Guerrero Dataset

Historically the Guerrero salt mats were thought to be very phylogenetically sparse - very high saline location which seemed to be extremely niche. Culture methods bore out the limited diversity in the location. However, after expanding the diversity measurements to include high-throughput metagenomic/microbiomic sequence data, a different picture was discovered. Turns out this is one of the *most* phylogenetically diverse communities on the planet, with over 40 different phylum present.

Beginning analysis on the guerrero dataset. There are 10 files associated with the Guerrero Mat data. Each file corresponds to a different level from a column taken from the mat. The files are 16s gene data which was sanger sequenced, and therefore the full gene sequence. Used an outside program to transpose the matrix from the uclust table output: transpose

mat_01 - first layer (top)

mat_02 - 2nd layer (top)

mat_03 - 3rd layer (top)

mat_04 - 4th layer (middle)

mat_05 - 5th layer (middle)

mat_06 - 6th layer (middle)

mat_07 - 7th layer (bottom)

mat_08 - 8th layer (bottom)

mat_09 - 9th layer (bottom)

mat_10 - 10th layer (bottom)

Data input is fasta format of sanger sequenced 16s genes (truncated view):

```
>gi|364589549|gb|JN437545.1|;barcodelabel=mat_01 Uncultured organism clone SBXZ_6525 16S ribosomal RNA gene, partial sequence
ACACATGCAAGTCGAACGAAGCCCTCGGGCGTAGTGGCGGACGGGTGAGTAACGCGTGAGAATCTACCTT
>gi|364589548|gb|JN437544.1|;barcodelabel=mat_01 Uncultured organism clone SBXZ_6524 16S ribosomal RNA gene, partial sequence
CTTACACATGCAAGTCGAACGCAACGTTCGGGTTGAGTGGCGGACGGGTGAGTAACGCGTGAGAATCTGC
>gi|364589547|gb|JN437543.1|;barcodelabel=mat_01 Uncultured organism clone SBXZ_6523 16S ribosomal RNA gene, partial sequence
TGGCCCTACGGGGGAAAGATTATCGCCATAGGATGGGCCCGCGTCAGATTAGCTAGTTGGTAGGGTAACG
```

I used the UCLUST algorithms, provided by Robert Edgar (which is part of the QIIME package, so you could do all the clustering through QIIME as well). You can get the program here The commands I used to cluster the OTUs are as follows:

```
$ usearch -derep_fulllength fasta_reads.fasta -output derep_fasta.fasta -sizeout

$ usearch -sortbysize derep_fasta.fasta -output sorted -minsize 2

$ usearch -cluster_otus sorted.fasta -otus old_otus.fasta

$ python ~/programs/usearch/fasta_number.py old_otus.fasta OTU_ > otus.fasta

$ usearch -usearch_global fasta_reads.fasta -db otus.fasta -strand plus -id 0.97 -uc readmap

$ python ~/programs/usearch/uc2otutab.py readmap > table

$ usearch8 -utax otus.fasta -db ~/Documents/rdp_16s.fa -strand plus -utax_rawscore -tt ~/Documents/rdp_16s.tt -utaxout utax

$ transpose -i 6000x20 -t table >transposed_table
```

```r
# install packages (Run this before trying to knit - ctrl+enter on the line commented out to prevent error during knitting)
#install.packages(c("vegan", "ecodist", "labdsv", "ape", "ade4", "smacof", "GUniFrac"))

# load packages
library(vegan)
```

```
## Loading required package: permute
## Loading required package: lattice
## This is vegan 2.2-1
```

```r
library(ecodist)
```

```
##
## Attaching package: 'ecodist'
```

```
##
## The following object is masked from 'package:vegan':
##
##     mantel
```

```
library(labdsv)
```

```
## Loading required package: mgcv
## Loading required package: nlme
## This is mgcv 1.8-3. For overview type 'help("mgcv-package")'.
## Loading required package: MASS
##
## Attaching package: 'labdsv'
##
## The following objects are masked from 'package:ecodist':
##
##     nmds, pco
##
## The following object is masked from 'package:stats':
##
##     density
```

```
library(ape)
library(ade4)
```

```
##
## Attaching package: 'ade4'
##
## The following object is masked from 'package:vegan':
##
##     cca
```

```
library(smacof)
```

```
## Loading required package: rgl
```

```
library(GUniFrac)
```

Great! Now we are ready to get our data in Rows=sample locations, Columns=OTUs

```
dat <- read.table("~/projects/guerrero/transposed_table", header=TRUE,row.names=1, quote="\"")
dat[1:5]
```

```
##        OTU_4973 OTU_41 OTU_43 OTU_206 OTU_14
## mat_01        1     14     26      57    163
## mat_02        0      1      4      28     65
## mat_03        0      1     18       8     12
## mat_04        0      0     22       2      5
## mat_05        0      0     17       5      6
## mat_06        0      0     25       5      8
## mat_07        0      0     43       1      1
## mat_08        0      0     55       2      2
## mat_09        0      0    378       3      8
## mat_10        0      0     68       2      9
```

If you've got a table in the form of the one above, congrats, you can pretty much start running these metrics immediately. The normal way of calculating a distance matrix in R is:

```
dat.dist <-dist(dat)
#Formatting looks bad, lets round this a bit
round(dat.dist,1)
```

```
##        mat_01 mat_02 mat_03 mat_04 mat_05 mat_06 mat_07 mat_08 mat_09
## mat_02  641.7
## mat_03 1213.1  861.7
## mat_04 1362.6 1091.4  684.1
## mat_05 1392.1 1184.7  974.3  775.5
## mat_06 1336.2 1161.7 1022.0  773.7  540.8
## mat_07 1460.6 1278.6 1055.7  911.0  603.3  692.4
## mat_08 1507.5 1378.6 1177.0 1187.9  914.7 1019.6  726.7
## mat_09 1656.0 1576.0 1418.9 1493.3 1311.3 1384.6 1215.3  764.8
## mat_10 1425.4 1317.0 1179.2 1272.3 1130.7 1173.0 1079.6  846.4  799.7
```

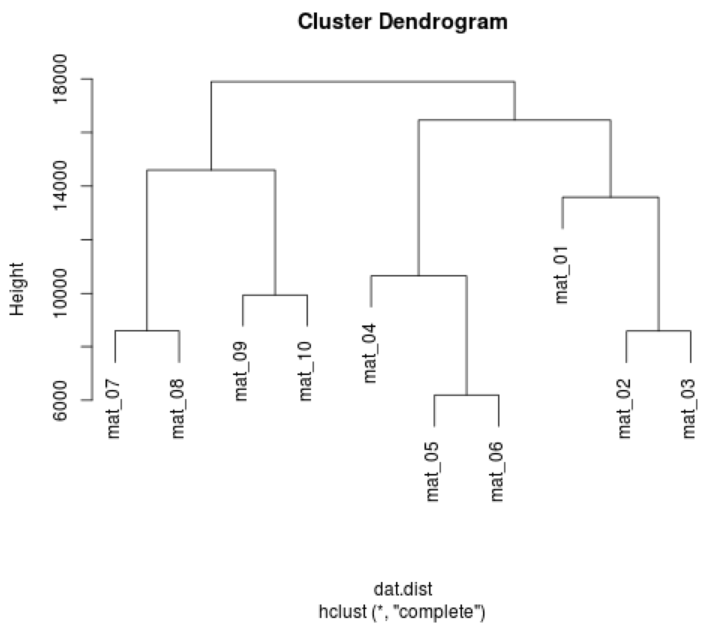
You can use several different ways of calculating a distance with 'dist', including: "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". For example:

```
dat.dist <-dist(dat, method = "manhattan")
dat.dist
```

```
##        mat_01 mat_02 mat_03 mat_04 mat_05 mat_06 mat_07 mat_08 mat_09
## mat_02   8804
## mat_03  13588   8580
## mat_04  14718  12380   9890
## mat_05  16460  14700  13182  10652
## mat_06  13892  12558  11592   9474   6198
## mat_07  17895  16331  14787  13175   8941   8953
## mat_08  17208  16490  15488  15050  11116  10678   8587
## mat_09  17601  17495  16853  17431  14521  13533  12784   8939
## mat_10  17230  17024  16768  17768  15542  14238  14599  11778   9943
```

Build a little heirarchical tree from this? No problem

```
hc <- hclust(dat.dist)
plot(hc)
```



In Vegan, you can pretty much do the same thing, except you want to use vegdist, which gives you access to some different distance metrics, the full set of which are: "manhattan", "euclidean", "canberra", "bray", "kulczynski", "jaccard", "gower", "altGower", "morisita", "horn", "mountford", "raup" , "binomial", "chao", "cao" or "mahalanobis"

```
dat.vdist <-vegdist(dat, method = "jaccard")
#Formatting looks bad, so lets round this up a bit
round(dat.vdist, 3)
```

```
##        mat_01 mat_02 mat_03 mat_04 mat_05 mat_06 mat_07 mat_08 mat_09
## mat_02  0.617
## mat_03  0.795  0.589
## mat_04  0.825  0.744  0.624
## mat_05  0.877  0.823  0.751  0.647
## mat_06  0.847  0.800  0.740  0.640  0.469
## mat_07  0.888  0.844  0.777  0.716  0.547  0.587
## mat_08  0.907  0.887  0.836  0.814  0.671  0.699  0.537
## mat_09  0.897  0.895  0.857  0.865  0.775  0.788  0.689  0.566
## mat_10  0.900  0.895  0.867  0.887  0.820  0.827  0.762  0.697  0.604
```

Alpha Diversity

There are multiple different ways of doing alpha diversity, see here for more information.

Also, for more discussion on the meaning of these different indicies, please see here

Shannon's Index (entropy):

Takes into account the number of individuals as well as number of taxa. Varies from 0 for communities with only a single taxon to high values for communities with many taxa, each with few individuals.

```
#Shannon's index
shannon_H <- diversity(dat, "shannon")
shannon_H
```

```
##   mat_01   mat_02   mat_03   mat_04   mat_05   mat_06   mat_07   mat_08
## 5.783008 5.887231 6.062203 5.837662 5.847422 5.839118 5.976024 5.758044
##   mat_09   mat_10
## 5.699611 5.953711
```

Simpson's Index:

Ranges from 0 (all taxa are equally present) to 1 (one taxon dominates the community completely)

```
#Simpson's index
simpson_H <- diversity(dat, "simpson")
simpson_H
```

```
##    mat_01    mat_02    mat_03    mat_04    mat_05    mat_06    mat_07
## 0.9842560 0.9876994 0.9927627 0.9916735 0.9929158 0.9899170 0.9936123
##    mat_08    mat_09    mat_10
## 0.9913589 0.9879163 0.9923796
```

Inverse Simpson's Index:

```
#Inverse Simpson's index
invsimpson_H <- diversity(dat, "invsimpson")
invsimpson_H
```

```
##    mat_01    mat_02    mat_03    mat_04    mat_05    mat_06    mat_07
##  63.51634  81.29678 138.17399 120.09826 141.15846  99.17659 156.55135
##    mat_08    mat_09    mat_10
## 115.72617  82.75621 131.22593
```

Chao1

Way of estimating the number of unseen species and adding them to the observed species richness

```
groups = c("top","top","top","middle","middle","middle","bottom","bottom","bottom","bottom")
spec_richness <- specpool(dat)
spec_richness$chao
```

```
## [1] 11701.51
```

Vegan doesn't implement evenness, but we can calculate that (Pielous's eveness) ourselves with this code:

(specnumber is a vegan function that gives the # of species in each sample)

This measures the evenness with which individuals are divided among the taxa present.

```
#Pielou's Evenness: J=H/log(S)
J <- shannon_H/log(specnumber(dat))
J
```

```
##    mat_01    mat_02    mat_03    mat_04    mat_05    mat_06    mat_07
## 0.7829847 0.7972967 0.8188779 0.8049667 0.8067075 0.8110248 0.8131496
##    mat_08    mat_09    mat_10
## 0.8031473 0.7749299 0.8104686
```

Fisher's Alpha

defined implicitly by the formula S=a*ln(1+n/a) where S is number of taxa, n is number of individuals and a is the Fisher's alpha.

```
alpha <- fisher.alpha(dat)
```

```
alpha
```

```
##   mat_01   mat_02   mat_03   mat_04   mat_05   mat_06   mat_07   mat_08
## 547.1715 547.1969 540.3055 428.7533 425.0961 434.8595 467.6549 385.0910
##   mat_09   mat_10
## 483.9053 488.3705
```

Fisher's Log Series

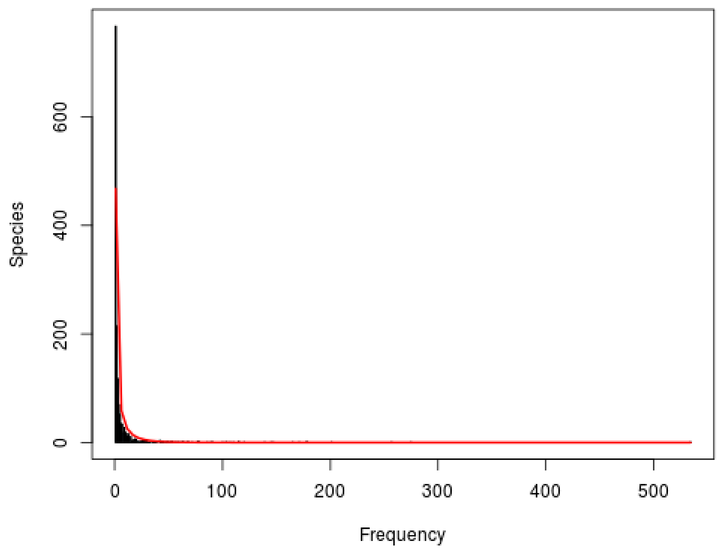
This is the expected number of species f_hat with n individuals - where Fisher's Alpha comes from

```
k <- sample(nrow(dat), 1)
fish <- fisherfit(dat[k,])
fish
```

```
##
## Fisher log series model
## No. of species: 1550
## Fisher alpha:   488.3705
```

Randomly selected location Fisher's log series

```
plot(fish)
```



Beta Diversity

Beta diversity seems like an especially tricky subject, please see the paper in this github repo by Allen et. al for addition information

Vegan implements many different beta diversity metrics, if you would like to see which ones are available, check out 'betadiver', which has many, many implementations:

```
betadiver(help=TRUE)
```

```
## 1 "w" = (b+c)/(2*a+b+c)
## 2 "-1" = (b+c)/(2*a+b+c)
## 3 "c" = (b+c)/2
## 4 "wb" = b+c
## 5 "r" = 2*b*c/((a+b+c)^2-2*b*c)
## 6 "I" = log(2*a+b+c) - 2*a*log(2)/(2*a+b+c) - ((a+b)*log(a+b) +
## (a+c)*log(a+c)) / (2*a+b+c)
## 7 "e" = exp(log(2*a+b+c) - 2*a*log(2)/(2*a+b+c) - ((a+b)*log(a+b)
## + (a+c)*log(a+c)) / (2*a+b+c))-1
## 8 "t" = (b+c)/(2*a+b+c)
## 9 "me" = (b+c)/(2*a+b+c)
## 10 "j" = a/(a+b+c)
## 11 "sor" = 2*a/(2*a+b+c)
## 12 "m" = (2*a+b+c)*(b+c)/(a+b+c)
## 13 "-2" = pmin(b,c)/(pmax(b,c)+a)
## 14 "co" = (a*c+a*b+2*b*c)/(2*(a+b)*(a+c))
## 15 "cc" = (b+c)/(a+b+c)
## 16 "g" = (b+c)/(a+b+c)
## 17 "-3" = pmin(b,c)/(a+b+c)
## 18 "l" = (b+c)/2
## 19 "19" = 2*(b*c+1)/((a+b+c)^2+(a+b+c))
## 20 "hk" = (b+c)/(2*a+b+c)
## 21 "rlb" = a/(a+c)
## 22 "sim" = pmin(b,c)/(pmin(b,c)+a)
## 23 "gl" = 2*abs(b-c)/(2*a+b+c)
## 24 "z" = (log(2)-log(2*a+b+c)+log(a+b+c))/log(2)
```

The Sorensen index of dissimilarity (aka, binary Bray-Curtis)

```
beta <- vegdist(dat, binary=TRUE)
mean(beta)
```

```
## [1] 0.5590099
```

Unifrac Distance

This is deceptively more complex. It is important to note a couple of things, first there is no unifrac distance calculator in vegan (that I could find), so we need a new package (which I called earlier in the script, aka GUniFrac). Second unifrac distances is actually based on a tree, and how the sequences from each sample fall on that tree. Therefore, you need to build a tree! And perhaps more importantly, you need to build an alignment first, in order to build the tree. I did so with mafft, and then used FastTree to compute the tree. I would highly recommend you NOT do it this way, a better way would be to align sequences with Pynast (find in the QIIME package), which is slower but will help avoid issues like expoentially increasing gaps in your alignment. FastTree or Raxml are your best options probably for building a tree from this alignment (my preference would be Raxml, but only because I've had good experiences with it in the past)

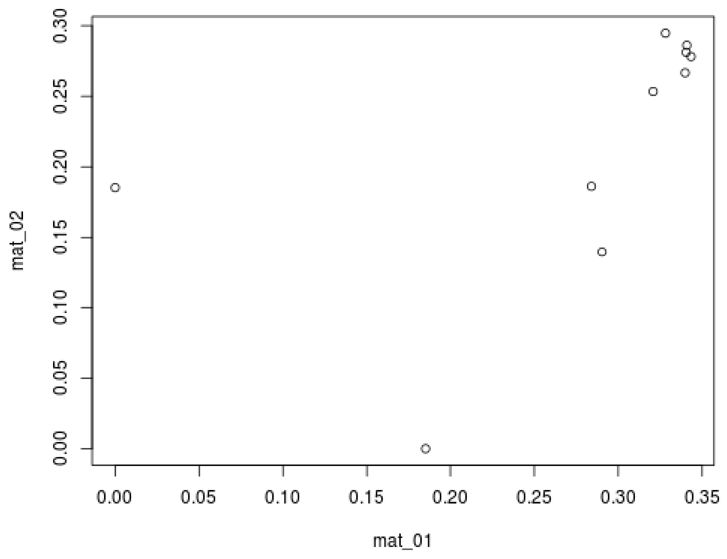
First load in your tree from the file (here it is in newick format)

Then you need to root it, since this is exploratory, choosing one of the longest branch lengths is about as good as you can hope for -- choosing a root is complicated and deserves it's own research. Additionally you have to kill any polytomies as ape does not allow you to have a rooted tree with a polytomy (no zero length branches). the function multi2di will randomly 'fix' this - questionable to downright wrong for 'real' data, so be aware

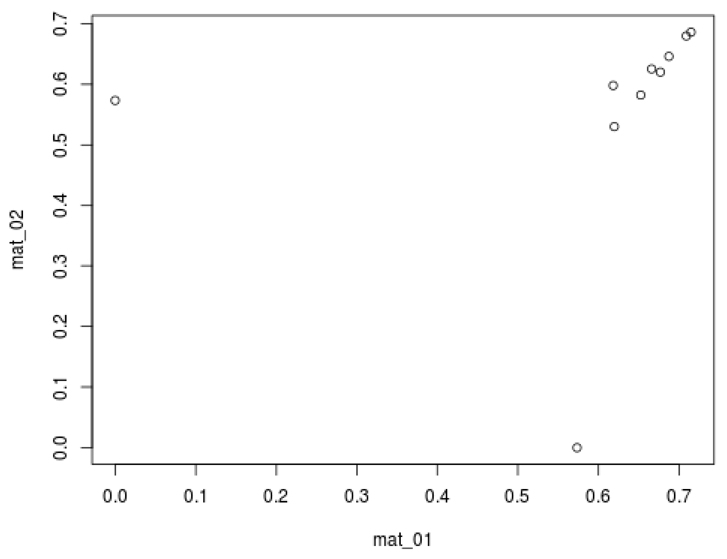
```
tree <- read.tree("~/projects/guerrero/otus.tree")
rooted_tree = multi2di(root(tree, "OTU_5140"))
```

Then use the GUniFrac function to calculate your distances matrices, hopefully easy to see which is which from the code below:

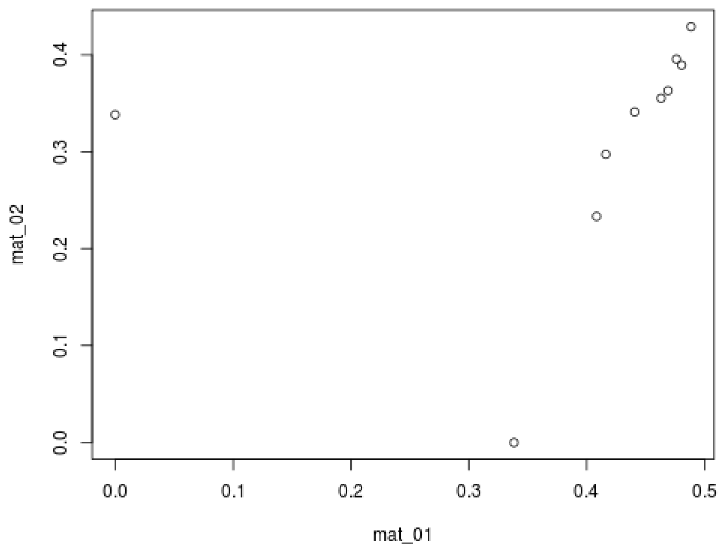
```
dat.mds.unifrac <- GUniFrac(dat, rooted_tree, alpha=c(0,0.5,1))$unifracs
dw <- dat.mds.unifrac[,,"d_1"] # Weighted UniFrac
du <- dat.mds.unifrac[, , "d_UW"] # Unweighted UniFrac
dv <- dat.mds.unifrac[, , "d_VAW"] # Variance adjusted weighted UniFrac
d0 <- dat.mds.unifrac[, , "d_0"] # GUniFrac with alpha 0
d5 <- dat.mds.unifrac[, , "d_0.5"] # GUniFrac with alpha 0.5
plot(dw)
```



```r
plot(du)
```



```r
plot(dv)
```



Gamma Diversity

This is just the total number of unique species seen in all sites (not their abundance, just binary, i.e. "there was an elephant" not "there were 20 elephants"). This is very easy to calculate, you just need the number of columns:

```
d <- dim(dat)
d[2]
```

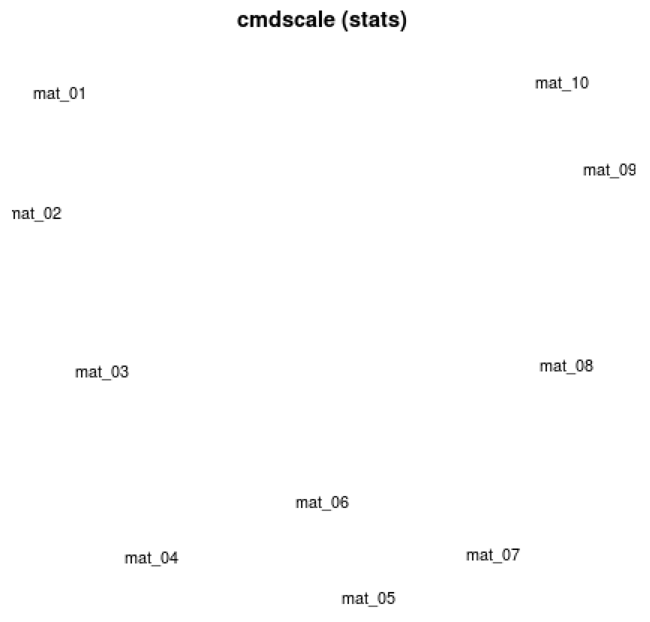
```
## [1] 5794
```

Metric Multidimensional scaling:

There are multiple different ways of doing MDS, these are metric methods, which you can see examples of here

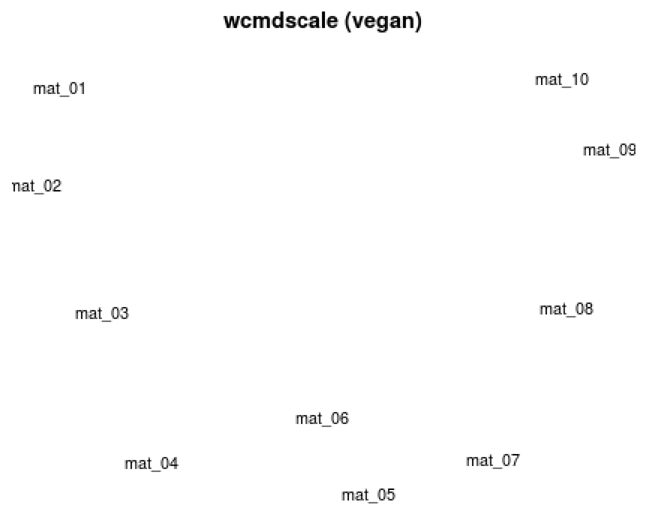
MDS with cmdscale

```
# 1) MDS 'cmdscale'
mds1 = cmdscale(dat.dist, k=2)
# plot
plot(mds1[,1], mds1[,2], type = "n", xlab = "", ylab = "", axes = FALSE,
     main = "cmdscale (stats)")
text(mds1[,1], mds1[,2], labels(dat.dist), cex=0.9)
```



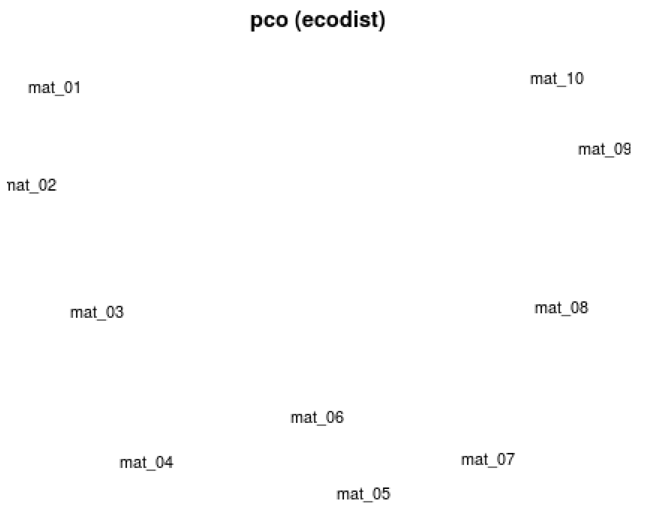
MDS with wcmdscale - Weighted Classical Multidimensional Scaling

```
# 1) MDS 'wcmdscale'
mds1 = wcmdscale(dat.dist, k=2)
# plot
plot(mds1[,1], mds1[,2], type = "n", xlab = "", ylab = "", axes = FALSE,
     main = "wcmdscale (vegan)")
text(mds1[,1], mds1[,2], labels(dat.dist), cex=0.9)
```



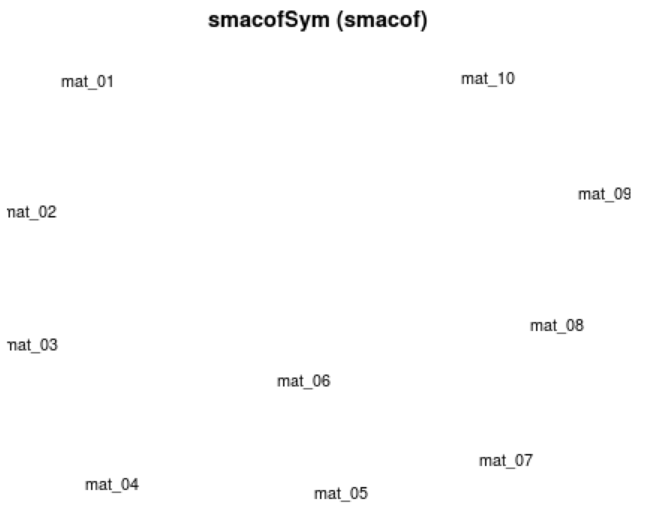
MDS with pco - Principal Coordinates Analysis

```
# 1) MDS
mds3 = pco(dat.dist, k=2)
# plot
plot(mds3$points[,1], mds3$points[,2], type = "n", xlab = "", ylab = "",
     axes = FALSE, main = "pco (ecodist)")
text(mds3$points[,1], mds3$points[,2], labels(dat.dist), cex = 0.9)
```



MDS with smocofSym - multidimensional scalling with stress minimization

```
# 7) MDS 'smacofSym'
mds7 = smacofSym(dat.dist, ndim=2)
# plot
plot(mds7$conf[,1], mds7$conf[,2], type = "n", xlab = "", ylab = "",
     axes = FALSE, main = "smacofSym (smacof)")
text(mds7$conf[,1], mds7$conf[,2], labels(dat.dist), cex = 0.9)
```

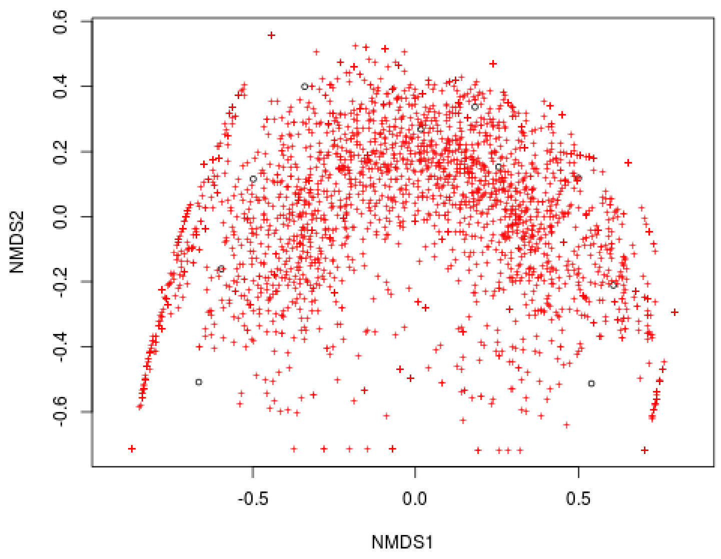


Non-metric Multidimensional scaling:- see here for more info

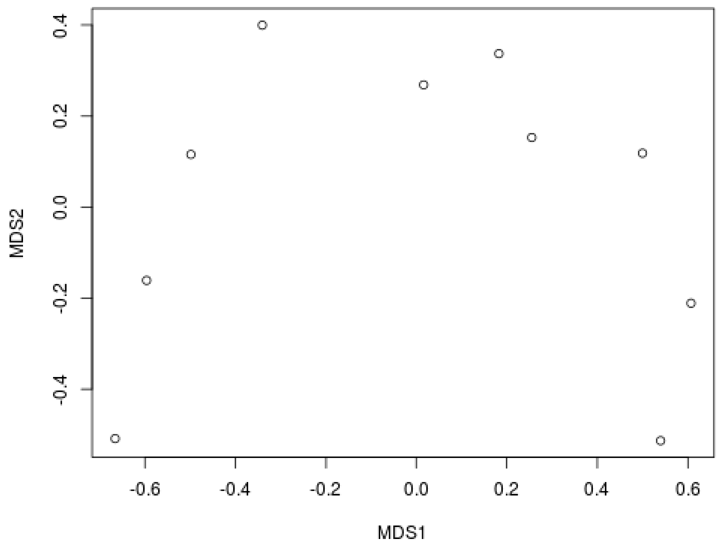
```
dat.mds <- metaMDS(dat,trymax=50)
```

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.03603821
## Run 1 stress 0.2225634
## Run 2 stress 0.03603834
## ... procrustes: rmse 0.0001243186  max resid 0.0002321515
## *** Solution reached
```

```
plot(dat.mds)
```



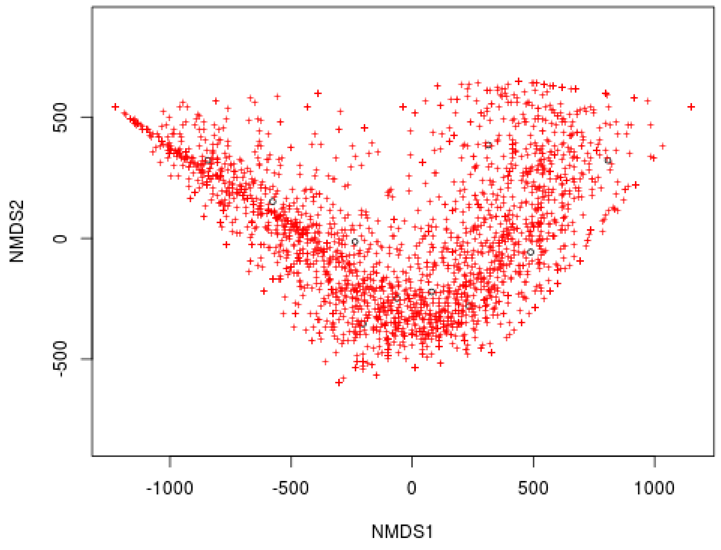
```r
plot(dat.mds$points)
```



```r
dat.mds.eucl <- metaMDS(dat, distance="euclidean", k=3, trymax=50, autotransform=FALSE)
```

```
## Run 0 stress 0.004202912
## Run 1 stress 0.004240669
## ... procrustes: rmse 0.08639502  max resid 0.1426599
## Run 2 stress 0.004451662
## ... procrustes: rmse 0.08789674  max resid 0.1926391
## Run 3 stress 0.004920031
## Run 4 stress 0.003928516
## ... New best solution
## ... procrustes: rmse 0.08322731  max resid 0.1589895
## Run 5 stress 0.005080326
## Run 6 stress 0.004545118
## Run 7 stress 0.003922729
## ... New best solution
## ... procrustes: rmse 0.01082996  max resid 0.01731728
## Run 8 stress 0.005215579
## Run 9 stress 0.004374366
## ... procrustes: rmse 0.03410134  max resid 0.06043715
## Run 10 stress 0.004537617
## Run 11 stress 0.004738947
## Run 12 stress 0.003954761
## ... procrustes: rmse 0.01548277  max resid 0.02898129
## Run 13 stress 0.004281013
## ... procrustes: rmse 0.0323294  max resid 0.06994574
## Run 14 stress 0.004245854
## ... procrustes: rmse 0.08304218  max resid 0.142809
## Run 15 stress 0.005205154
## Run 16 stress 0.005434852
## Run 17 stress 0.005410778
## Run 18 stress 0.005069492
## Run 19 stress 0.005362216
## Run 20 stress 0.00391332
## ... New best solution
## ... procrustes: rmse 0.0004493563  max resid 0.0009641842
## *** Solution reached
```

```
plot(dat.mds.eucl)
```

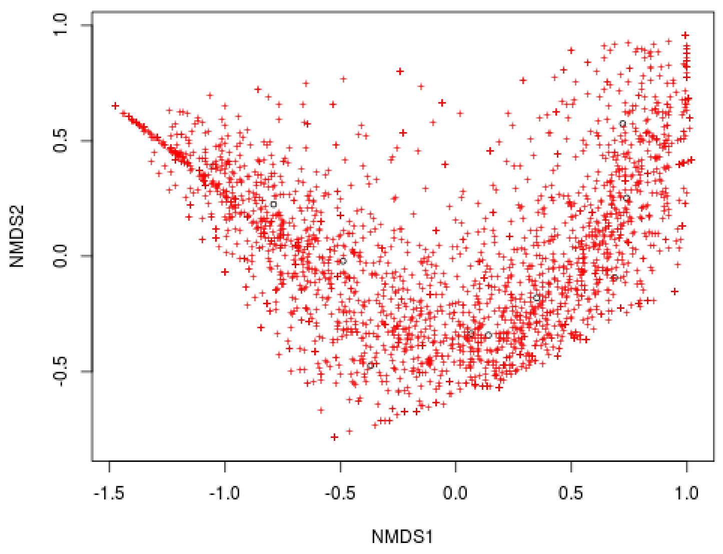


```
dat.mds.bray <- metaMDS(dat, distance="bray", k=3, trymax=50, autotransform=FALSE)
```

```
## Run 0 stress 0.0001187749
## Run 1 stress 9.590614e-05
## ... New best solution
## ... procrustes: rmse 0.05590461  max resid 0.07305668
## Run 2 stress 0.0001917627
## ... procrustes: rmse 0.05578392  max resid 0.1011119
## Run 3 stress 0.001979325
## Run 4 stress 0.007766502
## Run 5 stress 0.0001576429
## ... procrustes: rmse 0.04637877  max resid 0.1128965
## Run 6 stress 0.002703025
## Run 7 stress 0.007752519
## Run 8 stress 0.0006486684
## Run 9 stress 0.003378726
## Run 10 stress 0.003745529
## Run 11 stress 0.0006435522
## Run 12 stress 9.362878e-05
## ... New best solution
## ... procrustes: rmse 0.04015184  max resid 0.05539912
## Run 13 stress 0.002160468
## Run 14 stress 0.001192489
## Run 15 stress 0.007778327
## Run 16 stress 0.0006907404
## Run 17 stress 0.003267057
## Run 18 stress 0.007776055
## Run 19 stress 0.004328434
## Run 20 stress 0.008876923
## Run 21 stress 0.001360378
## Run 22 stress 9.62734e-05
## ... procrustes: rmse 0.0340675  max resid 0.05324446
## Run 23 stress 6.781335e-05
## ... New best solution
## ... procrustes: rmse 0.04524802  max resid 0.0801243
## Run 24 stress 0.000231864
## ... procrustes: rmse 0.05589756  max resid 0.1007269
## Run 25 stress 0.0002920302
## ... procrustes: rmse 0.05737813  max resid 0.10735
## Run 26 stress 0.00320794
## Run 27 stress 0.0004537213
## ... procrustes: rmse 0.05403301  max resid 0.1187507
## Run 28 stress 0.01075658
## Run 29 stress 0.002060967
## Run 30 stress 0.04030979
## Run 31 stress 0.0006008584
## Run 32 stress 0.000427778
## ... procrustes: rmse 0.0293691  max resid 0.0420305
## Run 33 stress 0.007787734
## Run 34 stress 0.00881426
## Run 35 stress 9.867655e-05
## ... procrustes: rmse 0.04879694  max resid 0.07125715
## Run 36 stress 0.002516518
## Run 37 stress 0.0001157417
## ... procrustes: rmse 0.02529695  max resid 0.04481577
## Run 38 stress 0.0003960894
## ... procrustes: rmse 0.02820087  max resid 0.04072111
## Run 39 stress 0.1969263
## Run 40 stress 0.001179261
## Run 41 stress 0.002504151
## Run 42 stress 0.00274767
## Run 43 stress 0.0003151533
## ... procrustes: rmse 0.05756593  max resid 0.1091009
## Run 44 stress 9.951257e-05
## ... procrustes: rmse 0.04432237  max resid 0.08359037
## Run 45 stress 0.001250756
## Run 46 stress 0.04032642
## Run 47 stress 0.008486592
## Run 48 stress 9.56097e-05
## ... procrustes: rmse 0.03210604  max resid 0.06594101
## Run 49 stress 0.004312166
## Run 50 stress 0.002389316
```

```
## Warning in metaMDS(dat, distance = "bray", k = 3, trymax = 50,
## autotransform = FALSE): Stress is (nearly) zero - you may have
## insufficient data
```

```
plot(dat.mds.bray)
```



ANOSIM:- see here for more info

```
dat.vegdist <- vegdist(dat)
groups = c("top","top","top","middle","middle","middle","bottom","bottom","bottom","bottom")
dat.anosim <- anosim(dat.vegdist, grouping=groups, permutations = 999)
dat.anosim
```

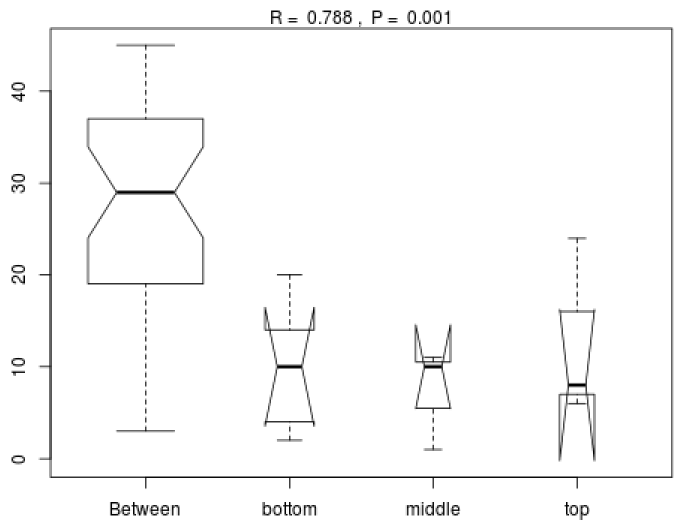
```
##
## Call:
## anosim(dat = dat.vegdist, grouping = groups, permutations = 999)
## Dissimilarity: bray
##
## ANOSIM statistic R: 0.7879
##       Significance: 0.001
##
## Permutation: free
## Number of permutations: 999
```

```
summary(dat.anosim)
```

```
##
## Call:
## anosim(dat = dat.vegdist, grouping = groups, permutations = 999)
## Dissimilarity: bray
##
## ANOSIM statistic R: 0.7879
##       Significance: 0.001
##
## Permutation: free
## Number of permutations: 999
##
## Upper quantiles of permutations (null model):
##   90%   95% 97.5%   99%
## 0.253 0.338 0.419 0.525
##
## Dissimilarity ranks between and within classes:
##         0%   25% 50%   75% 100%  N
## Between  3 19.00  29 37.00   45 33
## bottom   2  4.75  10 13.75   20  6
## middle   1  5.50  10 10.50   11  3
## top      6  7.00   8 16.00   24  3
```

```
plot(dat.anosim)
```

```
## Warning in bxp(structure(list(stats = structure(c(3, 19, 29, 37, 45, 2, :
## some notches went outside hinges ('box'): maybe set notch=FALSE
```

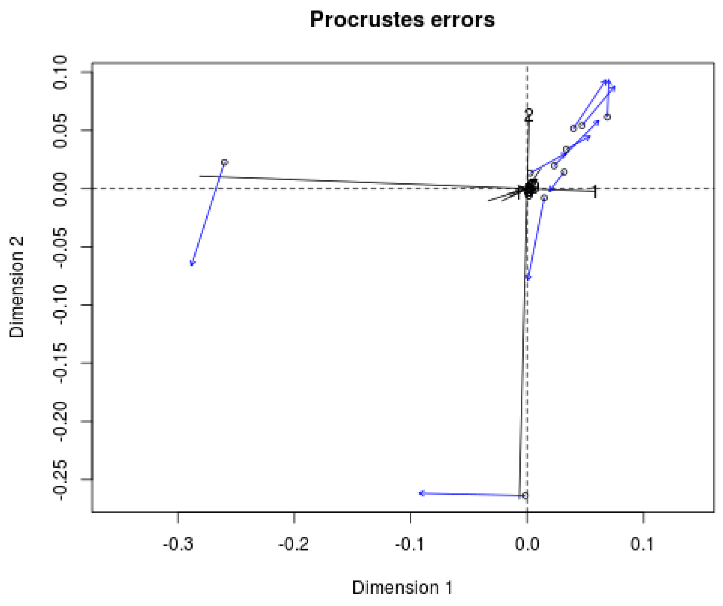


Procrustes- Comparing Ordinations : see here for more info

Also, see here for information on the unifrac distance method

Here we will compare using a unifrac distance matrix to a Bray-curtis distance matrix.

```
#procrustes(dat.mds.bray, dat.mds.eucl)
dat.mds.bray <-vegdist(dat, "bray")
pro_test <-protest(dat.mds.bray, du)
plot(pro_test)
```



```
pro_test
```

```
##
## Call:
## protest(X = dat.mds.bray, Y = du)
##
## Procrustes Sum of Squares (m12 squared):        0.1596
## Correlation in a symmetric Procrustes rotation: 0.9167
## Significance:  0.001
##
## Permutation: free
## Number of permutations: 999
```