Quantitative Latent Shape Phenotyping in R

Mitchell J. Feldmann

Introduction

This tutorial relies on only a few select R packages; namely `MOMOCS` and `LAVAAN` and has six sections, each of which are concerned with a different perspective of shape.

The purpose of this tutorial is not to describe how to process or interpret results, although a little bit of time will be spent on each. Hopefully someone in this very series will talk about image segmentation and different types of geometric and radiometric calibrations... fingers crossed.

The data that I have provided are binary maps of individual fruit that have been scaled to be 1000 x 1000 pixels. This means that the largest dimension of the 2D projection from all images in equal.

All of the analyses described here are reported in the paper that I have provided with the data. Feldmann et al "Multi-dimensional Machine Learning Approaches for Fruit Shape Phenotyping in Strawberry."

Confession

My segmentation approach was very very very brute force, animalistic even. You can find the imageJ macro on github, but I really don't suggest using it unless you have a lot of time to sit and correct and curate the output. I think I had to redo 450-600 / 6874 segmentation attempts. I used a black background and some strawberries were so dark that they failed to be segmented. I am still looking for a very flexible, automated approach for this. I used the Simple Object Interactive Extractor (SIOX) that is implemented in ImageJ, which is easy to deploy, but not super good when your targets are variable.

I also didn't realize until much later that I had placed each better in the same location, so instead of manually clicking through images to get measurements from imageJ, I could have automated that process and TBH I wish I had. So, while you can find these approaches on Github, I highly suggest going a different route which is why I am not demoing those aspects here.

Doing this again

All fruit would be imaged on an aruco checkerboard so that I could use CASS by Amy Tabb to correct the geometry of each image. My images do have some radial distortion so absolute measurements impossible.

I might include a Color Checker in the images to take advantage of some of the recent color calibration approaches, especially because my approach is in a highly standardized, artificial environment with out shadows or too much glare. The illumination uniformity could have been better.

Analyses

1. Elliptical Fourier Analysis of Ellipsoids

In this, the very first section, we are going to perform Elliptical Fourier Analysis using the MOMOCS package. This package has been around for a while and makes this and several other morphometric analyses a piece of cake! Which is good.

For this section we only need `MOMOCS`.

First we load all of our packages and the image data. There are 6,874 images in this dataset that are from two harvests, of 3 replicates of 574 genotypes in Salinas, CA. The list of files doesn't take long to load.

This is the most important thing that you will do for this workshop. Find your data and LIST THOSE FILES!!

```
lf <- list.files("/Volumes/Elements/Desktop/Reading/Strawberry_Research/Heterosis/Salinas/Data/Images/R
```

This next chuck of code loads the jpgs in a serial method. `import_jpg()` prints the number of files that are loading.

There are images that seem to break the loader. I was able to trace this to an issue with roughness of the edges in the images and had to manually check them.

My suggestion is to "open: the image prior to running these analyses. Opening an image entails first erosion followed by dilation of the same magnitude. So basically small (very thin) artifacts will be removed from an image. This likely comes from the fact that I am using jpg (to save space), which are quite lossy and do not propagate edges well. Using a larger, lossless image format is obviously preferred but also more expensive.

Regardless, this approach worked pretty well but is really ugly to look at.

```
coo <- import_jpg(lf[-c(471,2880,2895,3283,3752,3940,3963,4362,5362,5367,5485,1686,2274,2689,3616,3860,4

coo.0 = import_jpg(lf[1686],threshold = 0.001)
coo.1 = import_jpg(lf[2274],threshold = 0.001)
coo.2 = import_jpg(lf[2689],threshold = 0.01)
coo.3 = import_jpg(lf[3616],threshold = 0.015)
coo.4 = import_jpg(lf[3860],threshold = 0.015)
coo.5 = import_jpg(lf[4408],threshold = 0.001)
coo.6 = import_jpg(lf[4763],threshold = 0.001)
coo.7 = import_jpg(lf[4764],threshold = 0.005)
coo.8 = import_jpg(lf[5089],threshold = 0.005)
coo.9 = import_jpg(lf[5474],threshold = 0.005)
coo.10 = import_jpg(lf[5817],threshold = 0.005)
coo.11 = import_jpg(lf[5821],threshold = 0.005)
coo.12 = import_jpg(lf[6284])
coo.13 = import_jpg(lf[471],threshold = 0.01)
coo.14 = import_jpg(lf[2880],threshold = 0.02)
coo.15 = import_jpg(lf[2895],threshold = 0.01)
coo.16 = import_jpg(lf[3283],threshold = 0.001)
coo.17 = import_jpg(lf[3752],threshold = 0.015)
coo.18 = import_jpg(lf[3940],threshold = 0.001)
coo.19 = import_jpg(lf[3963],threshold = 0.01)
coo.20 = import_jpg(lf[4362])
coo.21 = import_jpg(lf[5362],threshold = 0.01)
coo.22 = import_jpg(lf[5367],threshold = 0.01)
coo.23 = import_jpg(lf[5485],threshold = 0.001)

coo = append(coo,coo.0)
coo = append(coo,coo.1)
coo = append(coo,coo.2)
coo = append(coo,coo.3)
coo = append(coo,coo.4)
coo = append(coo,coo.5)
coo = append(coo,coo.6)
coo = append(coo,coo.7)
coo = append(coo,coo.8)
```

```
coo = append(coo,coo.9)
coo = append(coo,coo.10)
coo = append(coo,coo.11)
coo = append(coo,coo.12)
coo = append(coo,coo.13)
coo = append(coo,coo.14)
coo = append(coo,coo.15)
coo = append(coo,coo.16)
coo = append(coo,coo.17)
coo = append(coo,coo.18)
coo = append(coo,coo.19)
coo = append(coo,coo.20)
coo = append(coo,coo.21)
coo = append(coo,coo.22)
coo = append(coo,coo.23)

strw <- Out(coo)
```

If I had done a better job with the segmentation or "opened" the image prior to reading the data in, this could have looked like this nice one-liner :

```
coo <- Out(import_jpg(lf))
```

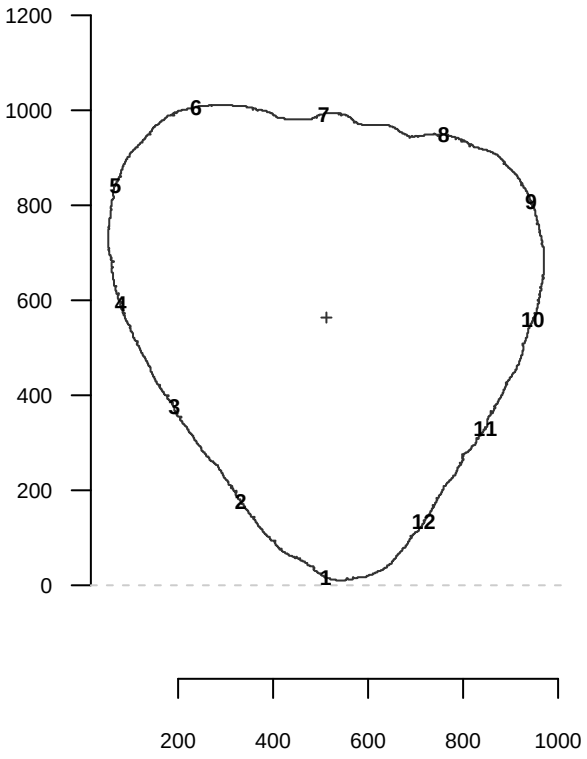
SO NICE! The gods of 2020 will not permit nice things (this was written in 2018/2019). Do you want to know the truth? I followed tutorials and messed around with stuff until it worked, and in the case of this tutorial, the slog was getting the data to read in properly... of course...

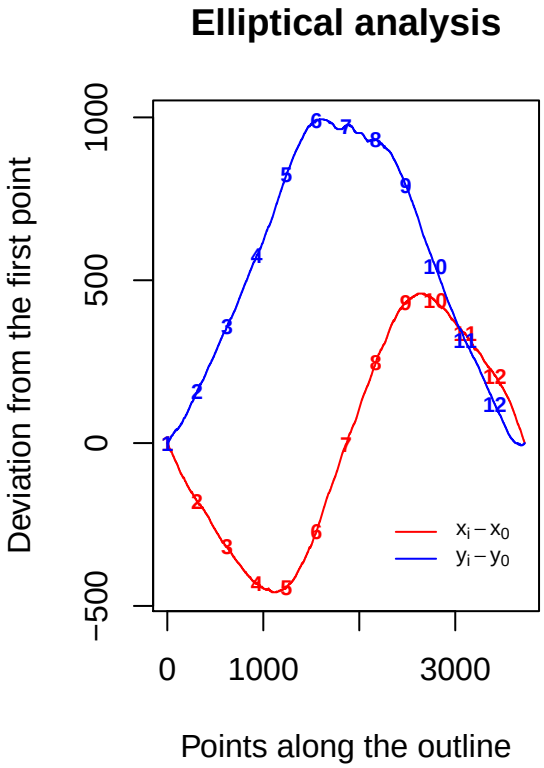
The output of the previous chunk is an `out` object (an outline), which is a closed outline object that is used in `MOMOCS`. Cool.

From here, we have the fun stuff. Checking out images, checking out the curvature on the x and y axis, power, reconstruction, WOW! everything good in the world.

(1) `coo_oscillo` prints an image of the object and a plot with the x and y coordinates starting at position 1 and ending at position 1.

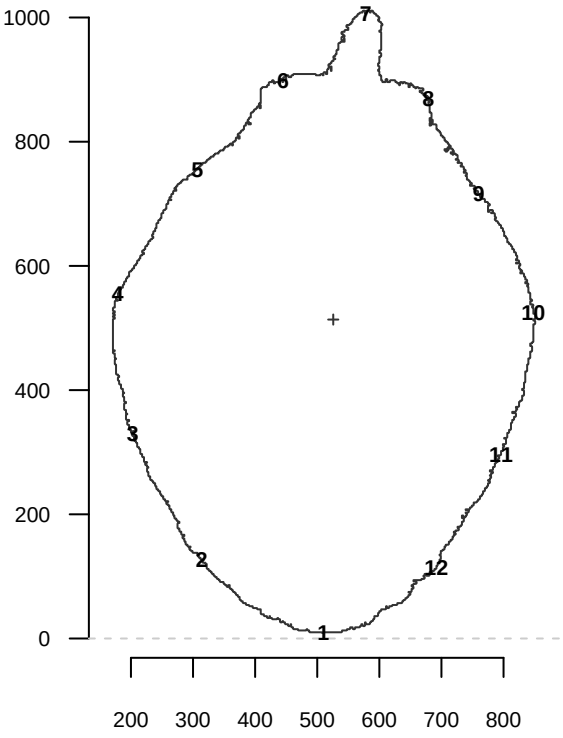
```
coo_oscillo(strw[1], "efourier")
```

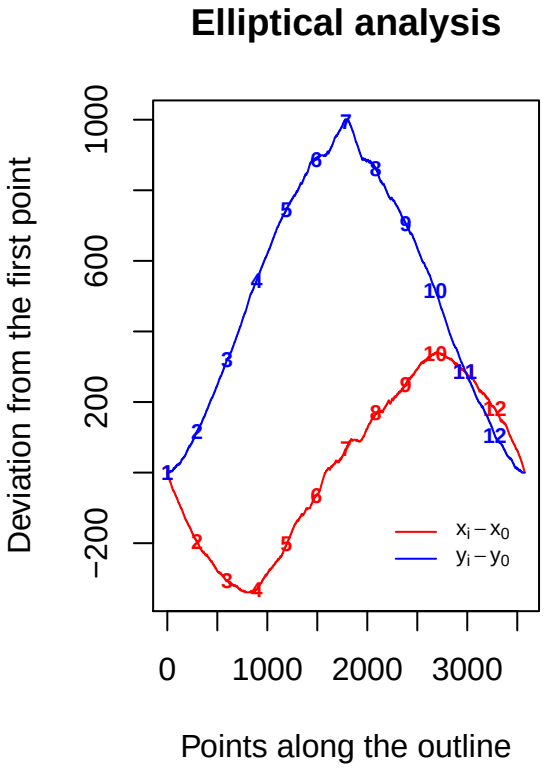





```
## # A tibble: 3,724 x 2
##       dx    dy
##    <dbl> <dbl>
##  1     0     0
##  2    -1    -1
##  3    -2    -1
##  4    -3     0
##  5    -4     1
##  6    -4     2
##  7    -4     3
##  8    -5     3
##  9    -6     3
## 10    -7     3
## # ... with 3,714 more rows
```

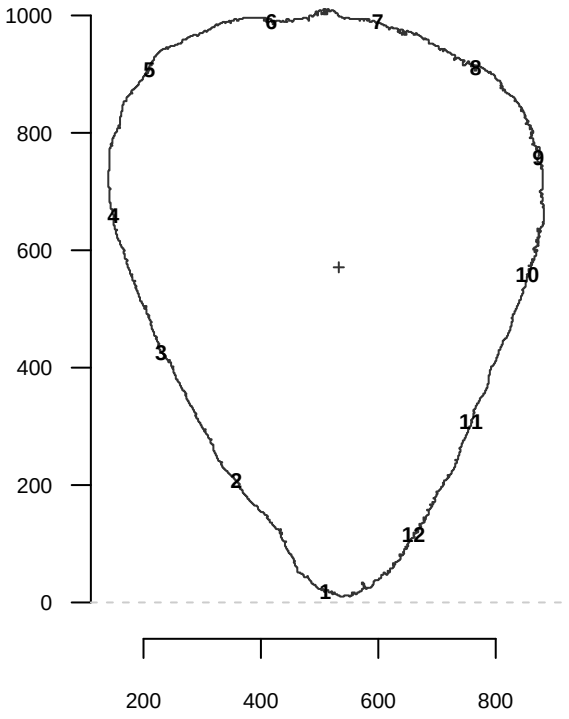
```
coo_oscillo(strw[10], "efourier")
```

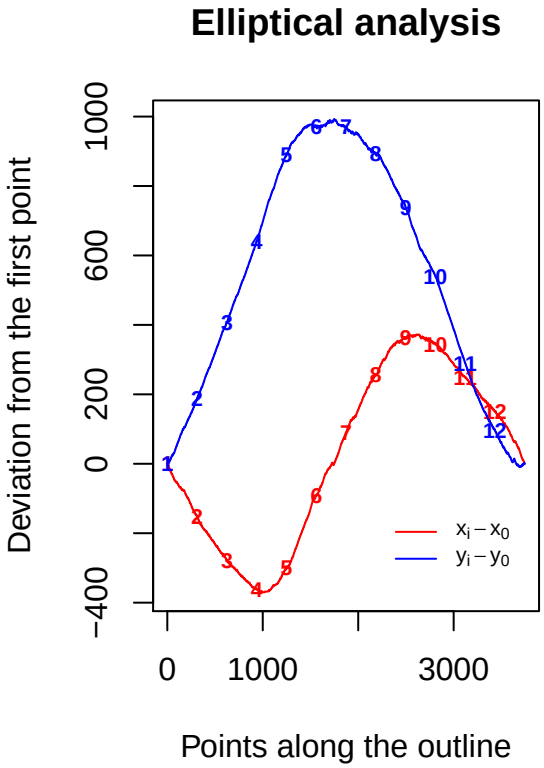





```
## # A tibble: 3,575 x 2
##       dx    dy
##    <dbl> <dbl>
##  1     0     0
##  2    -1    -1
##  3    -2    -1
##  4    -3     0
##  5    -4     0
##  6    -5     0
##  7    -6     0
##  8    -7     0
##  9    -8     0
## 10    -9     0
## # ... with 3,565 more rows
```

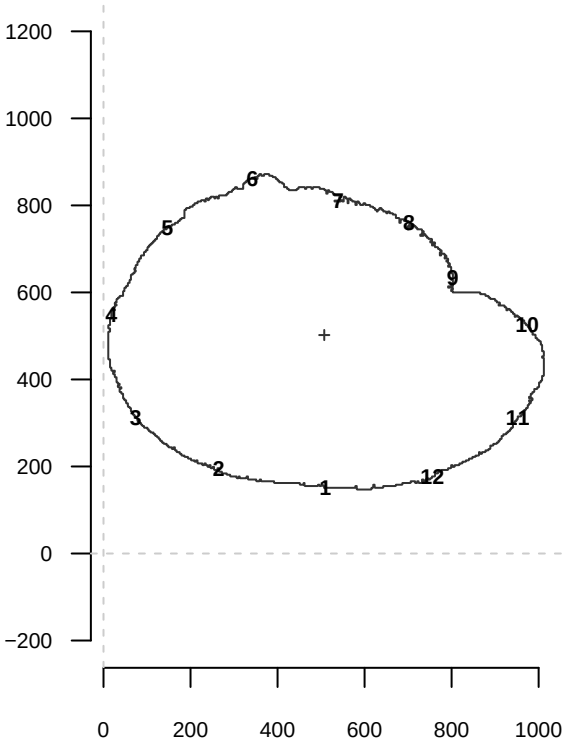
```
coo_oscillo(strw[100], "efourier")
```

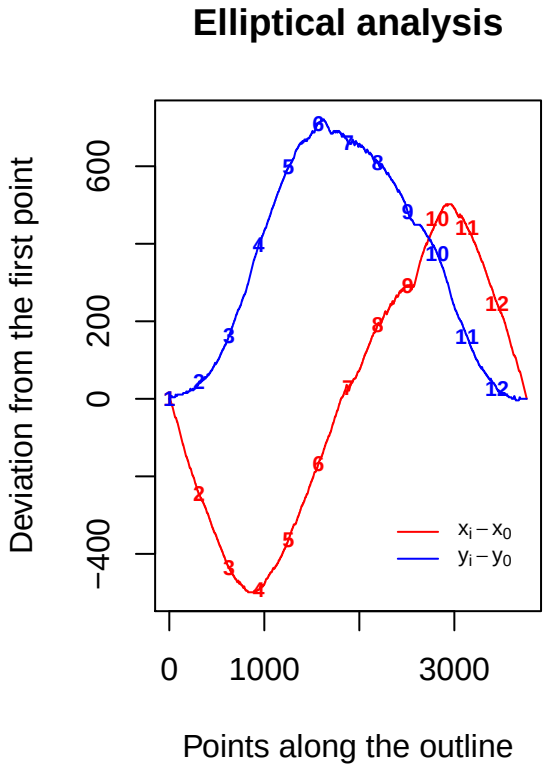





```
## # A tibble: 3,745 x 2
##       dx    dy
##    <dbl> <dbl>
##  1     0     0
##  2    -1    -1
##  3    -2    -1
##  4    -3     0
##  5    -3     1
##  6    -2     2
##  7    -3     2
##  8    -4     3
##  9    -4     4
## 10    -5     4
## # ... with 3,735 more rows
```

```
coo_oscillo(strw[1000], "efourier")
```

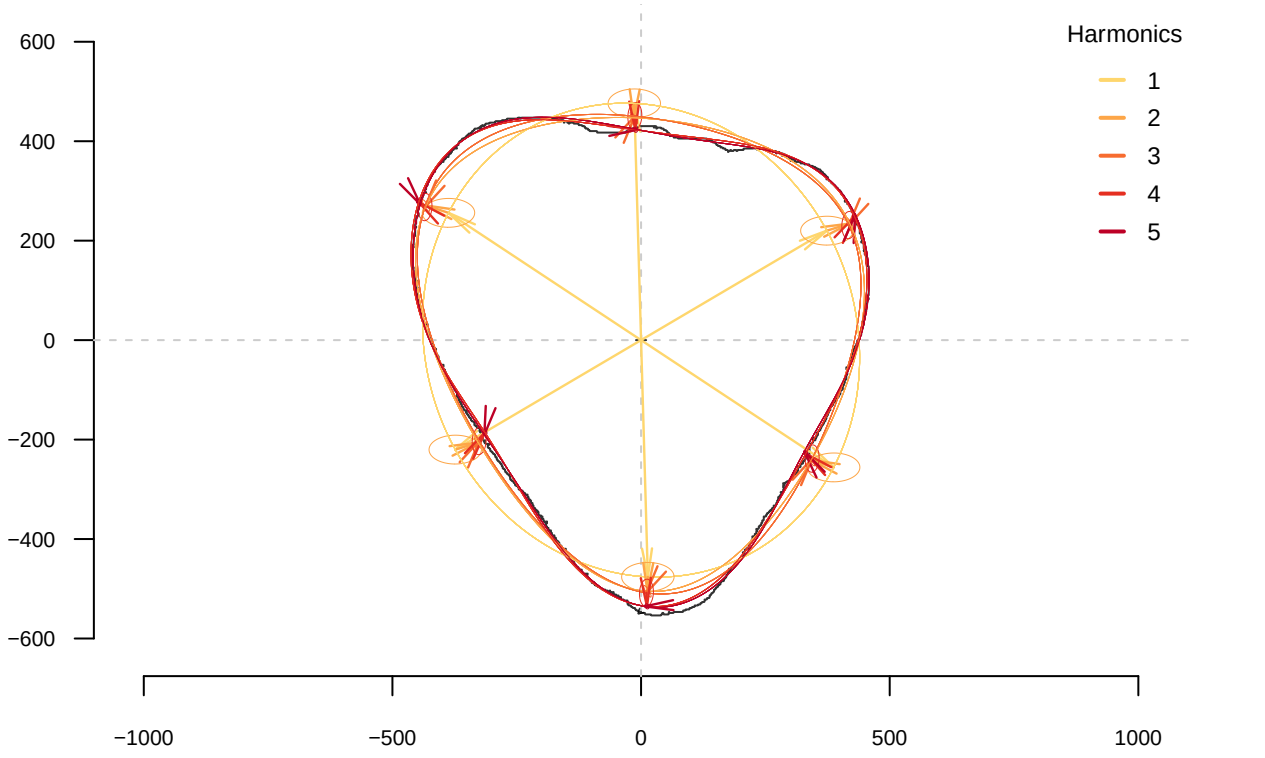




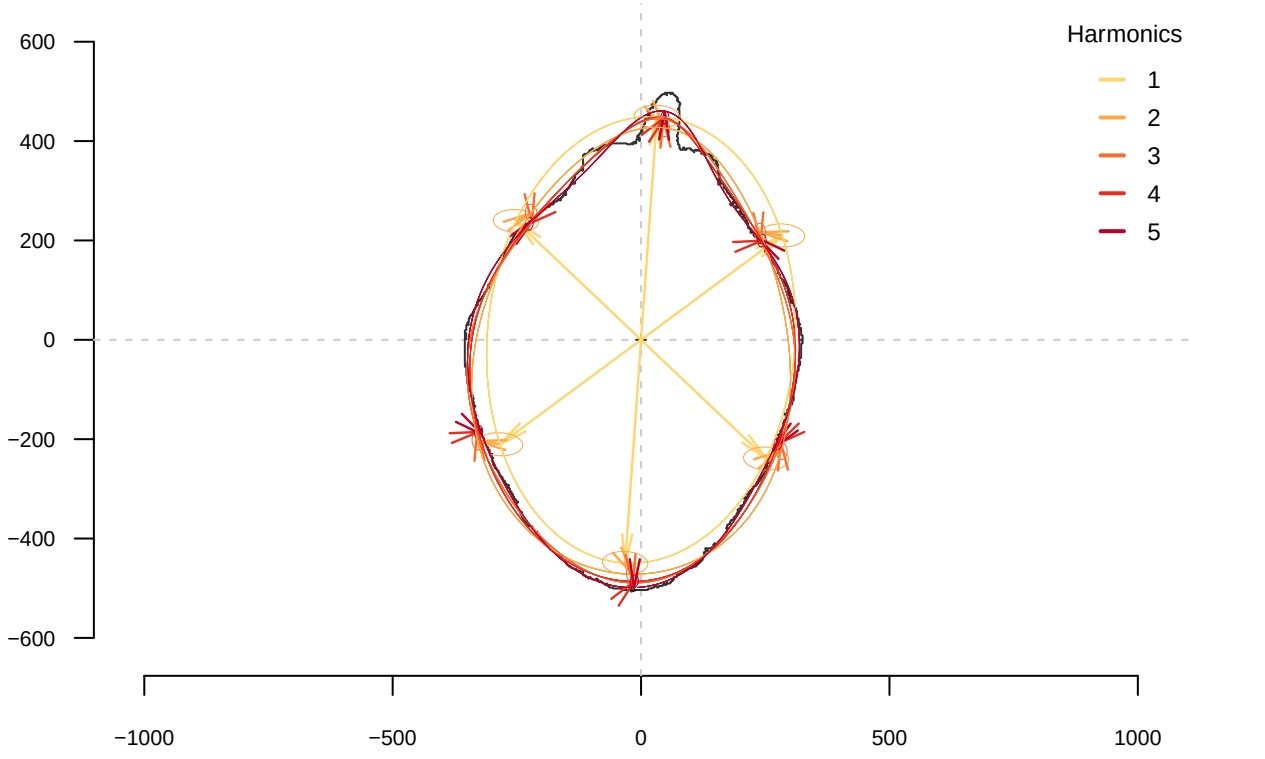

```
## # A tibble: 3,762 x 2
##       dx    dy
##    <dbl> <dbl>
##  1     0     0
##  2    -1    -1
##  3    -2    -1
##  4    -3     0
##  5    -3     1
##  6    -3     2
##  7    -3     3
##  8    -3     4
##  9    -4     4
## 10    -5     4
## # ... with 3,752 more rows
```

(2) `Ptolemy` plots the different ellipses formed from the number of harmonics `nb.h` so you can get an idea of how well the fit model reconstructs the input.

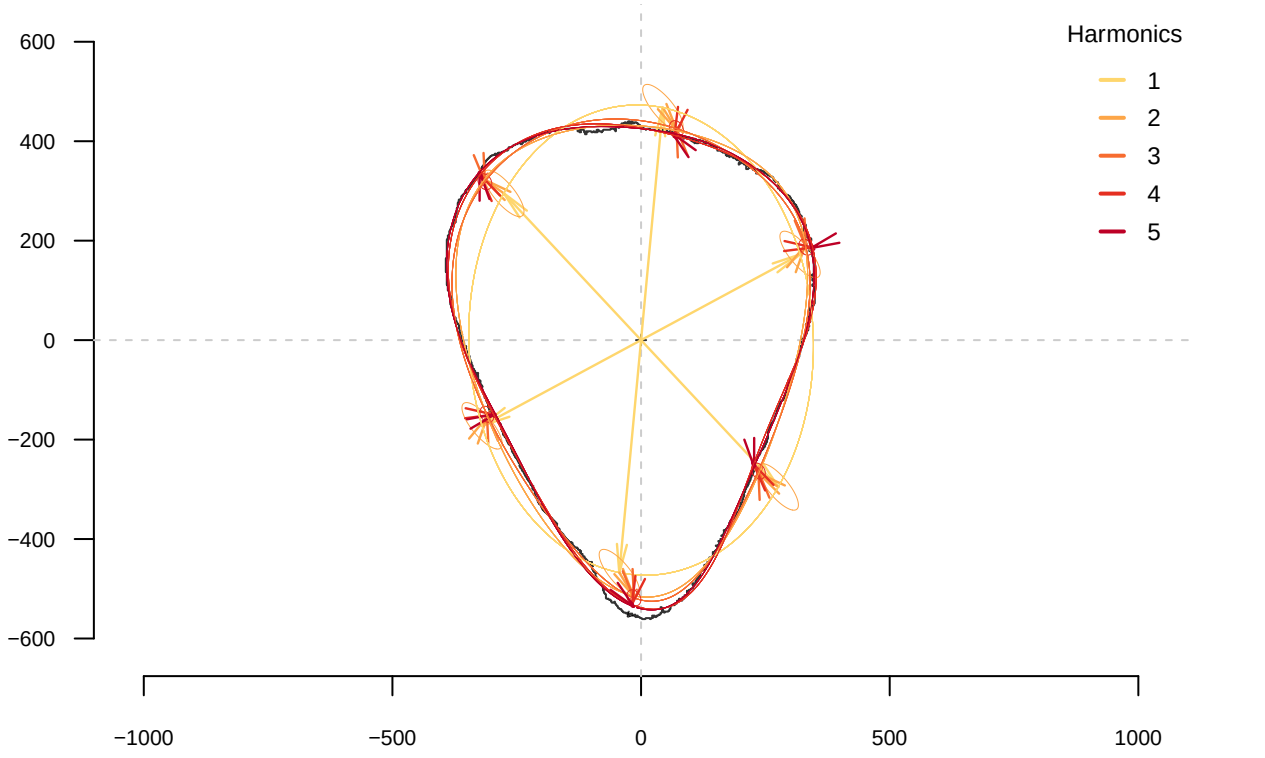
```
Ptolemy(strw[1],nb.h = 5)
```



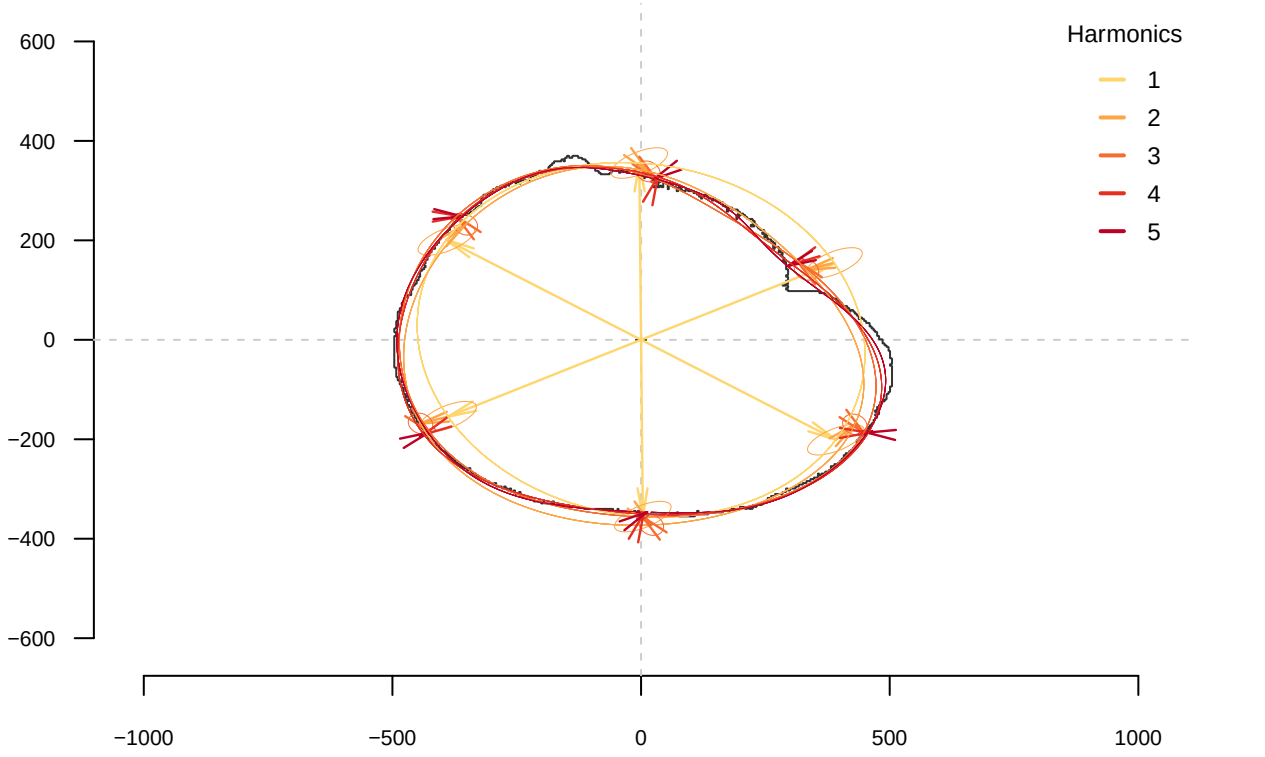

```
Ptolemy(strw[10],nb.h = 5)
```




```
Ptolemy(strw[100],nb.h = 5)
```




```
Ptolemy(strw[1000],nb.h = 5)
```

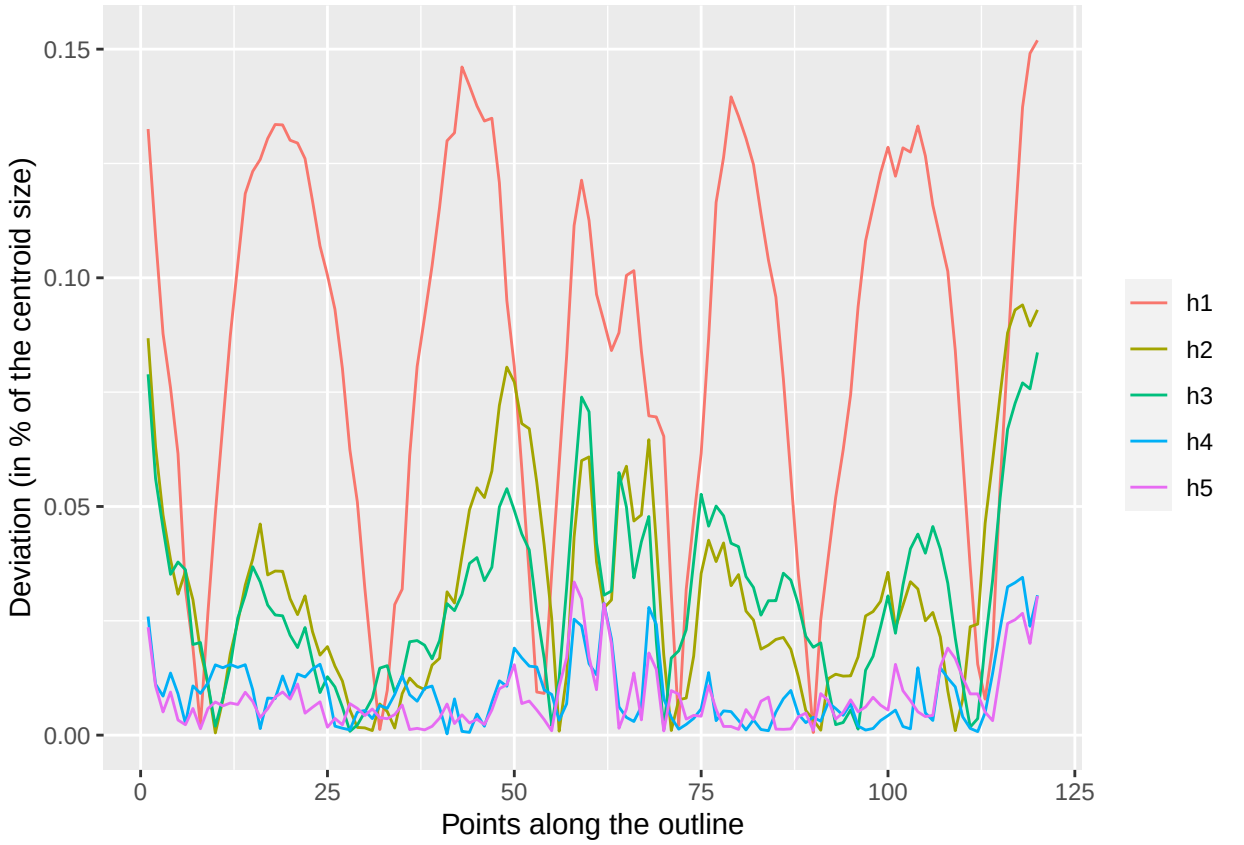


(3) `calibrate_deviations_efourier` calculates the deviation from the outline, so it quantifies the `Ptolemy` figure.

(4) `calibrate_deviations_efourier` indicates how many harmonics you need to reconstruct and image.

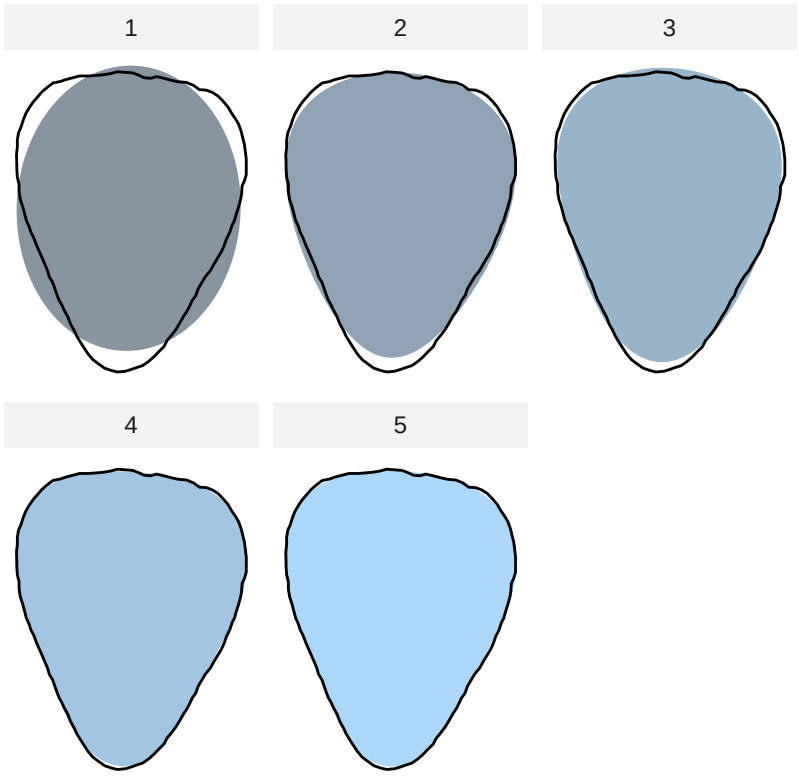
(5) `calibrate_reconstructions_efourier` you can think of this function as a multi-faceted version of `Ptolemy`. cool.

```
calibrate_deviations_efourier(strw,range = c(1:5),plot=T)
```




```
#calibrate_harmonicpower_efourier(strw,nb.h = 5)

calibrate_reconstructions_efourier(strw,range = c(1:5))
```

1483_mN_2246_16C035P032



Here we actually perform Elliptical FOurier ANalysis (EFA) I chose not to normalize the coefficients because i like that they are reflective of the amplitude of teh sine/cosine functions in pixels. COOL!

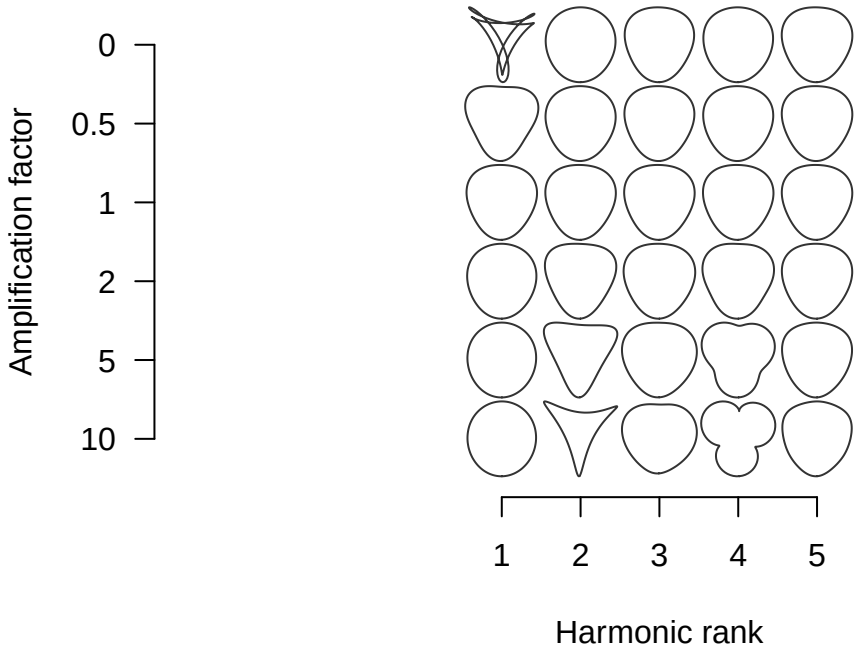
We can also view, with pre-specified scales the what each harmonic level does to the shape. The earlier harmonics are more important to the shape and are more coarse, so if they are not included you end up with very unrealistic reconstructions.

```
strw.f <- efourier(strw, nb.h=5,norm=F,start=F)

mean.shape <- MSHAPES(strw.f,nb.pts = 500)
```

```
## no 'fac' provided, returns meanshape
```

```
hcontrib(strw.f,harm.r = 1:5,mp.r = c(0,1,2,5,10))
```

```
## no 'id' provided, working on the meanshape
```



I do not care what any one says, PCA is unsupervised machine learning. It is also the easiest to implement and the components are ranked by their importance (eigenvalue), which is cool and I like that. PCA is super common to try to make sense of the 4 parameters (traits) per harmonic that you get from EFA of a 2D outline. The idea is that there are unique, uncorrelated component (principal components) that describe the data better, and more succinctly than the original, possibly correlated, axes.

We can use `MOMOCS` to view the contribution of each PC in the overall variance of shape using `PCcontrib`. Interpreting is as easy as looking and thinking, which isn't easy all the time, especially as the values become more and more minuscule.

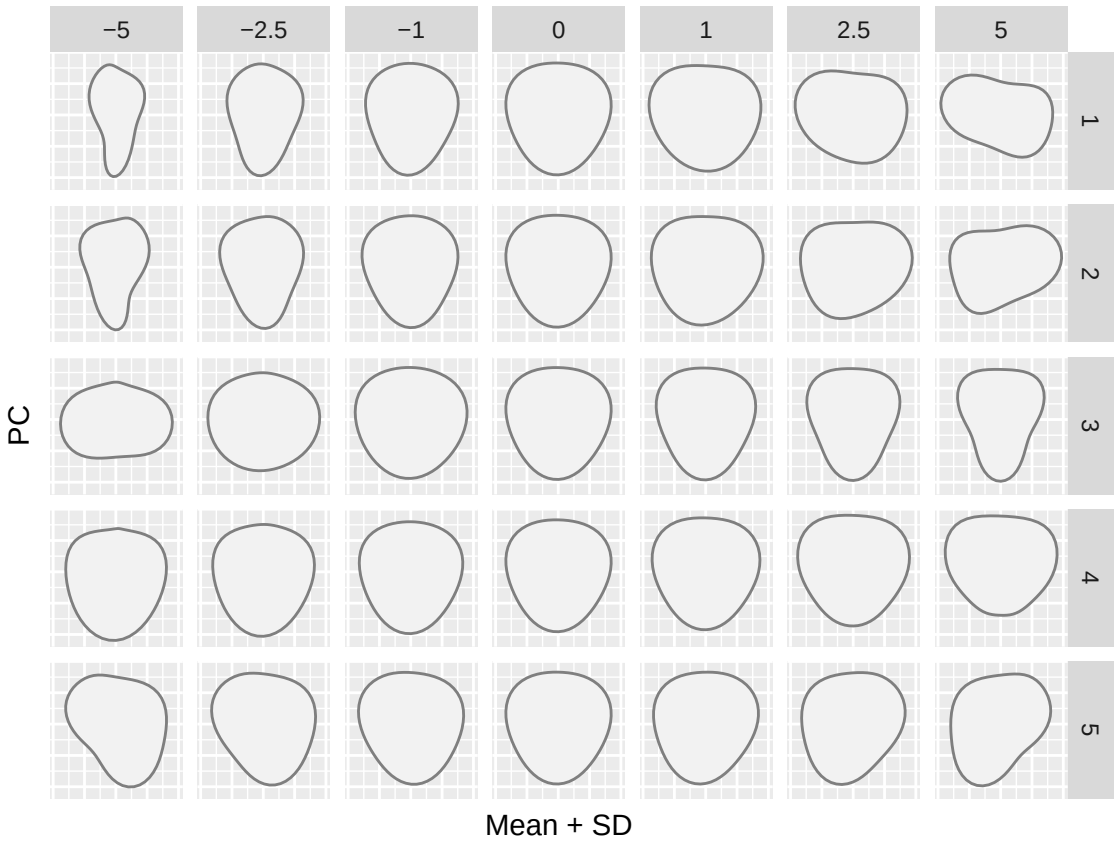
For each harmonic, there are 4 coefficients. Those coefficients can change regardless of the others and so you have a nice multidimensional description of shape. The fact that there is heritable variation in the eigenvectors suggests that there is genetic factor to shape. However, PCA does not assume that there is any latent "factor" underlying the data, unlike other methods that may have been appropriate for this paper.

```
efd.x <- strw.f[]

strw.p <- PCA(strw.f)

PCA.x <- strw.p$x
```

`MOMOCS` has a function for that...

```
PCcontrib(strw.p,nax = 1:5,sd.r = c(-5,-2.5,-1,0,1,2.5,5))
```

```
## Warning: `mutate_()` is deprecated as of dplyr 0.7.0.
## Please use `mutate()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



Again, PC1 is the most variable axis and is associated with the ratio of length and width, it turns out that this PC and the Aspect Ratio are positively correlated ($p < 0.05 / 68$). The more negative value the thinner the berry, and the opposite for positive values. COOL!

2. Generalized Procrustes Analysis of Lardmark Free Forms and our first (and only latent factor model)

Prucrustes Analysis is another classic morphometrics analysis where the distance between homologous landmarks is assessed. Simple enough. We can use and modify the same initial `coo` object from EFA for this. However, this time we are extracting 50 points along that outline. Why 50? IDK it seems reasonable and "saturating" without really exploding the dimensionality of the problem.

It is important to note that this approach relies on similarity and not homology. If you want to have homological landmarks you need to manually, or through machine learning, select and register those points prior to any analysis. The more landmarks, the more complex the registration process.

Strawberry fruit just doesn't have homologous landmarks on the 2D outline, hence this approach.

```
coo2 <- list()

for(i in 1:length(coo)){
  coo2[[i]] <- coo[[i]][round(seq(1,nrow(coo[[i]]),by=nrow(coo[[i]])/50),digits=0),]
}

strw2 <- Ldk(coo2)
```

The next chuck is all it takes...

This next chuck is a big dumb doosie of a toosie and I am not wholly convinced that it was a "good" idea.

The idea was fine, but like... not reasonable. The concept is that each landmark (remember, there are 50) is a reflection of a single latent factor (2D shape, or shape for short). This lantent factor is the cause of all of the observations that we make for each landmark. So, this is the latent factor approach that is used in this manuscript.

First we take the x and y coordinates and perform PCA on each one individually. We extract the first and second (there are only 2) PCs for each point.

```
dist.mat <- matrix(rep(0,50*length(strw.pr)),ncol=50,nrow = length(strw.pr))
disp.mat <- matrix(rep(0,50*length(strw.pr)),ncol=50,nrow = length(strw.pr))

for(i in 1:50){ #grab (x,y) of i pseudo land mark from all objects
  print(i)
  x=c()
  y=c()
  for(j in 1:length(strw.pr)){
    x[j] <- matrix(unlist(strw.pr[[1]][j]),ncol=2)[i,1]
    y[j] <- matrix(unlist(strw.pr[[1]][j]),ncol=2)[i,2]
  }
  mean.x <- mean(x)
  mean.y <- mean(y)

  dist.mat[,i] <- sqrt(((x - mean.x)**2) + ((y -mean.y)**2))
  disp.mat[,i] <- atan((y - mean.y)/(x -mean.x))

  for(k in 1:length(x)){
    ifelse((x[k]-mean.x > 0 && y[k]-mean.y > 0),
           disp.mat[k,i] <-  disp.mat[k,i],
      ifelse((x[k]-mean.x < 0 && y[k]-mean.y > 0),
             disp.mat[k,i] <-  pi + disp.mat[k,i],
        ifelse((x[k]-mean.x < 0 && y[k]-mean.y < 0),
               disp.mat[k,i] <-  pi + disp.mat[k,i],
              ifelse((x[k]-mean.x > 0 && y[k]-mean.y < 0),
                     disp.mat[k,i] <-  2*pi + disp.mat[k,i],
                            disp.mat[k,i] <-  disp.mat[k,i])))))
  }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
```

```
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
```

```
x <- matrix(unlist(strw.pr[[1]][10]),ncol=2)[,1]
y <- matrix(unlist(strw.pr[[1]][10]),ncol=2)[,2]

rownames(dist.mat) <- names(coo)
rownames(disp.mat) <- names(coo)

real.mat <- dist.mat*cos(disp.mat)
img.mat <- dist.mat*sin(disp.mat)

rownames(real.mat) <- names(coo)
rownames(img.mat) <- names(coo)
```

I thought that I was being creative by calcualting the distance and displacement of each point from the mean of that point and then transforming those into real and imaginary components to try to capture the 2 connected dimensions of the problem. But it turns out that the Real and imaginary components are just teh x and y coordinates any way... Someone else can confirm this on their own! challenge issued.

3. Shape as a Multidimensional Latent Variable and Structural Equation Models

This is the PCA on each landmark point...

```
pca.x=matrix(rep(NA,50*nrow(real.mat)),ncol=50)
pca.y=matrix(rep(NA,50*nrow(img.mat)),ncol=50)
var.x <- c()
var.y <- c()
theta <- c()

for(i in 1:50){
  print(i)
  Re_Im <- data.frame(Re = real.mat[,i],
                      Im = img.mat[,i])
  pca_dat <- prcomp(Re_Im,center=TRUE, scale.=F)

  pca.x[,i] <- pca_dat$x[,1]
  pca.y[,i] <- pca_dat$x[,2]

  theta[i] = round(acos(pca_dat$rotation[1,1]) * (180/pi),digits=1)
  var.x[i] <- var(pca.x[,i])
  var.y[i] <- var(pca.y[,i])
}
```

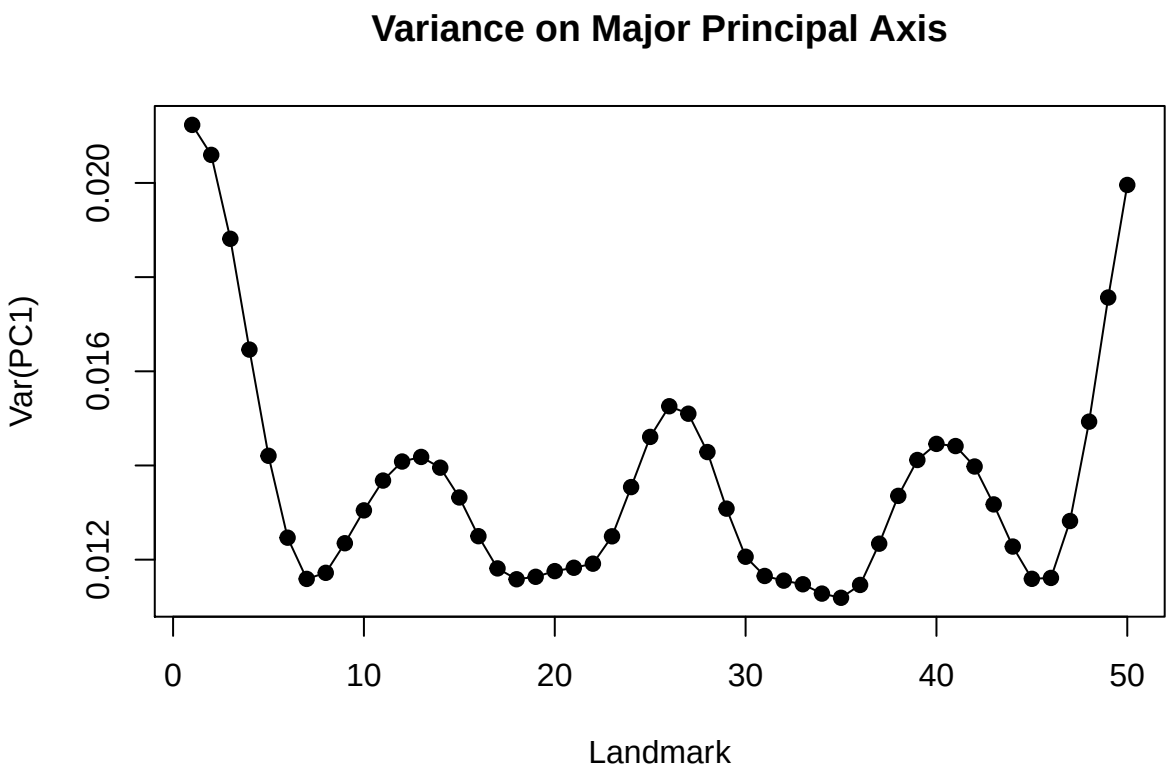
```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
```

```
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
```

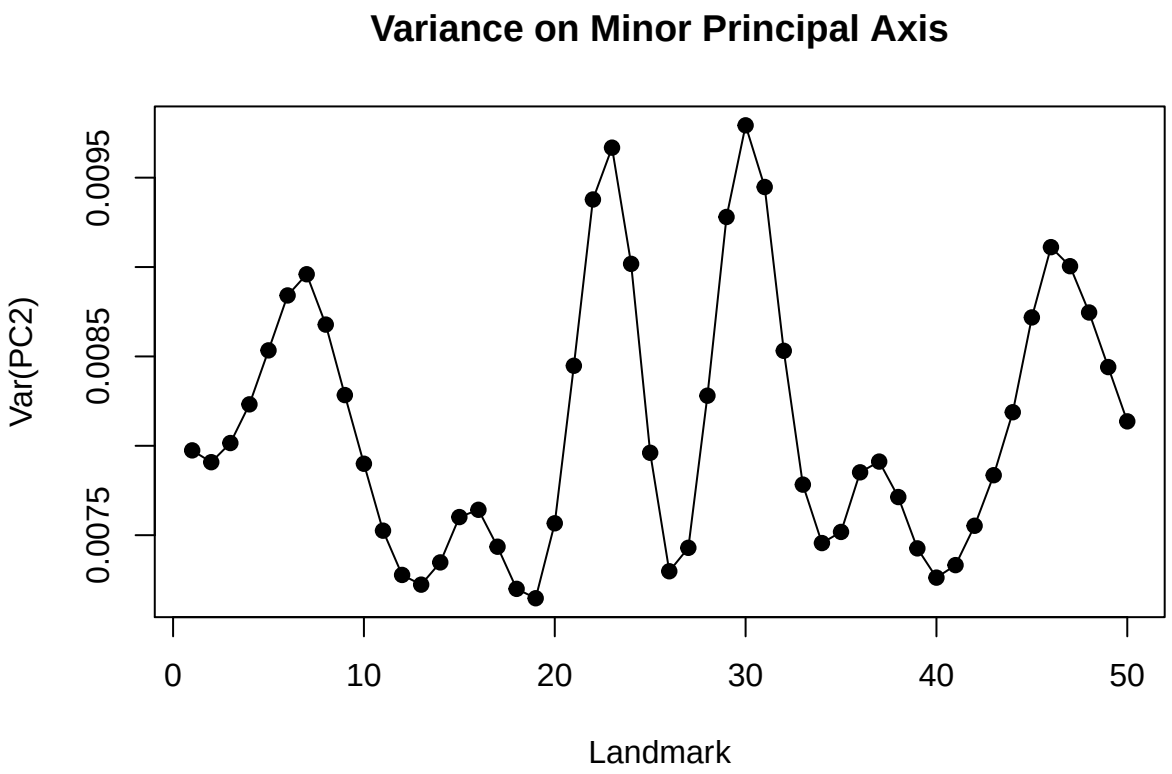
```
pca <- cbind(pca.x,pca.y)
```

these are graphs! These graphs demonstrate something very interesting about strawberry shape (see below).

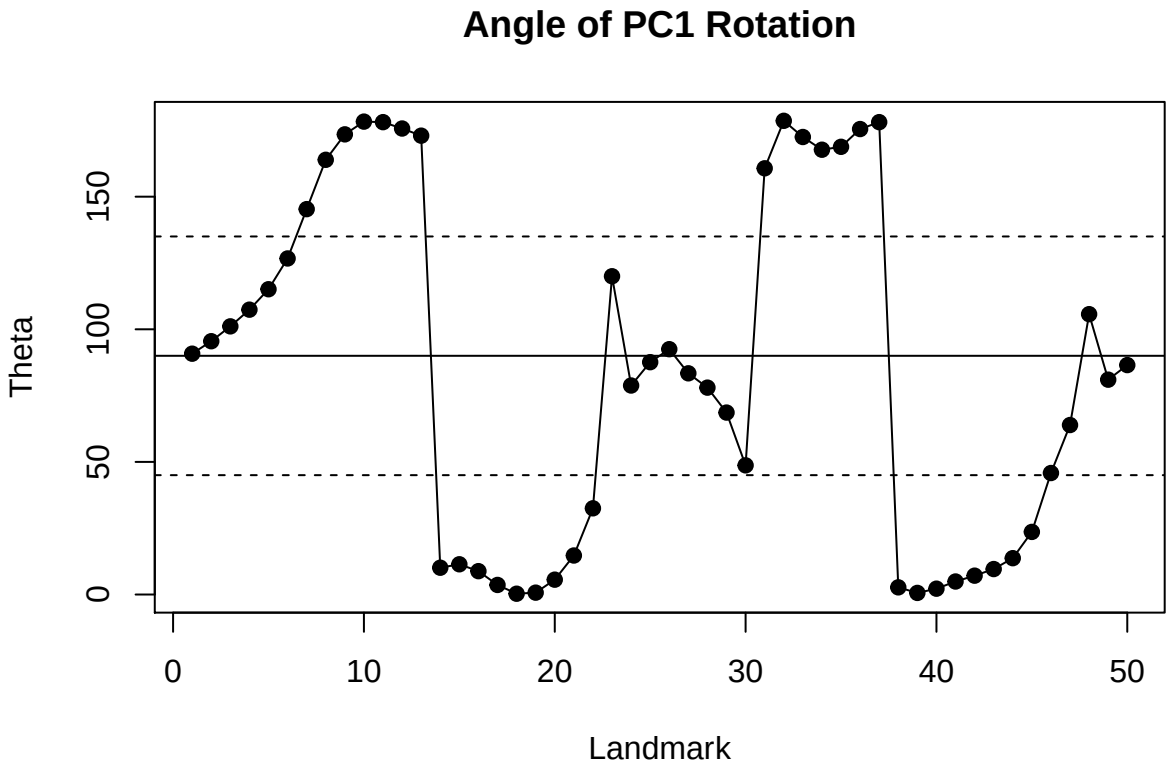
```
plot(sqrt(var.x),type="o",pch=19,
     ylab="Var(PC1)", xlab="Landmark", main = "Variance on Major Principal Axis")
```




```
plot(sqrt(var.y),type="o",pch=19,
     ylab="Var(PC2)", xlab="Landmark", main = "Variance on Minor Principal Axis")
```




```
plot(theta,type="o",pch=19, xlab="Landmark",main="Angle of PC1 Rotation",ylab="Theta")
abline(h=90)
abline(h=135,lty=2)
abline(h=45,lty=2)
```



Strawberry form and the variance of landmarks on that form is cyclic! WOW!

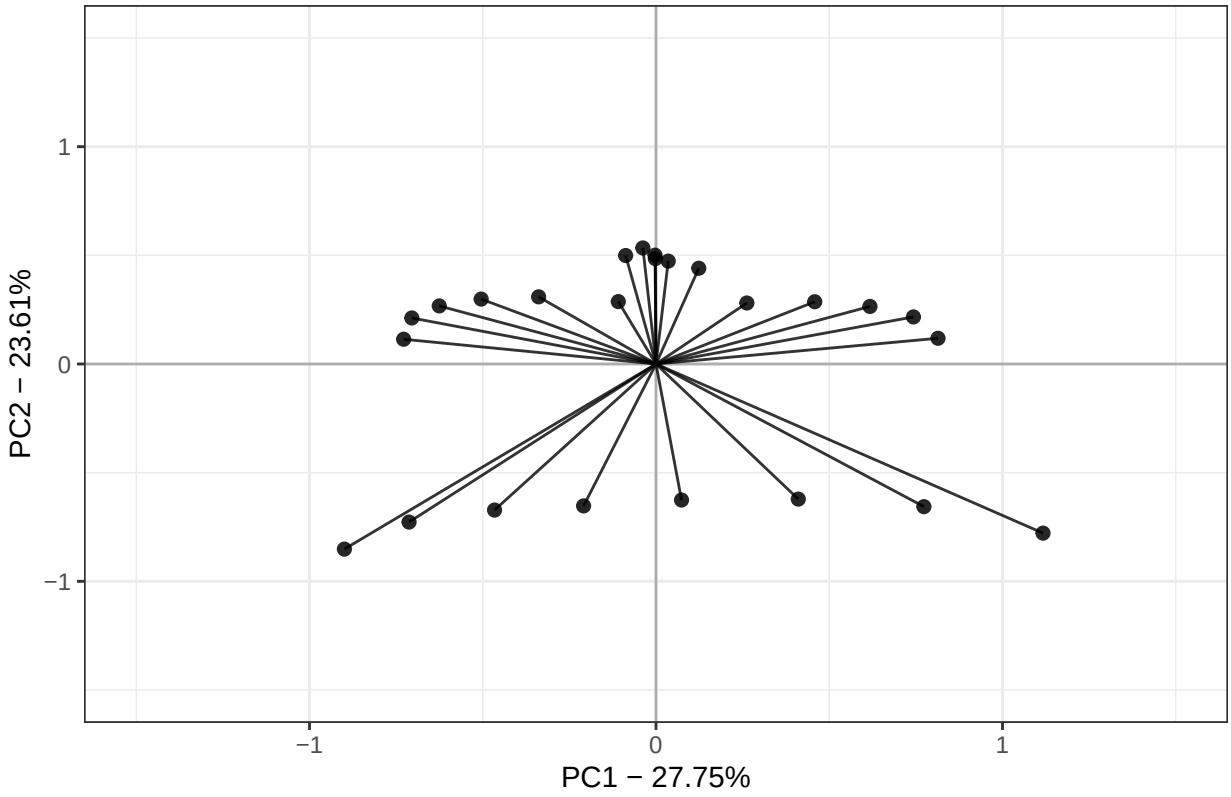
OK so it isn't that cool, but it basically tells me that there are only a few regions in the strawbverry shape that we need to capture to quantify the shape accurately.

Here, we flip flop some of the relationships to make them make a bit more sense. I have a good reason for this. PCA doesn't care if it is 100 or -100, only that 100 and -100 are opposite in magnitude. So multiplying a PC by -1 doesn't change the relationship with any other PC (uncorrelated) and this case makes a lot of sense.

```
dat = as.data.frame(scale(pca[,c(1:50)],center=T,scale=T))

dat[,11] <- -1 * dat[,11]
dat[,12] <- -1 * dat[,12]
dat[,13] <- -1 * dat[,13]
dat[,26] <- -1 * dat[,26]
dat[,24] <- -1 * dat[,24]
dat[,25] <- -1 * dat[,25]
dat[,42] <- -1 * dat[,42]
dat[,43] <- -1 * dat[,43]
dat[,38] <- -1 * dat[,38]
dat[,39] <- -1 * dat[,39]
dat[,40] <- -1 * dat[,40]
dat[,41] <- -1 * dat[,41]
dat[,48] <- -1 * dat[,48]
```

Let's PCA these PCs! and again... Nether!

```
mat = cov(scale(dat[,c( 1,  2,  3,  4,
                        5, 11, 12, 13, 14,
                        15, 24, 25, 26, 27,
                        28, 29, 38, 39, 40,
                        41, 42, 43, 48, 49, 50)],
                center = T, scale = T))
pc.dat = prcomp(mat)
scores = as.data.frame(pc.dat$x)

ggplot(data = scores, aes(x = PC1, y = PC2, label = colnames(mat))) +
  geom_hline(yintercept = 0, colour = "gray45",alpha=0.5,size=0.5) +
  geom_vline(xintercept = 0, colour = "gray45",alpha=0.5,size=0.5) +
  xlim(-1.5,1.5) +
  ylim(-1.5,1.5) +
  geom_segment(x = c(rep(0,25)),
               y = c(rep(0,25)),
               xend = c(pc.dat$x[,1]) ,
               yend = c(pc.dat$x[,2]),col = "black",alpha=0.8) +
  geom_point(x = c(pc.dat$x[,1]) ,
             y = c(pc.dat$x[,2]),alpha=0.85,col="black",size=2)+
  ggtitle("") +
  xlab("PC1 - 27.75%") + ylab("PC2 - 23.61%")+
  theme_bw()
```



This beautiful wonderful plots clearly shows 4 clusters of relatedness and anticorrelation. Very cool, Mitchell he says to him self. From these we say that there are 4 components to shape and that these 4 components give rise to "shape".

```
SHP.model <- ' Tip    =~ V1  + V2  + V3  + V4 + V5 + V48 + V49 + V50
               Side_L =~ V15 + V11 + V12 + V13 + V14
               Neck   =~  V24 + V25 + V26 + V27 + V28 + V29
               Side_R =~ V38 + V39 + V40 + V41 + V42 + V43
               Shape =~ Tip + Side_L + Neck + Side_R
               Tip ~~ 1*Tip
               Side_L ~~ 1*Side_L
               Side_R ~~ 1*Side_R
               Neck ~~ 1*Neck
               Shape ~~ 1*Shape
               '
```

```
fit <- lavaan::sem(SHP.model, data = dat,std.lv=T,estimator="DWLS")
summary(fit)
```

```
## lavaan 0.6-7 ended normally after 95 iterations
##
##   Estimator                                         DWLS
##   Optimization method                           NLMINB
##   Number of free parameters                         54
##
##   Number of observations                          6874
##
## Model Test User Model:
##
##   Test statistic                              9724.756
```

```
##   Degrees of freedom                                271
##   P-value (Chi-square)                            0.000
##
## Parameter Estimates:
##
##   Standard errors                             Standard
##   Information                                 Expected
##   Information saturated (h1) model        Unstructured
##
## Latent Variables:
##                    Estimate  Std.Err  z-value  P(>|z|)
##   Tip =~
##     V1                0.542    0.007   75.632    0.000
##     V2                0.551    0.007   75.985    0.000
##     V3                0.532    0.007   76.209    0.000
##     V4                0.470    0.006   75.902    0.000
##     V5                0.344    0.005   72.832    0.000
##     V48               0.362    0.005   71.579    0.000
##     V49               0.466    0.006   74.594    0.000
##     V50               0.520    0.007   75.391    0.000
##   Side_L =~
##     V15               0.455    0.006   73.706    0.000
##     V11               0.526    0.007   75.094    0.000
##     V12               0.526    0.007   74.527    0.000
##     V13               0.513    0.007   74.145    0.000
##     V14               0.487    0.007   73.895    0.000
##   Neck =~
##     V24               0.393    0.007   58.262    0.000
##     V25               0.430    0.007   58.402    0.000
##     V26               0.446    0.008   58.334    0.000
##     V27               0.447    0.008   58.413    0.000
##     V28               0.429    0.007   58.563    0.000
##     V29               0.384    0.007   58.438    0.000
##   Side_R =~
##     V38               0.403    0.006   66.813    0.000
##     V39               0.431    0.006   66.877    0.000
##     V40               0.453    0.007   66.920    0.000
##     V41               0.466    0.007   67.104    0.000
##     V42               0.465    0.007   67.346    0.000
##     V43               0.444    0.007   67.539    0.000
##   Shape =~
##     Tip               1.388    0.019   74.660    0.000
##     Side_L            1.615    0.023   68.921    0.000
##     Neck              1.800    0.033   55.157    0.000
##     Side_R            1.862    0.030   61.987    0.000
##
## Variances:
##                    Estimate  Std.Err  z-value  P(>|z|)
##    .Tip               1.000
##    .Side_L            1.000
##    .Side_R            1.000
##    .Neck              1.000
##     Shape             1.000
##    .V1                0.140    0.034    4.092    0.000
```

```
##    .V2                0.111    0.033    3.366    0.001
##    .V3                0.171    0.030    5.610    0.000
##    .V4                0.354    0.027   13.049    0.000
##    .V5                0.653    0.025   25.836    0.000
##    .V48               0.616    0.027   22.628    0.000
##    .V49               0.363    0.032   11.312    0.000
##    .V50               0.209    0.034    6.138    0.000
##    .V15               0.254    0.021   12.061    0.000
##    .V11               0.003    0.021    0.151    0.880
##    .V12               0.000    0.020    0.014    0.989
##    .V13               0.051    0.020    2.550    0.011
##    .V14               0.144    0.020    7.204    0.000
##    .V24               0.346    0.025   13.586    0.000
##    .V25               0.216    0.024    9.043    0.000
##    .V26               0.156    0.023    6.716    0.000
##    .V27               0.154    0.023    6.716    0.000
##    .V28               0.219    0.023    9.666    0.000
##    .V29               0.374    0.023   16.233    0.000
##    .V38               0.274    0.020   13.394    0.000
##    .V39               0.170    0.020    8.586    0.000
##    .V40               0.084    0.020    4.200    0.000
##    .V41               0.032    0.020    1.595    0.111
##    .V42               0.035    0.020    1.740    0.082
##    .V43               0.121    0.020    6.083    0.000
```

```
inspect(fit, "sample")$cov
```

```
##     V1    V2    V3    V4    V5    V48   V49   V50   V15   V11   V12   V13
## V1  1.000
## V2  0.965 1.000
## V3  0.872 0.957 1.000
## V4  0.718 0.839 0.947 1.000
## V5  0.487 0.632 0.789 0.931 1.000
## V48 0.698 0.568 0.415 0.231 0.013 1.000
## V49 0.870 0.763 0.621 0.436 0.198 0.939 1.000
## V50 0.966 0.889 0.767 0.593 0.352 0.822 0.955 1.000
## V15 0.524 0.590 0.653 0.694 0.675 0.215 0.359 0.457 1.000
## V11 0.660 0.682 0.673 0.612 0.462 0.413 0.550 0.624 0.759 1.000
## V12 0.632 0.671 0.688 0.660 0.550 0.354 0.498 0.584 0.851 0.976 1.000
## V13 0.599 0.650 0.687 0.687 0.611 0.299 0.448 0.541 0.923 0.928 0.979 1.000
## V14 0.557 0.618 0.670 0.693 0.649 0.247 0.396 0.493 0.974 0.857 0.931 0.978
## V24 0.464 0.459 0.428 0.367 0.251 0.320 0.413 0.456 0.624 0.707 0.701 0.688
## V25 0.536 0.532 0.504 0.441 0.319 0.389 0.484 0.525 0.643 0.697 0.692 0.683
## V26 0.561 0.563 0.540 0.479 0.356 0.404 0.503 0.547 0.640 0.689 0.685 0.676
## V27 0.555 0.565 0.548 0.492 0.376 0.376 0.481 0.534 0.631 0.676 0.675 0.667
## V28 0.507 0.521 0.508 0.459 0.351 0.330 0.430 0.484 0.593 0.647 0.645 0.635
## V29 0.418 0.431 0.418 0.372 0.272 0.259 0.348 0.397 0.508 0.598 0.589 0.571
## V38 0.609 0.564 0.492 0.379 0.210 0.665 0.681 0.652 0.432 0.585 0.550 0.510
## V39 0.637 0.596 0.526 0.411 0.235 0.650 0.688 0.672 0.463 0.650 0.611 0.565
## V40 0.663 0.632 0.567 0.455 0.277 0.621 0.685 0.687 0.500 0.705 0.666 0.617
## V41 0.681 0.660 0.605 0.501 0.325 0.569 0.664 0.689 0.535 0.745 0.708 0.660
## V42 0.681 0.675 0.633 0.540 0.371 0.490 0.618 0.671 0.567 0.772 0.739 0.693
## V43 0.652 0.666 0.642 0.568 0.418 0.364 0.531 0.617 0.591 0.773 0.748 0.709
##     V14   V24   V25   V26   V27   V28   V29   V38   V39   V40   V41   V42
## V1
```

```
## V2
## V3
## V4
## V5
## V48
## V49
## V50
## V15
## V11
## V12
## V13
## V14 1.000
## V24 0.664 1.000
## V25 0.667 0.903 1.000
## V26 0.660 0.770 0.915 1.000
## V27 0.651 0.678 0.790 0.916 1.000
## V28 0.617 0.624 0.703 0.794 0.915 1.000
## V29 0.545 0.563 0.606 0.657 0.753 0.901 1.000
## V38 0.467 0.529 0.607 0.638 0.648 0.657 0.629 1.000
## V39 0.512 0.562 0.630 0.658 0.668 0.680 0.667 0.975 1.000
## V40 0.560 0.586 0.645 0.672 0.681 0.693 0.686 0.929 0.980 1.000
## V41 0.601 0.603 0.654 0.679 0.690 0.700 0.696 0.863 0.935 0.979 1.000
## V42 0.634 0.610 0.654 0.678 0.690 0.697 0.695 0.774 0.862 0.928 0.976 1.000
## V43 0.656 0.600 0.636 0.657 0.673 0.680 0.677 0.654 0.754 0.839 0.912 0.969
##     V43
## V1
## V2
## V3
## V4
## V5
## V48
## V49
## V50
## V15
## V11
## V12
## V13
## V14
## V24
## V25
## V26
## V27
## V28
## V29
## V38
## V39
## V40
## V41
## V42
## V43 1.000
```

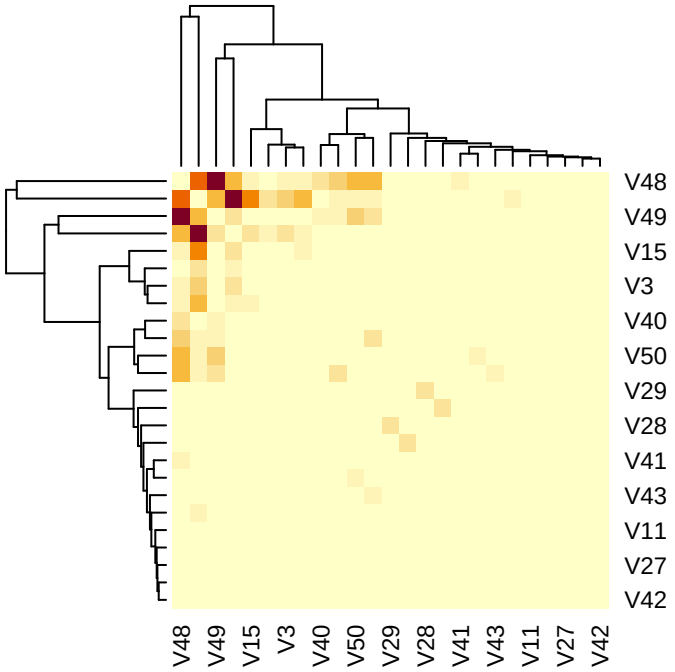
```
inspect(fit, "sigma")
```

```
##     V1    V2    V3    V4    V5    V48   V49   V50   V15   V11   V12   V13
## V1  1.000
```

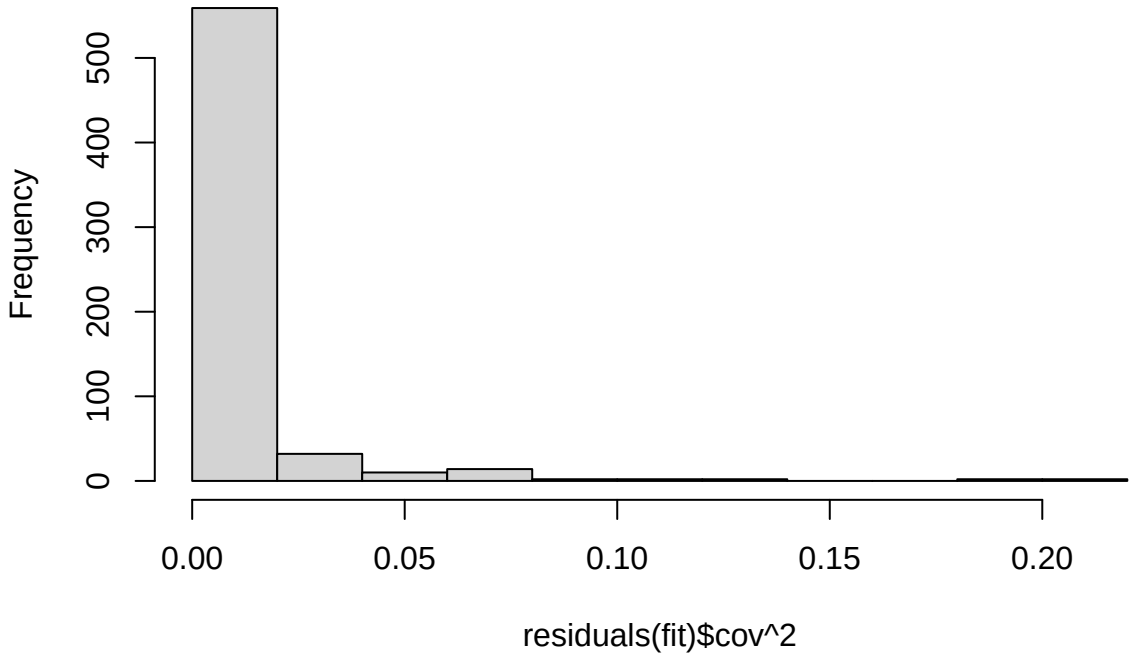
```
## V2  0.875 1.000
## V3  0.845 0.859 1.000
## V4  0.745 0.758 0.732 1.000
## V5  0.546 0.555 0.536 0.473 1.000
## V48 0.575 0.584 0.564 0.498 0.365 1.000
## V49 0.740 0.752 0.727 0.641 0.470 0.494 1.000
## V50 0.825 0.839 0.810 0.715 0.524 0.551 0.710 1.000
## V15 0.553 0.562 0.543 0.479 0.351 0.369 0.476 0.530 1.000
## V11 0.639 0.650 0.627 0.554 0.406 0.427 0.550 0.613 0.863 1.000
## V12 0.640 0.651 0.628 0.554 0.406 0.427 0.550 0.614 0.864 0.998 1.000
## V13 0.623 0.634 0.612 0.540 0.396 0.416 0.536 0.598 0.842 0.973 0.974 1.000
## V14 0.592 0.602 0.581 0.513 0.376 0.395 0.509 0.568 0.799 0.924 0.925 0.901
## V24 0.532 0.541 0.522 0.461 0.338 0.355 0.458 0.510 0.519 0.600 0.601 0.586
## V25 0.582 0.592 0.572 0.505 0.370 0.389 0.501 0.559 0.569 0.657 0.658 0.641
## V26 0.604 0.614 0.593 0.524 0.384 0.404 0.520 0.580 0.590 0.682 0.683 0.665
## V27 0.605 0.615 0.594 0.524 0.384 0.404 0.521 0.580 0.591 0.683 0.684 0.666
## V28 0.581 0.591 0.571 0.504 0.369 0.388 0.500 0.558 0.567 0.656 0.657 0.640
## V29 0.520 0.529 0.511 0.451 0.330 0.348 0.448 0.499 0.508 0.587 0.588 0.573
## V38 0.565 0.575 0.555 0.490 0.359 0.377 0.486 0.542 0.552 0.637 0.638 0.622
## V39 0.604 0.614 0.593 0.523 0.383 0.403 0.520 0.579 0.590 0.681 0.682 0.665
## V40 0.635 0.645 0.623 0.550 0.403 0.424 0.546 0.609 0.620 0.716 0.717 0.699
## V41 0.652 0.663 0.641 0.565 0.414 0.436 0.561 0.626 0.637 0.736 0.737 0.718
## V42 0.651 0.662 0.640 0.564 0.414 0.435 0.560 0.625 0.636 0.735 0.736 0.717
## V43 0.622 0.632 0.610 0.539 0.395 0.415 0.535 0.596 0.607 0.701 0.702 0.684
##     V14   V24   V25   V26   V27   V28   V29   V38   V39   V40   V41   V42
## V1
## V2
## V3
## V4
## V5
## V48
## V49
## V50
## V15
## V11
## V12
## V13
## V14 1.000
## V24 0.556 1.000
## V25 0.609 0.716 1.000
## V26 0.631 0.743 0.813 1.000
## V27 0.632 0.744 0.814 0.845 1.000
## V28 0.607 0.714 0.782 0.812 0.813 1.000
## V29 0.544 0.640 0.700 0.727 0.728 0.699 1.000
## V38 0.591 0.531 0.581 0.603 0.604 0.580 0.519 1.000
## V39 0.631 0.567 0.621 0.644 0.645 0.620 0.555 0.776 1.000
## V40 0.663 0.596 0.653 0.677 0.678 0.651 0.583 0.816 0.872 1.000
## V41 0.682 0.613 0.671 0.696 0.697 0.670 0.600 0.839 0.896 0.942 1.000
## V42 0.681 0.612 0.670 0.695 0.696 0.669 0.599 0.837 0.895 0.940 0.967 1.000
## V43 0.650 0.584 0.639 0.663 0.664 0.638 0.571 0.799 0.854 0.897 0.923 0.921
##     V43
## V1
## V2
## V3
```

```
## V4
## V5
## V48
## V49
## V50
## V15
## V11
## V12
## V13
## V14
## V24
## V25
## V26
## V27
## V28
## V29
## V38
## V39
## V40
## V41
## V42
## V43 1.000
```

```
heatmap(residuals(fit)$cov**2,symm = T,margins = c(8,8))
```




```
hist(residuals(fit)$cov**2)
```



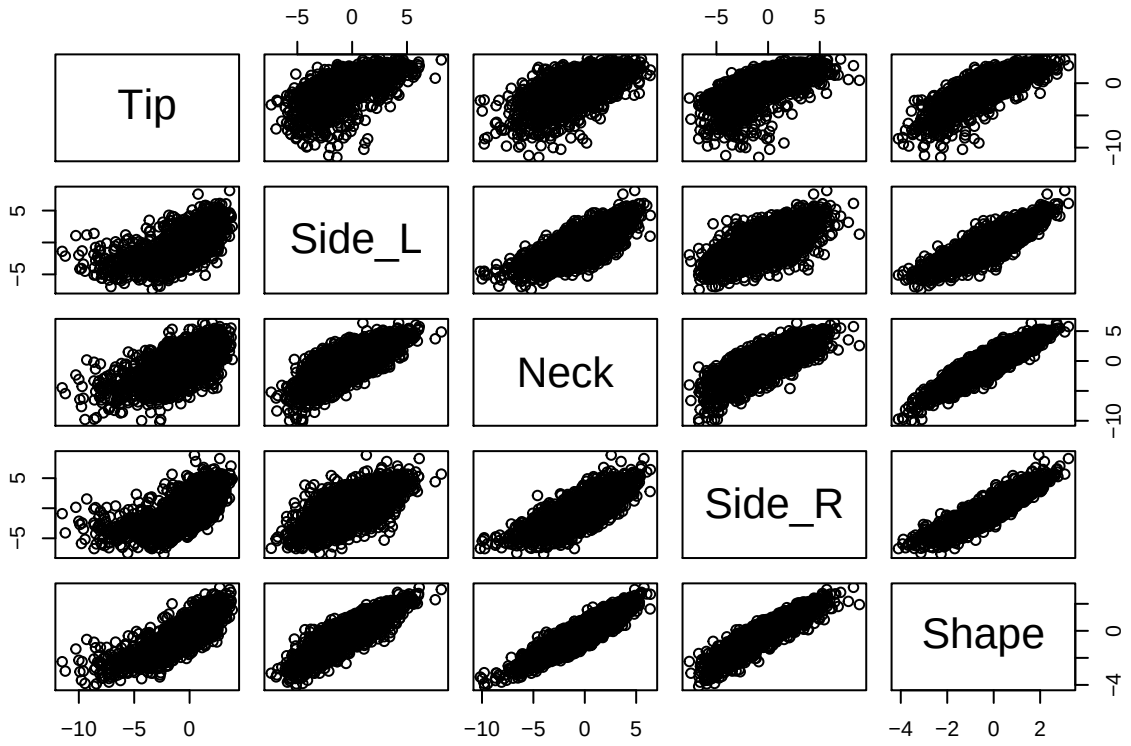

```
parameterestimates(fit,standardized = T)
```

```
##       lhs op    rhs   est    se      z pvalue ci.lower ci.upper std.lv std.all
## 1     Tip =~     V1 0.542 0.007 75.632  0.000    0.528    0.556  0.927   0.927
## 2     Tip =~     V2 0.551 0.007 75.985  0.000    0.537    0.565  0.943   0.943
## 3     Tip =~     V3 0.532 0.007 76.209  0.000    0.519    0.546  0.911   0.911
## 4     Tip =~     V4 0.470 0.006 75.902  0.000    0.458    0.482  0.804   0.804
## 5     Tip =~     V5 0.344 0.005 72.832  0.000    0.335    0.353  0.589   0.589
## 6     Tip =~    V48 0.362 0.005 71.579  0.000    0.352    0.372  0.620   0.620
## 7     Tip =~    V49 0.466 0.006 74.594  0.000    0.454    0.479  0.798   0.798
## 8     Tip =~    V50 0.520 0.007 75.391  0.000    0.506    0.533  0.890   0.890
## 9  Side_L =~    V15 0.455 0.006 73.706  0.000    0.443    0.467  0.864   0.864
## 10 Side_L =~    V11 0.526 0.007 75.094  0.000    0.512    0.539  0.998   0.998
## 11 Side_L =~    V12 0.526 0.007 74.527  0.000    0.512    0.540  1.000   1.000
## 12 Side_L =~    V13 0.513 0.007 74.145  0.000    0.499    0.526  0.974   0.974
## 13 Side_L =~    V14 0.487 0.007 73.895  0.000    0.474    0.500  0.925   0.925
## 14   Neck =~    V24 0.393 0.007 58.262  0.000    0.379    0.406  0.809   0.809
## 15   Neck =~    V25 0.430 0.007 58.402  0.000    0.415    0.444  0.885   0.885
## 16   Neck =~    V26 0.446 0.008 58.334  0.000    0.431    0.461  0.918   0.918
## 17   Neck =~    V27 0.447 0.008 58.413  0.000    0.432    0.462  0.920   0.920
## 18   Neck =~    V28 0.429 0.007 58.563  0.000    0.415    0.443  0.884   0.884
## 19   Neck =~    V29 0.384 0.007 58.438  0.000    0.371    0.397  0.791   0.791
## 20 Side_R =~    V38 0.403 0.006 66.813  0.000    0.391    0.415  0.852   0.852
## 21 Side_R =~    V39 0.431 0.006 66.877  0.000    0.418    0.444  0.911   0.911
## 22 Side_R =~    V40 0.453 0.007 66.920  0.000    0.440    0.466  0.957   0.957
## 23 Side_R =~    V41 0.466 0.007 67.104  0.000    0.452    0.479  0.984   0.984
## 24 Side_R =~    V42 0.465 0.007 67.346  0.000    0.451    0.478  0.982   0.982
## 25 Side_R =~    V43 0.444 0.007 67.539  0.000    0.431    0.456  0.938   0.938
## 26  Shape =~    Tip 1.388 0.019 74.660  0.000    1.352    1.425  0.811   0.811
## 27  Shape =~ Side_L 1.615 0.023 68.921  0.000    1.569    1.661  0.850   0.850
```

```
## 28  Shape =~   Neck 1.800 0.033 55.157  0.000    1.736    1.864  0.874   0.874
## 29  Shape =~ Side_R 1.862 0.030 61.987  0.000    1.803    1.921  0.881   0.881
## 30    Tip ~~    Tip 1.000 0.000     NA     NA    1.000    1.000  0.342   0.342
## 31 Side_L ~~ Side_L 1.000 0.000     NA     NA    1.000    1.000  0.277   0.277
## 32 Side_R ~~ Side_R 1.000 0.000     NA     NA    1.000    1.000  0.224   0.224
## 33   Neck ~~   Neck 1.000 0.000     NA     NA    1.000    1.000  0.236   0.236
## 34  Shape ~~  Shape 1.000 0.000     NA     NA    1.000    1.000  1.000   1.000
## 35     V1 ~~     V1 0.140 0.034  4.092  0.000    0.073    0.207  0.140   0.140
## 36     V2 ~~     V2 0.111 0.033  3.366  0.001    0.046    0.175  0.111   0.111
## 37     V3 ~~     V3 0.171 0.030  5.610  0.000    0.111    0.230  0.171   0.171
## 38     V4 ~~     V4 0.354 0.027 13.049  0.000    0.301    0.407  0.354   0.354
## 39     V5 ~~     V5 0.653 0.025 25.836  0.000    0.604    0.703  0.653   0.653
## 40    V48 ~~    V48 0.616 0.027 22.628  0.000    0.563    0.670  0.616   0.616
## 41    V49 ~~    V49 0.363 0.032 11.312  0.000    0.300    0.426  0.363   0.363
## 42    V50 ~~    V50 0.209 0.034  6.138  0.000    0.142    0.275  0.209   0.209
## 43    V15 ~~    V15 0.254 0.021 12.061  0.000    0.212    0.295  0.254   0.254
## 44    V11 ~~    V11 0.003 0.021  0.151  0.880   -0.037    0.043  0.003   0.003
## 45    V12 ~~    V12 0.000 0.020  0.014  0.989   -0.039    0.040  0.000   0.000
## 46    V13 ~~    V13 0.051 0.020  2.550  0.011    0.012    0.089  0.051   0.051
## 47    V14 ~~    V14 0.144 0.020  7.204  0.000    0.105    0.184  0.144   0.144
## 48    V24 ~~    V24 0.346 0.025 13.586  0.000    0.296    0.396  0.346   0.346
## 49    V25 ~~    V25 0.216 0.024  9.043  0.000    0.169    0.263  0.216   0.216
## 50    V26 ~~    V26 0.156 0.023  6.716  0.000    0.111    0.202  0.156   0.156
## 51    V27 ~~    V27 0.154 0.023  6.716  0.000    0.109    0.199  0.154   0.154
## 52    V28 ~~    V28 0.219 0.023  9.666  0.000    0.175    0.264  0.219   0.219
## 53    V29 ~~    V29 0.374 0.023 16.233  0.000    0.329    0.419  0.374   0.374
## 54    V38 ~~    V38 0.274 0.020 13.394  0.000    0.234    0.314  0.274   0.274
## 55    V39 ~~    V39 0.170 0.020  8.586  0.000    0.131    0.209  0.170   0.170
## 56    V40 ~~    V40 0.084 0.020  4.200  0.000    0.045    0.123  0.084   0.084
## 57    V41 ~~    V41 0.032 0.020  1.595  0.111   -0.007    0.071  0.032   0.032
## 58    V42 ~~    V42 0.035 0.020  1.740  0.082   -0.004    0.074  0.035   0.035
## 59    V43 ~~    V43 0.121 0.020  6.083  0.000    0.082    0.160  0.121   0.121
##    std.nox
## 1    0.927
## 2    0.943
## 3    0.911
## 4    0.804
## 5    0.589
## 6    0.620
## 7    0.798
## 8    0.890
## 9    0.864
## 10   0.998
## 11   1.000
## 12   0.974
## 13   0.925
## 14   0.809
## 15   0.885
## 16   0.918
## 17   0.920
## 18   0.884
## 19   0.791
## 20   0.852
## 21   0.911
```

```
## 22   0.957
## 23   0.984
## 24   0.982
## 25   0.938
## 26   0.811
## 27   0.850
## 28   0.874
## 29   0.881
## 30   0.342
## 31   0.277
## 32   0.224
## 33   0.236
## 34   1.000
## 35   0.140
## 36   0.111
## 37   0.171
## 38   0.354
## 39   0.653
## 40   0.616
## 41   0.363
## 42   0.209
## 43   0.254
## 44   0.003
## 45   0.000
## 46   0.051
## 47   0.144
## 48   0.346
## 49   0.216
## 50   0.156
## 51   0.154
## 52   0.219
## 53   0.374
## 54   0.274
## 55   0.170
## 56   0.084
## 57   0.032
## 58   0.035
## 59   0.121
```

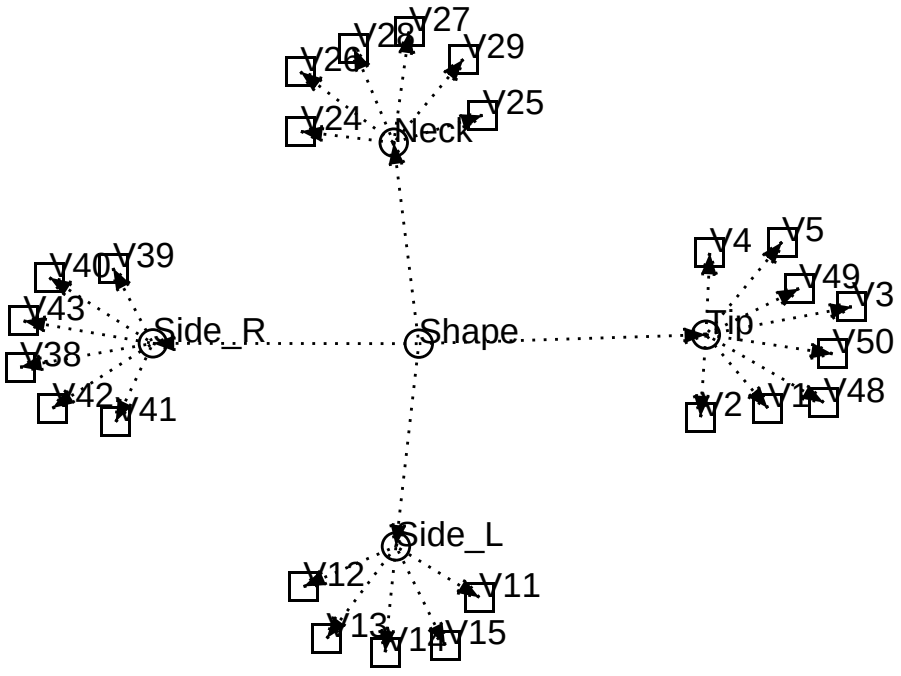
```
table = lavPredict(fit)

pairs(table)
```



Correlated features? I have a few.

```
ggsem(fit)
```

```
## Joining, by = "metric"
```



This is the structure that I determined shape has from this perspective and I feel that it seems reasonable, but is it? No idea. The function can be modified to print the covariances between each component but find that to be really really hard to read and ugly. This isn't *that* much better though. Moving on!!

4. Biomass Profile Analysis

"Morphological features of the [fruit] were quantified from the portion of the binarized image that lay above the horizontal line marking the root–shoot junction" from Turner et al (2018) An Automated Image Analysis Pipeline Enables Genetic Studies of Shoot and Root Morphology in Carrot (Daucus carota L.). I really like this paper and I really think Dr. Turner-Hissong is one of the best people you could know in quant/pop gen.

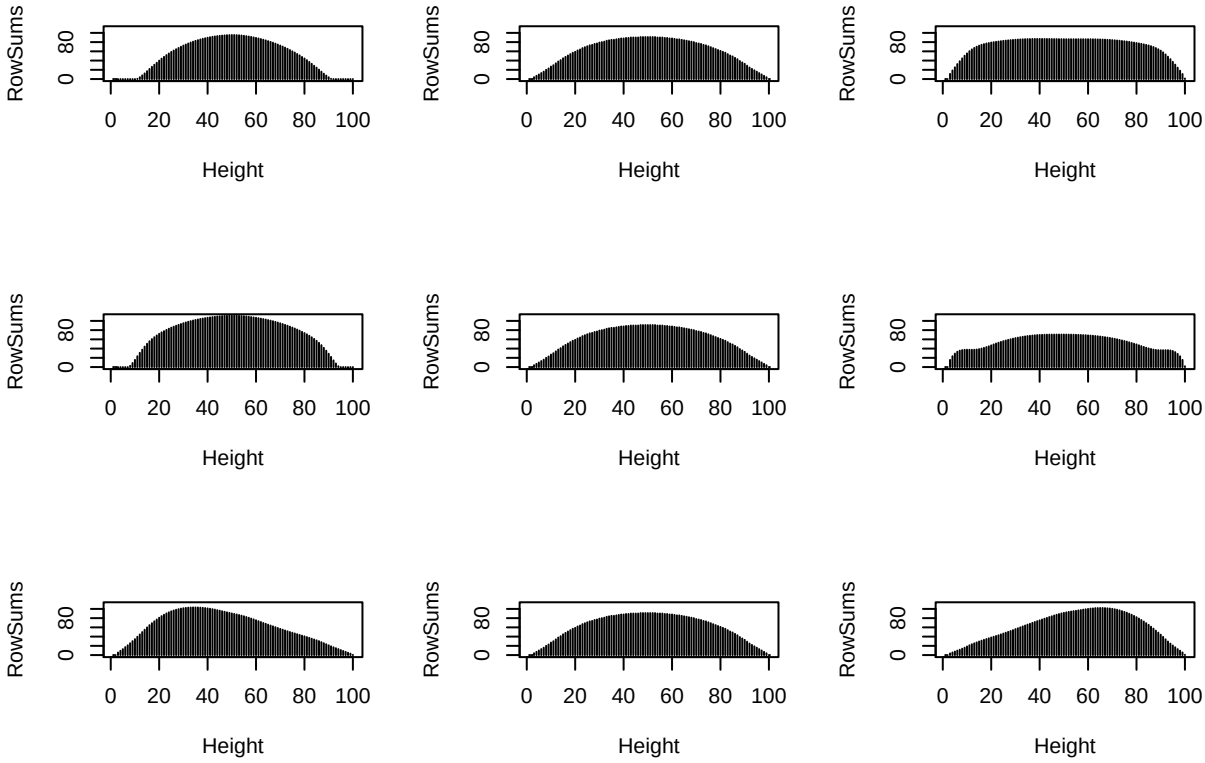
This chunk transform images from 1000x1000 to 100x100 and turns all of that into a matrix.

```
load_and_resize = function(path){image = load.image(path);
resize(im = image,size_x = 100, size_y = 100)}

list_of_images = lapply(lf, load_and_resize)

list_of_images.mat <- lapply(list_of_images, as.matrix)
```

We could the rows and columns are object so each iamge is summarized as a vector of either 100 row or column sums. We do PCA! This is getting kind of old... This analysis assumes in a way that objects are symmetric if you think of this as a shape descriptor... which, like, isn't a bad assumption to have.

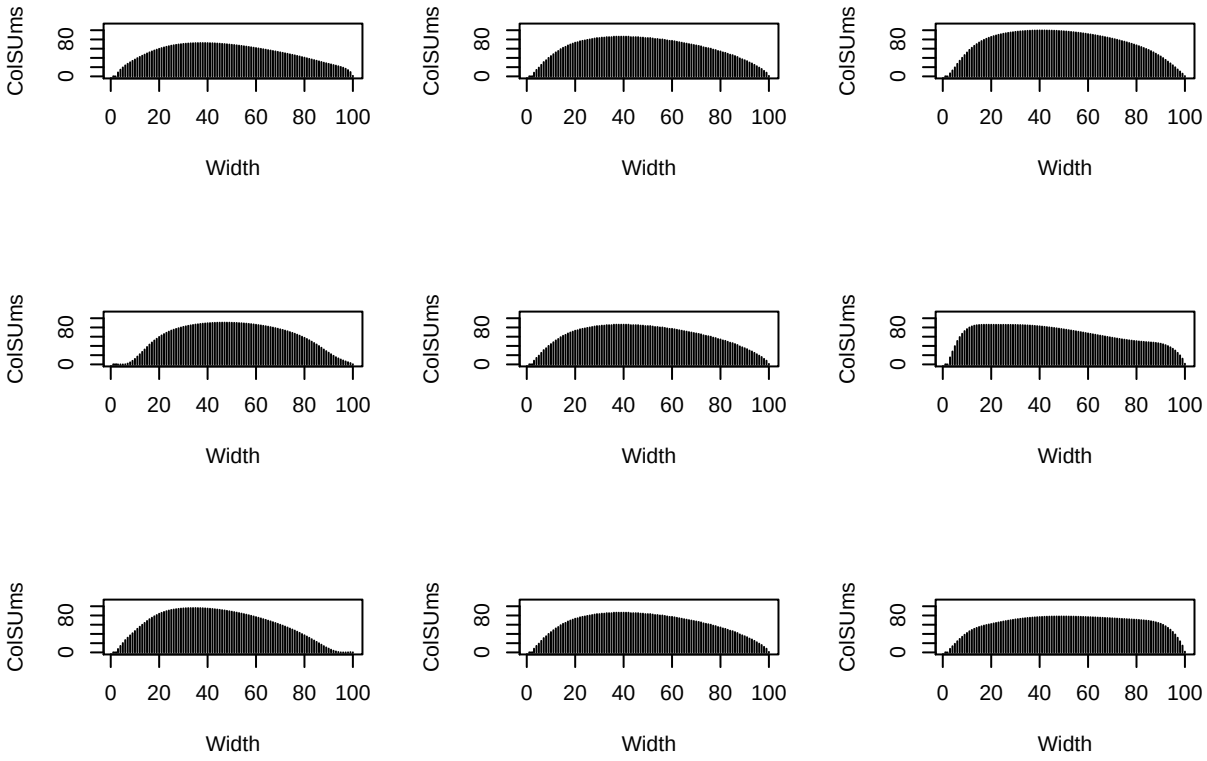
FIGURES! I should have snapped the outline to the edge of the image instead of leaving some wiggle room (10 px) on the outline as it causes some artifacts in the analysis here.

```
sweep <- c(seq(from = -1.5*v_pca_dat$sdev[1], to = 1.5*v_pca_dat$sdev[1], length.out = 3))
fruit1 <- t(sweep[1] * matrix(v_pca_dat$rotation[,1],
                              ncol=100) +
              matrix(colMeans(v.biomass),ncol=100))
fruit2 <- t(sweep[3] * matrix(v_pca_dat$rotation[,1],
                              ncol=100) +
              matrix(colMeans(v.biomass),ncol=100))
fruit3 <- t(sweep[1] * matrix(v_pca_dat$rotation[,2],
                              ncol=100) +
              matrix(colMeans(v.biomass),ncol=100))
fruit4 <- t(sweep[3] * matrix(v_pca_dat$rotation[,2],
                              ncol=100) +
              matrix(colMeans(v.biomass),ncol=100))
fruit5 <- t(sweep[1] * matrix(v_pca_dat$rotation[,3],
                              ncol=100) +
              matrix(colMeans(v.biomass),ncol=100))
fruit6 <- t(sweep[3] * matrix(v_pca_dat$rotation[,3],
                              ncol=100) +
              matrix(colMeans(v.biomass),ncol=100))

fruit.mean = t(matrix(round(colMeans(v.biomass)),ncol=100))

par(mfrow=c(3,3))
plot(fruit1,type="h",ylim=c(0,110),xlab="Height",ylab="RowSums")
plot(fruit.mean,type="h",ylim=c(0,110),xlab="Height",ylab="RowSums")
plot(fruit2,type="h",ylim=c(0,110),xlab="Height",ylab="RowSums")
plot(fruit3,type="h",ylim=c(0,110),xlab="Height",ylab="RowSums")
plot(fruit.mean,type="h",ylim=c(0,110),xlab="Height",ylab="RowSums")
plot(fruit4,type="h",ylim=c(0,110),xlab="Height",ylab="RowSums")
plot(fruit5,type="h",ylim=c(0,110),xlab="Height",ylab="RowSums")
plot(fruit.mean,type="h",ylim=c(0,110),xlab="Height",ylab="RowSums")
plot(fruit6,type="h",ylim=c(0,110),xlab="Height",ylab="RowSums")
```



MORE FIGURES!

```
sweep2 <- c(seq(from = -1.5*h_pca_dat$sdev[1], to = 1.5*h_pca_dat$sdev[1], length.out = 3))
fruit1 <- t(sweep2[1] * matrix(h_pca_dat$rotation[,1],ncol=100) + matrix(colMeans(h.biomass),ncol=100))
fruit2 <- t(sweep2[3] * matrix(h_pca_dat$rotation[,1],ncol=100) + matrix(colMeans(h.biomass),ncol=100))
fruit3 <- t(sweep2[1] * matrix(h_pca_dat$rotation[,2],ncol=100) + matrix(colMeans(h.biomass),ncol=100))
fruit4 <- t(sweep2[3] * matrix(h_pca_dat$rotation[,2],ncol=100) + matrix(colMeans(h.biomass),ncol=100))
fruit5 <- t(sweep2[1] * matrix(h_pca_dat$rotation[,3],ncol=100) + matrix(colMeans(h.biomass),ncol=100))
fruit6 <- t(sweep2[3] * matrix(h_pca_dat$rotation[,3],ncol=100) + matrix(colMeans(h.biomass),ncol=100))


fruit.mean = t(matrix(round(colMeans(h.biomass)),ncol=100))

par(mfrow=c(3,3))
plot(fruit1,type="h",ylim=c(0,110),xlab="Width",ylab="ColSUms")
plot(fruit.mean,type="h",ylim=c(0,110),xlab="Width",ylab="ColSUms")
plot(fruit2,type="h",ylim=c(0,110),xlab="Width",ylab="ColSUms")
plot(fruit3,type="h",ylim=c(0,110),xlab="Width",ylab="ColSUms")
plot(fruit.mean,type="h",ylim=c(0,110),xlab="Width",ylab="ColSUms")
plot(fruit4,type="h",ylim=c(0,110),xlab="Width",ylab="ColSUms")
plot(fruit5,type="h",ylim=c(0,110),xlab="Width",ylab="ColSUms")
plot(fruit.mean,type="h",ylim=c(0,110),xlab="Width",ylab="ColSUms")
plot(fruit6,type="h",ylim=c(0,110),xlab="Width",ylab="ColSUms")
```



5. Eigen Shape Analysis from Standardized Binary Maps

This analysis was inspired by facial recognition literature in the 80s and 90s. Basically the idea is that each pixel is a correlated variable and we can summarize all of the information in each image by performing PCA on a flattened binary image.

This flattenes and transposes our input matrix (n x pixel = 6874 x 10000)

```
image_matrix = do.call('cbind', lapply(list_of_images.mat, as.numeric))
rm(list_of_images)
dim(image_matrix)
```

```
## [1] 10000  6874
```

```
image_matrix = t(image_matrix)
dim(image_matrix)
```

```
## [1]  6874 10000
```

This takes for ever to run... like 15-20 minutes

To speed this up, filter out pixels that are fixed as either 1 or 0. These pixels have 0 variance, so they cannot be scaled... Looking back now, it would have made sense. 675 pixels have the same value across all images (either 0 or 1).

```
pca_dat = prcomp(image_matrix,scale=F)

cumsum(pca_dat$sdev[1:27]**2) / sum(pca_dat$sdev**2)
```

```
##  [1] 0.2680337 0.4044117 0.4567744 0.5021916 0.5400965 0.5660503 0.5875990
##  [8] 0.6065500 0.6225548 0.6380206 0.6503474 0.6605316 0.6698543 0.6776451
## [15] 0.6851858 0.6923747 0.6987430 0.7050767 0.7111398 0.7169249 0.7221627
## [22] 0.7268743 0.7315591 0.7359398 0.7402175 0.7442233 0.7479795
```

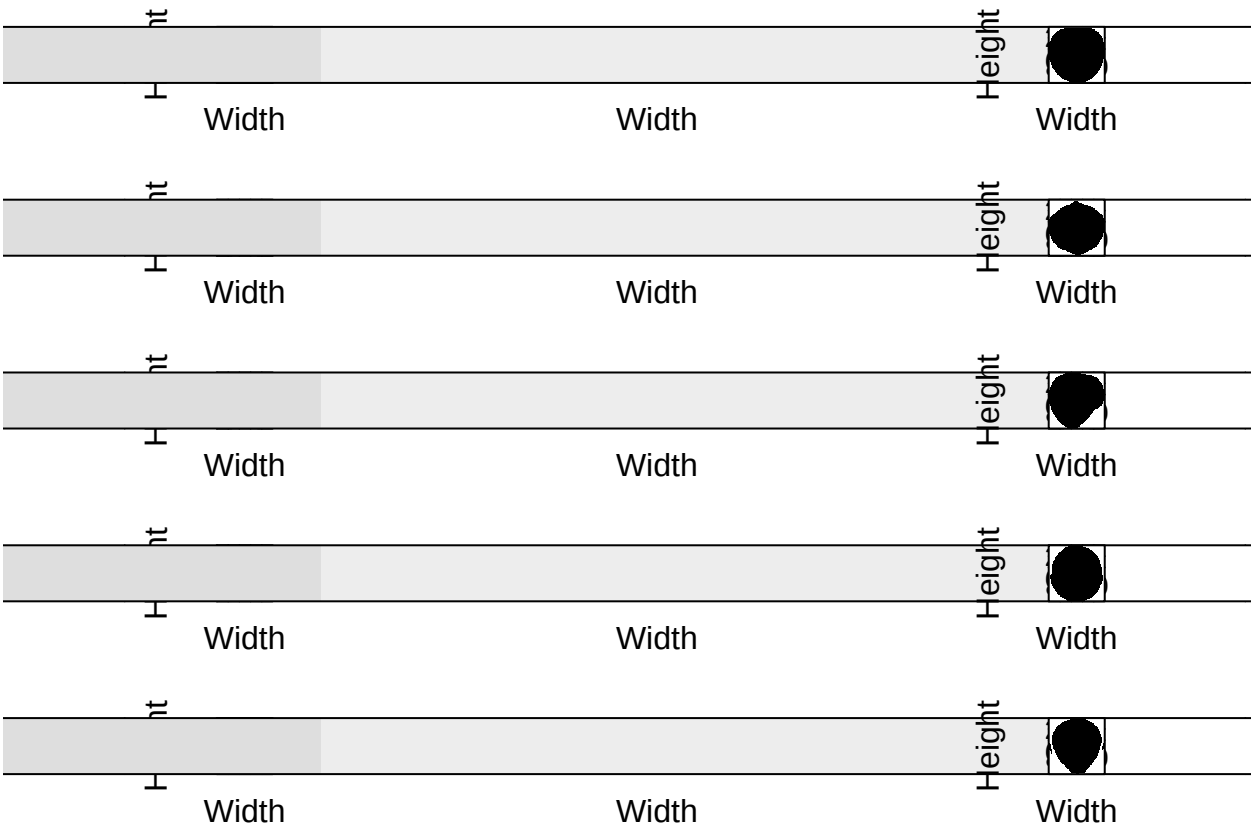
```
pca.x <- pca_dat$x[,c(1:20)]
pca.x <- as.data.frame(pca.x)
```

MORE MORE FIGURES! PCA can return some impossibilities, which is fun, but it means that these must be interpretted in a reasonable span.

```
sweep <- c(seq(from = -1.5*pca_dat$sdev[1], to = 1.5*pca_dat$sdev[1], length.out = 3))
fruit1 <- sweep[1] * matrix(pca_dat$rotation[,1],ncol=100,nrow = 100) +
  matrix(colMeans(image_matrix),ncol=100,nrow=100);fruit1[fruit1 >=0.9] = 1;
fruit1[fruit1 < 0.9] = 0
fruit2 <- sweep[3] * matrix(pca_dat$rotation[,1],ncol=100,nrow = 100) +
  matrix(colMeans(image_matrix),ncol=100,nrow=100);fruit2[fruit2 >=0.9] = 1;
fruit2[fruit2 < 0.9] = 0
fruit3 <- sweep[1] * matrix(pca_dat$rotation[,2],ncol=100,nrow = 100) +
  matrix(colMeans(image_matrix),ncol=100,nrow=100);fruit3[fruit3 >=0.9] = 1;
fruit3[fruit3 < 0.9] = 0
fruit4 <- sweep[3] * matrix(pca_dat$rotation[,2],ncol=100,nrow = 100) +
  matrix(colMeans(image_matrix),ncol=100,nrow=100);fruit4[fruit4 >=0.9] = 1;
fruit4[fruit4 < 0.9] = 0
fruit5 <- sweep[1] * matrix(pca_dat$rotation[,3],ncol=100,nrow = 100) +
  matrix(colMeans(image_matrix),ncol=100,nrow=100);fruit5[fruit5 >=0.9] = 1;
fruit5[fruit5 < 0.9] = 0
fruit6 <- sweep[3] * matrix(pca_dat$rotation[,3],ncol=100,nrow = 100) +
  matrix(colMeans(image_matrix),ncol=100,nrow=100);fruit6[fruit6 >=0.9] = 1;
fruit6[fruit6 < 0.9] = 0
fruit7 <- sweep[1] * matrix(pca_dat$rotation[,4],ncol=100,nrow = 100) +
  matrix(colMeans(image_matrix),ncol=100,nrow=100);fruit7[fruit7 >=0.9] = 1;
fruit7[fruit7 < 0.9] = 0
fruit8 <- sweep[3] * matrix(pca_dat$rotation[,4],ncol=100,nrow = 100) +
  matrix(colMeans(image_matrix),ncol=100,nrow=100);fruit8[fruit8 >=0.9] = 1;
fruit8[fruit8 < 0.9] = 0
fruit9 <- sweep[1] * matrix(pca_dat$rotation[,5],ncol=100,nrow = 100) +
  matrix(colMeans(image_matrix),ncol=100,nrow=100);fruit9[fruit9 >=0.9] = 1;
fruit9[fruit9 < 0.9] = 0
fruit10 <- sweep[3] * matrix(pca_dat$rotation[,5],ncol=100,nrow = 100) +
  matrix(colMeans(image_matrix),ncol=100,nrow=100);fruit10[fruit10 >=0.9] = 1;
fruit10[fruit10 < 0.9] = 0


plotm = levelplot((fruit.mean),ylab="Height",xlab="Width",
                  col.regions = gray(0:100/100),colorkey=FALSE )
plot1 <- levelplot((fruit1),ylab="Height",xlab="Width",
                   col.regions = gray(0:100/100),colorkey=FALSE )
plot2 <- levelplot((fruit2),ylab="Height",xlab="Width",
                   col.regions = gray(0:100/100),colorkey=FALSE )
plot3 <- levelplot((fruit3),ylab="Height",xlab="Width",
                   col.regions = gray(0:100/100),colorkey=FALSE )
plot4 <- levelplot((fruit4),ylab="Height",xlab="Width",
                   col.regions = gray(0:100/100),colorkey=FALSE )
plot5 <- levelplot((fruit5),ylab="Height",xlab="Width",
                   col.regions = gray(0:100/100),colorkey=FALSE )
plot6 <- levelplot((fruit6),ylab="Height",xlab="Width",
                   col.regions = gray(0:100/100),colorkey=FALSE )
plot7 <- levelplot((fruit7),ylab="Height",xlab="Width",
```

```
                    col.regions = gray(0:100/100),colorkey=FALSE )
plot8 <- levelplot((fruit8),ylab="Height",xlab="Width",
                    col.regions = gray(0:100/100),colorkey=FALSE )
plot9 <- levelplot((fruit9),ylab="Height",xlab="Width",
                    col.regions = gray(0:100/100),colorkey=FALSE )
plot10 <- levelplot((fruit10),ylab="Height",xlab="Width",
                     col.regions = gray(0:100/100),colorkey=FALSE )
```

```
grid.arrange(plot1,plotm,plot2,
             plot3,plotm,plot4,
             plot5,plotm,plot6,
             plot7,plotm,plot8,
             plot9,plotm,plot10,
             ncol=3,nrow=5)
```



PCA of fruit images and resulting eigen fruit provide a large number of quantitative variables for quant gen approaches and classification/prediction tasks directly from pixels.

It is important to remember that Eigenfruit, leaves, shapes are uncorrelated with one another and it takes many changes in multiple eigen dimensions to emulate a change that was observed in a true form.

6. K-means Clustering of Standardized Binary Maps

K-means clustering isn't new, but it might be new to you. Basically, it is an unsupervised approach to cluster data. The excellent thing is that you can define metrics of performance, overfit, etc. However, you NEED to decide how many clusters are in your data or test multiple to find what the best fit is.

K-means will find clusters for K = 2 . . . n-1, where n is the number of data points being clustered. So, it is important to find the right balance. To do that I did K=2 through 10

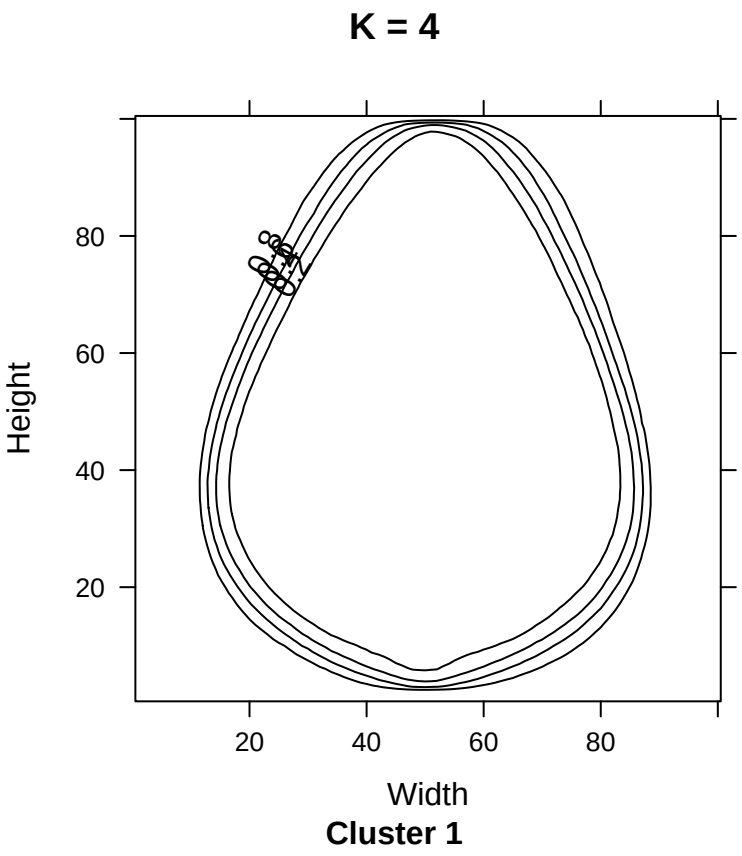
```
fit2 = kmeans(image_matrix, centers = 2)
fit3 = kmeans(image_matrix, centers = 3)
fit4 = kmeans(image_matrix, centers = 4)
fit5 = kmeans(image_matrix, centers = 5)
fit6 = kmeans(image_matrix, centers = 6)
fit7 = kmeans(image_matrix, centers = 7)
fit8 = kmeans(image_matrix, centers = 8)
fit9 = kmeans(image_matrix, centers = 9)
fit10 = kmeans(image_matrix, centers = 10)
```

This code extracts the centers of each cluster, or in this case the mean fruit shape.

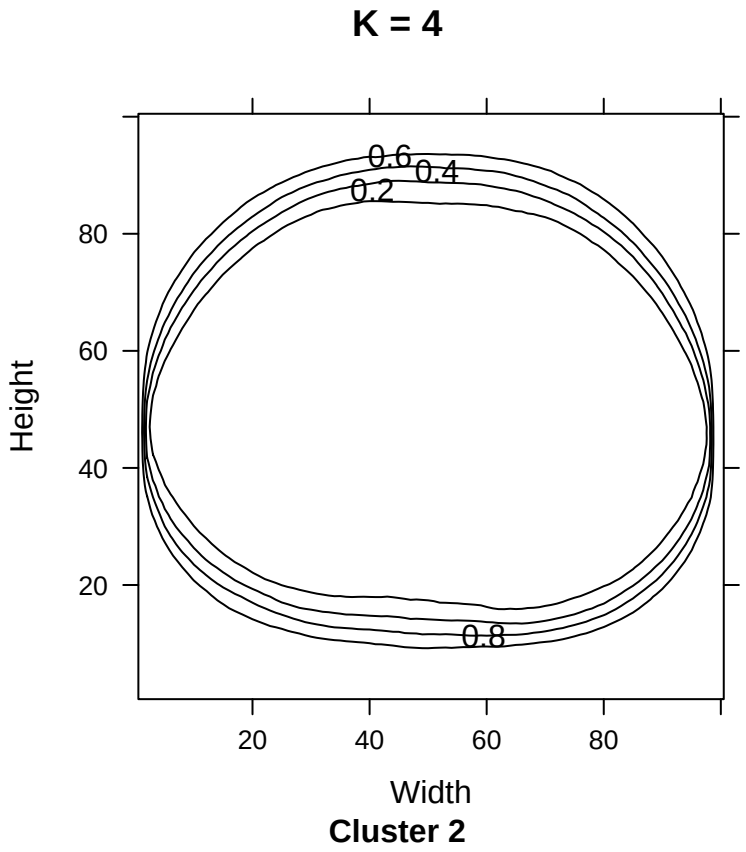
```
centroids2 = fit2$centers
centroids3 = fit3$centers
centroids4 = fit4$centers
centroids5 = fit5$centers
centroids6 = fit6$centers
centroids7 = fit7$centers
centroids8 = fit8$centers
centroids9 = fit9$centers
centroids10 = fit10$centers
```

This will make a figure of the centroid. Centroid is a word that means the mean, or the center of mass.

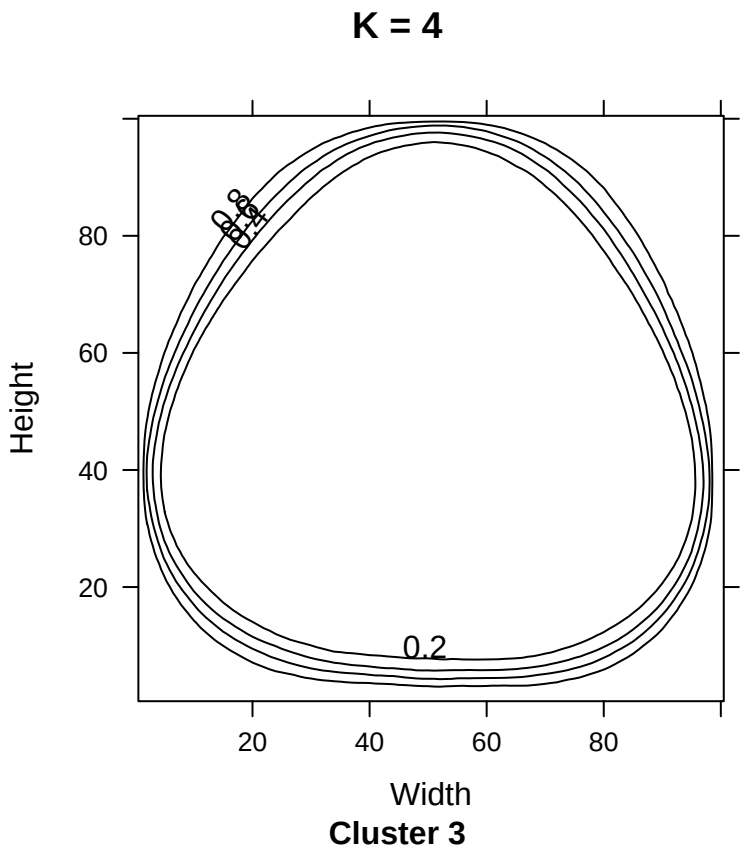
```
x4.1 <- matrix(centroids4[1,],nrow=100,ncol=100,byrow=TRUE);
contourplot(t(x4.1[,100:1]),aspect = "iso",
            ylab="Height",xlab="Width",
            main="K = 4",sub="Cluster 1")
```



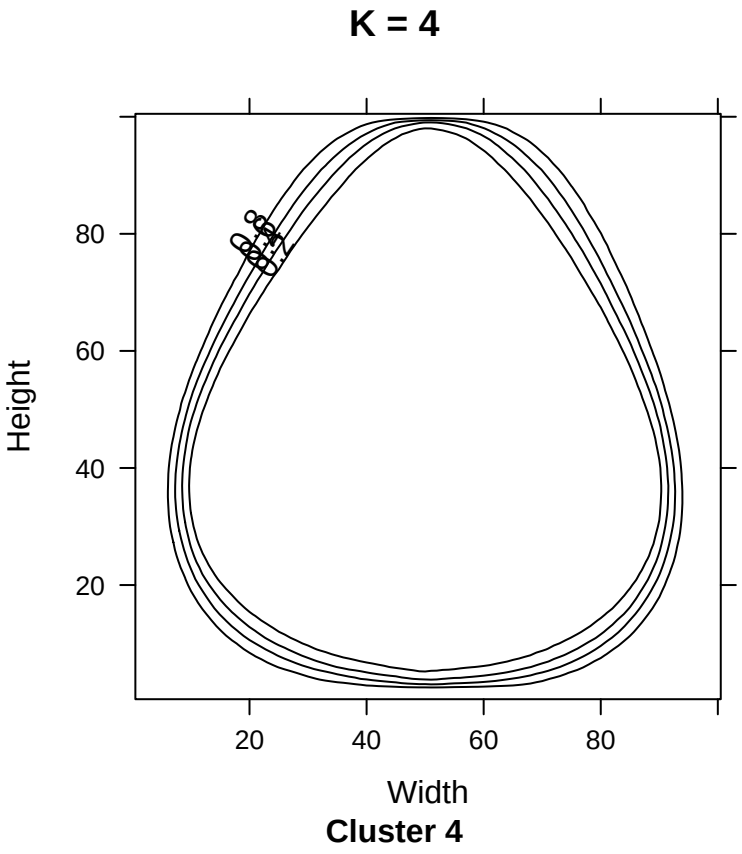
```
x4.2 <- matrix(centroids4[2,],nrow=100,ncol=100,byrow=TRUE);
contourplot(t(x4.2[,100:1]),aspect = "iso",
            ylab="Height",xlab="Width",
            main="K = 4",sub="Cluster 2")
```




```
x4.3 <- matrix(centroids4[3,],nrow=100,ncol=100,byrow=TRUE);
contourplot(t(x4.3[,100:1]),aspect = "iso",
            ylab="Height",xlab="Width",
            main="K = 4",sub="Cluster 3")
```




```
x4.4 <- matrix(centroids4[4,],nrow=100,ncol=100,byrow=TRUE);
contourplot(t(x4.4[,100:1]),aspect = "iso",
            ylab="Height",xlab="Width"
            ,main="K = 4",sub="Cluster 4")
```



We will determine the best number of clusters. We use several metrics, within group sum of squares, R2, AIC, and BIC.

For R2 and WSS, we are looking for an elbow in the plot as the R2 and WSS will always "improve" when more predictors are added.

AIC and BIC are different and can actually get worse as over fit becomes more prevalent as they are penalized.

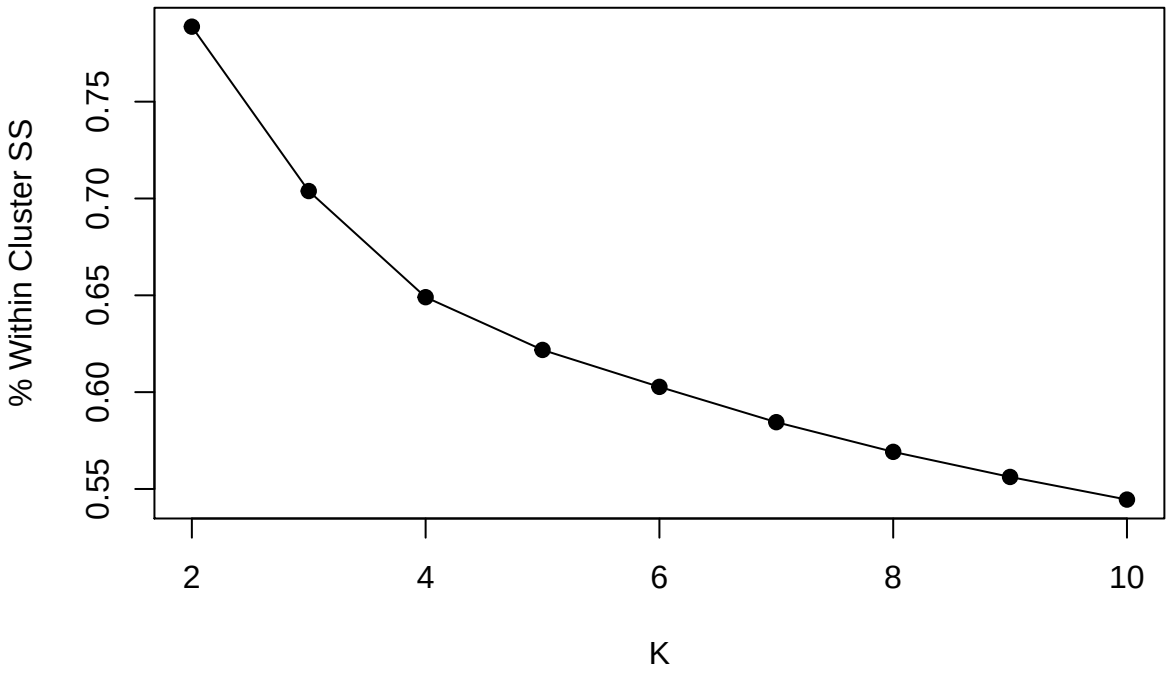
```
K = c(2,3,4,5,6,7,8,9,10)
WSS = c(fit2$tot.withinss / fit2$totss,
      fit3$tot.withinss / fit3$totss,
      fit4$tot.withinss / fit4$totss,
      fit5$tot.withinss / fit5$totss,
      fit6$tot.withinss / fit6$totss,
      fit7$tot.withinss / fit7$totss,
      fit8$tot.withinss / fit8$totss,
      fit9$tot.withinss / fit9$totss,
      fit10$tot.withinss / fit10$totss)
```

```
kmeansAIC = function(fit){

m = ncol(fit$centers)
n = length(fit$cluster)
k = nrow(fit$centers)
D = fit$tot.withinss
return(D + 2*m*k)
}
kmeansBIC = function(fit){
```

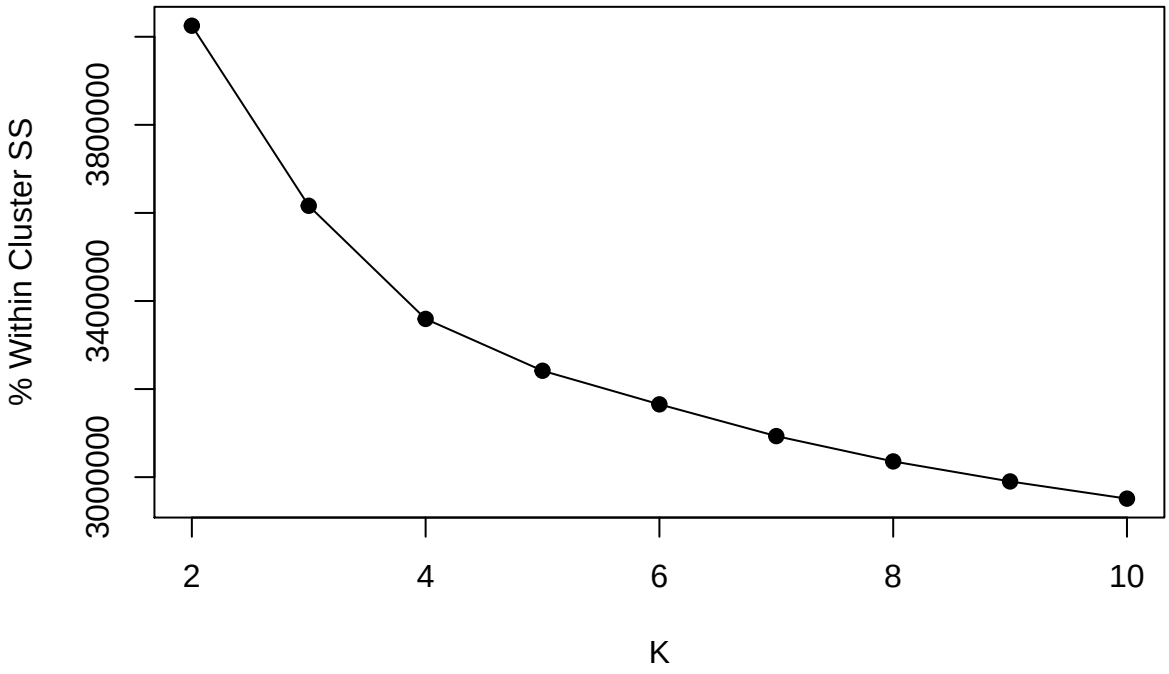
```
m = ncol(fit$centers)
n = length(fit$cluster)
k = nrow(fit$centers)
D = fit$tot.withinss
return(D + log(n)*m*k)
}

AIC = c(kmeansAIC(fit2),
        kmeansAIC(fit3),
        kmeansAIC(fit4),
        kmeansAIC(fit5),
        kmeansAIC(fit6),
        kmeansAIC(fit7),
        kmeansAIC(fit8),
        kmeansAIC(fit9),
        kmeansAIC(fit10))

BIC = c(kmeansBIC(fit2),
        kmeansBIC(fit3),
        kmeansBIC(fit4),
        kmeansBIC(fit5),
        kmeansBIC(fit6),
        kmeansBIC(fit7),
        kmeansBIC(fit8),
        kmeansBIC(fit9),
        kmeansBIC(fit10))
```

Ok, so here it looks like the best cluster is between 4-6. When I did this for the paper, I chose 4 after performing 80/20 cross validation. So, with less data, there are less clusters, but there are the same number of relevant clusters across all of the CV folds. So, 4 is good to me.

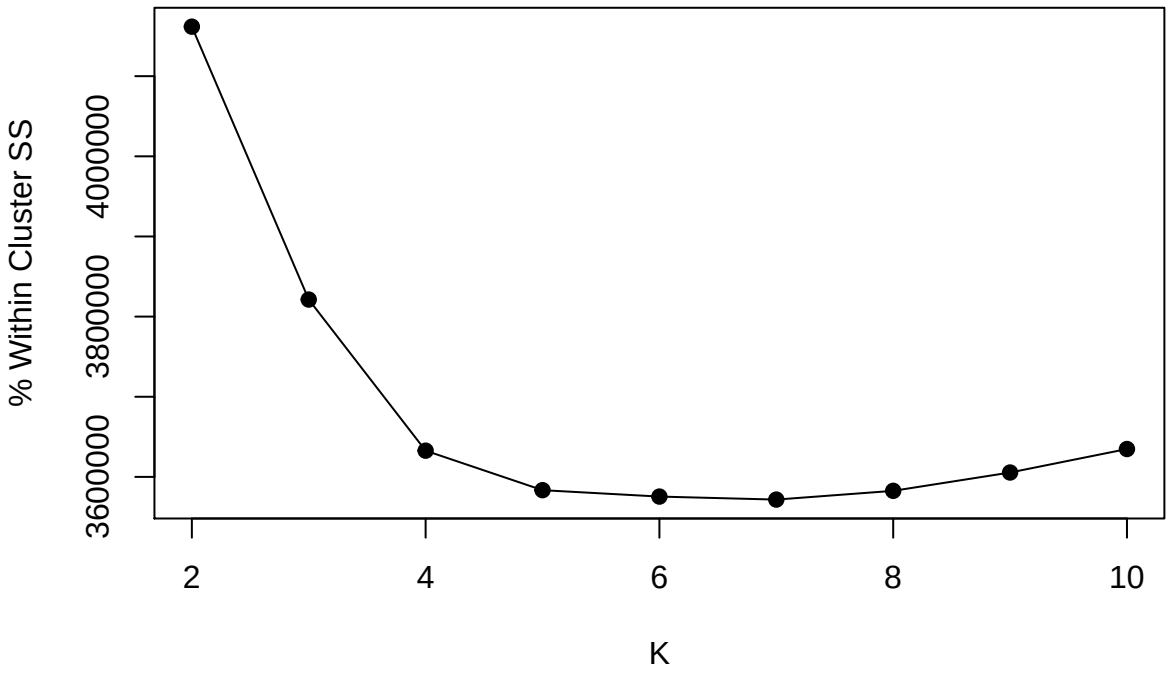
```
plot(K,WSS,type="o",pch=19,
     xlab="K",ylab="% Within Cluster SS")
```




```
plot(K,AIC,type="o",pch=19,
     xlab="K",ylab="% Within Cluster SS")
```




```
plot(K,BIC,type="o",pch=19,
     xlab="K",ylab="% Within Cluster SS")
```



We need this next data frame to perform the Principal Progression of K Clusters.

```
km <- data.frame(name = lf,
                 K2=fit2$cluster,
                 K3=fit3$cluster,
                 K4=fit4$cluster,
                 K5=fit5$cluster,
                 K6=fit6$cluster,
                 K7=fit7$cluster,
                 K8=fit8$cluster,
                 K9=fit9$cluster,
                 K10=fit10$cluster)
```

7. The Principal Progression of K Clusters (PPKC)

This was a silly idea that was bred by laziness and pseudo-neccesity. Basically, ordinal scales are the norm for fruit shape in breeding activities and in a lot of programs that study fruit shape. Ordinal scales are preferred by breeding groups, because they are quick to assess visually. But like, they kind of suck because you need specialized models to appropriately analyze the data.

Let me pose to your a thought question: Yes or No scale, what is in between?

Ordinal is an extension of binary and there isn't a value between the groups, because groups are categorical. Threshold models help account for this and rely on the group membership to infer a continuous liability function to the data.

```
#km <- read.csv("~/Box/Desktop/Reading/Strawberry_Research/1_UnsupervisedQuantization/Zenodo/Fruit_imag

km$K2 <- as.factor(km$K2) # 1 2
km$K3 <- as.factor(km$K3) # 1 3 2
km$K4 <- as.factor(km$K4) # 1 2 3 4
km$K5 <- as.factor(km$K5) # 4 5 1 3 2
km$K6 <- as.factor(km$K6) # 5 1 6 3 4 2
km$K7 <- as.factor(km$K7) # 7 3 6 1 2 4 5
```

```
km$K8 <- as.factor(km$K8) # 6 2 1 5 3 8 4 7
km$K9 <- as.factor(km$K9) # 5 3 2 1 8 7 9 4 6
km$K10 <- as.factor(km$K10) # 9 2 1 3 8 10 7 5 6 4

colnames(km)
```

```
##  [1] "name" "K2"   "K3"   "K4"   "K5"   "K6"   "K7"   "K8"   "K9"   "K10"
```

```
km2 = km[,-c(1)] #columns with clusters.
```

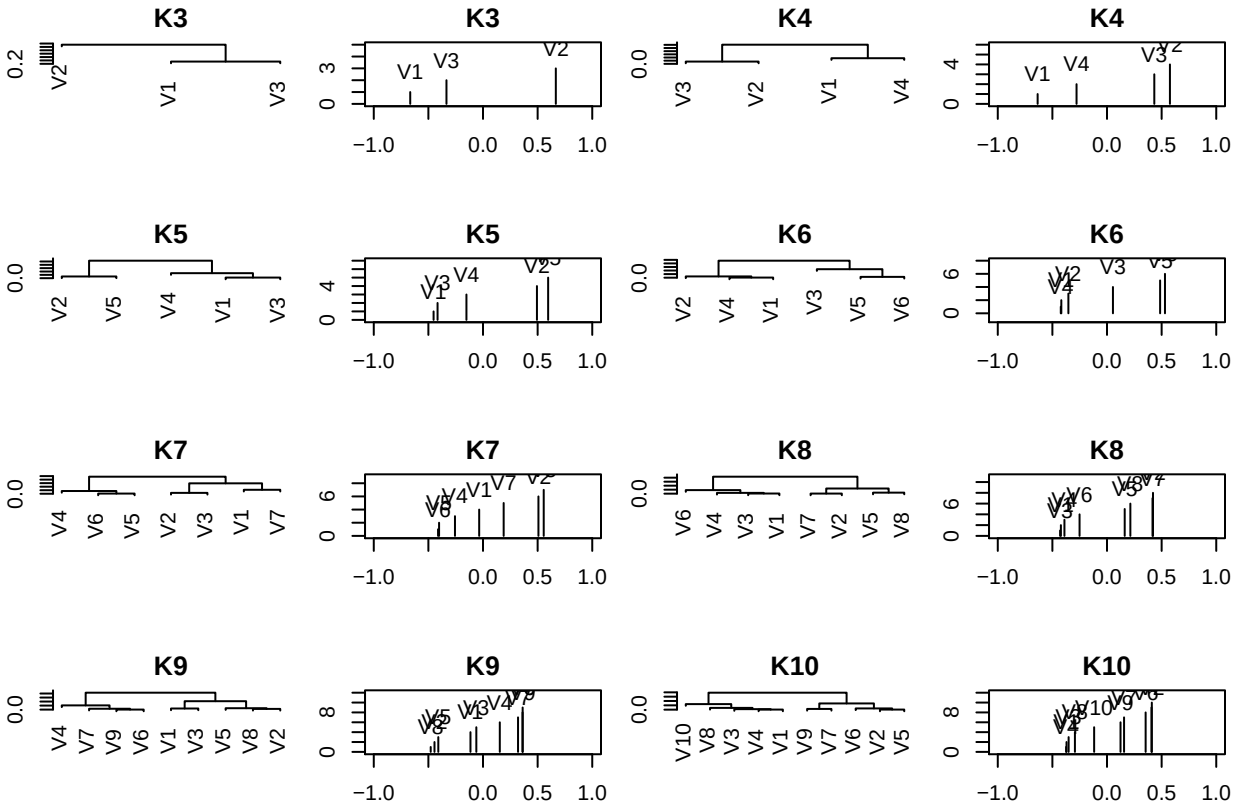
This function is the PPKC function!

```
PPKC=list()
for(i in 2:ncol(km2)){
  lev = levels(km2[,i])
  prev = c(1:(i-1))

  n = ((i)^2 + (i))/ 2 -1
  x = c()
  for(j in 1:length(lev)){
    for(k in 1:max(prev)){

      y = table(km2[which(km2[,i] == lev[j]),k]) / length(which(km2[,i] == lev[j]))
      x = c(x,y)
    }
  }
  M = matrix(x,ncol=i+1, nrow=n)
  CM = as.data.frame(cov(M))
  PPKC[[i-1]] = sort(prcomp(CM)$rotation[,1],decreasing=F)
}

save(list=ls(),file="Feldmann_PhenomeForce_RData.Rdata")
```

Some plots!

```
par(mfrow=c(4,4),mar=c(4,2,2,1))
plot(hclust(dist(PPKC[[1]])),xlab="",ylab="",main="K3")
plot(PPKC[[1]],1:3,pch=19,ylab="",xlab="",xlim=c(-1,1),ylim=c(0,5),type="h",main="K3")
text(PPKC[[1]],1:3, names(PPKC[[1]]),pos=3,cex=1)
#dev.off()

plot(hclust(dist(PPKC[[2]])),xlab="",ylab="",main="K4")
plot(PPKC[[2]],1:4,pch=19,ylab="",xlab="",main="K4",xlim=c(-1,1),type="h",ylim=c(0,6))
text(PPKC[[2]],1:4,  names(PPKC[[2]]),pos=3,cex=1)
#dev.off()
plot(hclust(dist(PPKC[[3]])),xlab="",ylab="",main="K5")
plot(PPKC[[3]],1:5,pch=19,ylab="",xlab="",main="K5",xlim=c(-1,1),type="h",ylim=c(0,7))
text(PPKC[[3]],1:5,  names(PPKC[[3]]),pos=3,cex=1)
#dev.off()

plot(hclust(dist(PPKC[[4]])),xlab="",ylab="",main="K6")
plot(PPKC[[4]],1:6,pch=19,ylab="",xlab="",main="K6",type="h",ylim=c(-1,8),xlim=c(-1,1))
text(PPKC[[4]],1:6,  names(PPKC[[4]]),pos=3,cex=1)
#dev.off()

plot(hclust(dist(PPKC[[5]])),xlab="",ylab="",main="K7")
```

```
plot(PPKC[[5]],1:7,pch=19,ylab="",xlab="",main="K7",xlim=c(-1,1),type="h",ylim=c(0,9))
text(PPKC[[5]],1:7,  names(PPKC[[5]]),pos=3,cex=1)
#dev.off()
plot(hclust(dist(PPKC[[6]])),xlab="",ylab="",main="K8")
plot(PPKC[[6]],1:8,pch=19,ylab="",xlab="",xlim=c(-1,1),type="h",ylim=c(0,11),main="K8")
text(PPKC[[6]],1:8,  names(PPKC[[6]]),pos=3,cex=1)

plot(hclust(dist(PPKC[[7]])),xlab="",ylab="",main="K9")
plot(PPKC[[7]],1:9,pch=19,ylab="",xlab="",xlim=c(-1,1),type="h",ylim=c(0,12),main="K9")
text(PPKC[[7]],1:9,  names(PPKC[[7]]),pos=3,cex=1)

plot(hclust(dist(PPKC[[8]])),xlab="",ylab="",main="K10")
plot(PPKC[[8]],1:10,pch=19,ylab="",xlab="",xlim=c(-1,1),type="h",ylim=c(0,12),main="K10")
text(PPKC[[8]],1:10,  names(PPKC[[8]]),pos=3,cex=1)
```



We're DONE!

Future Directions

Autoencoders seem pretty cool for this exact purpose and I am aching to put these to the test compared to the PCA approach. Gage et al 2019 does this, and it turns out (for his use case) that they are basically equivalent for predicting manually measured traits. Also, there are a couple of packages in R that seem of interest (e.g., ANN2 and Autoencoder), but also Keras/Tensorflow, Fastai, or whatever backbone you want to rely on is probably dope. MacOS kind of sucks for this stuff, because of the limitation of AMD Radeon graphics. If MacOS was still compatible with NVidia drivers, I would be a very happy man indeed.

Sparse Factor Models are also of incredible interest to me. Check out BGSF and MegaLMM by Dan Runcie and others for inspiration. I think that these factor approaches make a lot of sense for not only understanding

how different aspects of multidimensional measurements are related, but also in simplifying the response space of your analyses. 100 x 100 pixel images have 10,000 correlated values that are represented by MANY fewer continuously distributed characters that are much easier to make statistical sense of, but are more challenging to interpret.

Conclusions

The growth of the field in recent years, the availability of powerful open source software, and the competency to acquire complex datasets propounded the need for a thorough discussion of the state of the art. As with all new techniques and applications, careful consideration of the experimental procedures is required to deploy latent phenotyping approaches. All of the challenges of traditional methods still exist; the data must be collected systematically and in such a way to contain relevant information for the questions and hypotheses related to the specific study. Sensors are much less subject to explicit biases than are humans; however, that doesn't mean that the implicit and explicit biases won't be reflected in the quantification of those sensor data. For these reasons, analyses and protocol need to be reported as precisely and as accurately as possible in the literature so that those best practices can be duplicated in other labs, in other countries, and by less experienced hands lest results become non-reproducible (see the reproducibility crisis). Furthermore, we stress that not all research questions require the utility of a latent phenotyping approach and that the lazy application of these methods can lead to nonsensical results and interpretations. These methods rely on mathematical and statistical models that all have unique and diverse assumptions about the input which may render certain methods irrelevant or inappropriate for a given data set. As practitioners, we have an obligation to discuss our learnings and interpretations so that meaning and utility may be derived from our efforts. This requires us to communicate and converse with each other and with the perceived end users, those who may benefit, to make sure that we are attempting to solve problems that have the promise to effect change in how we generate biological knowledge and improve our crop species. The final frontier, as we see it, is in true validation of these approaches in independent populations managed by independent groups with similar objectives. To our readers and fellow practitioners, we strongly advocate quality over quantity, paying close attention to details and methodological assumptions, and focusing on repeatability and reproducibility.