Supplementary R code

#Part I - LSA Code

#i: represents varying on unit i

#j: represents varying on unit j

#w: represents varying on timeslot w

#d: represents varying on delay

#Order of subscripts indicates order in array, row, column... etc..

#POPiw = population for unit i at timeslot k

#RNiw = rank normal score for unit i at timeslot k

#LSAij\_d = Local similarity score for unit i, unit j, delay d

#CORNij\_d = correlation using rank normal for unit i, unit j, delay d

#CORij\_d = correlation for unit i, unit j, delay d

#CORNSij\_d = Correlation (using rank normal) significance for unit i, unit j, delay d

#CORSij\_d = Correlation significance for unit i, unit j, delay d

###Note: I could split this up into 2 timeslots to track the exact pair of times that produced a particular

###Combination, but as of yet, I can't use this, better to make the array more manageable and just track 1 slot

###All I need right now is the series for a given delay, not all the exact times

#Tw = vector of absolute times associated with timeslot k

#returns LSA score

LSA <- function(x) {

yP = 0

yN = 0

yMaxP = 0

yMaxN = 0

tmp = x

tmp[is.na(tmp)]=0

for(i in tmp) {

yP = max(yP+i,0)

yN = min(yN+i,0)

yMaxP = max(yMaxP,yP)

yMaxN = min(yMaxN,yN)

}

if(yMaxP>abs(yMaxN)) {

return(yMaxP)

}

else {

return(yMaxN)

}

}

normalTransform <- function(POPiw) t(apply(POPiw,1,function(x) qnorm(rank(x,na.last="keep")/(1+sum(!is.na(x))))))

#widens Tk to account for gaps

expandTw <- function(Tw,res=1) seq(min(Tw),max(Tw),by=res)

#widens Matrix according to Tk

expandMatrixUsingTw <- function(timeMatrix,Tw,res=1) {

TwE <- expandTw(Tw,res=res)

timeMatrixNew <- matrix(NA,nrow=nrow(timeMatrix),ncol=length(TwE))

timeMatrixNew[,is.element(TwE,Tw)] = timeMatrix

return(timeMatrixNew)

}

#res should always divide maxdelay and the minimum time measurement

compute\_LSA\_COR <- function(POPiw,maxdelay = 1,res=1,Tw=c(),expand=!is.null(Tw)) {

POPiw\_ = POPiw

Tw\_ = Tw

#expand will expand the matrix given the time.vector to ensure resolution is standard

if(expand) {

POPiw\_=expandMatrixUsingTw(POPiw,Tw,res)

Tw\_ = expandTw(Tw)

}

#number of OTUS

n = dim(POPiw\_)[[1]]

#number of timeslots

m = dim(POPiw\_)[[2]]

RNiw = normalTransform(POPiw\_)

#Store the delay range

delrange = seq(0,maxdelay,by=res)

ndelay = length(delrange)

LSAij\_d = array(NA,dim=c(n,n,ndelay))

dimnames(LSAij\_d)[[1]] = paste("OTU",1:n,sep="")

dimnames(LSAij\_d)[[2]] = paste("OTU",1:n,sep="")

dimnames(LSAij\_d)[[3]] = paste("DELAY",delrange, sep="")

CORNij\_d = array(NA,dim=c(n,n,ndelay))

dimnames(CORNij\_d)[[1]] = paste("OTU",1:n,sep="")

dimnames(CORNij\_d)[[2]] = paste("OTU",1:n,sep="")

dimnames(CORNij\_d)[[3]] = paste("DELAY",delrange, sep="")

CORij\_d = array(NA,dim=c(n,n,ndelay))

dimnames(CORij\_d)[[1]] = paste("OTU",1:n,sep="")

dimnames(CORij\_d)[[2]] = paste("OTU",1:n,sep="")

dimnames(CORij\_d)[[3]] = paste("DELAY",delrange, sep="")

CORNSij\_d = array(NA,dim=c(n,n,ndelay))

dimnames(CORNSij\_d)[[1]] = paste("OTU",1:n,sep="")

dimnames(CORNSij\_d)[[2]] = paste("OTU",1:n,sep="")

dimnames(CORNSij\_d)[[3]] = paste("DELAY",delrange, sep="")

CORSij\_d = array(NA,dim=c(n,n,ndelay))

dimnames(CORSij\_d)[[1]] = paste("OTU",1:n,sep="")

dimnames(CORSij\_d)[[2]] = paste("OTU",1:n,sep="")

dimnames(CORSij\_d)[[3]] = paste("DELAY",delrange, sep="")

for(k in 1:ndelay) {

#We will only be taking part of the time series range

#to account for this, we need to remove data points when considering a delay

#in this case, positive delay means that the second is offset from the beginning,

#and the first is offset from the end

d = delrange[k]

wrange = 1:(m-d/res)

vrange = (1+d/res):m

#compute LSA and COR on every pair of rows and store by delay

LSAij = t(apply(RNiw[,wrange],1,function(x) apply(RNiw[,vrange],1,function(y) LSA(x\*y))))

CORNij = t(apply(RNiw[,wrange],1,function(x) apply(RNiw[,vrange],1,function(y) {

not\_na = !is.na(x) & !is.na(y)

if(sum(not\_na)>10) return(cor.test(x[not\_na], y[not\_na])$estimate)

else return(NA)

})))

CORij = t(apply(POPiw\_[,wrange],1,function(x) apply(POPiw\_[,vrange],1,function(y) {

not\_na = !is.na(x) & !is.na(y)

if(sum(not\_na)>10) return(cor.test(x[not\_na], y[not\_na])$estimate)

else return(NA)

})))

CORNSij = t(apply(RNiw[,wrange],1,function(x) apply(RNiw[,vrange],1,function(y) {

not\_na = !is.na(x) & !is.na(y)

if(sum(not\_na)>10) return(cor.test(x[not\_na], y[not\_na])$p.value)

else return(NA)

})))

CORSij = t(apply(POPiw\_[,wrange],1,function(x) apply(POPiw\_[,vrange],1,function(y) {

not\_na = !is.na(x) & !is.na(y)

if(sum(not\_na)>10) return(cor.test(x[not\_na], y[not\_na])$p.value)

else return(NA)

})))

dim(LSAij) = c(n,n)

dim(CORNij) = c(n,n)

dim(CORij) = c(n,n)

dim(CORij) = c(n,n)

dim(CORSij) = c(n,n)

LSAij\_d[,,k] = LSAij

CORNij\_d[,,k] = CORNij

CORij\_d[,,k] = CORij

CORNSij\_d[,,k] = CORNSij

CORSij\_d[,,k] = CORSij

}

return(list(lsa\_d = LSAij\_d,

corn\_d = CORNij\_d,

cor\_d = CORij\_d,

corns\_d = CORNSij\_d,

cors\_d = CORSij\_d

))

}

ConfidentPairs <- function(LSA\_COR\_obj,lsasighash\_file="LSA98hashtable.csv",origdata = m000.matrix[OTUS,],cinterv=0.95) {

lsadata = LSA\_COR\_obj$lsa\_d

corsdata = LSA\_COR\_obj$cors\_d

cornsdata = LSA\_COR\_obj$corns\_d

hashtable = read.csv(lsasighash\_file)

datazeros = apply(origdata,1,function(x) sum(x==0,na.rm=TRUE))

dim1 = dim(lsadata)

#significance level is determined by the number of ties in the original data

#we expect these ties to take place at 0, effecting the significance value of the score

#hence a full set of hashed permutations of the data is stored prior to the analysis

#this strongly cuts down on the number of duplicate permutation runs

#one way to change this would be to ensure that all vectors have rank score conserved

#but this may bias certain results

lsasiglevels = matrix(Inf,nrow=dim1,ncol=dim1)

for(i in 1:(dim1[[1]]-1)) for(j in (i+1):dim1[[2]]) lsasiglevels[i,j] = hashtable[datazeros[i],datazeros[j]]

#compute COR sig for i j as the minimum over all delays

COR\_strong = apply(corsdata,c(1,2),function(x) min(x,na.rm=TRUE))

#will contain numeric(0)

COR\_delay = apply(corsdata,c(1,2), function(x) which.max(x)-1)

#compute CORN sig for i j as the minimum over all delays

CORN\_strong = apply(cornsdata,c(1,2),function(x) min(x,na.rm=TRUE))

#will contain numeric(0)

CORN\_delay = apply(cornsdata,c(1,2), function(x) which.max(x)-1)

#compute LSA for i j as the maximum over all delays

LSA\_strong = apply(lsadata,c(1,2),function(x) max(abs(x),na.rm=TRUE))

#need to fix this so that it gives the proper delays, but currently, this should be alright

LSA\_delay = apply(lsadata,c(1,2),function(x) which.max(abs(x))-1)

sigLSA = LSA\_strong>lsasiglevels

sigCOR = COR\_strong<(1-cinterv)

sigCORN = CORN\_strong<(1-cinterv)

sigLSA[is.na(sigLSA)]=FALSE

sigCOR[is.na(sigCOR)]=FALSE

sigCORN[is.na(sigCORN)]=FALSE

#don't want the self-score or duplicate scores

selfscore = matrix(FALSE,dim1[[1]],dim1[[1]])

for(i in 1:dim1[[1]]) for(j in i:dim1[[1]]) selfscore[j,i]=TRUE

lsapairs = which(sigLSA & !selfscore,arr.ind=TRUE)

corpairs = which(sigCOR & !selfscore,arr.ind=TRUE)

cornpairs = which(sigCORN & !selfscore,arr.ind=TRUE)

allpairs = which((sigLSA | sigCOR | sigCORN) & !selfscore,arr.ind=TRUE)

sharedpairs = which(sigLSA & sigCOR & sigCORN & !selfscore,arr.ind=TRUE)

return(list(lsa\_pairs=lsapairs,cor\_pairs=corpairs,corn\_pairs=cornpairs,all\_pairs=allpairs,shared\_pairs=sharedpairs))

}

pairlistToCSV <- function(pairlist,filename="LSACORpairs.csv") {

pairs = pairlist$all\_pairs

totpairs = dim(pairs)[[1]]

csvoutput = matrix(ncol=3,nrow=totpairs)

csvoutput[,1] = pairs[,1]

csvoutput[,2] = "pu"

csvoutput[,3] = pairs[,2]

colnames(csvoutput) = c("Source","Interaction","Target")

write.csv(csvoutput,file = filename,quote=FALSE)

return(filename)

}

#Part II - Benjamini-Hochberg correction

extractmatrices <- function(LCO, delayrange=c(1), fdr=TRUE) {

corn\_d = LCO$corn\_d

corns\_d = LCO$corns\_d

if(fdr) {

#adjust p-value according to Benjamini-Hochberg to control FDR

dim(corns\_d) = c()

corns\_d = p.adjust(corns\_d,method="BH",n=sum(!is.na(corns\_d)))

dim(corns\_d) = dim(LCO$corns\_d)

}

otu1 = dim(corns\_d)[[1]]

otu2 = dim(corns\_d)[[2]]

delay = dim(corns\_d)[[3]]

finalmatrix = matrix(0,nrow=otu1,ncol=otu2)

delaymatrix = matrix(NA,nrow=otu1,ncol=otu2)

cormatrix = matrix(0,nrow=otu1,ncol=otu2)

for(i in 1:otu1) {

for(j in 1:otu2) {

smallestp = which.min(corns\_d[i,j,])

effectivp = which.min(corns\_d[i,j,delayrange])

if(length(effectivp)==0) next

if(corns\_d[i,j,effectivp]<=0.01) {

delaymatrix[i,j] = smallestp-1

cormatrix[i,j] = corn\_d[i,j,effectivp]

}

}

}

signmatrix = sign(cormatrix)

finalmatrix = abs(signmatrix)

return(list(adjm=finalmatrix,delay =delaymatrix, cor = cormatrix,sign = signmatrix))

}

LSAextract2 = extractmatrices(LSA\_COR\_obj2)

# Part III - Matrix to pairlist

adjntimes <- function(x,n) {

if(n>1) return(x%\*%adjntimes(x,n-1))

else return(x)

}

#Takes matrix, converts it to csv pairlist

matrixToPairlist <- function(signmat,delaymat) {

n = dim(signmat)[[1]]

m = dim(signmat)[[2]]

z = colnames(signmat)

res = c()

for(i in 1:n) {

for(j in 1:m) {

if(i==j) next

if(signmat[i,j]==0) next

interaction = paste(ifelse(signmat[i,j]>0,"p","n"), ifelse(delaymat[i,j]==0,"u","dr"),sep="")

res = rbind(res,c(z[i], interaction, z[j]))

print(c(i,j))

}

}

return(res)

}

applyOTUnames <- function(LSAextract,otunames) {

tmp = LSAextract

rownames(tmp$adjm) = otunames

colnames(tmp$adjm) = otunames

rownames(tmp$delay) = otunames

colnames(tmp$delay) = otunames

rownames(tmp$cor) = otunames

colnames(tmp$cor) = otunames

rownames(tmp$sign) = otunames

colnames(tmp$sign) = otunames

return(tmp)

}

# Part IV - Cumulative distribution

final4surf <- read.csv("final4\_confirmation\_000.csv")

final4surf <- final4surf[,-1]

final4deep <- read.csv("final4\_confirmation\_200.csv")

final4deep <- final4deep[,-1]

tmp <- rowSums(final4surf)

tmp2 <- density(tmp)

newy2 <- cumsum(tmp2$y[tmp2$x>0])

newy3 <- 1-(newy2/max(newy2))

par(mar=c(5.1,6.1,4.1,0.1))

plot(tmp2$x[tmp2$x>0], newy3, log="y", type="l", col="red" ,xlim=c(1,300),yaxt="n", xlab="Number of connections", ylab="",main="Cumulative Distribution Plot")

axis(2, las=2, at=c(0.000001,0.0001,0.01,1), labels=c("1X10^-4","1X10^-2","1","100"))

mtext("Percent cumulative sum (log scale)",side=2, line=4.5)

par(new=T)

tmp3 <- rowSums(final4deep)

tmp4 <- density(tmp3)

newy4 <- cumsum(tmp4$y[tmp4$x>0])

newy5 <- 1-(newy4/max(newy4))

plot(tmp4$x[tmp4$x>0], newy5, log="y", type="l", col="blue", axes=F, xlab="", ylab="")

surfcumsum <- cbind(tmp2$x[tmp2$x>0],newy3)

deepcumsum <- cbind(tmp4$x[tmp4$x>0],newy5)

write.csv(surfcumsum,file="surface cumulative sum.csv")

write.csv(deepcumsum,file="deep cumulative sum.csv")

y <- surfcumsum[,2]

x <- surfcumsum[,1]

dy <- deepcumsum[,2]

dx<- deepcumsum[,1]

#fit first degree polynomial equation:

fit <- lm(y~x)

#second degree

fit2 <- lm(y~poly(x,2,raw=TRUE))

#third degree

fit3 <- lm(y~poly(x,3,raw=TRUE))

#fourth degree

fit4 <- lm(y~poly(x,4,raw=TRUE))

#fifth degree

fit5 <- lm(y~poly(x,5,raw=TRUE))

#sixth degree

fit6 <- lm(y~poly(x,6,raw=TRUE))

#seventh degree

fit7 <- lm(y~poly(x,7,raw=TRUE))

#generate range of 100 numbers starting from 1 and ending at 300

xx <- seq(1,300, length=100)

plot(x,y,type="l",ylim=c(0.000001,1))

lines(xx, predict(fit4, data.frame(x=xx)), col="red")

lines(xx, predict(fit5, data.frame(x=xx)), col="green")

lines(xx, predict(fit6, data.frame(x=xx)), col="blue")

lines(xx, predict(fit7, data.frame(x=xx)), col="purple")

anova(fit,fit2)

anova(fit2,fit3)

anova(fit3,fit4)

summary(fit)

summary(fit2)

summary(fit3)

summary(fit4)

dfit <- lm(dy~dx)

#second degree

dfit2 <- lm(dy~poly(dx,2,raw=TRUE))

#third degree

dfit3 <- lm(dy~poly(dx,3,raw=TRUE))

#fourth degree

dfit4 <- lm(dy~poly(dx,4,raw=TRUE))

dfit5 <- lm(dy~poly(dx,5,raw=TRUE))

dfit6 <- lm(dy~poly(dx,6,raw=TRUE))

dfit7 <- lm(dy~poly(dx,7,raw=TRUE))

dfit8 <- lm(dy~poly(dx,8,raw=TRUE))

#generate range of 100 numbers starting from 0 and ending at 300

dxx <- seq(0,300, length=100)

plot(dx,dy,type="l",ylim=c(0,1))

lines(dxx, predict(dfit4, data.frame(dx=dxx)), col="red")

lines(dxx, predict(dfit5, data.frame(dx=dxx)), col="green")

lines(dxx, predict(dfit6, data.frame(dx=dxx)), col="blue")

lines(dxx, predict(dfit7, data.frame(dx=dxx)), col="purple")

#Can start with other values of p1 and p2 but end up at the same fitted values but with different #iterations to convergence

p1 <- 0.05

p2 <- 0.1

> fite1 <- nls(y ~ (x^(-p1))\*(exp(-(p2)\*(x))),start=list(p1=p1,p2=p2))

> summary(fite1)

Formula: y ~ (x^(-p1)) \* (exp(-(p2) \* (x)))

Parameters:

Estimate Std. Error t value Pr(>|t|)

p1 0.0867127 0.0028833 30.07 <2e-16 \*\*\*

p2 0.0262209 0.0003044 86.14 <2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.01596 on 484 degrees of freedom

Number of iterations to convergence: 8

Achieved convergence tolerance: 9.94e-06

> plot(x,y,type="l",ylim=c(0.000001,1),log="y")

> lines(xx, predict(fite1, data.frame(x=xx)), col="blue")

> fite2 <- nls(dy ~ (dx^(-p1))\*(exp(-(p2)\*(dx))),start=list(p1=p1,p2=p2))

> plot(dx,dy,type="l",ylim=c(0.000001,1),log="y")

> lines(dxx, predict(fite2, data.frame(dx=dxx)), col="blue")

> summary(fite2)

Formula: dy ~ (dx^(-p1)) \* (exp(-(p2) \* (dx)))

Parameters:

Estimate Std. Error t value Pr(>|t|)

p1 0.0627748 0.0026595 23.60 <2e-16 \*\*\*

p2 0.0264308 0.0002916 90.63 <2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.01846 on 484 degrees of freedom

Number of iterations to convergence: 7

Achieved convergence tolerance: 9.428e-06

> confint(fite1)

Waiting for profiling to be done...

2.5% 97.5%

p1 0.08062648 0.09276734

p2 0.02556533 0.02689100

> confint(fite2)

Waiting for profiling to be done...

2.5% 97.5%

p1 0.05749339 0.06800879

p2 0.02582636 0.02704914

> pdf(file="cumulative\_dist.pdf")

par(mar=c(5.1,6.1,4.1,0.1))

plot(x, y, log="y", type="l", col="red" ,xlim=c(1,300),yaxt="n", xlab="Number of connections", ylab="",main="Cumulative Distribution Plot")

axis(2, las=2, at=c(0.000001,0.0001,0.01,1), labels=c("1X10^-4","1X10^-2","1","100"))

mtext("Percent cumulative sum (log scale)",side=2, line=4.5)

par(new=T)

plot(dx, dy, log="y", type="l", col="black", axes=F, xlab="", ylab="")

lines(xx[0:87], predict(fite1, data.frame(x=xx[0:87])), col="blue")

lines(dxx[0:87], predict(fite2, data.frame(dx=dxx[0:87])), col="green")

legend("topright",c("Suface","200 m","Surface power law fit","200 m power law fit"),lty=c(1,1,1,1),lwd=2,col=c("red","black","blue","green"))

# Part V - Linear model diagnostics

#Written for deep (200 m) samples. For surface samples, replace “deep”

#with “surf” and grab appropriate files

combSPdeep <- read.csv("combinedSP\_200.csv")

combSPdeep <- combSPdeep[,-1]

table(as.matrix(combSPdeep))

combSPdeep <- as.matrix(combSPdeep)

(sum(combSPdeep)-sum(diag(combSPdeep)))/2 #number of 1’s in the upper triangle

#use matrix=surface\_popcounts, boolean=combSPdeep for function below.

# combined\_SpearmanPearson\_surface.csv is sum of adjacency matrix output from LSA for #Spearman and Pearson correlations consisting of 0’s and 1’s.

#This script uses linear modeling to confirm the relationship between two NTUs that are judged #related by the LSA analysis and creates a new upper triangle matrix with 1’s confirming a #relationship and 0’s eliminating a relationship.

s\_matrixlm <- function(matrix, boolean,fn="lmconfirmation\_200.csv") {

s\_mat <- matrix(NA,nrow=nrow(boolean),ncol=ncol(boolean))

for(i in 1:(dim(matrix)[1])) {

for(j in 1:(dim(matrix)[1])) {

if(i>j) next

#only considers upper triangle

if(i == j) next

#ignores the diagonal

if(boolean[i,j] < 0.5) next

#only looks at putative correlations

print(paste(i, "and", j))

xval <- unlist(matrix[i,])

yval <- unlist(matrix[j,])

tmp <- lm(xval~yval)

tmp2 <- anova(tmp)

tmp3 <- tmp2$"Pr(>F)"[1]

s\_mat[i,j] <- ifelse(tmp3<=0.05,1,0)

}

}

write.csv(s\_mat,file=fn)

}

s\_matrixlm(matrix=deep\_popcounts, boolean=combSPdeep)

lmconfirmdeep <- read.csv("lmconfirmation\_200.csv")

lmconfirmdeep <- lmconfirmdeep [,-1] #removes 1st column of 1-1374

table(as.matrix(lmconfirmdeep))

lmconfirmdeep[is.na(lmconfirmdeep)] <- 0 #replace NA’s with 0’s

#Check assumptions of linear modeling that residuals are normally distributed by observing the #autocorrelation function. Need a modified dataset that has full years of data (trim away partial #years and impute data for 1-2 month gaps). Transposed the data but that is not strictly #necessary. The autocorrelation function is sensitive to the order of the variables in the linear #model (F test and p-value are insensitive to order) so generate data for both cases. Difference #the data by season (lag=12) to pre-whiten the data.

deep\_popcounts\_diff <- read.csv("BATS\_200m\_timeseries\_cellcounts\_transposed\_differenced.csv", header=T)

#Use matrix=deep\_popcounts\_diff,boolean=lmconfirmdeep in next function

#At least 3 of 9 significant values in autocorrelations of residuals if there is structure in the #residuals so use that as criteria for identifying relationships to discard (1 in matrix to discard, 0 #if OK, NA elsewhere).

s\_matrixacf <- function(matrix, boolean,fn="acf\_resid\_diff2way\_200.csv") {

s\_mat <- matrix(NA,nrow=nrow(boolean),ncol=ncol(boolean))

for(i in 1:(dim(matrix)[2])) {

for(j in 1:(dim(matrix)[2])) {

if(i>j) next

if(i == j) next

if(boolean[i,j] <0.5) next

xval <- unlist(matrix[,i])

yval <- unlist(matrix[,j])

tmp <- lm(xval~yval)

tmp3 <- lm(yval~xval)

tmp2 <- acf(tmp$resid, plot=F)

tmp4 <- acf(tmp3$resid, plot=F)

s\_mat[i,j] <- ifelse((sum((abs(tmp2$acf[2:10]))>=0.20))>=3,1,( ifelse((sum((abs(tmp4$acf[2:10]))>=0.20))>=3,1,0)))

}

}

write.csv(s\_mat,file=fn)

}

s\_matrixacf(matrix=deep\_popcounts\_diff,boolean=lmconfirmdeep)

acfd2waydeep <- read.csv("acf\_resid\_diff2way\_200.csv")

acfd2waydeep <- acfd2waydeep [,-1]

table(as.matrix(acfd2waydeep))

acfd2waydeep [is.na(acfd2waydeep)] <- 0

lmconfirmdeep\_2 <- lmconfirmdeep - acfd2waydeep #subtract correlated NTUs with non-normal linear #model residuals

table(as.matrix(lmconfirmdeep\_2))

#Initial data screening suggested that most data did not need to be differenced seasonally first. #Simulations show that structure in residuals for one NTU only does not affect type I error. #However, structure in two NTU’s can greatly affect type I error. Check autocorrelation #functions for undifferenced modified data (as before). Look at row and column sums for each #NTU. Most NTUs have only a few acf’s with structure. Most of the acf’s with structure are #found in the top 5% of NTUs (sorted by the number of problem acfs). If both NTUs are in the #upper 5%, their type I error rate is probably elevated so they should be eliminated. Others are #more likely to be OK.

deep\_popcounts\_trans <- read.csv("BATS\_200m\_timeseries\_cellcounts\_transposed.csv")

# use matrix=deep\_popcounts\_trans, boolean=lmconfirmdeep\_2 in next function

s\_matrixacf2 <- function(matrix, boolean,fn="acf\_resid\_norm\_200.csv") {

s\_mat <- matrix(NA,nrow=nrow(boolean),ncol=ncol(boolean))

for(i in 1:(dim(matrix)[2])) {

for(j in 1:(dim(matrix)[2])) {

if(i>j) next

if(i == j) next

if(boolean[i,j] <0.5) next

xval <- unlist(matrix[,i])

yval <- unlist(matrix[,j])

tmp <- lm(xval~yval)

tmp3 <- lm(yval~xval)

tmp2 <- acf(tmp$resid, plot=F)

tmp4 <- acf(tmp3$resid, plot=F)

s\_mat[i,j] <- ifelse((sum((abs(tmp2$acf[2:10]))>=0.20))>=3,1,( ifelse((sum((abs(tmp4$acf[2:10]))>=0.20))>=3,1,0)))

}

}

write.csv(s\_mat,file=fn)

}

s\_matrixacf2(matrix=deep\_popcounts\_trans, boolean=lmconfirmdeep\_2)

acfnormdeep <- read.csv("acf\_resid\_norm\_200.csv")

acfnormdeep <- acfnormdeep [,-1]

table(as.matrix(acfnormdeep))

acfnormdeep [is.na(acfnormdeep)] <- 0

rs <- rowSums(acfnormdeep)

cs <- colSums(acfnormdeep)

totaldeep <- rs + cs

totaldeep <- as.matrix(totaldeep)

total2 <- sum(totaldeep>0)

total3 <- total2\*0.05

sum(totaldeep>X) #find X by trial and error such that the sum is = total3. Fill in X in next script.

s\_matrixdualacf <- function(matrix, boolean,fn="acf\_resid\_dual\_200.csv") {

s\_mat <- matrix(NA,nrow=nrow(boolean),ncol=ncol(boolean))

for(i in 1:(dim(boolean)[2])) {

for(j in 1:(dim(boolean)[2])) {

if(i>j) next

if(i == j) next

if(boolean[i,j] <0.5) next

s\_mat[i,j] <- ifelse(matrix[i,]>=X,(ifelse(matrix[j,]>=X,1,0)),0)

}

}

write.csv(s\_mat,file=fn)

}

s\_matrixdualacf(matrix=totaldeep, boolean=acfnormdeep)

acfdualdeep <- read.csv("acf\_resid\_dual\_200.csv")

acfdualdeep <- acfdualdeep [,-1]

table(as.matrix(acfdualdeep))

acfdualdeep [is.na(acfdualdeep)] <- 0

lmconfirmdeep\_3 <- lmconfirmdeep\_2 - acfdualdeep

#subtract correlated NTUs with likely dual NTUs

#with structure in residuals

table(as.matrix(lmconfirmdeep\_3))

write.csv(lmconfirmdeep\_3, file="final4\_confirmation\_200.csv")

# Part VI - ARIMA structure simulation

#both white noise

one\_sim <- function(){

x <- arima.sim(list(), n = 100)

y <- 1 + arima.sim(list(), n = 100)

fit <- arima(y, order = c(0, 0, 0), xreg = x)

abs( fit$coef["x"] /

sqrt(diag(fit$var.coef)["x"]) ) > 1.96

}

reject <- replicate(1000, one\_sim())

table(reject)

reject

FALSE TRUE

954 46

#one white noise, one structured

one\_sim <- function(){

x <- arima.sim(list(order = c(1,0,0), ar = 0.7), n = 100)

y <- 1 + arima.sim(list(), n = 100)

fit <- arima(y, order = c(0, 0, 0), xreg = x)

abs( fit$coef["x"] /

sqrt(diag(fit$var.coef)["x"]) ) > 1.96

}

reject <- replicate(1000, one\_sim())

table(reject)

reject

FALSE TRUE

950 50

#both structured

one\_sim <- function(){

x <- arima.sim(list(order = c(1,0,0), ar = 0.7), n = 100)

y <- 1 + arima.sim(list(order = c(1,0,0), ar = 0.9), n = 100)

fit <- arima(y, order = c(0, 0, 0), xreg = x)

abs( fit$coef["x"] /

sqrt(diag(fit$var.coef)["x"]) ) > 1.96

}

reject <- replicate(1000, one\_sim())

table(reject)

reject

FALSE TRUE

663 337

# Part VII - Weighted NTU correlation similarity

x <- read.delim("~/Desktop/BermudaCodeOrg/v4.output.txt")

#v4.output.txt is a file of tree distances for each NTU compared to the 5

#final nodes to either side of that NTU. Final nodes are nodes with

#sequences assigned by Phyloassigner so each distance is between two

#nodes containing some number of sequences. This is essentially a moving

#window for phylogenetic distance. The adjacency matrix comes from the

#linear modeling filtering script. Otunames (or Ntunames) is a list of

#NTU identifiers for each node. Run this script by providing the

#adjacency matrix to num\_sim\_metric\_tree.

#extractOBJ <- c()

#extractOBJ$adjm <- (adjacency matrix)

#num\_sim\_metric\_tree(extractOBJ)

#remove last element

x = x[-13411,]

x\_comb = x

#y = x[order(x[,1]),]

#n = length(unique(x[,1]))

n = length(otunames)

n\_comb = 2\*n

distadj <- function(xdata = x, n = length(otunames))

{

eta = exp(-10)

ymat = matrix(nrow=n,ncol=n)

rownames(ymat) = otunames

colnames(ymat) = otunames

for(i in 1:length(x[,1])) {

ymat[as.character(x[i,1]),as.character(x[i,2])] = x[i,3]

ymat[as.character(x[i,1]),as.character(x[i,1])] = 0

}

ymat = exp(-5\*ymat)

ymat[is.na(ymat)] = 0

return(ymat)

}

ymatrix = distadj()

#need a second version for combined

distadj\_comb <- function(xdata = x\_comb, n = 2\*length(otunames))

{

eta = exp(-10)

ymat = matrix(nrow=n,ncol=n)

rownames(ymat) = c(paste(otunames,"\_000",sep=""), paste(otunames,"\_200",sep=""))

colnames(ymat) = c(paste(otunames,"\_000",sep=""), paste(otunames,"\_200",sep=""))

for(i in 1:length(x[,1])) {

xa = paste(as.character(x[i,1]), "\_000", sep="")

xb = paste(as.character(x[i,2]), "\_000", sep="")

x2a = paste(as.character(x[i,1]), "\_200", sep="")

x2b = paste(as.character(x[i,2]), "\_200", sep="")

ymat[xa,xb] = x[i,3]

ymat[xb,xa] = x[i,3]

ymat[x2a,x2b] = x[i,3]

ymat[x2b,x2a] = x[i,3]

#ymat[xa,x2b] = x[i,3]

#ymat[x2b,xa] = x[i,3]

#ymat[x2a,xb] = x[i,3]

#ymat[xb,x2a] = x[i,3]

ymat[xa,xa] = 0

#ymat[xa,x2a] = 0

#ymat[x2a,xa] = 0

ymat[x2a,x2a] = 0

}

ymat = exp(-5\*ymat)

ymat[is.na(ymat)] = 0

return(ymat)

}

ymatrix\_comb = distadj\_comb()

num\_sim\_metric\_tree <- function(extractOBJ,group = c(), ymat = ymatrix,nodenames = otunames) {

tmp\_adjm = extractOBJ$adjm

tmp\_nodenames = nodenames

tmp\_ymatrix = ymat

if(!is.null(group)) {

#filter to just the group we're interested in

tmp\_adjm = tmp\_adjm[group,group]

tmp\_ymatrix = tmp\_ymatrix[group,group]

tmp\_nodenames = tmp\_nodenames[group]

}

#coerce nodes to numbers

adj\_nodenames = as.character(tmp\_nodenames)

adj\_nodenames = as.numeric(adj\_nodenames)

#select those nodes which have numbered values

tmpset = which(!is.na(adj\_nodenames))

group\_nodenames = tmp\_nodenames[tmpset]

#smaller adjacency matrix

tmp\_adjm = tmp\_adjm[tmpset,tmpset]

tmp\_ymatrix = tmp\_ymatrix[tmpset,tmpset]

rownames(tmp\_adjm) = group\_nodenames

colnames(tmp\_adjm) = group\_nodenames

tmp\_mavem = matrix(nrow = length(tmpset),ncol = length(tmpset))

rownames(tmp\_mavem) = group\_nodenames

colnames(tmp\_mavem) = group\_nodenames

tmp\_mavem = tmp\_adjm%\*%(tmp\_ymatrix)

sim\_matrix = matrix(nrow = length(tmpset),ncol = length(tmpset))

for(i in 1:length(tmpset)) {

for(j in 1:length(tmpset)) {

totpoints = sum((tmp\_mavem[i,] + tmp\_mavem[j,])\*\*2)

diffpoints = sum((tmp\_mavem[i,] - tmp\_mavem[j,])\*\*2)

ipoints = sum(tmp\_mavem[i,])

jpoints = sum(tmp\_mavem[j,])

if(ipoints == 0 | jpoints == 0) next

sim\_matrix[i,j] = diffpoints/totpoints

}

}

y = t(1/(sim\_matrix+1))

rownames(y) = group\_nodenames

colnames(y) = group\_nodenames

return(y)

}

sim\_metric\_csv <- function(LSAextract,fname,groupA) {

tmp = num\_sim\_metric\_tree(LSAextract,group = groupA )

print(paste("Writing ", fname))

write.csv(tmp, file=fname)

}

sim\_metric\_csv\_mangroup <- function(LSAextract,fname,nodenames=otunames) {

tmp = num\_sim\_metric\_tree(LSAextract)

print(paste("Writing ", fname))

adj\_nodenames = as.character(nodenames)

adj\_nodenames = as.numeric(nodenames)

#select those nodes which have numbered values

tmpsetind = which(!is.na(adj\_nodenames))

tmpset = adj\_nodenames[tmpsetind]

write.csv(tmp[intersect(group1surf,tmpset),intersect(group1surf,tmpset)], file=paste(fname,"\_1.csv",sep=""))

write.csv(tmp[intersect(group2surf,tmpset),intersect(group2surf,tmpset)], file=paste(fname,"\_2.csv",sep=""))

write.csv(tmp[intersect(group3surf,tmpset),intersect(group3surf,tmpset)], file=paste(fname,"\_3.csv",sep=""))

}

# Part VIII - Spectral clustering script

surf <- read.csv("surface\_weighted\_sim.csv")

surf[1:5,1:5]

surf <- surf[,-1]

dim(surf)

ntu <- read.csv("otunames.csv")

ntu[1:5,]

ntu <- rbind(0,ntu)

ntunames <- ntu[,1]

length(ntunames)

rownames(surf) <- ntunames

colnames(surf) <- ntunames

#matrix is a similarity matrix; trying to find clusters in the data

library(kernlab)

ntulist <- function(matrix){

ntus <- c()

for(i in 1:nrow(matrix)){

ifelse(sum(matrix[i,]) == 0,next, ntus <- c(ntus,rownames(matrix)[i]))

}

return(ntus)

}

#vector1 is from output of ntulist

samp\_vector <- function(vector1){

vector2 <- sample(vector1, round(0.9\*(length(vector1))))

return(vector2)

}

#vector1 is full ntu list, vector2 is subsample of 90% of ntus

newmatrix <- function(matrix,vector1,vector2){

tmp\_matrix <- matrix(nrow=length(vector2),ncol=length(vector2))

rownames(tmp\_matrix) <- vector2

colnames(tmp\_matrix) <- vector2

for(i in 1:length(vector2)){

for(j in 1:length(vector2)){

tmp\_matrix[vector2[i],vector2[j]] <- matrix[vector2[i],vector2[j]]

}

}

return(tmp\_matrix)

}

tmp\_clust <- function(tmp\_matrix,centersnum,kerneltype){

clust <- specc(tmp\_matrix,centers=centersnum,kernel="kerneltype")

return(clust)

}

#vector1 is ntus and vector2 is sample of ntus

all\_clust <- function(clust,vector1,vector2){

clust\_matrix <- matrix(nrow=1,ncol=length(vector1))

colnames(clust\_matrix) <- vector1

for(k in 1:length(vector2)){

clust\_matrix[1,vector2[k]] <- clust@.Data[k]

}

return(clust\_matrix)

}

#input matrix is result from bootstrap\_clustering

clust\_sim <- function(matrix, matrix2, fn2){

vector1 <- ntulist(matrix)

sim\_mat <- matrix(nrow=ncol(matrix2),ncol=ncol(matrix2))

rownames(sim\_mat) <- vector1

colnames(sim\_mat) <- vector1

for(i in 1:ncol(matrix2)){

for(j in 1:ncol(matrix2)){

match <- 0

total <- 0

for(k in 1:nrow(matrix2)){

ifelse(is.na(matrix2[k,i]+matrix2[k,j]),next,total <- total+1)

if((matrix2[k,i]+matrix2[k,j])/2 == (matrix2[k,i])) match <- match+1

}

sim\_mat[i,j] <- match/total

}

}

write.csv(sim\_mat,file=fn2)

}

#run previous functions in proper order and iterate

bootstrap\_clustering <- function(matrix,iteration,fn1){

vector1 <- ntulist(matrix)

clust\_matrix\_all <- matrix(nrow=iteration,ncol=length(vector1))

colnames(clust\_matrix\_all) <- vector1

for(t in 1:iteration){

smallvector <- samp\_vector(vector1)

temp\_mat <- newmatrix(matrix,vector1,vector2=smallvector)

temp\_cl <- tmp\_clust(tmp\_matrix=temp\_mat,centersnum=6,kerneltype=rbfdot)

for(k in 1:length(smallvector)){

clust\_matrix\_all[t,smallvector[k]] <- temp\_cl@.Data[k]

}

}

write.csv(clust\_matrix\_all,file=fn1)

return(clust\_matrix\_all)

}

tmp <- bootstrap\_clustering(matrix=surf4, iteration=100, fn1="surface\_unweighted\_rbf\_100.csv")

tmp2 <- clust\_sim(matrix=surf4, matrix2=tmp, fn2="surface\_unweighted\_rbf\_100\_sim.csv")

surf\_rbf\_100 <- bootstrap\_clustering(matrix=surf,iteration=100, fn1="surf\_rbf\_100.csv",fn2="sim\_surf\_rbf\_100.csv")

matchlist <- function(matrix){

n <- 1

k <- 1

clusts <- list()

for(i in 1:k){

for(j in 1:ncol(matrix)){

if(i>=j) next

if(length(unlist(clusts[n]))==0){

if(matrix[i,j]==1){

clusts[n] <- colnames(matrix)[i]

clusts[[n]] <- c(clusts[[n]],colnames(matrix)[j])

if(length(unlist(clusts[n])) >0) next

}

}

if(length(unlist(clusts[n]))>=2){

if(matrix[i,j]==1){

clusts[[n]] <- c(clusts[[n]],colnames(matrix)[j])

}

}

}

}

return(clusts)

}

ifelse(length(unlist(clusts[n])) ==0, k <- k+1, break)

}

for(i in k+1:ncol(matrix)){

for(j in 1:ncol(matrix)){

if(i>=j) next

n <- n+1

for(c in 1:n-1){

if(length(unlist(clusts[c])) != 0){

for(x in 1:length(unlist(clusts[c]))){

if(unlist(clusts[[c]])[x] == colnames(matrix[i])){

if(matrix[i,j]==1) clusts[[c]] <- c(clusts[[c]], colnames(matrix[j]))

}

}

}

}

if(length(unlist(clusts[n]))==0){

if(matrix[i,j]==1){

clusts[n] <- colnames(matrix[i])

clusts[[n]] <- c(clusts[[n]],colnames(matrix[j]))

}

}

if(length(unlist(clusts[n])) >0){

if(matrix[i,j]==1){

if(clusts[[n]][2]!=colnames(matrix[j])){

clusts[[n]] <- c(clusts[[n]],colnames(matrix[j]))

}

}

}

}

}

return(clusts)

}

firstclust <- function(i,n,clusts,matrix){

for(j in 1:ncol(matrix)){

if(i>=j) next

if(length(unlist(clusts[n]))==0){

if(matrix[i,j]==1){

clusts[n] <- colnames(matrix)[i]

clusts[[n]] <- c(clusts[[n]],colnames(matrix)[j])

if(length(unlist(clusts[n])) >0) next

}

}

if(length(unlist(clusts[n]))>=2){

if(matrix[i,j]==1){

clusts[[n]] <- c(clusts[[n]],colnames(matrix)[j])

}

}

}

return(clusts)

}

matchlist <- function(matrix){

n <- 1

i <- 1

clusts <- list()

clust1 <- firstclust(i,n,clusts,matrix)

while(length(unlist(clust1[n]))==0){

i <- i+1

clust1 <-firstclust(i,n,clusts,matrix)

}

return(clust1)

}

# vector to lists

#converts a matrix of ntu names and cluster assignments to a list of ntus for each cluster

#start with a two column matrix where column 1 is ntu names and column 2 is cluster assignment

vtol <- function(matrix){

matrix[is.na(matrix)] <- 0

ls <- list()

for(i in 1:max(matrix[,2])){

tmp <- c()

x <- 1

for(j in 1:nrow(matrix)){

if (i==matrix[j,2]){

tmp[x] <- matrix[j,1]

if (length(tmp)==x){

x <- x+1

}

}

ls[[i]] <- tmp

}

}

return(ls)

}