full<-Read.NTS("Belemnite_SmoothedOutline.nts")
class(Belemnite.Full)
```

```
## [1] "array"
```

```
dim(Belemnite.Full)
```

```
## Number        Number
##     70      2    174
```

The 'class()' and 'dim()' functions show us, that 'Belemnite.Full' is an array with 70 rows (one per outline semi-landmark), 2 columns (x and y coordinates), and 174 layers (2 times 87 hooks). We are interested to separate the dataset into the two sets for replication 1 and replication 2 respectively.

We could separate these datasets by simple numerical values, as we know that the first 87 layers of the array are from replication 1 and the second 87 are from replication 2. So, the following code would do the trick

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

However, it is never a good idea to hard-code these values, as datasets may change over time and then you must remember to change these numbers by hand or your data will be wrong. Rather, we can use the names of the objects and some regular expressions to always automatically find the correct data.

```
Specimens<-unlist(dimnames(Belemnite.Full)[3])
Belemnite.R1<-Belemnite.Full[,,str_detect(Specimens, "R1")]
Belemnite.R2<-Belemnite.Full[,,str_detect(Specimens, "R2")]
```

4 Conducting outline analyses

4.1 Elliptic Fourier analysis

Elliptic Fourier analysis is an established method for outline data analysis. Unfortunately, 'Momocs' still has several issues with its functions. For instance, when calculating the normalized elliptic Fourier harmonics with 'efourier()' the returned object is incompatible with the function 'efourier_i' that would reconstruct the outlines from the coefficients. We therefore use the functions provided in 'OutlineAnalysis_Functions.r' for the moment.

```
EFA<-list()
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]
}
```

EXERCISE 1: How would you evaluate the quality of the EFA solution?

4.1.1 EFA quality evaluation

The first thing to check is whether the shapes are well aligned and reconstructed by the chosen parameters. For this, we can plot the first shape and then plot the others into the same plot (Fig. 2). The function 'efourier_i()' takes harmonics as returned by 'efourier()' and reconstructs the shape. It can be used for this purpose.

```
#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]]
}
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])
}
```

4.1.2 Determining the ideal number of harmonics

The ideal number of harmonics can be calculated using a power analysis. This can theoretically be done using the 'harm_pow()' function in 'Momocs', but as the 'efourier_i()' function, this fails with normalized EFA calculated in 'Momocs' due to the hard-coding of expected elements names that were not correctly implemented in 'harm_pow()'. We can instead use the function 'Power()' from 'OutlineAnalysis_Functions.r' (Fig. 3).

```
#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)
```

EXERCISE 2: This calculates the power for only the first belemnite hook. Find a way to make the same calculation for all hooks and use the results for the hook with least best performance to recalculate your EFA for later use.

```
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
}
```

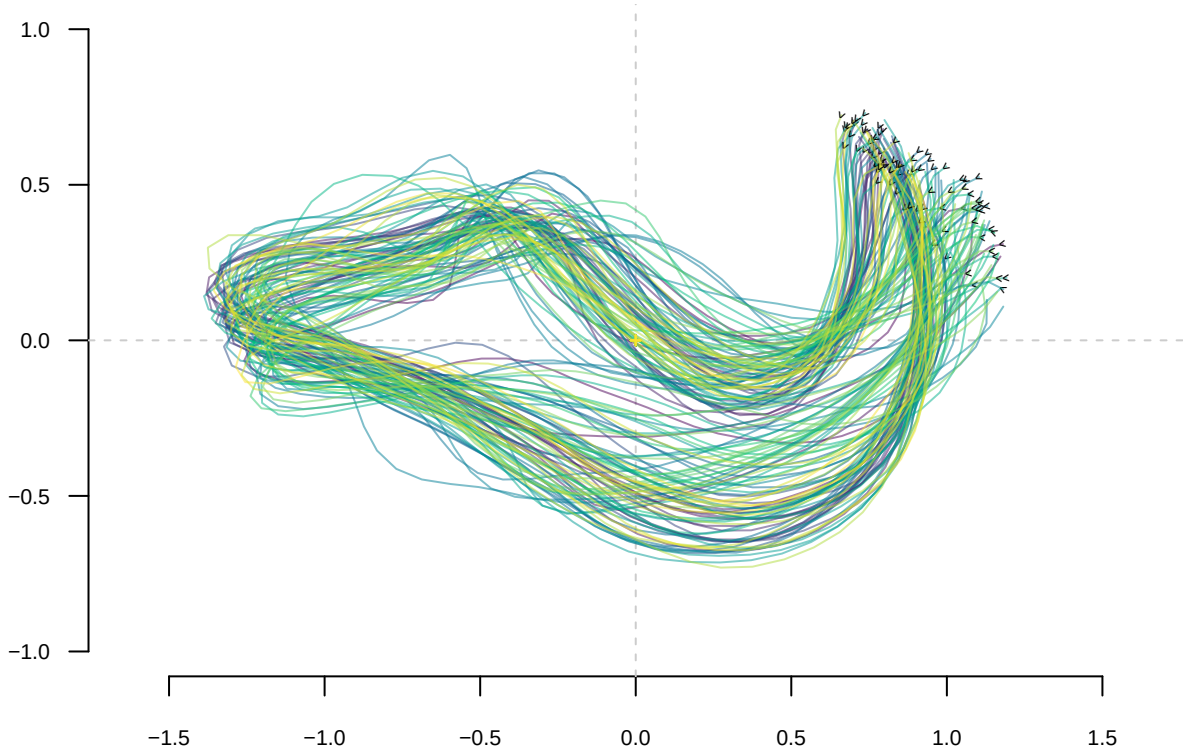

Figure 2: Visual test of superimposition of aligned belemnite hooks.

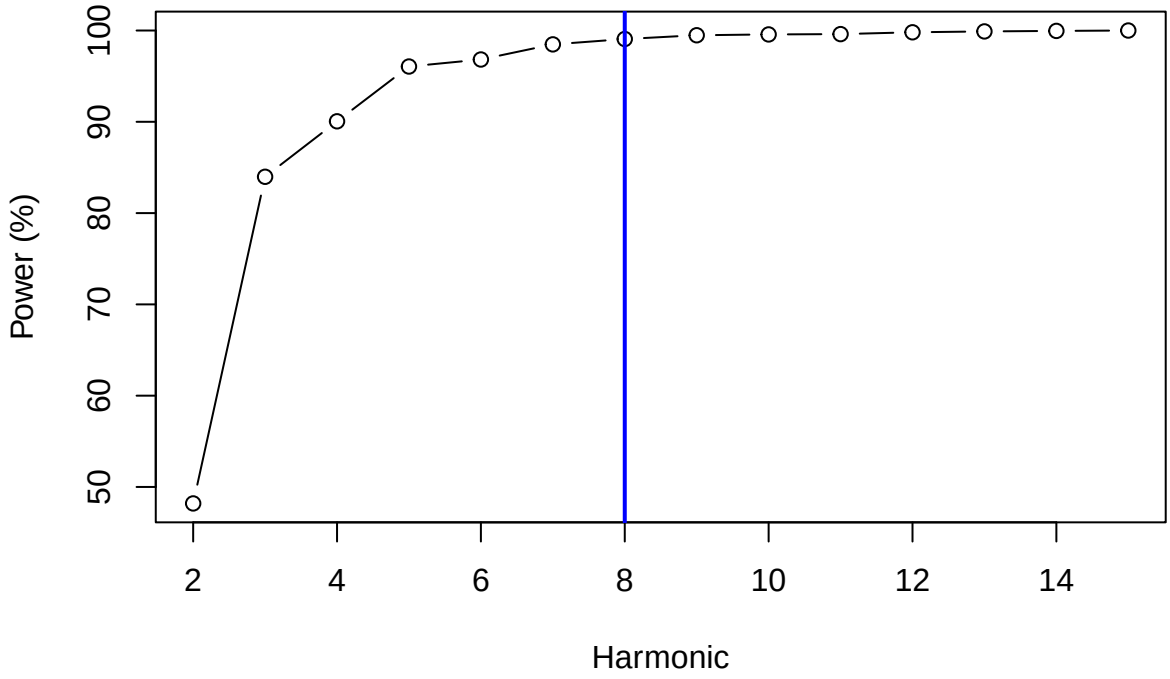

Figure 3: Power analysis for the EFA of the first belemnite hook.

```
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")
  }
}
```

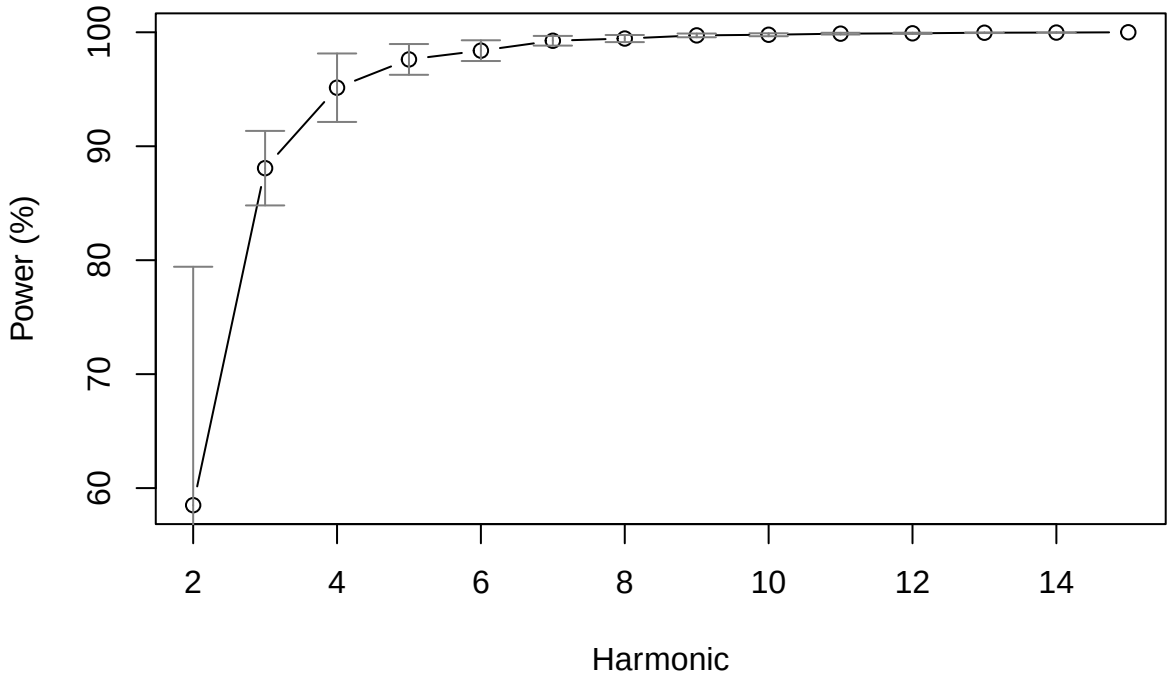

Figure 4: Power analysis for the EFA of all belemnite hooks with standard deviation of power across specimens (grey error bars).

The analysis shows, that 7 is a sufficient number of harmonics to catch 99% of overall shape information in the belemnite hooks.

EXERCISE 3: Have some fun with the other Fourier functions in 'Momocs': 'sfourier()' for radii variation Fourier analysis and 'tfourier()' for Zahn–Roskies Fourier analysis. The latter is also available in 'OutlineAnalysis_Functions.r' as 'ZRfourier()', but lacks the option for normalization there. 'OutlineAnalysis_Functions.r' also contains a function 'OutlineAverage()'. We will have a look at that later, but feel free to try it out already.

4.2 Fast Fourier transform

Fast Fourier transform is another form of outline analyses. In R, so far, it is only implemented in the package 'hangler' that is currently in development on GitHUB.

The package requires to first calculate the tangents on all points (Fig. 5).

```
Bel1.Tangents<-computeTangents(Belemnite.R1[,"x",1], Belemnite.R1[,"y",1])
plot(Bel1.Tangents)
lines(Bel1.Tangents, col="blue")
```

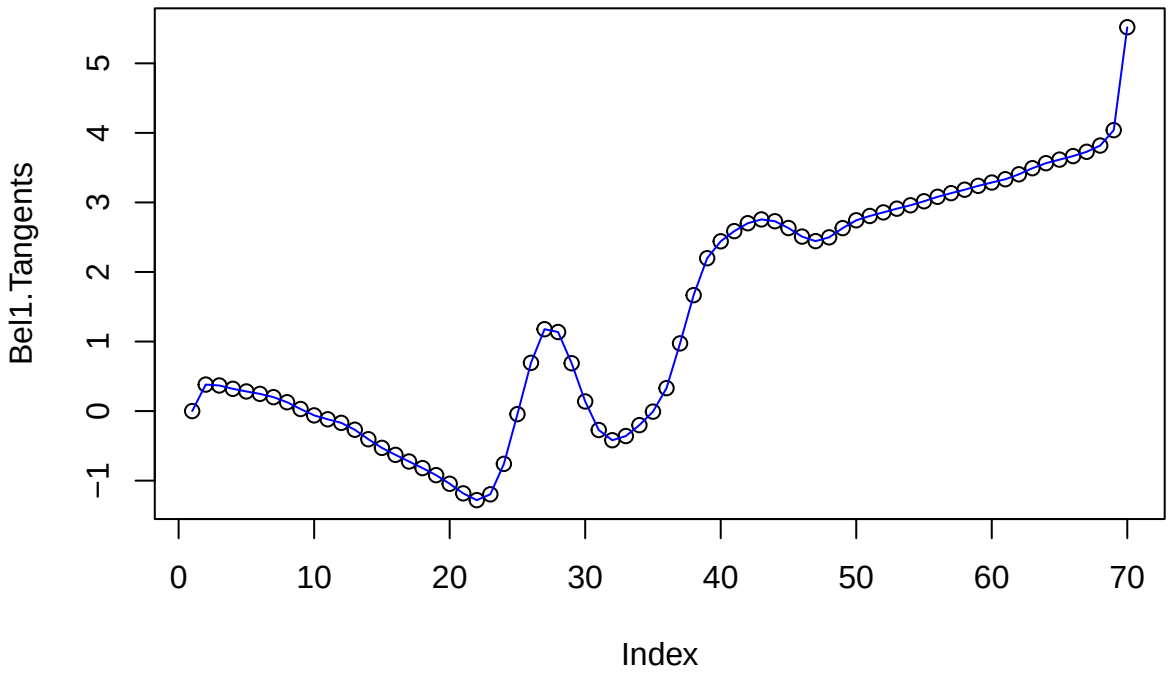

Figure 5: Tangent decomposition of first belemnite hook.

To evaluate the deconstruction, we can reform the shape from the tangents we calculated (fig. 6).

```
XY.Recon<-matrix(unlist(tangentsToXY(Bel1.Tangents)), length(Bel1.Tangents)+1, 2)
plot(XY.Recon[,1], XY.Recon[,2], type="l", asp=1, xlab="X", ylab="Y")
```

We now need to resample the outline (Fig. 7), as fast Fourier transform is extremely sensitive to even minor deviations from curvelinear equal spacing of outline points. Unfortunately, the 'resampleTangents()' was not properly added to the 'hangler'-namespace, so we have to copy the raw function code here for it to work.

```
#Copy function
resampleTangents <- function(x,y,tangents,length.out=1024,
                             integral.iter=5, solver.error= 1e-10,
                             solver.max.iter=1000){
  nPoints = length(tangents)

  # Try approximate the deli values for the spline
  deli = sapply(1:nPoints, function(i) {
    inext = i + 1
    inext = ifelse(inext > nPoints, inext - nPoints, inext)

    tryCatch(solveDeli(x[inext] - x[i], y[inext] - y[i], tangents[i], tangents[inext],
                       maxIter=solver.max.iter, targetError = solver.error,
                       integralIter=integral.iter),
      error = function(e){
                stop(paste(e,"\nFailed to compute deli for spline between",i,
                           "and",inext))
              }
```

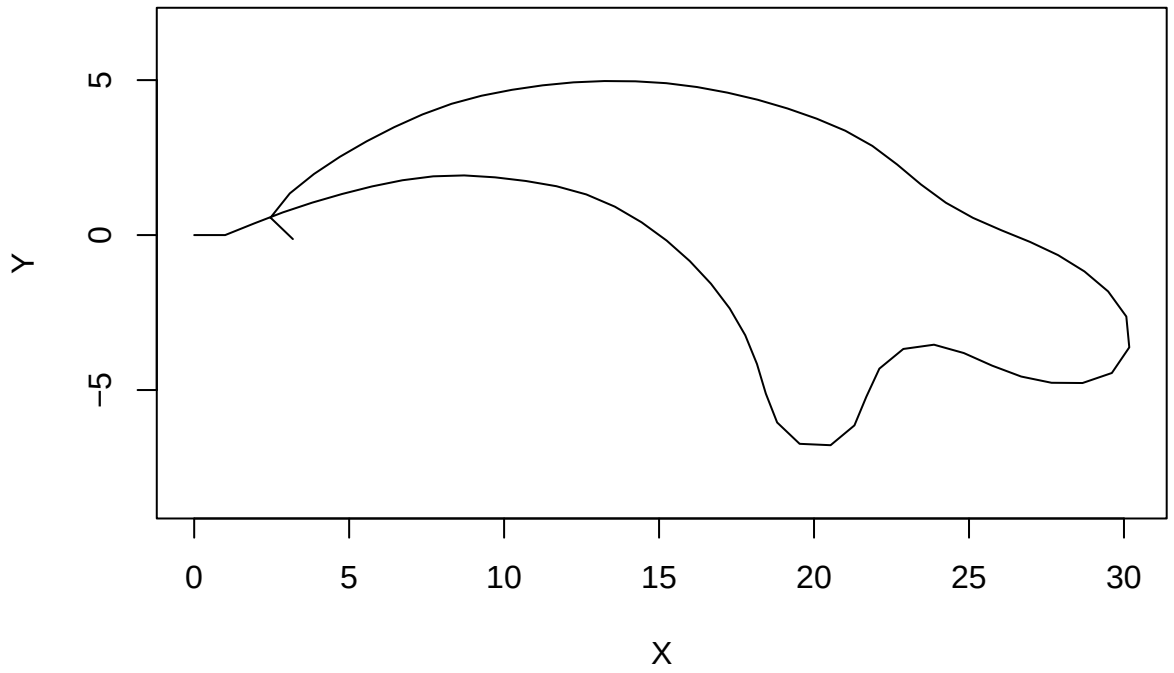

Figure 6: Reconstruction of tangents on first belemnite hook. The algorithm is not yet perfect, but for now, we roll with it.

```
  )
})

# Using the deli values try approximate the ds values
ds = sapply(1:nPoints, function(i) {
  inext = i + 1
  inext = ifelse(inext > nPoints, inext - nPoints, inext)

  tryCatch(solveDs(x[inext] - x[i], deli[i], tangents[i], tangents[inext],
                   integralIter = integral.iter),
    error = function(e){
              stop(paste(e, "\nFailed to compute ds for spline between",i,
                         "and", inext))
            }
  )
})

# Need them to be positive
ds = abs(ds)

# si are the cumulative sum of ds value
si = cumsum(c(0,ds))

# Recompute the right number of tangents from the spline
samples = sapply(seq(0,si[length(si)],length = length.out), function(s){
  i = which(si < s)
  if(length(i)==0){
    i = 1
```

```
    }else{
      i = max(i)
    }

    inext = i + 1
    sinext = ifelse(inext > length(si), inext-length(si), inext)
    tinext = ifelse(inext > length(tangents), inext-length(tangents), inext)

    computeSpline(s, si[i], si[sinext], tangents[i], tangents[tinext], deli[i])
  })

  return(samples)
}

#Resample tangents
Bel1.Tangents.resample<-resampleTangents(Belemnite.R1[,"x",1], Belemnite.R1[,"y",1],
                                         Bel1.Tangents, length.out=1024)

#Check resampling
layout(matrix(1:2, 1, 2))
plot(Bel1.Tangents.resample)
lines(Bel1.Tangents.resample, col="blue")

XY.Recon<-matrix(unlist(tangentsToXY(Bel1.Tangents.resample)),
                 length(Bel1.Tangents.resample)+1, 2)
plot(XY.Recon[,1], XY.Recon[,2], type="l", asp=1, xlab="X", ylab="Y")
```

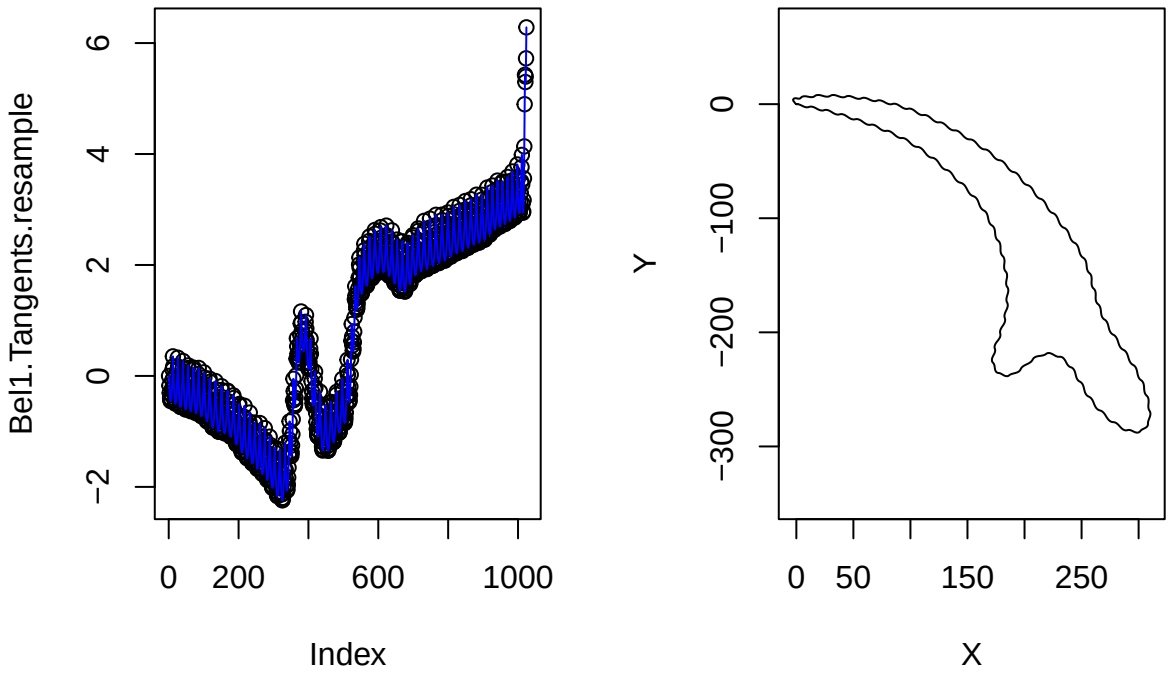

Figure 7: Tangent resampling of the first belemnite hook.

Unfortunately, the resampling process currently adds quite a bit of noise to the data. Part of this noise is removed by a follow-up flattening procedure (Fig. 8). This is necessaty, as FFT evaluates shapes as deviation from a circle. Unfortunately, this function as well was not properly added to the 'hangler'-namespace.

```
#Import function
flattenTangents <- function(tangents) {
  circle = seq(0, 2*pi, length.out=length(tangents))
  return(tangents - circle)
}

#Flatten curve
Bel1.Tangents.resample<-flattenTangents(Bel1.Tangents.resample)

#Check results
plot(Bel1.Tangents.resample)
lines(Bel1.Tangents.resample, col="blue")
```

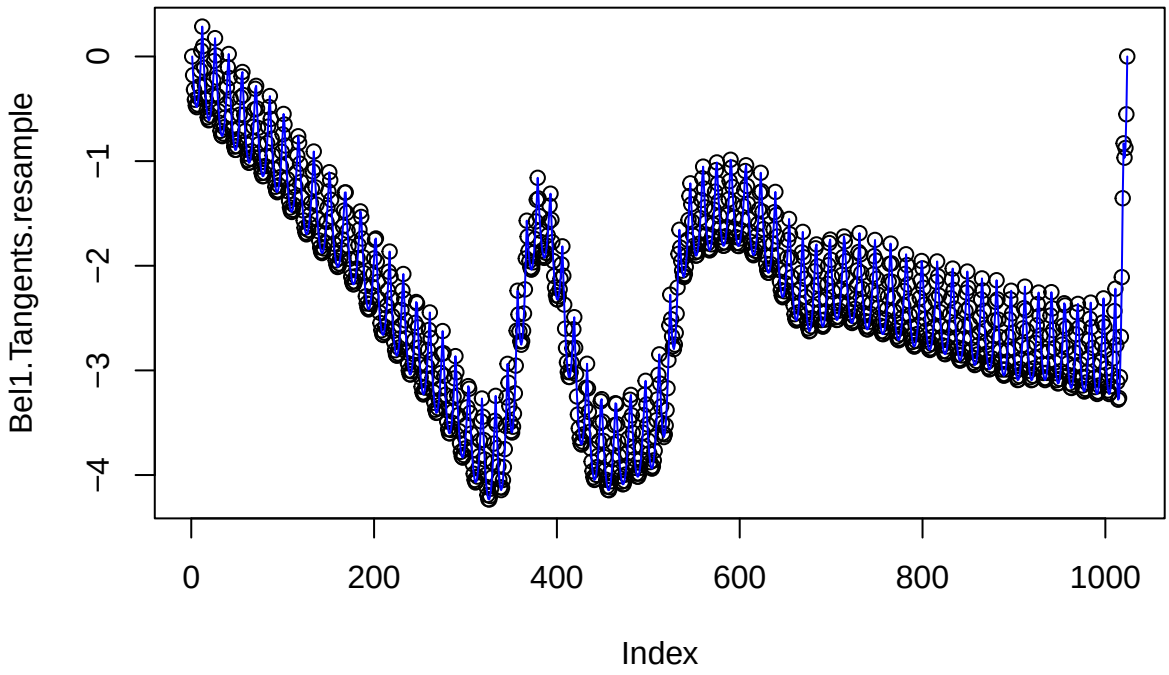

Figure 8: Tangent flattening of the first belemnite hook.

From these data, the Fourier coefficients of a fast Fourier transform can be computed.

```
FFT<-fft(Bel1.Tangents.resample)/length(Bel1.Tangents.resample)
```

With the FFT coefficients, we can now try to reconstruct the original form of the belemnite hook (Fig. 9).

```
#Convert FFT coefficients back into curve outline
Recon.FFT<-fftoToCoeffs(FFT)
Recon.FFT.Outline<-sapply(seq(0, 2*pi, length.out=length(Bel1.Tangents.resample)),
      function(s) getTangentAtS(s, Recon.FFT))

#Plot results
layout(matrix(1:2, 1, 2))
plot(Recon.FFT.Outline)
lines(Recon.FFT.Outline, col="blue")

XY.Recon<-matrix(unlist(tangentsToXY(Recon.FFT.Outline)), length(Recon.FFT.Outline)+1, 2)
plot(XY.Recon[,1], XY.Recon[,2], type="l", asp=1, xlab="X", ylab="Y")
```

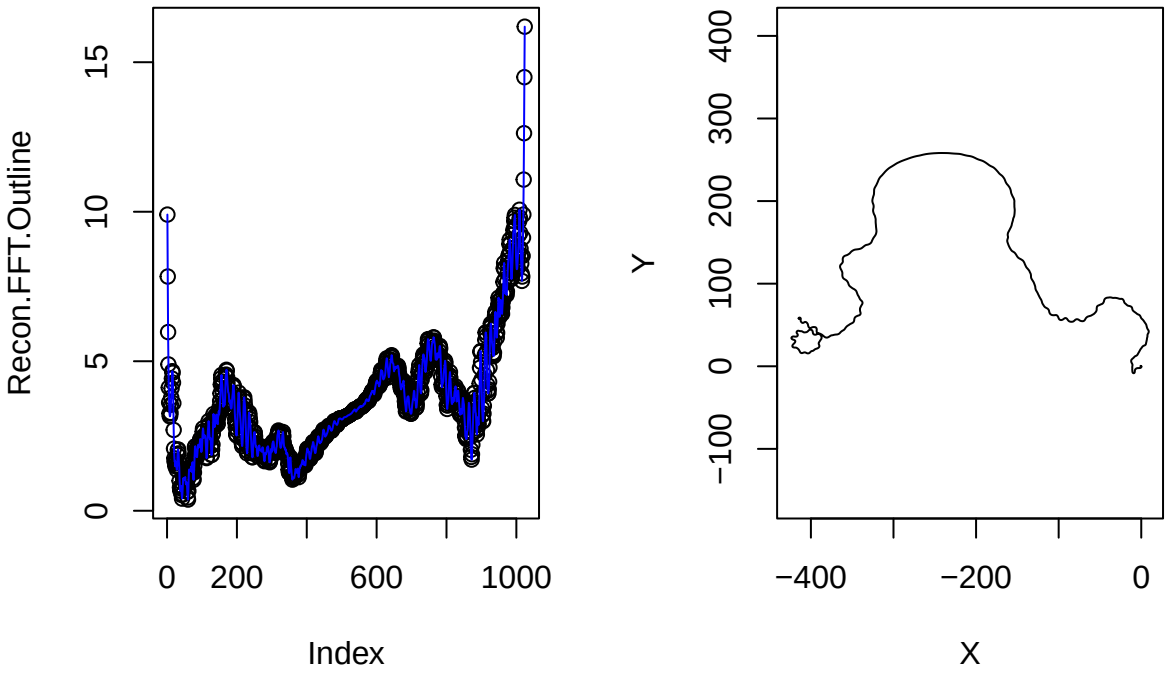

Figure 9: For forms like this, the current FFT implementation in R is unfortunately not very useful.

As you can see, this procedure in its current implementation is not very useful for forms like belemnite hooks. It does work a bit better with more roundish forms that do not have tips and with forms with radii that do not leave the shape (similar to radii variation Fourier analysis). I think it is worth keeping an eye on that package, in case the implementation gets better in the future.

Unfortunately, the last update was five years ago and the project may as well be abandoned at that time. You may salvage the existing functions for your own work in the future.