A look at Momocs

installing

```
install.packages("devtools")
devtools::install_github("MonX/Momocs")
devtools::install_github("MomX/Momit",bulid_vignettes=TRUE)
browseVignettes("Momit")
```

"Momit" is a extra project of Momocs including more type of files inporting and functions in morphometrics, which I didn't use here. I put it here so that we can try those functions someday (maybe after our contour extraction complete? because it is said that Momit can directly import .jpg files and do angle detection work).

```
library(Momocs)
```

data wragling

```
long<-read.csv("RW_vases_long.csv")
```

keep main features we may use

```
long_1<-long[,c("x","y","vaseseries","timestep")]
head(long_1)
```

```
##          x    y vaseseries timestep
## 1 -6.219953 10          0        0
## 2 -8.238827 11          0        0
## 3 -9.092390 12          0        0
## 4 -9.647357 13          0        0
## 5 -9.980561 14          0        0
## 6 -10.255564 15         0        0
```

have a look at vase series 1

```
long_1.1<-subset(long_1,long_1$vaseseries==1)
head(long_1.1)
```

```
##    x    y vaseseries timestep
## 192  -8.567492 10          1        1
## 193 -10.405003 11          1        1
## 194 -11.551042 12          1        1
## 195 -12.236296 13          1        1
## 196 -12.569428 14          1        1
## 197 -12.812451 15          1        1
```

Divide vase series 1 into lists by different timesteps, because In Momocs, a shape should be a matrix of (x, y) coordinates. When they are plenty, they are usually gathered in a Coo object which is, essentially, a list with

at least a $coo component.

```
a<-split(long_1.1,long_1.1$timestep)
head(a)
```

Here we remove column "timestep" and "vaseseries" so that we keep every element in list b is 2-col matrix, which Momocs can read.

```
b<-lapply(a, function(x){as.matrix(x[, c(1,2)])})
```

check the class

```
class(b[1])
## [1] "list"
class(b[[1]])
## [1] "matrix"
```

So here elements in b are still lists instead of matrics, we transform b into "Out" type, which means "outline"

```
o<-Out(b)
o
## Out (outlines)
##     - 200 outlines, 191 +/- 0 coords (in $coo)
##     - 0 classifiers (in $fac):
## # A tibble: 0 x 0
##     - also: $ldk
```

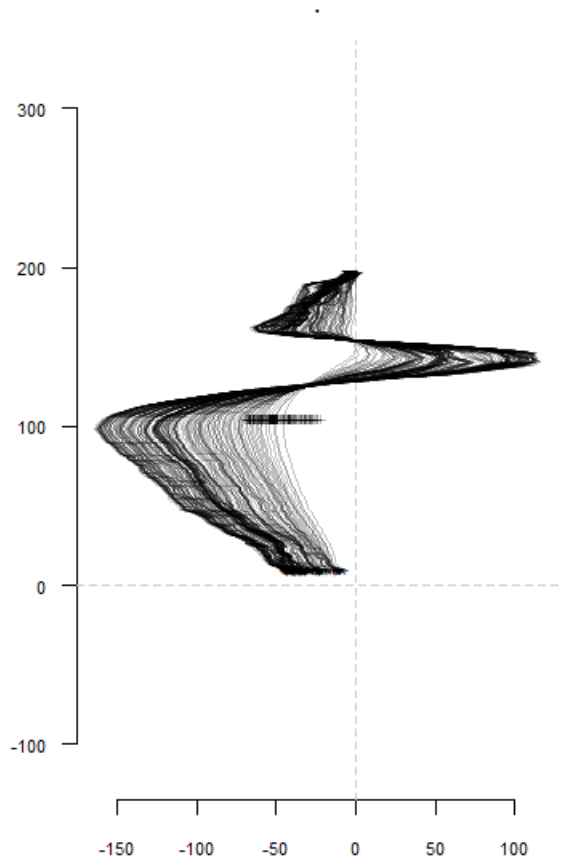
Check our data o

```
# a single shape
o[1] %>% is_shp() # tests if it is a 2-col matrix without NAs
## [1] TRUE

o[1] %>% head()
##                    x  y
## 192   -8.567492 10
## 193 -10.405003 11
## 194 -11.551042 12
## 195 -12.236296 13
## 196 -12.569428 14
## 197 -12.812451 15

# a collection of shapes
o %>% class()   # Out for outlines, but also a Coo. See ?Coo
## [1] "Out" "Coo"

o %>% is.list   # essentially a list
## [1] TRUE
```

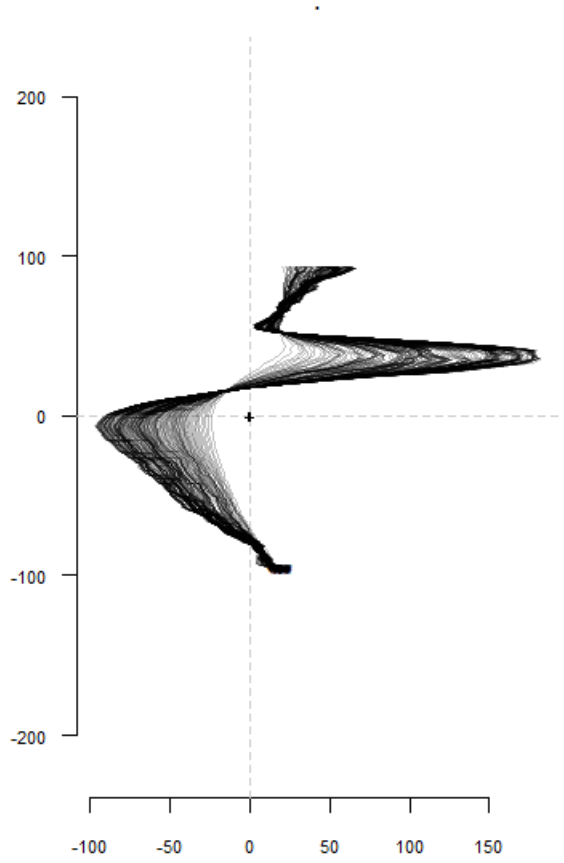
```
o$coo %>% class() # with a $coo component
## [1] "list"
```

Geometric operations

```
o %>% stack
## will soon be deprecated, see ?pile
```




```
o %>% coo_center %>% stack()
## will soon be deprecated, see ?pile
```

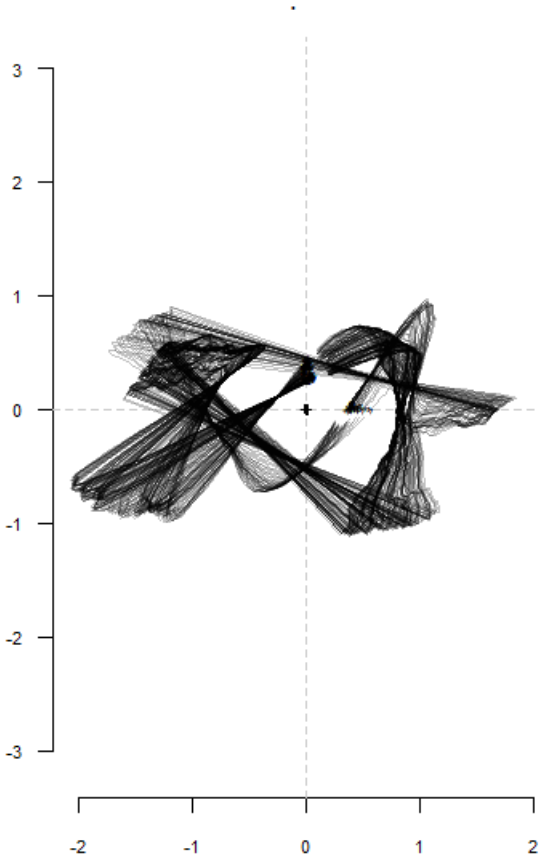


What I didn't mentioned in former codes is that there are other $coo functions we can use to do isotropic or anisotropic reshape, alignment, translation, rotation, sampling, smoothing, slicing, indexing and so on. These different functions can be used in our project for bar chopping, angle detection and standardization in our later work.

A small example, the most familiar operation can directly be applied on Coo objects:

```
o %>%
coo_center %>% coo_scale %>%
coo_alignxax %>% coo_slidedirection("up")%T>%
print()%>%stack()
```

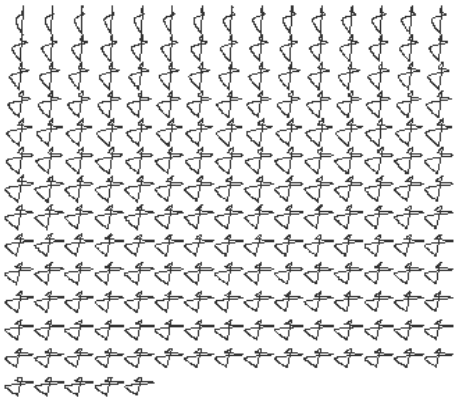
```
## Out (outlines)
##     - 200 outlines, 191 +/- 0 coords (in $coo)
##     - 0 classifiers (in $fac):
## # A tibble: 0 x 0
##     - also: $ldk
## will soon be deprecated, see ?pile
```



Morphometrics

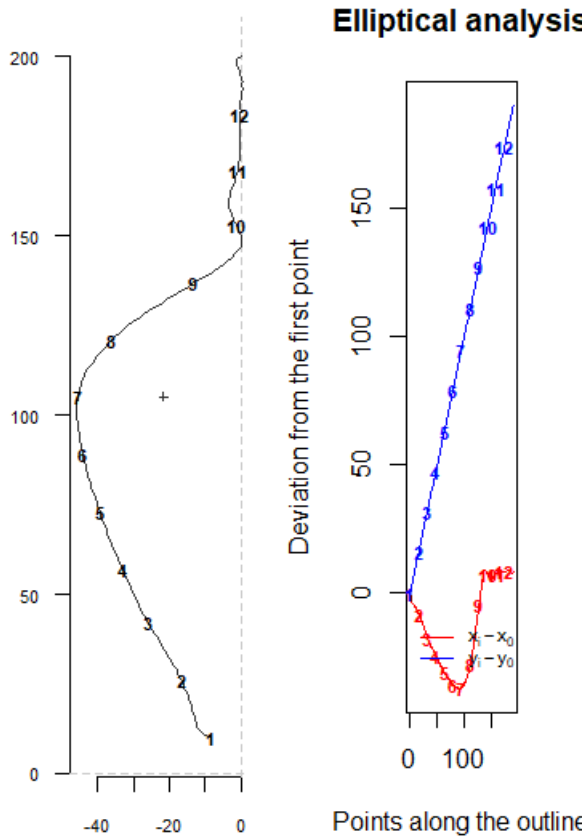
Outline analysis

```
panel(o)    # a brief looking of all contours we have in vs==1
```



Here, we will illustrate outline analysis with elliptical Fourier transforms (but the less used and tested rfourier, sfourier and tfourier are also implemented).
The idea behind elliptical Fourier transforms is to fit the x and y coordinates separately, that is the blue and red curves below:

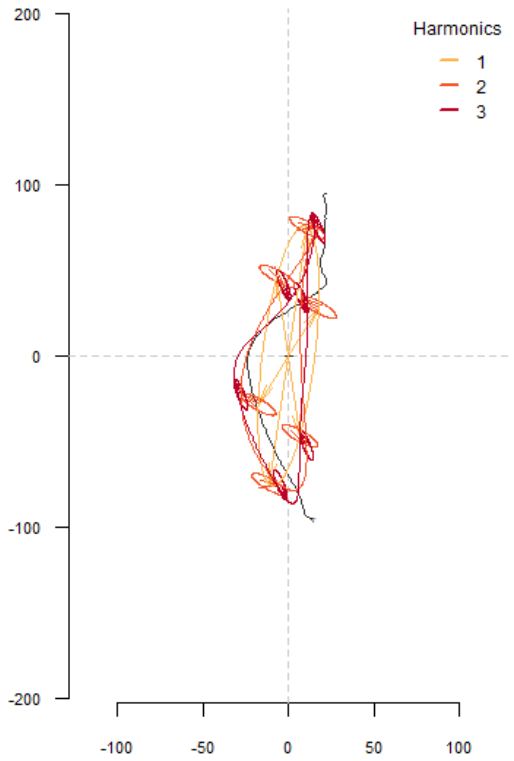
```
coo_oscillo(o[1],"efourier")
```




```
## # A tibble: 191 x 2
##          dx      dy
##       <dbl> <dbl>
##    1   0         0
##    2  -1.84      1
##    3  -2.98      2
##    4  -3.67      3
##    5  -4.00      4
##    6  -4.24      5
##    7  -4.46      6
##    8  -4.69      7
##    9  -5.03      8
## 10  -5.25        9
## # ... with 181 more rows
```

Graphically, this is equivalent to fitting Ptolemaic ellipses on the plane:

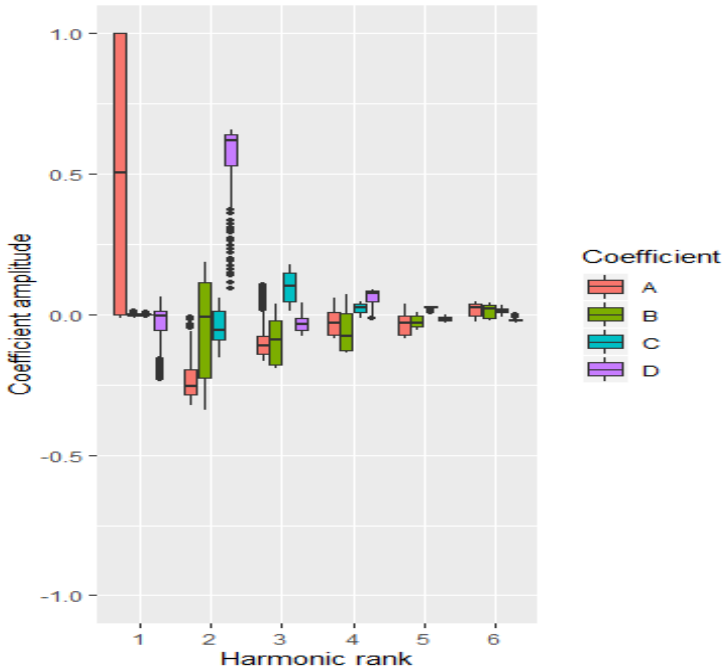
```
Ptolemy(o[1])
```




```
o.f<-efourier(o,nb.h = 10)
o.f
## An OutCoe object [ elliptical Fourier analysis ]
## --------------------
##    - $coe: 200 outlines described, 10 harmonics
## # A tibble: 0 x 0
```

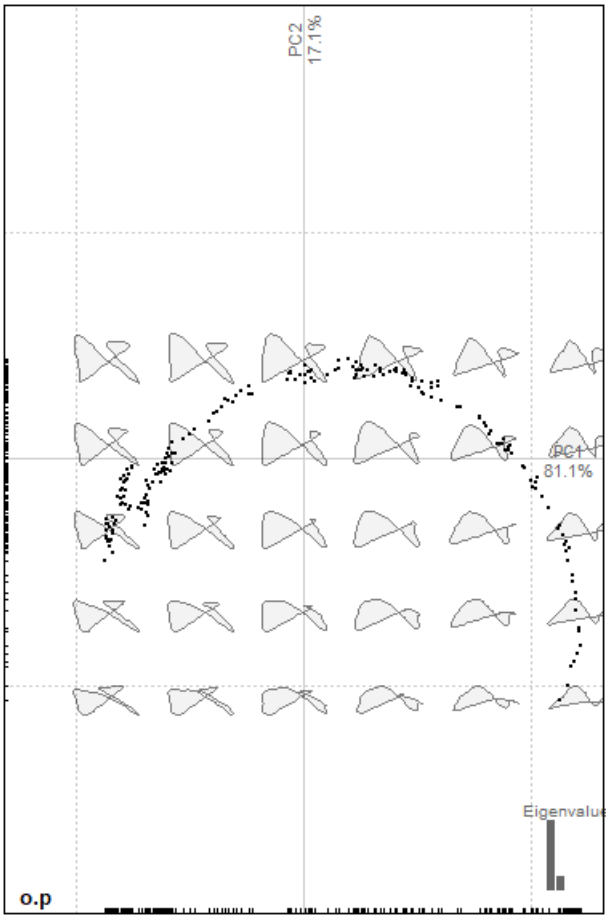
o.f is a Coe object (and even an OutCoe), we can have a look to the amplitude of fitted coefficients with:

```
boxplot(o.f)
```



we can calculate a PCA on the Coe object and plot it, along with morphospaces, calculated on the fly.

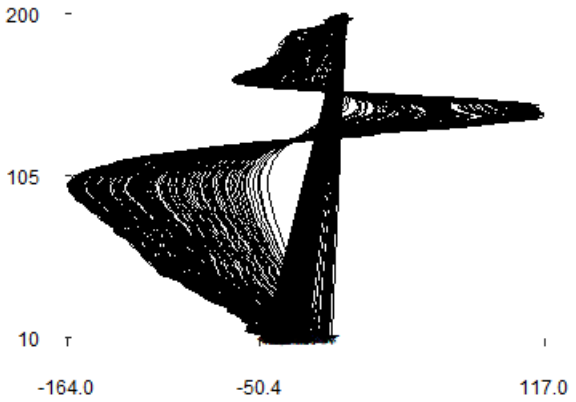
```
o.p<- PCA(o.f)
plot(o.p)
```



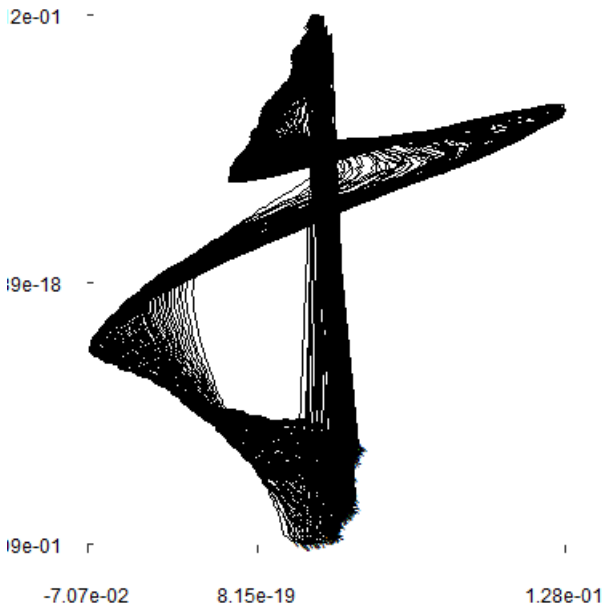
we can also add landmarks in our graphs so that we can control some important spots (e.g. vases' bottom

part may change less than belly because the bottom should suspend whole vase so it can't be too small)

```
pile(o)
```



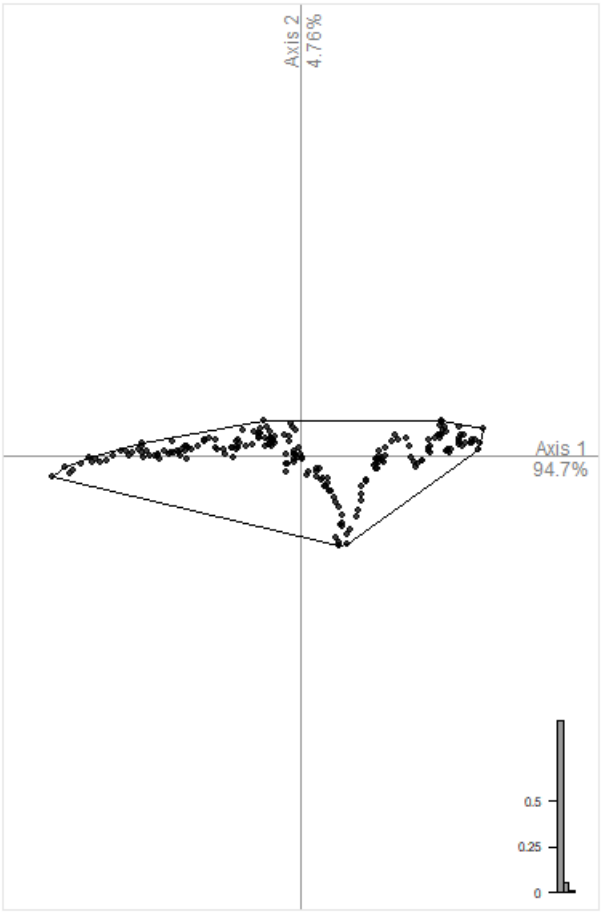

```
options(Momocs_verbose = FALSE)    # to silent Momocs
o.al<-fgProcrustes(o)
pile(o.al)
```



Functions above are still experimenting, but it provides a tool.

Momocs can also describe these vases with scalar descriptors: area (if we make a close contour after we extract half of them), circularity and the distance between the 1st and the 100st bumps of the hearts. measure is of great help. Note the loadings.

```
ot<-measure(o,coo_area,coo_circularity,d(1,100))
ot %>% PCA() %>% plot_PCA()
```

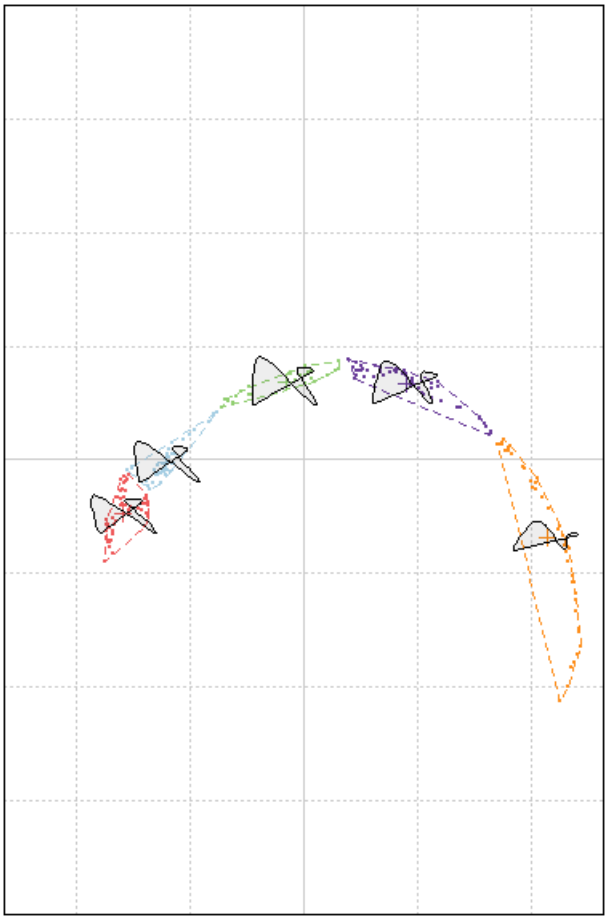


A very minimal k-means clustering is implemented:

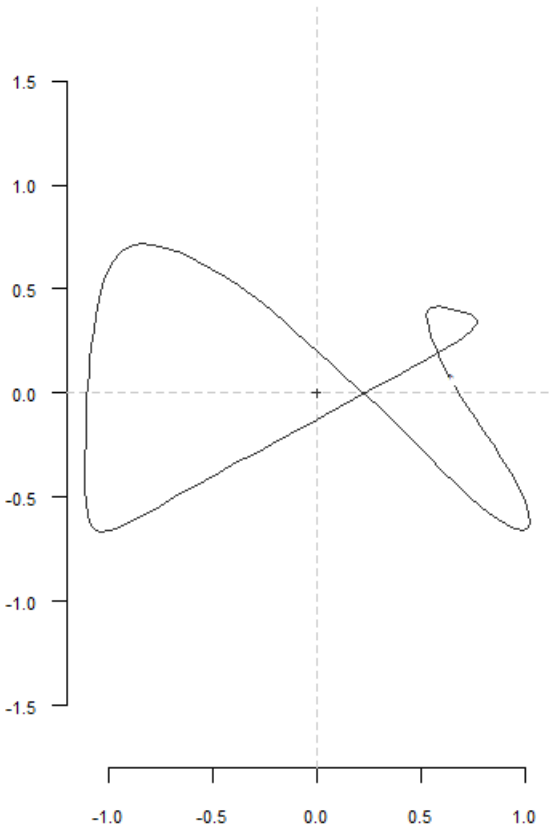
```
KMEANS(o.p,centers = 5)
```

```
## K-means clustering with 5 clusters of sizes 37, 27, 49, 39, 48

## Cluster means:
##                PC1         PC2
## 1 -0.23402002 -0.00287542
## 2 -0.03110714  0.13253567
## 3 -0.30794381 -0.09152274
## 4  0.41499305 -0.13282681
## 5  0.17506565  0.12901640

## Clustering vector:
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
##  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
## 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
##  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
## 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
##  4  4  4  4  4  4  4  4  4  5  5  5  5  5  5
```

```
##  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75
##   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5
##  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   5   5   5   5   5   5   5   5   5   5   5   2   5   2   2
##  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
##   2   2   2   2   2   2   2   2   2   1   1   1   1   1   1
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
##   1   1   1   1   3   3   3   3   3   3   3   3   3   1   1
## 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165
##   1   1   1   1   1   1   1   3   3   3   3   3   3   3   3
## 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
##   3   1   3   3   3   1   3   3   3   3   3   1   3   3   3
## 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
## 196 197 198 199 200
##   3   3   3   3   3

## Within cluster sum of squares by cluster:
## [1] 0.08009991 0.11849713 0.09415505 0.78900101 0.29691824
##  (between_SS / total_SS =  92.6 %)

## Available components:

## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```



```
o.f%>%MSHAPES()%>% coo_plot()
## no 'fac' provided, returns meanshape
```


1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-1.0
-0.5
0.0
0.5
1.0