FARM: simulating the spread of agriculture through cultural phylogenies

Users manual for running simulations in R and on Linux clusters

Ty Tuff, Bruno Vilela, and Carlos Botero

project began: 15 May 2016, document updated: 06 February 2018

Contents

Chapter 1

Introduction

While we have information on where nodes should be located, we still lack data on how cultural nodes should be connected. To compensate for a lack of data describing these connections between societies, we randomized the number of edges between nodes for different simulations. We created 11 different networks, each with a different minimum number of neighbors ranging from 4 to 14 computed using the function … in the R package … . This function builds a network where the minimum number of connections is achieved for every node, but some nodes may have more connections than that minimum in order to achieve the minimum connections for every node across the network. At the beginning of each simulation, the algorithm draws one of the 11 available networks and uses that network as the arena for carrying out that simulation.

Each node occupies a unique geographic location that is assigned a binary value indicating whether the environment of the region is more favorable to foragers or domesticators based on the results from Vilela et al (unpublished work). This designation was determined by the potential richness of domesticated species in each location estimated using ecological niche models. The threshold richness, above which we considered the region more favorable to domesticators, correspond to the potential number of species where farming societies become the dominant form of subsistence. Only some of these societies were inside of the known origins of agriculture as reported by Larson et al 2014 so, we assigned each node a value of 1 or 0 to distinguish between those inside or outside of those origins.

Chapter 2

Spatial scaffolding for the simulations

Using the ~8000 currently extant cultures cataloged in the glottolog repository (http://glottolog.org), we calculated a symetrical spatial proximity network connecting them. This network provides the scaffolding for the simulation to grow on.

```
library(FARM)
library(spdep)
library(geosphere)
library(maps)
```

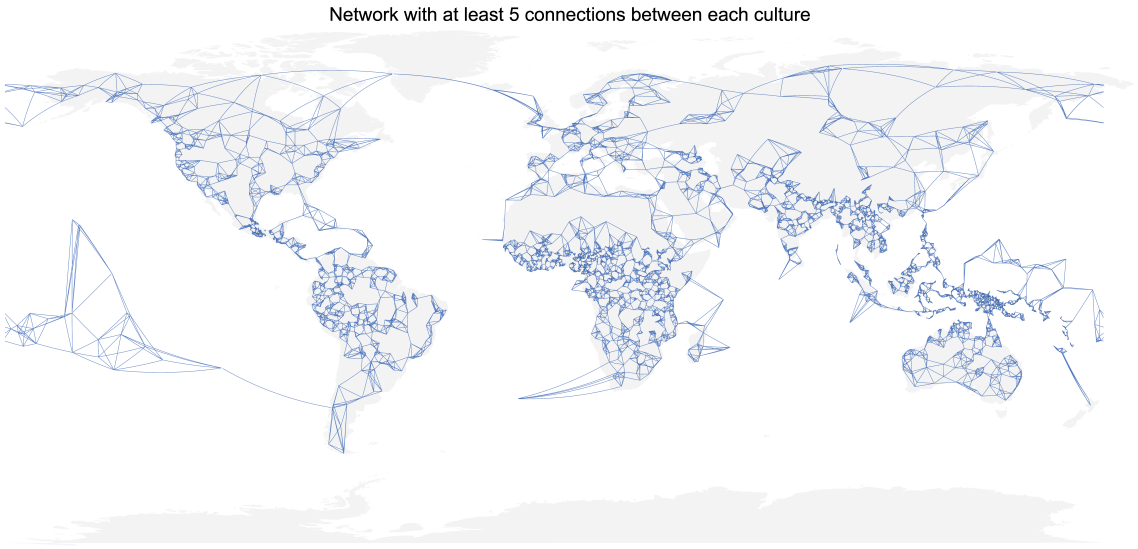
We constructed this network using the functions knearneigh() and knn2nb() in the package spdep. The user identifies the number of neighbors and the algorythm identifies that many nearest neighbors calculated based on proximity. Using those neighbor identifiers, we then calculated the great circle distance between each point using the gcIntermediate() function from the geosphere package and ploted those lines over a map.

Through a seperate analysis based on ecological niche models of agricultue, we determined that networks with 7 neighbors produced spatial distributions with the lowest level of autocorrelation. Accordingly, we ran the simulation on networks with 5-9 connections between neighbors to represent some variation in this parameter but centered these estimates around 7. At the begining of each simualation, one network was chosen from the 5 possible arrangements and used as simulation scaffolding. The number of neighbors assigned to a particular simulation replicate is recorded in the metadata so it can be extracted and used as a covariate in later models.

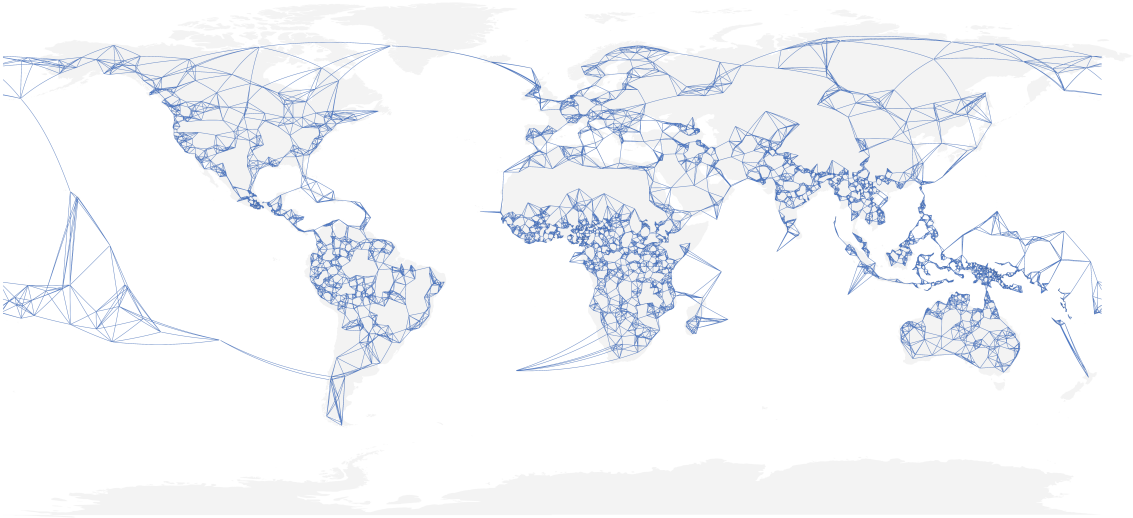
Here is the code for producing and plotting these networks. The object language_centroids is in the FARM package, so make sure that package is loaded so R can find the required data. Below the code are plots of the 5 possible graph arrangements used in the FARM simulations.

```
coords <- as.matrix(apply(language_centroids[, 3:4], 2, as.numeric)) #coords
conds <- ifelse(suitability2 == 0, 1, 2)
conds[is.na(conds)] <- sample(c(1, 2), sum(is.na(conds)), replace = TRUE)
origins <- language_centroids[, 5]



nbs_number <- 2
this_row <- 1
number_of_neighbors <- 7
for(i in 5:9){

number_of_neighbors <- i
nbs <- knearneigh(coords, k = number_of_neighbors, longlat = TRUE)
```

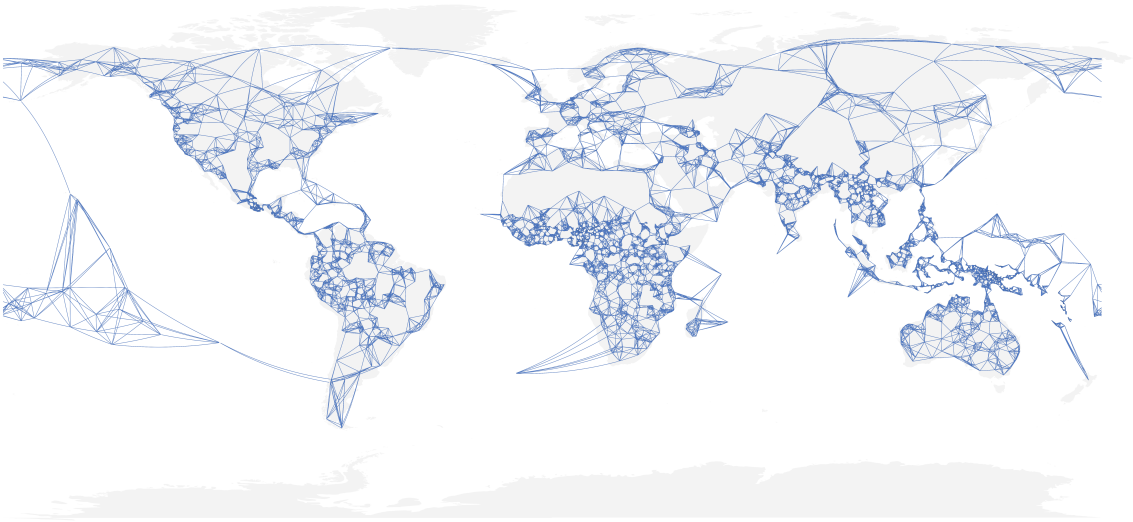
```
nbs_list <- knn2nb(nbs, sym = TRUE)


png(paste("network_scratch_", number_of_neighbors, ".png"), width=11, height=5.5, res=1000, units = "in


par(mar=c(0,0,0,0))

plot(0,0, xlim=c(-180,180), ylim=c(-90,90), type="n")
map("world", col=adjustcolor("lightgrey", alpha=0.2), interior = FALSE, fill=TRUE, border=NA)
#map("state", col="lightgrey")
for(this_row in 1:length(nbs_list)){
for(nbs_number in 1:length(nbs_list[[this_row]])){
#pointser <- NULL
pointser <- gcIntermediate(c(coords[nbs_list[[this_row]][nbs_number],1], coords[nbs_list[[this_row]][nbs
    length(pointser)
        if(length(pointser) == 200){lines(pointser, col=adjustcolor("black", alpha=1), lwd=.3)}
if(length(pointser) == 2){lines(pointser[[1]], col=adjustcolor("black", alpha=1), lwd=.3)
    lines(pointser[[2]], col=adjustcolor("black", alpha=1), lwd=.3)}

    #lines(pointser, col="red", lwd=.5)
if(length(pointser) == 200){lines(pointser, col="cornflowerblue", lwd=.25)}
if(length(pointser) == 2){lines(pointser[[1]], col="cornflowerblue", lwd=.25)
    lines(pointser[[2]], col="cornflowerblue", lwd=.25)}


    mtext(paste0("Network with at least ", number_of_neighbors, " connections between each culture"), 3
}}

dev.off()


}
```



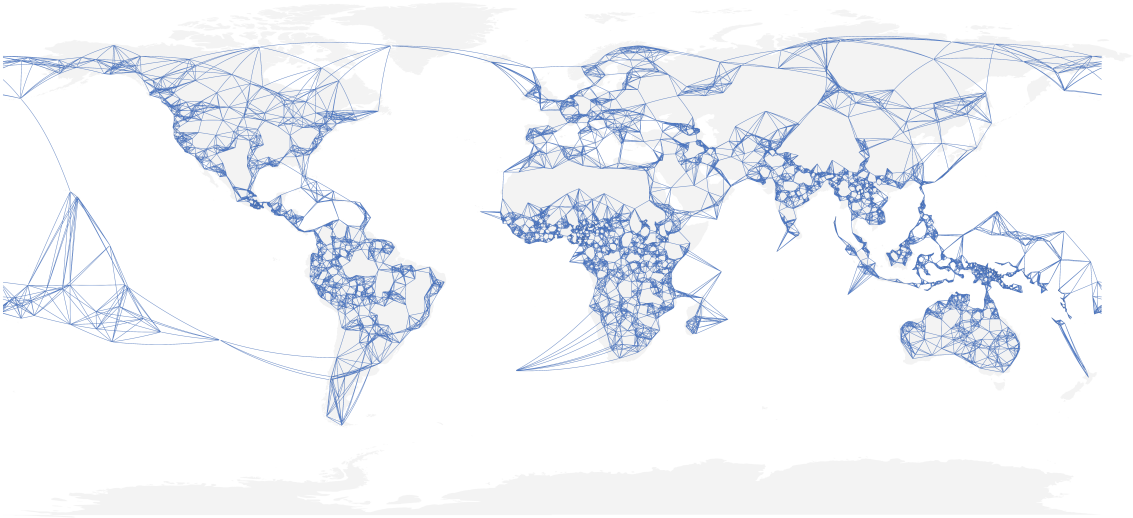
Network with at least 6 connections between each culture



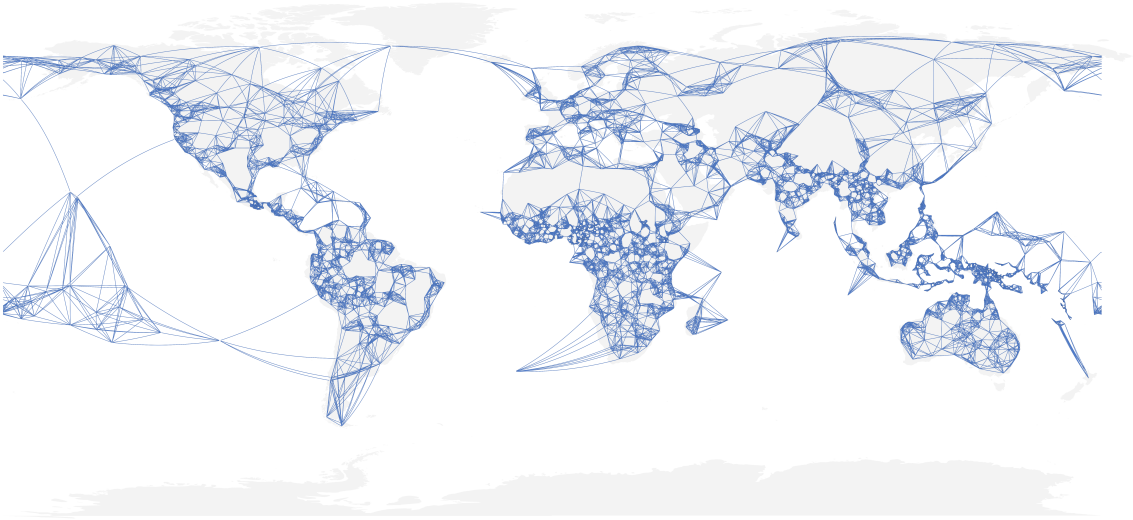
Network with at least 7 connections between each culture



Network with at least 8 connections between each culture



Network with at least 9 connections between each culture



Chapter 3

Rule sets

```
##
## The downloaded binary packages are in
##  /var/folders/4v/qwc_jrrd1wd07klywdz9qv8c0000gq/T//RtmpMihxdG/downloaded_packages
##
## The downloaded binary packages are in
##  /var/folders/4v/qwc_jrrd1wd07klywdz9qv8c0000gq/T//RtmpMihxdG/downloaded_packages
##
## The downloaded binary packages are in
##  /var/folders/4v/qwc_jrrd1wd07klywdz9qv8c0000gq/T//RtmpMihxdG/downloaded_packages
```

3.1 Simulation:

Building a model that simulates the spread of agriculture across human cultures took several independently built and validated components. The simulations play out in an arena defined by a spatial network of cultural centroids and those networks needed to be defined. Assigning agricultural suitability measures to each society required first modeling agricultural suitability across all environments and then assigning suitability values to each node in the spatial network. Events unfold as simulations progress through time and those events must be tracked from beginning to end. Running these simulations required the development of a stand-alone system for generating input parameters that control the experimental design. Rules for how simulations operate were carefully designed to describe the behavior of each hypothesized mechanism. Each replicate simulation produced a spatial pattern and phylogenetic tree that are too complex to compare to each other directly so, each simulation output is summarized by 12 summary statistics that can be passed to the random forest machine learning algorithm for categorization. These six components each play an important role in the design and operation such a large-scale simulation. We describe each component of the simulation in more detail below.

There are 5 general behavioral continuation rules that correspond to the 5 categories of input variables described above. The first 3 rules are included in all models. Rules 4 and 5 are either included or excluded from the model to define different model types corresponding to different hypothesized mechanisms. These rules describe the possible ways that each node can change status (e.g. change from or to extant, extinct, forager, or domesticator) over the course of one time step and we define them as follows:

3.1.0.1 1. Arisal -

A society switches their subsistence mode without any influence from ancestors or neighbors.

3.1.0.2 2. Extinction -

A society dies out. This terminates their branch of the phylogeny and resets their node to unoccupied. The probability of extinction is higher when the society's subsistence mode doesn't match their environment.

3.1.0.3 3. Speciation -

A society expands spatially across the network by sending a descendent diaspora to colonize an unoccupied adjoining node. This expansion into unoccupied territory creates a speciation event, where a new branch is added to the phylogeny showing a new society as a descendent of th e original society. If all adjoining nodes are occupied, then speciation cannot occur. The decedent society inherits their subsistence mode from the parent society, regardless of environment.

3.1.0.4 4. Diffusion -

A society converts an occupied adjoining society to match its own subsistence mode. If there are no occupied neighboring nodes with an opposing subsistence mode, then diffusion cannot occur.

3.1.0.5 5. Takeover -

A society expands spatially across the network by sending a descendent diaspora to colonize an occupied adjoining node. If the target society is connected to an unoccupied node, then that society can flee this expansion and avoid extinction by abandoning their current node and colonizing that unoccupied node. If the target society is surrounded by occupied nodes, then they will go extinct when the expanding society colonizes their node. This extinction terminates their branch of the phylogeny and resets their spatial node to unoccupied. As with speciation, the expansion of the conquering society's diaspora creates a speciation event, where a new branch is added to the phylogeny showing a new society as a descendent of the original society. If all adjoining nodes are unoccupied, then takeover cannot occur. The decedent society of the conquer inherits their subsistence mode from the parent society, regardless of environment.

To avoid any implied hierarchy between these rules or between nodes, both were randomized regularly. Within a single time step, the order of the rules were randomized before applying them to any nodes and then they were applied one-rule-at-a-time randomly (sampled without replacement) across all the nodes. One time step was complete when all the rules had been applied to all of the individual nodes. When one time step turns over to another, the order of rules is redrawn and they are again applied randomly across the nodes. This process was repeated for 30,000 time steps, which preliminary experiments showed was long enough for the system to reach equilibrium. Using preliminary trials of the simulations, we concluded that the model had reached equilibrium by plotting summary statistics through time and showing that statistics that asymptote usually do so between 10 and 20 thousand time steps so they are stabilized well before 30,000 time steps.

“‘

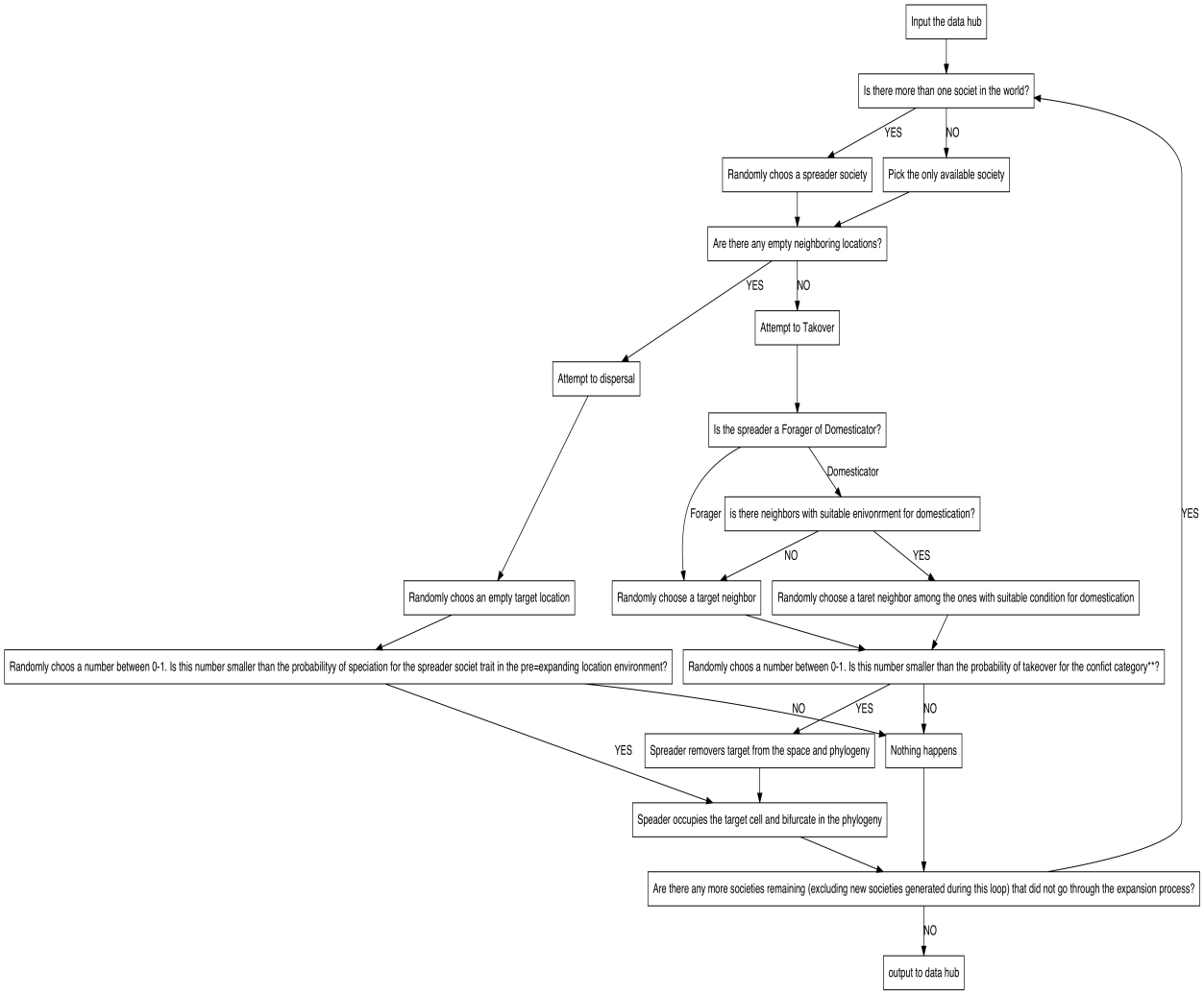


Figure 3.1: Rule set for the Takeover model

Chapter 4

Module 1: Simulation to produce a world and a tree

After the spatial network was built to host the simulations, the machinery for tracking simulation progress was in place, and the input parameters were assigned, then each simulation could launch according to a set of initial launch rules and then play out according to a separate set of continuing behavioral rules. The initial launch rules begin by stipulating that the first human culture will originate within modern-day Ethiopia and radiate outward from that point. All of these original cultures use foraging as their only mode of subsistence until at least 80% of the available nodes are occupied by a culture and then agriculture can start to arise by foragers switching to agriculturalists. The ability to switch from forager to agriculturalist without any outside influence is restricted to within known origins of agriculture as defined by Larson et al. (2014). Switching from forager to agriculturalist can only occur on nodes within those known origins, but agriculturalists can switch to foragers with equal probability on any node. Nodes can only interact via edges so, edges define neighboring nodes. These initial launch rules are the same for all simulations, regardless of the hypothesized mechanism they are simulating.

When simulations were terminated at 30,000 time steps, they will have produced a spatial pattern of subsistence modes distributed across the geographic network and a phylogeny describing the relatedness of those societies through time. These two objects contain a great deal of information about the trajectory of particular replicates but that complexity makes it difficult to compare replicates directly. No test currently exists to compare a combination of spatial and phylogenetic patterns between replicated simulations. To simplify these outputs and allow them to be compared directly to each other, we calculated a suite of 12 summary statistics to describe different features of both the spatial distribution and the phylogeny. During preliminary tests of the simulations, we used 19 summary statistics but later eliminated 7 of those original statistics because they didn't stabilize over time to where they could be described by a linear model. The 12 remaining statistics stabilized by either increasing regularly through time or came to an asymptote as the simulation reached equilibrium (SI figure…). After all of the simulations were complete, these summary statistics were concatenated into a single dataset, labelled with the rule set (mechanism) used to create each one, and then passed on to the random forest algorithm for analysis.

```
# Install the most recent version of FARM from a .zip file
install.packages(file.choose(), repos=NULL)
```

```
library(FARM)
ls("package:FARM")
```

```
##  [1] "Arisal"                "bd"
##  [3] "BuildWorld"            "combo_of_choice"
##  [5] "coords"                "coords.austronisian"
##  [7] "coords.bantu"          "coords.uto"
```

```
##  [9] "Diffusion"               "DivDep"
## [11] "DropTip"                 "Dsig"
## [13] "evol.distinct2"          "extinct"
## [15] "Extinction"              "getTargets"
## [17] "language_centroids"      "makePhy"
## [19] "Module_2"                "NewTip"
## [21] "parameters"              "parameters.table"
## [23] "plot.myworld"            "RunSim"
## [25] "RunSim.push"             "RunSim2"
## [27] "RunSim2.push"            "RunSimUltimate"
## [29] "RunSimUltimate.push"     "RunSimUltimate2"
## [31] "RunSimUltimate2.push"    "speciate"
## [33] "Speciation"              "SpeciationTakeOver"
## [35] "SpeciationTakeOver.push" "sub.TakeOver"
## [37] "sub.TakeOver.push"       "suitability"
## [39] "suitability2"            "TakeOver"
## [41] "TakeOver.push"           "TheOriginOfSpecies"
```

4.1 Inputs

The specific behavior of any particular replicate simulation is controlled by 17 input parameters. All of these values are unique to each replicate simulation, but don't change from the beginning to the end of a simulation. These parameters are all constrained between 0 and 1 and fall within five general categories: speciation, extinction, cultural diffusion, diffusion by takeover, and arisal. The first two categories, speciation and extinction, each require four parameters to describe the different ways that farmers and foragers can interact with environments that are suitable or unsuitable for agriculture. Diffusion by takeover also requires four parameters to describe the four ways that farmers or foragers can displace their neighbors to expand their subsistence mode. Cultural diffusion is simpler than diffusion by takeover and only requires two input parameters: diffusion from farmers to foragers and diffusion from foragers to farmers. Arisal follows the same convention as the cultural diffusion inputs but the probability of switching from foragers to farmers is constraining to within the known origins of agriculture (see Figure 2).

Most of the input parameters are drawn randomly from a uniform distribution constrained between 0 and 1, but some of these uniform random draws are constrained more narrowly. Parameter values in the speciation or extinction categories are ordered so that farmers in environments suitable for farming will have the highest speciation rate and lowest extinction rate, farmers in environments unsuitable for agriculture will have the lowest speciation rate and highest extinction rate, and all foragers will have an intermediate probability between them. The first value selected during a random draw is assigned to the high probability input, the second is constrained to be smaller than the first draw and assigned to the low probability input, the third is constrained between the first two values. Arisal is constrained so that the probability of switching from forager to farmer is always higher than switching from farmer to forager.

The input parameters are also where the model type is assigned to each simulation. The hypothesized mechanism prescribed to each simulation are relayed to the algorithm by setting specific sets of parameters to 0 so certain events have no chance of happening. The full ensemble model (+culture +takeover) assigns a value to all 17 parameters, the cultural diffusion only model (+culture) sets all takeover values to 0, the diffusion by takeover model (+takeover) sets all cultural diffusion values to 0, and the basic model sets all culture and takeover values to 0.

4.2 Module 1 functions

4.2.0.1 The first set of RunSim functions are the default pipeline where only one output is saved at the end of the simulation.

This first function controls error messages coming from the primary function below.

```
# Run the simulation function skiping the erros and atributing NA if it occurs
RunSimUltimate <- function(myWorld, P.extinction, P.speciation,
                           P.diffusion, P.Arisal, P.TakeOver, nbs, independent,
                           N.steps, multiplier,
                           silent = TRUE, start = NULL) {

  result <- try(RunSim(myWorld, P.extinction, P.speciation,
                       P.diffusion, P.Arisal, P.TakeOver, nbs,
                       independent, N.steps,
                       multiplier, start = start), silent = silent)
  if (class(result) == "try-error") {
    result <- NA
  }
  return(result)
}
```

This is the primary function running the simulation.

```
#=================================================================
# SimulationFunctions.R
#
# Contains a function for simulation of cultural evolution in space and time
# Allows for (1) Vertical Transmission (phylogenetic inheritance); (2) Horizontal
# Transmission (cultural diffusion); (3) Ecological selection (Both speciation and
# extinction are determined by the match between the state of a binary trait and the
# environment a societuy occupies).
#
# 7 Jun 2016
# Carlos A. Botero, Bruno Vilela & Ty Tuff
# Washington University in Saint Louis
#=================================================================
RunSim <- function(myWorld, P.extinction, P.speciation,
                   P.diffusion, P.Arisal, P.TakeOver, nbs, independent,
                   N.steps, multiplier, start) {
  # myWorld = The hexagonal world created with the function BuildWorld
  # P.extinction = Probability matrix of extinction
  # P.speciation = Probability matrix of speciation
  # P.diffusion = Probability matrix of diffusion
  # P.Arisal = Probability matrix of arisal
  # P.TakeOver = Probability matrix of takeover
  # N.steps = Number of steps in the model
  # multiplier = The number that will multiply the probabilities according
  # to environmetal fitness.
  # start = the point ID in 'myWorld' that will give risen to humans.
  # (humans origin will be in one of the existing positions)

  world.size <- nrow(myWorld)
  # Initialize parameters we will use later to build the phylogeny
```

```
  rootnode <-  world.size + 1 # standard convention for root node number

  # set the seed for simulation
  if (is.null(start)) {
  start <- sample(1:world.size, 1)
  }

  myWorld[start, 4:6] <- c(0, 0, 1) # Setting root(0), time(0), ancestral(1, forager)

  mytree <- TheOriginOfSpecies(world.size, start) # Empty tree
  myT <- 0 # Time starts at zero

  # Common input and output for all the internal modules
  input <- list(P.speciation, P.Arisal, P.diffusion, P.extinction, P.TakeOver,
                myWorld, mytree, myT, multiplier, nbs, independent)

  # Functions order to be randomized
  rand_order_func_run <- list("Extinction", "Diffusion", "SpeciationTakeOver", "Arisal")

  cat("0% [") # Time count

  for (steps in 1:N.steps) { # Starts the loop with 'n' steps

    if (steps %% round((N.steps / 10)) == 0) { # Time count
      cat('-') # Time count
    }# Time count
    if (steps == N.steps) { # Time count
      cat("] 100 %\n")# Time count
    }# Time count

    # Randomize functions order
    rand_order <- sample(rand_order_func_run)
    # Run the functions
    input <- do.call(rand_order[[1]], list(input = input))
    input <- do.call(rand_order[[2]], list(input = input))
    input <- do.call(rand_order[[3]], list(input = input))
    input <- do.call(rand_order[[4]], list(input = input))

  }
  # Trunsform the input/output into the final result and return it
  myWorld <- as.data.frame(input[[6]])
  myWorld[, 8] <- paste0("t", myWorld[, 8])
  mytree <- makePhy(input[[7]])
  mytree$edge.length <- mytree$edge.length / N.steps
  return(list('mytree' = mytree, 'myWorld' = myWorld))
}
```

We track each simulation from start to finish using two data storage objects, a spatial object storing the current subsistence mode of each node and a phylogenetic object tracking the relationship between all nodes through time. For each action taken during the simulation, the algorithm first checks the current state of these two objects, then prescribes a change to a single node within the network according to a set of rules, and then modifies that node in both storage objects before moving on to the next node. This process is repeated for each node within each time step and randomized across time steps to create an entire simulation. The phylogenetic object is built under standard evolutionary assumptions necessary for many of the summary

statistics used later, so the tree is always bifurcating, non-reticulated, and ultrametric.

4.3 Push versions

```
# Run the simulation function skiping the erros and atributing NA if it occurs
RunSimUltimate.push <- function(myWorld, P.extinction, P.speciation,
                                P.diffusion, P.Arisal, P.TakeOver, nbs, independent,
                                N.steps, multiplier,
                                silent = TRUE, start = NULL) {

  result <- try(RunSim.push(myWorld, P.extinction, P.speciation,
                            P.diffusion, P.Arisal, P.TakeOver, nbs,
                            independent, N.steps,
                            multiplier, start = start), silent = silent)
  if (class(result) == "try-error") {
    result <- NA
  }
  return(result)
}
```

```
#=================================================================
# SimulationFunctions.R
#
# Contains a function for simulation of cultural evolution in space and time
# Allows for (1) Vertical Transmission (phylogenetic inheritance); (2) Horizontal
# Transmission (cultural diffusion); (3) Ecological selection (Both speciation and
# extinction are determined by the match between the state of a binary trait and the
# environment a societuy occupies).
#
# 7 Jun 2016
# Carlos A. Botero, Bruno Vilela & Ty Tuff
# Washington University in Saint Louis
#=================================================================
RunSim.push <- function(myWorld, P.extinction, P.speciation,
                    P.diffusion, P.Arisal, P.TakeOver, nbs, independent,
                    N.steps, multiplier, start) {
  # myWorld = The hexagonal world created with the function BuildWorld
  # P.extinction = Probability matrix of extinction
  # P.speciation = Probability matrix of speciation
  # P.diffusion = Probability matrix of diffusion
  # P.Arisal = Probability matrix of arisal
  # P.TakeOver = Probability matrix of takeover
  # N.steps = Number of steps in the model
  # multiplier = The number that will multiply the probabilities according
  # to environmetal fitness.
  # start = the point ID in 'myWorld' that will give risen to humans.
  # (humans origin will be in one of the existing positions)

  world.size <- nrow(myWorld)
  # Initialize parameters we will use later to build the phylogeny
  rootnode <-  world.size + 1 # standard convention for root node number
```

```
  # set the seed for simulation
  if (is.null(start)) {
    start <- sample(1:world.size, 1)
  }

  myWorld[start, 4:6] <- c(0, 0, 1) # Setting root(0), time(0), ancestral(1, forager)

  mytree <- TheOriginOfSpecies(world.size, start) # Empty tree
  myT <- 0 # Time starts at zero

  # Common input and output for all the internal modules
  input <- list(P.speciation, P.Arisal, P.diffusion, P.extinction, P.TakeOver,
                myWorld, mytree, myT, multiplier, nbs, independent)

  # Functions order to be randomized
  rand_order_func_run <- list("Extinction", "Diffusion",
                              "SpeciationTakeOver.push", "Arisal")

  cat("0% [") # Time count

  for (steps in 1:N.steps) { # Starts the loop with 'n' steps

    if (steps %% round((N.steps / 10)) == 0) { # Time count
      cat('-') # Time count
    }# Time count
    if (steps == N.steps) { # Time count
      cat("] 100 %\n")# Time count
    }# Time count

    # Randomize functions order
    rand_order <- sample(rand_order_func_run)
    # Run the functions
    input <- do.call(rand_order[[1]], list(input = input))
    input <- do.call(rand_order[[2]], list(input = input))
    input <- do.call(rand_order[[3]], list(input = input))
    input <- do.call(rand_order[[4]], list(input = input))

  }
  # Trunsform the input/output into the final result and return it
  myWorld <- as.data.frame(input[[6]])
  myWorld[, 8] <- paste0("t", myWorld[, 8])
  mytree <- makePhy(input[[7]])
  mytree$edge.length <- mytree$edge.length / N.steps
  return(list('mytree' = mytree, 'myWorld' = myWorld))
}
```

4.3.0.1 The second set of RunSim functions save an output each timestep if we want to look at trends through time. We use this to make videos of the simulation running.

```
RunSimUltimate2 <- function (myWorld, P.extinction, P.speciation, P.diffusion, P.Arisal,
    P.TakeOver, nbs, independent, N.steps, multiplier, silent = TRUE,
    count, resolution = seq(1, N.steps, 100), P.Arisal0, start = NULL)
```

```
{
    result <- try(RunSim2(myWorld, P.extinction, P.speciation,
        P.diffusion, P.Arisal, P.TakeOver, nbs, independent,
        N.steps, multiplier, count = count, resolution = resolution,
        P.Arisal0 = P.Arisal0, start), silent = silent)
    if (class(result) == "try-error") {
        result <- NA
    }
    return(result)
}
```

```
RunSim2 <- function (myWorld, P.extinction, P.speciation, P.diffusion, P.Arisal,
    P.TakeOver, nbs, independent, N.steps, multiplier, count,
    resolution, P.Arisal0, start = NULL)
{
    folder <- paste0("./Module_1_outputs/myOut_rep_", formatC(count,
        width = 2, flag = 0), "_combo_", formatC(count, width = 2,
        flag = 0), "_", "params", "_P.speciation_", paste(formatC(P.speciation,
        width = 2, flag = 0), collapse = "_"), "_P.extinction_",
        paste(formatC(P.extinction, width = 2, flag = 0), collapse = "_"),
        "_P.diffusion_", paste(formatC(P.diffusion, width = 2,
            flag = 0), collapse = "_"), "_P.TO_", paste(formatC(P.TakeOver,
            width = 2, flag = 0), collapse = "_"), "_P.Arisal_",
        paste(formatC(P.Arisal0, width = 2, flag = 0), collapse = "_"),
        "_timesteps_", N.steps)
    world.size <- nrow(myWorld)
    rootnode <- world.size + 1
    if (is.null(start)) {
        start <- sample(1:world.size, 1)
    }
    myWorld[start, 4:6] <- c(0, 0, 1)
    mytree <- TheOriginOfSpecies(world.size, start)
    myT <- 0
    input <- list(P.speciation, P.Arisal, P.diffusion, P.extinction,
        P.TakeOver, myWorld, mytree, myT, multiplier, nbs, independent)
    rand_order_func_run <- list("Extinction", "Diffusion", "SpeciationTakeOver",
        "Arisal")
    cat("0% [")
    for (steps in 1:N.steps) {
        if (steps%%round((N.steps/10)) == 0) {
            cat("-")
        }
        if (steps == N.steps) {
            cat("] 100 %\n")
        }
        rand_order <- sample(rand_order_func_run)
        input <- do.call(rand_order[[1]], list(input = input))
        input <- do.call(rand_order[[2]], list(input = input))
        input <- do.call(rand_order[[3]], list(input = input))
        input <- do.call(rand_order[[4]], list(input = input))
        if (steps %in% resolution) {
            myWorld <- as.data.frame(input[[6]])
            myWorld[, 8] <- paste0("t", myWorld[, 8])
            if (nrow(na.omit(input[[7]])) > 1) {
```

```
                mytree <- makePhy(input[[7]])
            }
            else {
                mytree <- NA
            }
            myOut <- list(mytree = mytree, myWorld = myWorld)
            save(myOut, file = paste0(folder, "_", formatC(steps,
                10, flag = 0), ".Rdata"))
            stats <- Module_2(myOut)
            save(stats, file = paste0(folder, "_", formatC(steps,
                10, flag = 0), "_stats", ".Rdata"))
        }
    }
    myWorld <- as.data.frame(input[[6]])
    myWorld[, 8] <- paste0("t", myWorld[, 8])
    mytree <- makePhy(input[[7]])
    mytree$edge.length <- mytree$edge.length/N.steps
    return(list(mytree = mytree, myWorld = myWorld))
}
```

#..And the push version of saving each time step

```
# Run the simulation function skiping the erros and atributing NA if it occurs
RunSimUltimate2.push <- function(myWorld, P.extinction, P.speciation,
                            P.diffusion, P.Arisal, P.TakeOver, nbs, independent,
                            N.steps, multiplier,
                            silent = TRUE, count, resolution = seq(1, N.steps, 100),
                            P.Arisal0, start = NULL) {

  result <- try(RunSim2.push(myWorld, P.extinction, P.speciation,
                        P.diffusion, P.Arisal, P.TakeOver, nbs,
                        independent, N.steps,
                        multiplier, count = count, resolution = resolution,
                        P.Arisal0 = P.Arisal0, start),
                silent = silent)
  if (class(result) == "try-error") {
    result <- NA
  }
  return(result)
}
```

```
#=================================================================
RunSim2.push <- function(myWorld, P.extinction, P.speciation,
                    P.diffusion, P.Arisal, P.TakeOver, nbs, independent,
                    N.steps, multiplier, count, resolution, P.Arisal0,
                    start = NULL) {
  # myWorld = The hexagonal world created with the function BuildWorld
  # P.extinction = Probability matrix of extinction
  # P.speciation = Probability matrix of speciation
  # P.diffusion = Probability matrix of diffusion
  # P.Arisal = Probability matrix of arisal
  # P.TakeOver = Probability matrix of takeover
  # N.steps = Number of steps in the model
  # multiplier = The number that will multiply the probabilities according
  # to environmetal fitness.
```

```
# start = the point ID in 'myWorld' that will give risen to humans.
# (humans origin will be in one of the existing positions)
folder <- paste0("./Module_1_outputs/myOut_rep_",
                 formatC(count, width = 2,flag = 0),
                 "_combo_",
                 formatC(count, width = 2,flag = 0),
                 "_","params", "_P.speciation_",
                 paste(formatC(P.speciation, width = 2,flag = 0),
                       collapse="_"),"_P.extinction_",
                 paste(formatC(P.extinction, width = 2,flag = 0),
                       collapse="_"), "_P.diffusion_",
                 paste(formatC(P.diffusion, width = 2,flag = 0),
                       collapse="_"), "_P.TO_",
                 paste(formatC(P.TakeOver, width = 2,flag = 0),
                       collapse="_"),"_P.Arisal_",
                 paste(formatC(P.Arisal0, width = 2,flag = 0),
                       collapse="_"), "_timesteps_",
                 N.steps)
world.size <- nrow(myWorld)
# Initialize parameters we will use later to build the phylogeny
rootnode <-  world.size + 1 # standard convention for root node number

# set the seed for simulation
if (is.null(start)) {
  start <- sample(1:world.size, 1)
}

myWorld[start, 4:6] <- c(0, 0, 1) # Setting root(0), time(0), ancestral(1, forager)

mytree <- TheOriginOfSpecies(world.size, start) # Empty tree
myT <- 0 # Time starts at zero

# Common input and output for all the internal modules
input <- list(P.speciation, P.Arisal, P.diffusion, P.extinction, P.TakeOver,
              myWorld, mytree, myT, multiplier, nbs, independent)

# Functions order to be randomized
rand_order_func_run <- list("Extinction", "Diffusion",
                            "SpeciationTakeOver.push", "Arisal")

cat("0% [") # Time count

for (steps in 1:N.steps) { # Starts the loop with 'n' steps

  if (steps %% round((N.steps / 10)) == 0) { # Time count
    cat('-') # Time count
  }# Time count
  if (steps == N.steps) { # Time count
    cat("] 100 %\n")# Time count
  }# Time count

  # Randomize functions order
  rand_order <- sample(rand_order_func_run)
```

```
    # Run the functions
    input <- do.call(rand_order[[1]], list(input = input))
    input <- do.call(rand_order[[2]], list(input = input))
    input <- do.call(rand_order[[3]], list(input = input))
    input <- do.call(rand_order[[4]], list(input = input))
    # Save
    if(steps %in% resolution) {
      myWorld <- as.data.frame(input[[6]])
      myWorld[, 8] <- paste0("t", myWorld[, 8])
      if(nrow(na.omit(input[[7]])) > 1) {
        mytree <- makePhy(input[[7]])
      } else {
        mytree <- NA
      }
      myOut <- list('mytree' = mytree, 'myWorld' = myWorld)
      save(myOut, file= paste0(folder,"_", formatC(steps, 10, flag = 0), ".Rdata"))
      stats <- Module_2(myOut)
      save(stats, file= paste0(folder,"_", formatC(steps, 10, flag = 0),
                               "_stats", ".Rdata"))
    }
  }
  # Trunsform the input/output into the final result and return it
  myWorld <- as.data.frame(input[[6]])
  myWorld[, 8] <- paste0("t", myWorld[, 8])
  mytree <- makePhy(input[[7]])
  mytree$edge.length <- mytree$edge.length / N.steps
  return(list('mytree' = mytree, 'myWorld' = myWorld))
}
```

Chapter 5

Module 2: Space and phylogeny summary statistics

This is the master document for Module 2, a foundational function in our FARM package that analyzes results from Module 1, the other foundational function. Module 1 simulates a spatial pattern and a phylogenetic tree given a set of environmental and inheritance rules and then Module 2 summarizes those simulated results using a large set of targeted summary statistics. Here we describe our choice of summary statistics, justify those choices as part of a larger theoretical context, and provide our reproducable code for executing the anaylses yourself. These two parts are seperated into modules so that they can act independently. An combination of spatial pattern and associated phylogeny many be used as long as they are formatted correctly.

This pipeline was designed to analyze a simulated world where all the information is known about both the world and the tree. There is no missing information, just extinct trees. This is much different than our real tree that has loads of uncertainty unevenly destributed across it. The result you see demonstrated right now are one simulated result of many. I need to do a sister page to this were we do this entire analysis on the real tree, or best real tree we've got.

We have four types of data available for asking research questions using D-place data: phylogenies, spatial locations, trait identities, and environmental reconstructions. Any one of these four data types alone are relatively information poor, so we are searching for ways to model connections between these data types to draw stronger conclusions overall.

Other modules can use the summary statistics generated from this module to test hypotheses. We currently have a ABC and Random Forest module started but there will be more to come.

These are quantitative connections that we are assumed in the analyses, but we don't actually have any support in the data for doing so. 1. nearest neighbor connectivity measures 2. Abundance estimates 3. Pairwise influence (history) between cultures. 4. Environmental reconstruction validation evidence

5.1 Phylogenetic summary statistics

Whole tree vs. part of tree? These statistics are generally used to compare one sample to another. For example, an experimental contrast between two sites, two phylogenetic groups, or two communities in two different locations. Here we are calculating these statistics for the global langauage tree to compare against global trees created in our simulation. You still retain the ability to subset this tree or others and send only those subsets through this code to compare the values with each other afterwards.

- Introduction and framework

- Alpha Diversity metrics

1. Branch Length (richness and divergence)
2. Pairwise distance between tips (richness, divergence, and regularity)
3. Phylogenetic isolation (divergence, and regularity)
4. Nearest Neighbor (divergence, and regularity)

- Beta Diversity
- Tree topology
- Macroevolutionary rates

All trees are ultrametric.

5.1.1 Introduction and framework

The choice of phylogenetic analyses and organizational scheme is based on the suggestions of **?**. Here are a few images from that paper for an overview:

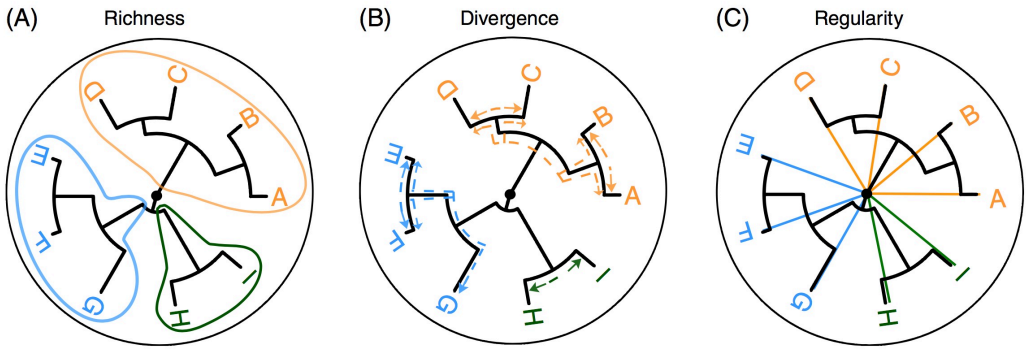


Fig. 1. Conceptual diagrams illustrating the calculation of each of the three dimensions of phylogenetic information: richness (A), divergence (B), and regularity (C). The branching diagram in each image is a phylogenetic tree representing the inferred evolutionary relationships among taxa A–I. Taxa A–I are grouped within three assemblages (A–D, orange; E–G, blue; and H–I, green). Tree branches represent accumulated differences between taxa.

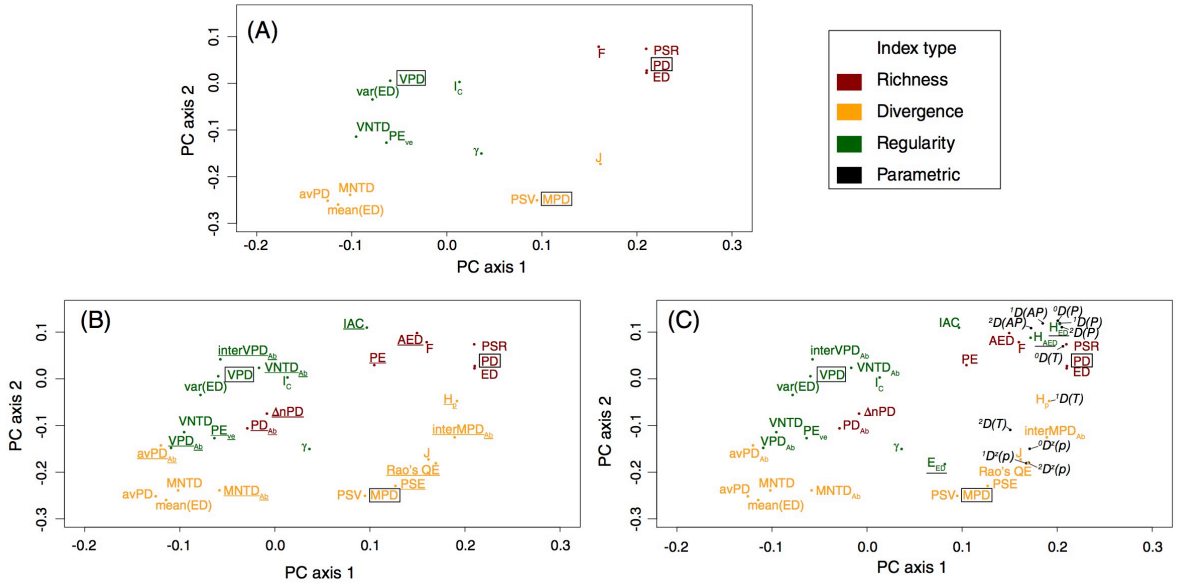


Fig. 2. Principal components analysis for Spearman's correlations between the α-diversity metrics shown in Table 1. Results represent measures taken from 800 simulated landscapes, based on 100 simulated phylogenetic trees and eight landscape types defined in Table 2 (see online Appendix S2 for detailed methods). (A) All metrics excluding abundance-weighted metrics and those classified as parametric indices. (B) As in A, but with abundance-weighted metrics included (underlined). (C) As in B, but with parametric indices (black), and indices that incorporate multiple dimensions (underlined) included (e.g. all α-diversity metrics). X and Y axes are scaled to reflect explained variance (PC1 = 41.8%; PC2 = 20.5% for the PCA performed with all metrics, shown in (C). Boxed metrics reflect 'anchor' metrics (PD, MPD and VPD) that align most closely with the richness, divergence and regularity dimensions, respectively. Where metrics are identified in Table 1 as mathematically identical, we include only one (e.g. MPD is plotted, but not AvTD). See Appendix S1 for equalities among indices.

```
library(knitr)
library(phytools)
library(FARM)
library(ROCR)
library(spdep)
```

```
load('~/Downloads/download.Rdata')

this_tree <- myOut$mytree
this_world <- myOut$myWorld
```

Dimension	**Richness:** *How much?*	**Divergence:** *How different?*	**Regularity:** *How regular?*
Definition	How much evolutionary history is associated with a set of tips?	How closely related (on average) are tips within a set?	How evenly is evolutionary history distributed between tips within a set?
Interpretation	*What is the total evolutionary history within (or between) assemblages?*	*How similar are species within (or between) assemblages?*	*How is similarity distributed within (or between) assemblages?*
Example Questions	**(i) Evolutionary diversity as dependent variable**		
	Is evolutionary history correlated with resource availability? How does compositional similarity decay with geographic distance? Do current protected areas adequately protect evolutionary history?	Is environmental filtering more important in high-elevation communities compared to low-elevation communities? Do geologically unstable regions have more recently diverged species? Where are concentrations of endemic evolutionary diversity?	Does biotic competition drive community structure? What is the relationship between evolutionary history and variation in function?
	(ii) Evolutionary diversity as independent variable		
	Are more diverse communities more productive? Does evolutionary diversity predict the spatial distribution of diversity? Do more diverse communities supply greater ecosystem services?	Are distinct species more invasive relative to community composition? Do island radiations influence bird co-occurence patterns?	Do patterns of niche space occupancy affect community stability? Does a community with evenly distributed evolutionary history have greater potential to adapt to future habitats?

Figure 5.1: From **?**

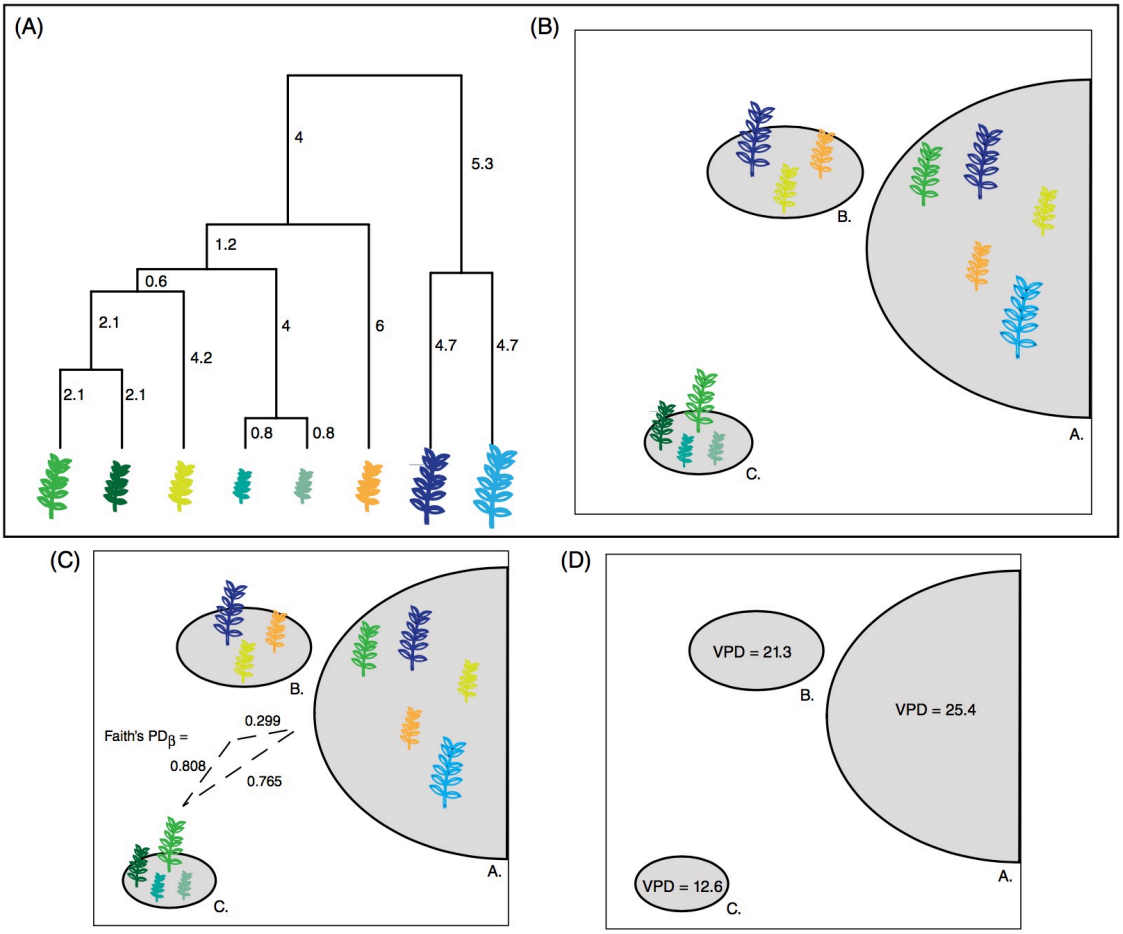


Fig. 6. Example involving the flora of an hypothetical island mainland system (top panel), for which a researcher wishes to choose appropriate phylo-diversity metrics to test how evolutionary diversity varies among the mainland and island sites. See Section V.2. Panel (A) and (B) shows the distribution of species among the sites, and their phylogenetic relatedness. Values on the phylogeny represent hypothetical distances between species (e.g. branch lengths, etc.). Panel (C) shows how evolutionary history is shared between sites (richness metric, β-diversity, Faith's PD_β). Panel (D) shows how evenly evolutionary history is distributed at each site (regularity metric, α-diversity VPD).

Figure 5.2: From **?**

```
str(this_world)
```

```
## 'data.frame':    1253 obs. of  8 variables:
##  $ cellID     : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ Longitude  : num  -60 12 28 -124 -63 ...
##  $ Latitude   : num  -25 10 -29 54 4 ...
##  $ Parent     : num  828 NA 216 616 901 NA NA NA NA NA ...
##  $ BirthT     : num  37.8 NA 37.6 31 36.2 ...
##  $ Trait      : num  1 NA 1 1 1 NA NA NA NA NA ...
##  $ Environment: num  2 2 2 1 2 1 2 1 1 1 ...
##  $ TipLabel   : chr  "t1" "t2" "t3" "t4" ...
```

```
str(this_tree)
```

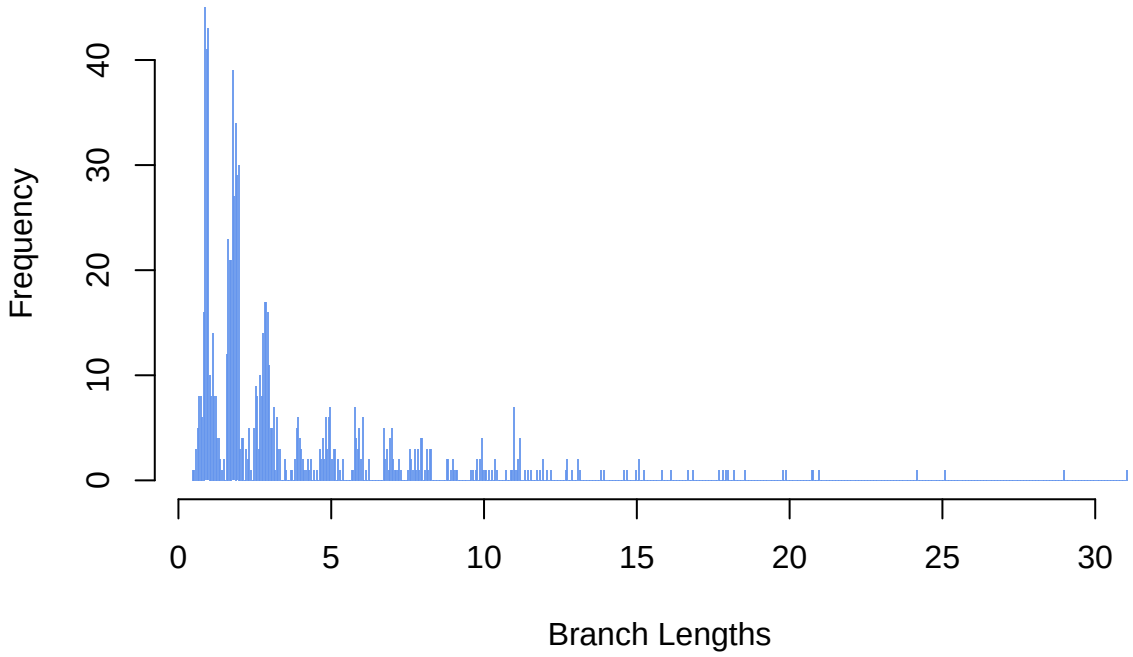
```
## List of 4
##  $ edge       : num [1:910, 1:2] 457 460 463 560 892 892 560 463 466 466 ...
##  $ tip.label  : chr [1:456] "t862" "t409" "t260" "t204" ...
##  $ edge.length: num [1:910] 2.667 4.433 20.973 10.011 0.916 ...
##  $ Nnode      : num 455
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

5.1.2 Alpha diversity metrics

5.1.2.1 Branch Lengths

Branch length data is embedded in the tree object provided to this function. The first step in summarizing the lengths is to extract those data from the tree object. These data are called 'edges' in the tree object. We extract branch lengths and create an object called 'Branch_lengths' for passing on to the other summary functions. The histogram below shows the frequency of different branch lengths found throughout the tree.

```
Branch_Lengths <- this_tree$edge.length
```



We can summarize branch lengths according to normal summary statistics, but it can be difficult to assign

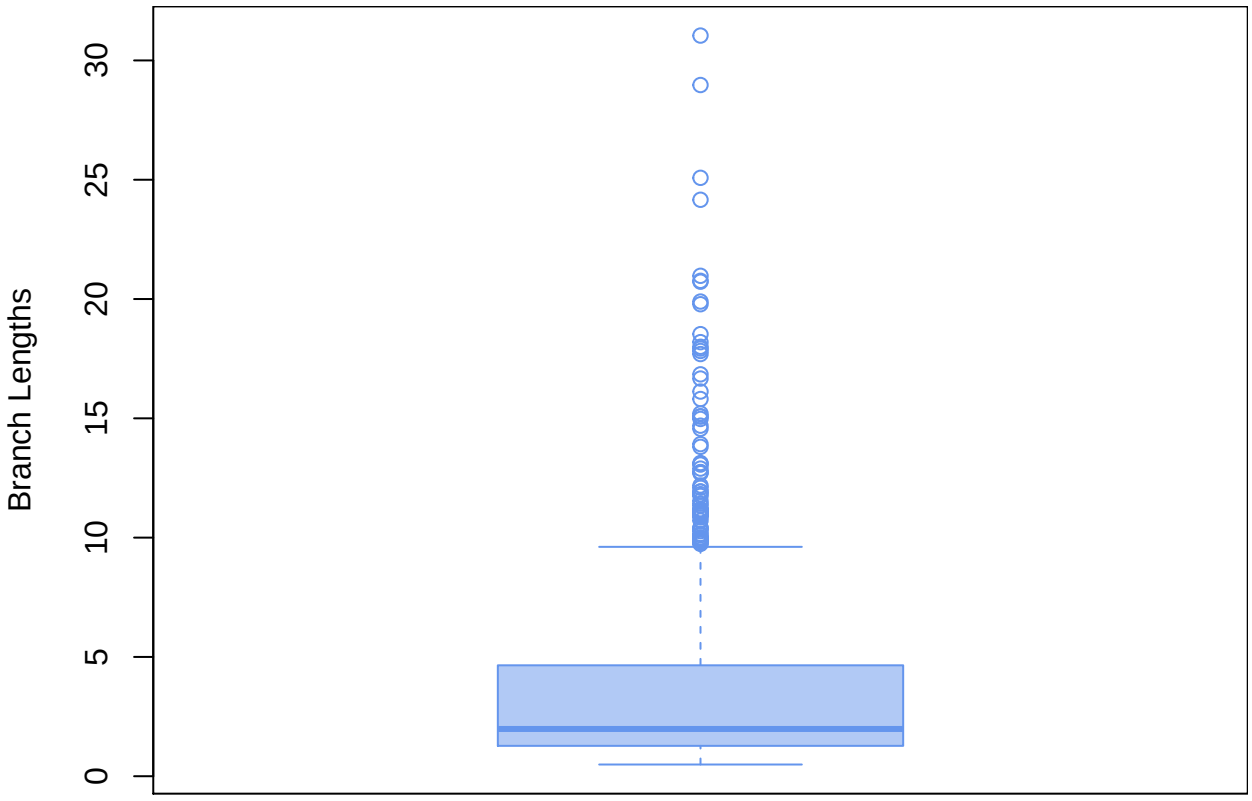
evolutionary meaning to some of these metrics and so they are not regularly used as best I can tell. This lack of meaning does not mean that these statistics couldn't be used to distinguish between large simulated trees.

```
mean_branch_length <- mean(Branch_Lengths)
variance_branch_length <- var(Branch_Lengths)
SD_branch_length <- sd(Branch_Lengths)
```

```
## [1] "mean branch length =  3.64753764473253"
```

```
## [1] "variance in branch lengths =  14.9947769344804"
```

```
## [1] "standard deviation in branch lengths =  3.87230899263998"
```



5.1.2.2 Phylogenetic diversity (PD)

Phylogenetic diversity (PD) is the summation ($\sum$) of all branch lengths connecting species together, where B_t is the set of included tips and L_b is Branch lengths (**?**). This is an anchor test, which means it is regularly used, well understood, and we should use it to anchor our work to past work. PD is a richness measure, it tells us how much evolutionary history is associated with a set of tips.

$$PD = \sum_{b \in B_t} L_b$$

```
# Anchor test = PD (Faith's phylogenetic diversity)
Pylo_diversity_is_sum_of_BL <- sum(Branch_Lengths)
Pylo_diversity_is_sum_of_BL
```

```
## [1] 3319.259
```

There are variations on this measure that we have NOT implemented here. It is popular to scale this measure according to some ecological driver. **?** scales branch lengths (L_b) by multiplying them against the abundance of individuals at at tip (A_b). Others (**?**), scale them by their range size instead (R_b).

$$\Delta nPD = \sum_{b \in B_t} A_b L_b$$

$$PE = \sum_{b \in B_t} \frac{L_b}{R_b}$$

Argueing that proportional abundance phylogenetic diversity (PD_{Ab}) is more effective than the standard PD calculated from raw abundance, **?** penned a new version of PD where B is the total number of branch lengths (L_b). Note: We don't have abundance data right now for the human project so this metric is not currently very helpful.

$$PD_{Ab} = B * \frac{\sum_{b \in B_t} A_b L_b}{\sum_{b \in B_t} A_b}$$

```
#Calculate B
number_of_branches <- length(Branch_Lengths)
number_of_branches
```

```
## [1] 910
```

5.1.2.3 Average phylogenetic diversity ($avPD$)

Average phylogenetic diversity ($avPD$) (**?**) is a branch length-based divergence indices where PD is divided by the total number of tips (S) in the tree.

$$avPD = \frac{PD}{S}$$

```
Number_of_tips <- length(this_tree$tip.label)
average_phylogenetic_diversity <- Pylo_diversity_is_sum_of_BL / Number_of_tips
average_phylogenetic_diversity
```

```
## [1] 7.279077
```

There is also a proportional abundance version of average phylogenetic diversity ($avPD_{Ab}$) (**?**). Again, we don't have abundance values yet for D-place.

$$avPD_{Ab} = \frac{B * \frac{\sum_{b \in B_t} A_b L_b}{\sum_{b \in B_t} A_b}}{S}$$

5.1.3 Pairwise distance between tips

This is the patristic distance, the sum of the branch lengths following the shortest distance between two tips in a tree, implemented as a distance matrix where every tip is compared to every other tip. This distance function can be anything. We use euclidean and environmental distance matrices heavily in the spatial analyses.

5.1.3.1 Calculate the patristic distance between two taxa, for all taxa

Calculate the patristic distance between two taxa using the R package 'phytools', this function takes a 'phylo' tree object and returns a distance matrix between tips. Need original citation.

```
## Pairwise distance between tips - From library(ape) in library(phytools)
Pairwise_dist <- cophenetic(this_tree)
```

yields a distance matrix (list of 2D matrices) of all distances between taxa.

```
##  num [1:456, 1:456] 0 57.9 78 57.9 72.7 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:456] "t862" "t409" "t260" "t204" ...
##   ..$ : chr [1:456] "t862" "t409" "t260" "t204" ...
```

5.1.3.2 Sum of all pairwise distances (F)

Now we can use a set of summary statistics to describe those pairwise distances. The sum of all pairwise distances, F, is formally called 'Extensive quadratic entropy'. (**?**). Just as it was with branch lengths, this is a richness measure and, accordingly, should be used to answer richness questions.

$$F = \sum_i \sum_j d_{ij}$$

```
# F -- Extensive quadratic entropy
F_quadratic_entropy_is_sum_of_PD <- sum(Pairwise_dist)
F_quadratic_entropy_is_sum_of_PD
```

```
## [1] 14047351
```

5.1.3.3 Mean pairwise distance (MPD)

Mean inter-species distances. The mean of all pairwise distances, MPD (a.k.a. $AvTD$, and Δ^+), is the mean distance between species. (**????**).

$$MPD = \frac{\sum_{ij} d_{ij}}{S(S-1)}$$

```
# Anchor test = MPD (mean pairwise distance)
Mean_pairwise_distance <-
  Pairwise_dist / (Number_of_tips * (Number_of_tips - 1) )
```

```
##  num [1:456, 1:456] 0 0.000279 0.000376 0.000279 0.00035 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:456] "t862" "t409" "t260" "t204" ...
##   ..$ : chr [1:456] "t862" "t409" "t260" "t204" ...
```

5.1.3.4 MPD anchored to the root

There is an extention to mean pairwise distance calculations from **?** called PSV, PSR, and PSE, phylogenetic species variability, phylogenetic species richness, and phylogenetic species evenness. These measures take the basic pairwise distance calculations and anchor them to the root of the tree so distances have a common denominator. This extention is implemented by using the same equations, just with a constrained set of d_{ij} conditions. Specifically,

$$PSV = MPD = \frac{\sum_{ij} d_{ij}}{S(S-1)}$$

$$PSR = \sum_i (\frac{1}{S-1} \sum_j d_{ij})$$

$$PSE = \frac{S}{S-1} \sum_{ij} d_{ij} p_i p_j$$

with these specific values of d_{ij}

$$d_{ji} = 0.5 * (c_{ii} + c_{jj} - c_{ij}) or d_{ij} = 1 - c_{ij}/(\sqrt{c_{ii} c_{jj}})$$

and

$c_{ii} = the\ sum\ of\ branch\ lengths\ from\ tip\ i\ to\ the\ root\ of\ the\ phylogenetic\ tree. c_{ij} = the\ sum\ of\ branch\ lengths\ from\ first\ c$

5.1.3.5 Average distance between two randomly chosen species

J, Intensive quadratic entropy, which is the average distance between two randomly chosen species (**?**)

$$J = \frac{\sum_{ij} d_{ij}}{S^2}$$

5.1.3.6 Simpson's diversity index for pairwise distance

There has been a long effort to pen a phylogenetic analogy to a Simpson's diversity index. (**???????**). The conclusion seems to be that this measure is equivilent to scaling MPD by abundance p_i and p_j to get MPD_{Ab}. This is also a special case of Rao's Quadratic Entropy, $Roe's QE$. Note: not using abundance measures yet for D-place data.

$$MPD_{Ab} = \sum_i \sum_j d_{ij} p_i p_j$$

5.1.3.7 Interspecific comparisons of pairwise distances

The interspecific variant (rather than the intraspecific default described above) defines the expected phylogenetic distance between two indivdiuals randomely drawn conditionally on the fact that they indivdulas from different species.

$$InterMPD_{Ab} = \frac{\sum_i \sum_{j \neq i} d_{ij} p_i p_j}{\sum_i \sum_{j \neq i} d_{ij} p_i p_j}$$

5.1.3.8 Variance in pairwise distances (VPD)

Variance in pairwise distances, VPD (a.k.a. $VarTD$ and Λ^+), is a regularity indices. **?** Variance is relative to tips, S, not to total branches (B from above). These are the residuals, they compare each individual pairwise connection to the overall mean.

$$VPD = \frac{1}{S(S-1)} (\sum_i \sum_{j \neq i} (d_{ij} - MPD)^2)$$

```
#need to adjust to equation above!

#Pairwise distance/all distances -- Variance of pairwise distances

# Anchor test = VPD (variation of pairwise distance)

variance_pairwise_distance <- var(as.vector(Pairwise_dist))
```

Variants of VPD are VPD_{ab} and $InterPVD_{Ab}$, where variance is scaled by abundance or compared in and out of species. These are also regularity indices.

$$VPD_{Ab} = (\sum_i \sum_j n_i n_j) * \frac{\sum_i \sum_j n_i n_j (d_{ij} - MPD_{Ab})^2}{(\sum_i \sum_j n_i n_j)^2 - \sum_i \sum_j (n_i n_j)^2} or InterVPD_{Ab} = (\sum_i \sum_{j \neq i} n_i n_j) * \frac{\sum_i \sum_{j \neq i} n_i n_j (d_{ij} - InterMPD_{Ab})^2}{(\sum_i \sum_{j \neq i} n_i n_j)^2 - \sum_i \sum_{j \neq i} (n_i n_j)^2}$$

5.1.4 Nearest phylogenetic neighbor

5.1.4.1 Divergence indices

Divergence indices using nearest distance: $MNTD$ and $MNTD_{Ab}$, Mean nearest taxon distance and Abundance-weighted MNTD (**???**).

$MNTD$, mean nearest taxon distance, is the mean shortest distance from a species to all other in the assemblage (**???**).

$$MNTD = \frac{1}{S} \sum_i d_{i_{min}}$$

$MNTD_{AB}$, abundance adjusted mean nearest taxon distance. Adjusted by species proportions (i.e. species' relative abundances) (**???**)

$$MNTD_{Ab} = \sum_{i=1}^{S} [d_{i_{min}} * p_i]$$

5.1.4.2 Regularity indices

Regularity indices using nearest distances: $VNTD$, $VNTD_{Ab}$, PE_{ev}.

$VNTD$, Variance in nearest taxon distances, is the variance in nearest pairwise distance (**?**).

$$VNTD = \frac{1}{S} \sum_{i-1}^{S} [(d_{i_{min}} - MNTD)^2]$$

$VNTD_{Ab}$, Abundance weighted variance in nearest taxon distances, is scales by abundance in the same way as descried above (**?**).

$$VNTD_{Ab} = \frac{(\sum_i n_i) \sum_i n_i (d_{i_{min}} - MNTD_{Ab})^2}{(\sum_i n_i)^2 - \sum_i n_i^2}$$

5.1.4.3 Phylogenetic version of the funtional FE_{ve} index

PE_{ve}, phylogenetic evenness is a phylogenetic version of the funtional FE_{ve} index. First a minimum spanning tree (MST) is computed using the cophenetic distance obtained from the phylogenetic tree. The MST contains $S-1$ Branches connection the S species. We denote l a branch on the MST, $dist(i,j)$ is the length the branch l that connects species i and j. n_i is, as defined above, the abundance of species i in the asseblage (**??**).

$$Weighted\,evenness: EW_i = \frac{dist(i,j)}{(n_i+n_j)/(\sum_{k=1}^{S} n_k)} Partial\,weighted\,evenness: PEW_l = \frac{EW_l}{\sum_{l=1}^{S-1} EW_l} Phylogenetic\,evenness$$

5.1.5 Phylogenetic isolation

A phylogenetic isolation index represents the relative isolation of a given species within a phylogenetic tree. Several indices have been proposed so far but we focus here on the evolutionary distinctiveness index called 'Fair Proportion' as proposed by **?** and **?**.

5.1.5.1 Evolutionary distinctiveness (richness indices)

ED, evolutionary distinctiveness is a richness indices. NOTE: not equal to Faith's PD because the ED_i are computed from the regional pool of species and sumed across a given assemblage (i.e. a subset of the regional species pool) (**????**).

$$ED = \sum_i ED_i where\ ED_i = \sum_{b \in B_{t_i}} \frac{L_b}{S_b}$$

AED, Abundance-weighted ED (**??**).

$$\sum_i AED_i where\ AED_i = \sum_{b \in B_{t_i}} \frac{L_b}{A_b} * p_i$$

```
# Bruno's function for ED. Provided in library(FARM)

evol.distinct2 <- function (tree, type = c("equal.splits", "fair.proportion"),
    scale = FALSE, use.branch.lengths = TRUE)
{
    type <- match.arg(type)
    if (is.rooted(tree) == FALSE)
        warning("A rooted phylogeny is required for meaningful output of this function",
            call. = FALSE)
    if (scale == TRUE) {
        if (is.ultrametric(tree) == TRUE)
            tree$edge.length <- tree$edge.length/(as.numeric(branching.times(tree)[1]))
        else tree$edge.length <- tree$edge.length/sum(tree$edge.length)
    }
    if (use.branch.lengths == FALSE)
        tree$edge.length <- rep(1, length(tree$edge.length))
    for (i in 1:length(tree$tip.label)) {
        spp <- tree$tip.label[i]
```

```
        nodes <- .get.nodes(tree, spp)
        nodes <- nodes[1:(length(nodes) - 1)]
        internal.brlen <- tree$edge.length[which(tree$edge[,
            2] %in% nodes)]
        if (length(internal.brlen) != 0) {
            internal.brlen <- internal.brlen * switch(type, equal.splits = sort(rep(0.5,
                length(internal.brlen))^c(1:length(internal.brlen))),
                fair.proportion = {
                  for (j in 1:length(nodes)) {
                    sons <- .node.desc(tree, nodes[j])
                    n.descendents <- length(sons$tips)
                    if (j == 1) portion <- n.descendents else portion <- c(n.descendents,
                      portion)
                  }
                  1/portion
                })
        }
        ED <- sum(internal.brlen, tree$edge.length[which.edge(tree,
            spp)])
        if (i == 1)
            w <- ED
        else w <- c(w, ED)
    }
    return(w)
}
```

Evolutionary distinctiveness is our basic measure of phylogenetic isolation. #This should likely be 'fair proportions' instead of 'equal.splits'.

```
# Calculate ED
# Using equal.splits method, faster computation
# Evolutionary_distinctiveness_i <- evol.distinct2(this_tree, type = "equal.splits")

# ED - Summed evolutionary distinctiveness
# Evolutionary_distinctiveness_sum <- sum(Evolutionary_distinctiveness_i)
```

```
#Evolutionary_distinctiveness_sum
```

We can run some standard summary statistics (mean and variance) on this ED measure. var(Ed) shows up close to VPD on the PCAs in the intro (**?**).

```
# mean(ED)
# mean_Phylogenetic_isolation <- mean(Evolutionary_distinctiveness_i)

# var(ED)
#variance_Phylogenetic_isolation <- var(Evolutionary_distinctiveness_i)
```

```
#mean_Phylogenetic_isolation
#variance_Phylogenetic_isolation
```

5.1.5.2 Mean evolutionary distinctiveness (divergence indices)

The divergence indices version for ED is mean evolutionary distinctiveness, MED. The mean of evolutionary distinctiveness (**??**).

$$MED = \frac{\sum_i ED_i}{S} with ED_i = \sum_{b \in B_{t_i}} \frac{L_b}{S_b}$$

Entropy measure of evolutonary distinctiveness (regularity indices) The regularity indices for ED/phylogenetic isolation are H_{ED}, E_{ED}, $var(ED)$, H_{AED}

H_{ED}, Entropy measure of evolutionary distinctiveness, is the shannon index applied to evolutionary distinctiveness values (**?**).

$$H_{ED} = -\sum_{i=1}^{S}((\frac{ED_i}{\sum_{i=1}^{S} ED_i}) * \ln(\frac{ED_i}{\sum_{i=1}^{S} ED_i}))$$

E_{ED}, Equitability of evolutionary distinctiveness, is H_{ED} controlled for species richness (**?**).

$$E_{ED} = \frac{H_{ED}}{\ln(S)}$$

$var(ED)$, Variance in evolutinoary distinctiveness, is the variance of species evolutionary distinctiveness (**?**).

$$var(ED) = \frac{1}{S-1} * \sum_{i=1}^{S}(ED_i - \frac{\sum_{i=1}^{S} ED_i}{S})^2$$

$H_{ED_{Ab}}$, Abundance-weighted version of H_{ED} (**?**).

$$H_{ED_{Ab}} = -\sum_{i=1}^{S}(\frac{n_i AED_i}{\sum_{i=1}^{S} n_i AED_i} * \ln(\frac{n_i AED_i}{\sum_{i=1}^{S} n_i AED_i}))$$

5.2 Beta diversity

We currently are not using any beta diversity metrics but there are many to choose from if we decide to add them later.

5.3 Tree topology

Tree topology is a measure of the shape of the overall tree. The tree can be lopsided side-to-side or front-to-back.

Our most trusted index for the tippy vs trunky of a tree is the gamma index, γ.The index characterizes the distribution of branching events within the tree. Trees with $\gamma < 0$ have relatively longer branches towards the tips of the phylogeny (tippy trees), whereas trees with $\gamma > 0$ have relatively longer inter-nodal distances towards the root of the phylogeny (stemmy trees). tk represents an 'evolutionary period' (limits are given by two speciation events) or equivalently an internode distance (**?**).

$$\gamma = \frac{(\frac{1}{S-2} * \sum_{i=2}^{S-1}(\sum_{k=2}^{i} Kt_k)) - \frac{1}{2} * \sum_{j=2}^{S} jt_j}{(\sum_{j=2}^{S} jt_j) * \sqrt{\frac{1}{12 * (S-2)}}}$$

```
# ltt function from library(phytools)
ltt <- function (tree, plot = TRUE, drop.extinct = FALSE, log.lineages = TRUE,
    gamma = TRUE, ...)
{
    tol <- 1e-06
    if (!inherits(tree, "phylo") && !inherits(tree, "multiPhylo"))
        stop("tree must be object of class \"phylo\" or \"multiPhylo\".")
    if (inherits(tree, "multiPhylo")) {
        obj <- lapply(tree, ltt, plot = FALSE, drop.extinct = drop.extinct,
            log.lineages = log.lineages, gamma = gamma)
        class(obj) <- "multiLtt"
    }
    else {
        tree <- reorder.phylo(tree, order = "cladewise")
        if (!is.null(tree$node.label)) {
            node.names <- setNames(tree$node.label, 1:tree$Nnode +
                Ntip(tree))
            tree$node.label <- NULL
        }
        else node.names <- NULL
        if (is.ultrametric(tree)) {
            h <- max(nodeHeights(tree))
            time <- c(0, h - sort(branching.times(tree), decreasing = TRUE),
                h)
            nodes <- as.numeric(names(time)[2:(length(time) -
                1)])
            ltt <- c(cumsum(c(1, sapply(nodes, function(x, y) sum(y ==
                x) - 1, y = tree$edge[, 1]))), length(tree$tip.label))
            names(ltt) <- names(time)
        }
        else {
            drop.extinct.tips <- function(phy) {
                temp <- diag(vcv(phy))
                if (length(temp[temp < (max(temp) - tol)]) >
                  0)
                  pruned.phy <- drop.tip(phy, names(temp[temp <
                    (max(temp) - tol)]))
                else pruned.phy <- phy
                return(pruned.phy)
            }
            if (drop.extinct == TRUE)
                tree <- drop.extinct.tips(tree)
            root <- length(tree$tip) + 1
            node.height <- matrix(NA, nrow(tree$edge), 2)
            for (i in 1:nrow(tree$edge)) {
                if (tree$edge[i, 1] == root) {
                  node.height[i, 1] <- 0
                  node.height[i, 2] <- tree$edge.length[i]
                }
                else {
                  node.height[i, 1] <- node.height[match(tree$edge[i,
                    1], tree$edge[, 2]), 2]
                  node.height[i, 2] <- node.height[i, 1] + tree$edge.length[i]
```

```
                }
            }
            ltt <- vector()
            tree.length <- max(node.height)
            n.extinct <- sum(node.height[tree$edge[, 2] <= length(tree$tip),
                2] < (tree.length - tol))
            node.height[tree$edge[, 2] <= length(tree$tip), 2] <- node.height[tree$edge[,
                2] <= length(tree$tip), 2] + 1.1 * tol
            time <- c(0, node.height[, 2])
            names(time) <- as.character(c(root, tree$edge[, 2]))
            temp <- vector()
            time <- time[order(time)]
            time <- time[1:(tree$Nnode + n.extinct + 1)]
            for (i in 1:(length(time) - 1)) {
                ltt[i] <- 0
                for (j in 1:nrow(node.height)) ltt[i] <- ltt[i] +
                  (time[i] >= (node.height[j, 1] - tol) && time[i] <=
                    (node.height[j, 2] - tol))
            }
            ltt[i + 1] <- 0
            for (j in 1:nrow(node.height)) ltt[i + 1] <- ltt[i +
                1] + (time[i + 1] <= (node.height[j, 2] + tol))
            names(ltt) <- names(time)
            ltt <- c(1, ltt)
            time <- c(0, time)
            time[length(time)] <- time[length(time)] - 1.1 *
                tol
        }
        if (!is.null(node.names)) {
            nn <- sapply(names(time), function(x, y) if (any(names(y) ==
                x))
                y[which(names(y) == x)]
            else "", y = node.names)
            names(ltt) <- names(time) <- nn
        }
        if (gamma == FALSE) {
            obj <- list(ltt = ltt, times = time, tree = tree)
            class(obj) <- "ltt"
        }
        else {
            gam <- gammatest(list(ltt = ltt, times = time))
            obj <- list(ltt = ltt, times = time, gamma = gam$gamma,
                p = gam$p, tree = tree)
            class(obj) <- "ltt"
        }
    }
    if (plot)
        plot(obj, log.lineages = log.lineages, ...)
    obj
}
<environment: namespace:phytools>
```

```
ltts <- ltt(this_tree, gamma = TRUE, plot = FALSE)
ltts
```

```
## Object of class "ltt" containing:
##
## (1) A phylogenetic tree with 456 tips and 455 internal nodes.
##
## (2) Vectors containing the number of lineages (ltt) and branching times (times) on the tree.
##
## (3) A value for Pybus & Harvey's "gamma" statistic of 1.7401, p-value = 0.0818.
```

```
str(ltts)
```

```
## List of 5
##  $ ltt  : Named num [1:457] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "names")= chr [1:457] "" "457" "458" "459" ...
##  $ times: Named num [1:457] 0.00 2.84e-13 1.00 2.00 2.67 ...
##   ..- attr(*, "names")= chr [1:457] "" "457" "458" "459" ...
##  $ gamma: num 1.74
##  $ p    : num 0.0818
##  $ tree :List of 4
##   ..$ edge       : num [1:910, 1:2] 457 460 463 560 892 892 560 463 466 466 ...
##   ..$ tip.label  : chr [1:456] "t862" "t409" "t260" "t204" ...
##   ..$ edge.length: num [1:910] 2.667 4.433 20.973 10.011 0.916 ...
##   ..$ Nnode      : num 455
##   ..- attr(*, "class")= chr "phylo"
##   ..- attr(*, "order")= chr "cladewise"
##  - attr(*, "class")= chr "ltt"
```

```
lineages_through_time <- as.numeric(ltts[[1]])
time_steps <- as.numeric(ltts[[2]])
#extract Gamma index
gamma <- ltts[[3]]
gamma_p_value <- ltts[[4]]
```

```
lineages_through_time
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102
## [103] 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
## [120] 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
## [137] 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
## [154] 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170
## [171] 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
## [188] 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204
## [205] 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221
## [222] 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238
## [239] 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
## [256] 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272
## [273] 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289
## [290] 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
## [307] 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
```

```
## [324] 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340
## [341] 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357
## [358] 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374
## [375] 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391
## [392] 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408
## [409] 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425
## [426] 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442
## [443] 443 444 445 446 447 448 449 450 451 452 453 454 455 456 456
```

```
time_steps
```

```
##   [1] 0.000000e+00 2.842171e-13 1.000000e+00 2.000000e+00 2.666667e+00
##   [6] 3.333333e+00 5.461538e+00 7.100000e+00 7.450000e+00 9.296296e+00
##  [11] 1.003030e+01 1.024242e+01 1.121429e+01 1.126190e+01 1.200000e+01
##  [16] 1.203509e+01 1.215789e+01 1.217544e+01 1.314815e+01 1.316667e+01
##  [21] 1.335185e+01 1.416667e+01 1.428788e+01 1.436364e+01 1.502222e+01
##  [26] 1.503333e+01 1.512222e+01 1.527778e+01 1.600000e+01 1.630588e+01
##  [31] 1.720588e+01 1.733333e+01 1.734314e+01 1.738235e+01 1.808036e+01
##  [36] 1.833036e+01 1.901613e+01 1.913710e+01 1.915323e+01 1.921774e+01
##  [41] 1.923387e+01 2.006504e+01 2.021138e+01 2.030081e+01 2.038211e+01
##  [46] 2.043902e+01 2.047154e+01 2.100000e+01 2.101361e+01 2.117687e+01
##  [51] 2.130612e+01 2.131973e+01 2.132653e+01 2.204065e+01 2.233333e+01
##  [56] 2.234146e+01 2.236585e+01 2.247967e+01 2.254472e+01 2.314872e+01
##  [61] 2.315897e+01 2.320513e+01 2.406011e+01 2.409290e+01 2.409836e+01
##  [66] 2.424044e+01 2.431694e+01 2.501579e+01 2.503158e+01 2.506842e+01
##  [71] 2.512105e+01 2.514737e+01 2.515263e+01 2.516842e+01 2.517895e+01
##  [76] 2.519474e+01 2.529474e+01 2.534737e+01 2.600985e+01 2.602956e+01
##  [81] 2.604433e+01 2.605419e+01 2.607882e+01 2.623153e+01 2.624138e+01
##  [86] 2.625123e+01 2.625616e+01 2.627586e+01 2.629064e+01 2.630049e+01
##  [91] 2.631034e+01 2.706135e+01 2.711043e+01 2.717791e+01 2.724540e+01
##  [96] 2.725153e+01 2.734356e+01 2.747239e+01 2.749693e+01 2.755828e+01
## [101] 2.757669e+01 2.800385e+01 2.802692e+01 2.804231e+01 2.807308e+01
## [106] 2.813077e+01 2.814615e+01 2.816154e+01 2.816923e+01 2.818462e+01
## [111] 2.820000e+01 2.821923e+01 2.825385e+01 2.830000e+01 2.901115e+01
## [116] 2.901487e+01 2.902230e+01 2.903346e+01 2.905204e+01 2.909294e+01
## [121] 2.912268e+01 2.915985e+01 2.923048e+01 2.938662e+01 3.000683e+01
## [126] 3.001365e+01 3.003413e+01 3.015358e+01 3.018771e+01 3.019795e+01
## [131] 3.021502e+01 3.022184e+01 3.028669e+01 3.032765e+01 3.101894e+01
## [136] 3.104545e+01 3.107955e+01 3.109848e+01 3.115152e+01 3.115909e+01
## [141] 3.117424e+01 3.118939e+01 3.120455e+01 3.120833e+01 3.122348e+01
## [146] 3.125000e+01 3.125758e+01 3.128030e+01 3.129924e+01 3.132197e+01
## [151] 3.138636e+01 3.140909e+01 3.141288e+01 3.141667e+01 3.142803e+01
## [156] 3.201587e+01 3.201905e+01 3.202222e+01 3.202540e+01 3.204127e+01
## [161] 3.205079e+01 3.206984e+01 3.209206e+01 3.210476e+01 3.215873e+01
## [166] 3.216825e+01 3.218095e+01 3.218730e+01 3.223810e+01 3.225714e+01
## [171] 3.228571e+01 3.233651e+01 3.302521e+01 3.303922e+01 3.305042e+01
## [176] 3.305602e+01 3.311204e+01 3.311765e+01 3.312885e+01 3.313445e+01
## [181] 3.314566e+01 3.314846e+01 3.315126e+01 3.315966e+01 3.318487e+01
## [186] 3.319328e+01 3.321289e+01 3.322129e+01 3.322409e+01 3.322689e+01
## [191] 3.322969e+01 3.323249e+01 3.323529e+01 3.323810e+01 3.327171e+01
## [196] 3.328852e+01 3.400290e+01 3.402899e+01 3.405217e+01 3.408406e+01
## [201] 3.409855e+01 3.410725e+01 3.412754e+01 3.413623e+01 3.415652e+01
## [206] 3.417101e+01 3.418261e+01 3.421449e+01 3.422609e+01 3.423478e+01
## [211] 3.425217e+01 3.425797e+01 3.427536e+01 3.427826e+01 3.428986e+01
## [216] 3.430145e+01 3.431304e+01 3.431884e+01 3.432464e+01 3.433043e+01
```

```
## [221] 3.435072e+01 3.435942e+01 3.436812e+01 3.500431e+01 3.503664e+01
## [226] 3.504310e+01 3.506897e+01 3.507112e+01 3.508190e+01 3.508836e+01
## [231] 3.509698e+01 3.509914e+01 3.510129e+01 3.510991e+01 3.511853e+01
## [236] 3.600000e+01 3.600651e+01 3.601303e+01 3.602280e+01 3.603257e+01
## [241] 3.603583e+01 3.606189e+01 3.606515e+01 3.606840e+01 3.607166e+01
## [246] 3.607492e+01 3.608143e+01 3.608795e+01 3.609772e+01 3.610098e+01
## [251] 3.610749e+01 3.611401e+01 3.612378e+01 3.614332e+01 3.614984e+01
## [256] 3.615309e+01 3.615635e+01 3.615961e+01 3.617590e+01 3.617915e+01
## [261] 3.618241e+01 3.618567e+01 3.619218e+01 3.619870e+01 3.620195e+01
## [266] 3.620847e+01 3.622150e+01 3.623127e+01 3.623779e+01 3.624104e+01
## [271] 3.624756e+01 3.626384e+01 3.627036e+01 3.629316e+01 3.629967e+01
## [276] 3.630293e+01 3.630945e+01 3.631922e+01 3.632248e+01 3.633225e+01
## [281] 3.634202e+01 3.637134e+01 3.637785e+01 3.640065e+01 3.640391e+01
## [286] 3.640717e+01 3.641368e+01 3.642020e+01 3.642345e+01 3.642671e+01
## [291] 3.642997e+01 3.643322e+01 3.643974e+01 3.644625e+01 3.644951e+01
## [296] 3.645277e+01 3.646254e+01 3.646580e+01 3.647231e+01 3.647557e+01
## [301] 3.648534e+01 3.648860e+01 3.649511e+01 3.650163e+01 3.652117e+01
## [306] 3.653094e+01 3.654397e+01 3.700000e+01 3.700279e+01 3.700559e+01
## [311] 3.700838e+01 3.701117e+01 3.701676e+01 3.702235e+01 3.703073e+01
## [316] 3.703911e+01 3.704749e+01 3.705307e+01 3.705587e+01 3.705866e+01
## [321] 3.706145e+01 3.706704e+01 3.706983e+01 3.707542e+01 3.707821e+01
## [326] 3.708101e+01 3.708380e+01 3.708659e+01 3.710056e+01 3.710335e+01
## [331] 3.710615e+01 3.710894e+01 3.711453e+01 3.711732e+01 3.712011e+01
## [336] 3.712570e+01 3.712849e+01 3.713408e+01 3.713687e+01 3.713966e+01
## [341] 3.714246e+01 3.714804e+01 3.715084e+01 3.715363e+01 3.715642e+01
## [346] 3.715922e+01 3.716201e+01 3.716480e+01 3.717039e+01 3.717877e+01
## [351] 3.718156e+01 3.718436e+01 3.718994e+01 3.719832e+01 3.720112e+01
## [356] 3.720391e+01 3.720670e+01 3.720950e+01 3.721229e+01 3.721788e+01
## [361] 3.722067e+01 3.722346e+01 3.722626e+01 3.722905e+01 3.723184e+01
## [366] 3.723464e+01 3.723743e+01 3.724022e+01 3.724302e+01 3.724581e+01
## [371] 3.724860e+01 3.725140e+01 3.725419e+01 3.726257e+01 3.726816e+01
## [376] 3.727095e+01 3.727933e+01 3.728212e+01 3.728771e+01 3.729050e+01
## [381] 3.729609e+01 3.729888e+01 3.730168e+01 3.730447e+01 3.730726e+01
## [386] 3.731006e+01 3.731285e+01 3.732402e+01 3.733240e+01 3.733520e+01
## [391] 3.733799e+01 3.734078e+01 3.734358e+01 3.734637e+01 3.736034e+01
## [396] 3.736313e+01 3.736592e+01 3.737151e+01 3.737430e+01 3.737709e+01
## [401] 3.737989e+01 3.738268e+01 3.739106e+01 3.739385e+01 3.739665e+01
## [406] 3.739944e+01 3.740223e+01 3.740503e+01 3.740782e+01 3.741061e+01
## [411] 3.741620e+01 3.742179e+01 3.742458e+01 3.743296e+01 3.800000e+01
## [416] 3.800393e+01 3.800589e+01 3.801572e+01 3.801768e+01 3.802358e+01
## [421] 3.802554e+01 3.802750e+01 3.803143e+01 3.803340e+01 3.803536e+01
## [426] 3.804322e+01 3.804519e+01 3.804715e+01 3.805108e+01 3.805697e+01
## [431] 3.806090e+01 3.806483e+01 3.806680e+01 3.807269e+01 3.807662e+01
## [436] 3.808251e+01 3.808448e+01 3.808841e+01 3.809234e+01 3.809627e+01
## [441] 3.809823e+01 3.810216e+01 3.810806e+01 3.811198e+01 3.811395e+01
## [446] 3.812181e+01 3.812377e+01 3.812574e+01 3.812770e+01 3.812967e+01
## [451] 3.813163e+01 3.813360e+01 3.813949e+01 3.814145e+01 3.814538e+01
## [456] 3.815128e+01 3.900000e+01
```

```
gamma
```

```
## [1] 1.740065
```

```
gamma_p_value
```

```
## [1] 0.08184766
```

There are two other regularly used metrics that include abundance measures. Note: we don't have abundance measures for D-place data.

IAC, imbalance of abundance at the clade level, quantifies the relative deviation in the abundance distribution from a null case where individuals are evenly partitioned between clade splits. v is the number of nodes in the phylogenetic tree. n_i is, as defined above, the abundance of species i in the assemblage. η_k is the expected abundance species i would have if the abundance was randomly split among lineages in the phylogenetic tree at each speciation event. is the number of lineages originating at node k in the set $s(k, root)$, which contains the nodes located on the path between node k and the root of the phylogenetic tree. N is the total assemblage abundance (**?**).

$$\frac{\sum_{i=1}^{S} |n_i - \hat{n_i}|}{v} where \hat{n_i} = \frac{N}{\prod_{K \in s(i,root)} \eta_k}$$

I_c, the Colless index, is the sum of the absolute differences in species richness between sister-clades at each internal node. For fully resolved trees, each internal node defines two sister-clades. S_{1k} is the number of species descending from the first clade defined by node k and S_{2k} that of the second clade. v is, as defined above, the number of nodes in the phylogenetic (**?**).

$$I_c = \sum_{k=1}^{v} |S_{1k} - S_{2k}|$$

5.4 Macroevolutionary rates

```
#function name = bd, function input = tree of type 'phylo'
bd <- function (tree)
{
    tree$edge.length <- tree$edge.length/max(tree$edge.length)
    x <- birthdeath(tree)
    b <- x$para[2]/(1 - x$para[1])
    d <- b - x$para[2]
    c(setNames(c(b, d), c("b", "d")), x$para)
}
```

```
 ## Speciation vs extinction rates and Net diversification
bds <- bd(this_tree)
speciation_rate <- bds[1]
extinction_rate <- bds[2]
extinction_per_speciation <- bds[3]
speciation_minus_extinction <- bds[4]
```

```
## Speciation vs extinction rates and Net diversification dependent on trait
# N.for.dom <- table(this_world[, 6])
#    if(length(N.for.dom) == 2) {
par.div.dep <- DivDep( mytree = this_tree, myWorld = this_world)
trait_1_speciation <- par.div.dep[1]
trait_2_speciation <- par.div.dep[2]
trait_1_extinction <- par.div.dep[3]
trait_2_extinction <- par.div.dep[4]
transition_from_trait_1_to_2 <- par.div.dep[5]
transition_from_trait_2_to_1 <- par.div.dep[6]
transition_rate_ratio_1to2_over_2to1 <- transition_from_trait_1_to_2/transition_from_trait_2_to_1
```

```
## Crown age per trait AUC and effect size
tip.length <- this_tree$edge.length[this_tree$edge[, 2] %in% 1:Ntip(this_tree)]
tip.length <- (tip.length - min(tip.length)) / (max(tip.length) - min(tip.length))
this_trait <- this_world[match(this_tree$tip.label, this_world[, 8]), 6]
tip.length.2 <- tip.length[this_trait == 2]
tip.length.1 <- tip.length[this_trait == 1]
model <- glm(as.factor(this_trait) ~ log(tip.length + 1),
             family = "binomial")
effect.size <- model$coefficients[2]
# plot(y = this_trait - 1, x= log(tip.length))
p <- predict(model, as.factor(this_trait), type = "resp")
# points(y = p, x = log(tip.length), col = "red")
pr <- prediction(p, as.factor(this_trait))
auc.model <- performance(pr, measure = "auc")@y.values[[1]]
```

```
## Phylogenetic signal (D)
Phylogenetic_signal <- Dsig(mytree = this_tree, myWorld = this_world)
```

5.5 Spatial Locations

```
## Spatial Analysis
nbs0 <- knearneigh(as.matrix(this_world[, 2:3]), k = 7, longlat = TRUE)
```

```
## Warning in knearneigh(as.matrix(this_world[, 2:3]), k = 7, longlat = TRUE):
## knearneigh: identical points found
```

```
nbs <- knn2nb(nbs0, sym = TRUE) # 7 symmetric neighbors
nbs.listw <- nb2listw(nbs)
factors.nbs <- as.factor(ifelse(is.na(this_world[, 6]), 3, this_world[, 6]))
spatial.tests <- joincount.test(fx = factors.nbs, listw = nbs.listw)
spatial.tests.fora <- spatial.tests[[1]]$statistic
spatial.tests.dom <- spatial.tests[[2]]$statistic
#prevalence <- (N.for.dom[1] - N.for.dom[2]) / sum(N.for.dom)
```

```
results_summary_matrix_1 <- cbind(

        number_of_branches,
        #Pylo_diversity_is_sum_of_BL,
        #average_phylogenetic_diversity_is_mean_of_BL,
        #variance_Pylo_diversity_is_variance_of_BL,

        F_quadratic_entropy_is_sum_of_PD,
        Mean_pairwise_distance,
        variance_pairwise_distance,

        #Evolutionary_distinctiveness_sum,
        #mean_Phylogenetic_isolation,
        #variance_Phylogenetic_isolation,

        gamma,
        gamma_p_value,
        speciation_rate,
```

```
      extinction_rate,
      extinction_per_speciation,
      speciation_minus_extinction,
      trait_1_speciation,
      trait_2_speciation ,
      trait_1_extinction ,
      trait_2_extinction ,
      transition_from_trait_1_to_2 ,
      transition_from_trait_2_to_1 ,
      transition_rate_ratio_1to2_over_2to1 ,
      Phylogenetic_signal,
      spatial.tests.fora,
      spatial.tests.dom,
     # prevalence,
     # auc.model,
      effect.size
    )
    #rownames(results_summary_matrix_1) <- 1

    #results_summary_matrix_2 <- cbind(
    #  c(Evolutionary_distinctiveness,NA),
    #  lineages_through_time,
    #  time_steps
    #)
    #colnames(results_summary_matrix_2) <- c("Evolutionary_distinctiveness", "lineages_through_time",
    #head(results_summary_matrix_2)

    ### Returns from function in list form
    #returns <- list(
      #Branch_Lengths,
      #Pairwise_dist,
    #  results_summary_matrix_1,
    #  results_summary_matrix_2

    #)

    #names(returns) <- c(
      #"Branch_Lengths",
      #"Pairwise_distance",
     # "results_summary_of_single_value_outputs",
     # "results_summary_matrix_of_multi_value_outputs"
    #)
```

5.6 Module2() returns these two matrices as a list

5.6.1 Here is the exact version in R

```
## This module analyzes the results from module 1 and returns a list based on how many values each stat
## Ty Tuff and Bruno Vilela
## 24 August 2016
```

```
###### Specify function ################################

Module_2 <- function(Module_1_output) {
  cat("\nAnalyzing: 0% [")
  if (any(is.na(Module_1_output))) {
    cat("----------]")
    return(NA)
  } else {

    this_tree <- Module_1_output$mytree
    this_world <- Module_1_output$myWorld

    ##### (0) Pull necessary variables from simulated trees and organize into a single object for all tl

    #str(all_trees)
    #str(this_tree)

    ## 0a) Branch lengths
    Branch_Lengths <- this_tree$edge.length
    number_of_branches <- length(Branch_Lengths)

    # Anchor test = PD (Faith's phylogenetic diversity)
    Pylo_diversity_is_sum_of_BL <- sum(Branch_Lengths)

    # avPD -- Average phylogenetic diversity
    average_phylogenetic_diversity_is_mean_of_BL <- mean(Branch_Lengths)

    variance_Pylo_diversity_is_variance_of_BL <- var(Branch_Lengths)
    cat("-")
    ## 0b) Pairwise distance between tips
    Pairwise_dist <- cophenetic.phylo(this_tree)
    cat("-")
    # 2b) Pairwise distance -- Sum of pairwise distances

    # F -- Extensive quadratic entropy
    F_quadratic_entropy_is_sum_of_PD <- sum(Pairwise_dist)

    #Mean inter-species distances

    # Anchor test = MPD (mean pairwise distance)

    Mean_pairwise_distance <- mean(Pairwise_dist)

    cat("-")
    #Pairwise distance/all distances -- Variance of pairwise distances

    # Anchor test = VPD (variation of pairwise distance)

    variance_pairwise_distance <- var(as.vector(Pairwise_dist))

    ## 0c) Phylogenetic isolation

    # Using equal.splits method, faster computation
```

```
  Evolutionary_distinctiveness <- evol.distinct2(this_tree, type = "fair.proportion")
  cat("-")
  # ED - Summed evolutionary distinctiveness

  Evolutionary_distinctiveness_sum <- sum(Evolutionary_distinctiveness)

  ## 3d) Phylogenetic isolation -- Mean of species evolutionary distinctiveness

  # mean(ED)

  mean_Phylogenetic_isolation <- mean(Evolutionary_distinctiveness)

  ## 4d) Phylogenetic isolation -- Variance of species isolation metrics

  #var(ED)

  variance_Phylogenetic_isolation <- var(Evolutionary_distinctiveness)
  cat("-")

  ## Tree topology

  #Gamma index

  ltts <- ltt(this_tree, gamma = TRUE, plot = FALSE)
  lineages_through_time <- as.numeric(ltts[[1]])
  time_steps <- as.numeric(ltts[[2]])
  gamma <- ltts[[3]]
  gamma_p_value <- ltts[[4]]
  cat("-")

  colless_stat <- colless(as.treeshape(this_tree))
  sackin_index <- sackin(as.treeshape(this_tree))
  tree_shape_stat <- shape.statistic(as.treeshape(this_tree))

  ##### (5) Tree metric -- Macroevolutionary - Rate and rate changes ###############
  ######################################################

  ## Speciation vs extinction rates and Net diversification
  bds <- bd(this_tree)
  speciation_rate <- bds[1]
  extinction_rate <- bds[2]
  extinction_per_speciation <- bds[3]
  speciation_minus_extinction <- bds[4]
  cat("-")

  ## Speciation vs extinction rates and Net diversification dependent on trait
  N.for.dom <- table(this_world[, 6])
  if(length(N.for.dom) == 2) {
    par.div.dep <- DivDep( mytree = this_tree, myWorld = this_world)
    trait_1_speciation <- par.div.dep[1]
    trait_2_speciation <- par.div.dep[2]
    trait_1_extinction <- par.div.dep[3]
    trait_2_extinction <- par.div.dep[4]
```

```
    transition_from_trait_1_to_2 <- par.div.dep[5]
    transition_from_trait_2_to_1 <- par.div.dep[6]
    transition_rate_ratio_1to2_over_2to1 <- transition_from_trait_1_to_2/transition_from_trait_2_to_1
    cat("-")

    ## Phylogenetic signal (D)
    Phylogenetic_signal <- Dsig(mytree = this_tree, myWorld = this_world)
    cat("-")

    ## Spatial Analysis
    nbs0 <- knearneigh(as.matrix(this_world[, 2:3]), k = 7, longlat = TRUE)
    nbs <- knn2nb(nbs0, sym = TRUE) # 7 symmetric neighbors
    nbs.listw <- nb2listw(nbs)
    factors.nbs <- as.factor(ifelse(is.na(this_world[, 6]), 3, this_world[, 6]))
    spatial.tests <- joincount.test(fx = factors.nbs, listw = nbs.listw)
    spatial.tests.fora <- spatial.tests[[1]]$statistic
    spatial.tests.dom <- spatial.tests[[2]]$statistic
    prevalence <- (N.for.dom[1] - N.for.dom[2]) / sum(N.for.dom)
    cat("-")
  } else {
    trait_1_speciation <- NA
    trait_2_speciation <- NA
    trait_1_extinction <- NA
    trait_2_extinction <- NA
    transition_from_trait_1_to_2 <- NA
    transition_from_trait_2_to_1 <- NA
    transition_rate_ratio_1to2_over_2to1 <- NA
    Phylogenetic_signal <- NA
    spatial.tests.fora <- NA
    spatial.tests.dom <- NA
    prevalence <- ifelse(names(table(this_world[, 6])[1]) == "1", 1,
                         -1)
    cat("---")

  }

  results_summary_matrix_1 <- cbind(

    number_of_branches,
    Pylo_diversity_is_sum_of_BL,
    average_phylogenetic_diversity_is_mean_of_BL,
    variance_Pylo_diversity_is_variance_of_BL,

    F_quadratic_entropy_is_sum_of_PD,
    Mean_pairwise_distance,
    variance_pairwise_distance,

    colless_stat ,
    sackin_index ,
    tree_shape_stat,

    Evolutionary_distinctiveness_sum,
    mean_Phylogenetic_isolation,
```

```
      variance_Phylogenetic_isolation,

      gamma,
      gamma_p_value,
      speciation_rate,
      extinction_rate,
      extinction_per_speciation,
      speciation_minus_extinction,
      trait_1_speciation,
      trait_2_speciation ,
      trait_1_extinction ,
      trait_2_extinction ,
      transition_from_trait_1_to_2 ,
      transition_from_trait_2_to_1 ,
      transition_rate_ratio_1to2_over_2to1 ,
      Phylogenetic_signal,
      spatial.tests.fora,
      spatial.tests.dom,
      prevalence
    )
    rownames(results_summary_matrix_1) <- 1

    results_summary_matrix_2 <- cbind(
      c(Evolutionary_distinctiveness,NA),
      lineages_through_time,
      time_steps
    )
    colnames(results_summary_matrix_2) <- c("Evolutionary_distinctiveness",
                                            "lineages_through_time", "time_steps")
    head(results_summary_matrix_2)

    ### Returns from function in list form
    returns <- list(
      #Branch_Lengths,
      #Pairwise_dist,
      results_summary_matrix_1,
      results_summary_matrix_2

    )

    names(returns) <- c(
      #"Branch_Lengths",
      #"Pairwise_distance",
      "results_summary_of_single_value_outputs",
      "results_summary_matrix_of_multi_value_outputs"
    )
    cat("] 100%")

    return(returns)

  }
}
```

```
#Module_2(myOut)
```

5.7 References

Chapter 6

Calling Module 1 and Module 2

Simulating 10000 replicates of each of the 4 hypothesized mechanisms (40000 total simulations) and then calculating summary statistics for all of those simulated worlds required a tremendous amount of computing power. We utilize a 1000-node cluster at Washington University in St. Louis and a 300-node cluster at MPI in Jena from August 2016 to January 2018 to first prototype the simulation and then run 1000 replicates of each hypothesized mechanism. Model development and prototyping took a full year and two major rebuilds to produce realistic worlds that could fit real world data. In that time, we changed modes of categorizing simulation outputs from approximate Bayesian computation (ABC) to random forest machine learning (CITE) because our simulations did not produce the linear posterior distributions required for ABC but we were able to work around those limitations using a supervised random forest algorithm.

There are two versions of this script, the first is for running each simulation to the end and then saving the final step as the output of the model and the second is to save outputs along the way so we can evaluate how different models change through time.

6.1 Run the simulation to a specified timestep and then save one output

Here is the first, and primary, version:

```
#########################################################################

# Run the full model in a cluster. This version writes files to a cluster output folder.
# rm(list = ls())
# install.packages("~/Desktop/FARM_1.0.tar.gz", repos=NULL, type="source")


#########################################################################
## need to document which functions we use from each of these libraries.
library(ape)
library(spdep)
library(Rcpp)
library(msm)
library(FARM)


sim_run_cluster <- function(replicate_cycle, myWorld, number_of_time_steps, nbs,
```

```
                            number_of_tips, number_of_neighbors, origins, start = NULL) {
 # Calls the full simulation script
 #
 # Purpose: Need to wrap the entire simulation script into a function so it can be called in parallel
 #
 # Args:
 #    replicate_cycle: An integer indicating the replicate number of a simulation. This variable is us
 #         the saved output file and control the number of replicates run by the cluster.
 #
 #    combo_number: An interger between 1 and 31 indicating the combinations of S, E, A, D, and T modu
 #         in the simulation. The full list of these combinations can be printed using the function co
 #         We are currently using combinations 25,28,29,and 31 as our four competing models for the sp
 #
 #    myWorld: Matrix that defines the scope of the available world and acts as a data hub for organiz
 #         results from the different elements of the simulation.
 #
 #    number_of_time_steps: An integer indicating how many iterations the simulation will calculated b
 #         file.
 #
 #    nbs: A list of the available neighbors for each spatial point. This is passed to the function fo
 #         of neighbors through time.
 #
 #    number_of_tips: An interger indicating the number of tree tips the simulation should be truncate
 #         include all the available tips (e.g. 1254 for human languages).
 #
 # Returns:
 #    myOut: A list object containing a 'phylo' tree object called mytree in the first position and th
 #         spatial and tree data in the second position
 #


 x1 <- 4 #Number of runs per core
 sampleer <- sample(c(1,2,5,6), x1)
 #if (replicate_cycle != 1) {
 #  replicate_cycle <- ((replicate_cycle - 1) * x1) + 1
# }
# replicate_cycle <- replicate_cycle:(replicate_cycle + (x1 - 1))
 for (count in sampleer) {
 independent <- 1


   # Probability of Arisal
   prob_choose_a <- rev(sort(rexp(4, rate = 9)))
   prob_choose_a <- prob_choose_a[c(sample(1:2, 2), sample(3:4, 2))]
   prob_choose_a[3] <- 0
   P.Arisal0  <- parameters(prob_choose_a[1], prob_choose_a[4],
                            prob_choose_a[3], prob_choose_a[2],
                            "Env_NonD", "Env_D",
                            "Evol_to_F", "Evol_to_D")
   # P.Arisal0 is the one you should change the parameters
   P.Arisal <- matrix(NA, ncol = 2, nrow = nrow(myWorld)) # probability per cell
   colnames(P.Arisal) <- c("Evolve_to_F", "Evolve_to_D")
   Env.Dom <- myWorld[, 7] == 2
```

```
P.Arisal[Env.Dom, 1] <- P.Arisal0[1, 2]
P.Arisal[!Env.Dom, 1] <- P.Arisal0[1, 1]
P.Arisal[Env.Dom, 2] <- P.Arisal0[2, 2]
P.Arisal[!Env.Dom, 2] <- P.Arisal0[2, 1]

colnames(P.Arisal) <- c("Prob_of_Foraging", "Prob_of_Domestication")
P.Arisal[which(origins == FALSE), 2]  <- 0

#####
#prob_choose <- runif(12, 0.01, 1)
#sub <- (prob_choose[1] - 0.01)
#sub <- ifelse(sub < .1, .1, sub)
#prob_choose[c(4)] <- runif(1, 0.01, sub)
#prob_choose[c(5)] <- runif(1, 0.1, 1) # High extinction
#prob_choose[c(6)] <- runif(1, 0, (prob_choose[3] - 0.01))
#prob_choose[c(9, 10, 12)] <- runif(3, 0.01, prob_choose[11])

####
prob_choose <- runif(12, 0, 1)
top <- min(prob_choose[c(1,3)], na.rm=TRUE)
prob_choose[c(2)] <- runif(1, 0, top)

prob_choose[c(5)] <- runif(1, 0, prob_choose[c(2)])
prob_choose[c(6)] <- runif(1, 0, prob_choose[c(5)])
prob_choose[c(4)] <- runif(1, prob_choose[c(6)], prob_choose[c(5)])


if (count == 1) {
  prob_choose[7:12] <- 0
}
if (count == 2) {
  prob_choose[9:12] <- 0
}
if (count == 3 | count == 5) {
  prob_choose[7:8] <- 0
  independent <- 0
}
if (count == 4 | count == 6) {
  independent <- 0
}


P.speciation <- parameters(prob_choose[1], prob_choose[1],
                           prob_choose[2], prob_choose[3],
                           "Env_NonD", "Env_D", "For", "Dom")

P.extinction  <- parameters(prob_choose[4], prob_choose[4],
                            prob_choose[5], prob_choose[6],
                            "Env_NonD", "Env_D", "For", "Dom")


P.diffusion <- parameters(0, prob_choose[7],
                          prob_choose[8], 0,
```

```
                              "Target_For", "Target_Dom",
                              "Source_For", "Source_Dom")

    P.TakeOver <- parameters(prob_choose[9], prob_choose[10],
                             prob_choose[11], prob_choose[12],
                             "Target_For", "Target_Dom",
                             "Source_For", "Source_Dom")
    multiplier <- 1 # always 1 now.
   if (count %in% 1:4) {
        myOut <- RunSimUltimate(myWorld, P.extinction, P.speciation,
                            P.diffusion, P.Arisal, P.TakeOver, nbs, independent,
                            N.steps = number_of_time_steps, silent = FALSE,
                            multiplier = multiplier, start = start)
                            }
   if (count %in% 5:6) {
        myOut <- RunSimUltimate.push(myWorld, P.extinction, P.speciation,
                            P.diffusion, P.Arisal, P.TakeOver, nbs, independent,
                            N.steps = number_of_time_steps, silent = FALSE,
                            multiplier = multiplier, start = start)
                            }


    # Count refers to the combo, 1 = null, 2 = diffusion, 3 = Takeover, 4 = full
    save(myOut,  file= paste0("./Module_1_outputs/myOut_rep_",
                              formatC(replicate_cycle, width = 2,flag = 0),
                              "_combo_",
                              formatC(count, width = 2,flag = 0),
                              "_","params", "_P.speciation_",
                              paste(formatC(P.speciation, width = 2,flag = 0), collapse="_"),"_P.extinc
                              paste(formatC(P.extinction, width = 2,flag = 0), collapse="_"), "_P.diffu
                              paste(formatC(P.diffusion, width = 2,flag = 0), collapse="_"), "_P.TO_",
                              paste(formatC(P.TakeOver, width = 2,flag = 0), collapse="_"),"_P.Arisal_"
                              paste(formatC(P.Arisal0, width = 2,flag = 0), collapse="_"),
                              "_timesteps_", number_of_time_steps, "_NBS_", number_of_neighbors, "_.Rda

    Sim_statistics <- Module_2(myOut)

    save(Sim_statistics, file= paste0("./Module_2_outputs/Sim_stats_rep_",
                                      formatC(replicate_cycle, width = 2,flag = 0),
                                      "_combo_",
                                      formatC(count, width = 2,flag = 0),
                                      "_","params", "_P.speciation_",
                                      paste(formatC(P.speciation, width = 2,flag = 0), collapse="_"),"_
                                      paste(formatC(P.extinction, width = 2,flag = 0), collapse="_"), "
                                      paste(formatC(P.diffusion, width = 2,flag = 0), collapse="_"), "_
                                      paste(formatC(P.TakeOver, width = 2,flag = 0), collapse="_"),"_P.
                                      paste(formatC(P.Arisal0, width = 2,flag = 0), collapse="_"),
                                      "_timesteps_", number_of_time_steps, "_NBS_", number_of_neighbors
  }

}

#########################################################################
```

```
coords <- as.matrix(apply(language_centroids[, 3:4], 2, as.numeric)) #coords
conds <- ifelse(suitability2 == 0, 1, 2)
conds[is.na(conds)] <- sample(c(1, 2), sum(is.na(conds)), replace = TRUE)
origins <- language_centroids[, 5]

##### Specify simulation parameters ##################################

number_of_tips <- length(coords[,1])
number_of_time_steps_a <- 30000
#replicate_cycle <- c(1)  #number of replicates
######################################################################
data("parameters.table")


system.time(
  myWorld <- BuildWorld(coords, conds)
)
number_of_neighbors <- sample(5:9,1)

nbs <- knn2nb(knearneigh(coords, k = number_of_neighbors, longlat = TRUE),
              sym = TRUE) # 7 symmetric neighbors
n.obs <- sapply(nbs, length)
seq.max <- seq_len(max(n.obs))
nbs <- t(sapply(nbs, "[", i = seq.max))

dim(myWorld)


# NAI <- 1000
args <- commandArgs(trailingOnly = FALSE)
print(args)
NAI <- as.numeric(args[7])
#setwd("~/Box Sync/colliding ranges/Simulations_humans/big world cluster outputs")


# Starting point
start <- sample((1:nrow(language_centroids))[as.logical(language_centroids[, 6])], 1)

#sim_run_cluster(replicate_cycle = NAI,
#                myWorld, number_of_time_steps = number_of_time_steps_a,
#                nbs, number_of_tips = nrow(myWorld), number_of_neighbors= number_of_neighbors, #origin
```

6.2 Run simulations for a specified time but run stats and save timesteps along the way

Here is the second version

```
######################################################################

# Run the full model in a cluster. This version writes files to a cluster output folder.
# rm(list = ls())
# install.packages("~/Desktop/FARM_1.0.tar.gz", repos=NULL, type="source")
```

```
#########################################################################
## need to document which functions we use from each of these libraries.
library(ape)
library(spdep)
library(Rcpp)
library(msm)
library(FARM)


sim_run_cluster <- function(replicate_cycle, myWorld, number_of_time_steps, nbs,
                            number_of_tips, number_of_neighbors, origins, start = NULL) {
  # Calls the full simulation script
  #
  # Purpose: Need to wrap the entire simulation script into a function so it can be called in parallel
  #
  # Args:
  #    replicate_cycle: An integer indicating the replicate number of a simulation. This variable is us
  #         the saved output file and control the number of replicates run by the cluster.
  #
  #    combo_number: An interger between 1 and 31 indicating the combinations of S, E, A, D, and T modu
  #         in the simulation. The full list of these combinations can be printed using the function co
  #         We are currently using combinations 25,28,29,and 31 as our four competing models for the sp
  #
  #    myWorld: Matrix that defines the scope of the available world and acts as a data hub for organiz
  #         results from the different elements of the simulation.
  #
  #    number_of_time_steps: An integer indicating how many iterations the simulation will calculated b
  #         file.
  #
  #    nbs: A list of the available neighbors for each spatial point. This is passed to the function fo
  #         of neighbors through time.
  #
  #    number_of_tips: An interger indicating the number of tree tips the simulation should be truncate
  #         include all the available tips (e.g. 1254 for human languages).
  #
  # Returns:
  #    myOut: A list object containing a 'phylo' tree object called mytree in the first position and th
  #         spatial and tree data in the second position
  #


  x1 <- 4 #Number of runs per core
  sampleer <- sample(c(1,2,5,6), x1)
  #if (replicate_cycle != 1) {
  #  replicate_cycle <- ((replicate_cycle - 1) * x1) + 1
 # }
 # replicate_cycle <- replicate_cycle:(replicate_cycle + (x1 - 1))
  for (count in sampleer) {
  independent <- 1


    # Probability of Arisal
```

```
prob_choose_a <- rev(sort(rexp(4, rate = 9)))
prob_choose_a <- prob_choose_a[c(sample(1:2, 2), sample(3:4, 2))]
prob_choose_a[3] <- 0
P.Arisal0  <- parameters(prob_choose_a[1], prob_choose_a[4],
                         prob_choose_a[3], prob_choose_a[2],
                         "Env_NonD", "Env_D",
                         "Evol_to_F", "Evol_to_D")
# P.Arisal0 is the one you should change the parameters
P.Arisal <- matrix(NA, ncol = 2, nrow = nrow(myWorld)) # probability per cell
colnames(P.Arisal) <- c("Evolve_to_F", "Evolve_to_D")
Env.Dom <- myWorld[, 7] == 2
P.Arisal[Env.Dom, 1] <- P.Arisal0[1, 2]
P.Arisal[!Env.Dom, 1] <- P.Arisal0[1, 1]
P.Arisal[Env.Dom, 2] <- P.Arisal0[2, 2]
P.Arisal[!Env.Dom, 2] <- P.Arisal0[2, 1]

colnames(P.Arisal) <- c("Prob_of_Foraging", "Prob_of_Domestication")
P.Arisal[which(origins == FALSE), 2]  <- 0

#####
#prob_choose <- runif(12, 0.01, 1)
#sub <- (prob_choose[1] - 0.01)
#sub <- ifelse(sub < .1, .1, sub)
#prob_choose[c(4)] <- runif(1, 0.01, sub)
#prob_choose[c(5)] <- runif(1, 0.1, 1) # High extinction
#prob_choose[c(6)] <- runif(1, 0, (prob_choose[3] - 0.01))
#prob_choose[c(9, 10, 12)] <- runif(3, 0.01, prob_choose[11])

####
prob_choose <- runif(12, 0, 1)
top <- min(prob_choose[c(1,3)], na.rm=TRUE)
prob_choose[c(2)] <- runif(1, 0, top)

prob_choose[c(5)] <- runif(1, 0, prob_choose[c(2)])
prob_choose[c(6)] <- runif(1, 0, prob_choose[c(5)])
prob_choose[c(4)] <- runif(1, prob_choose[c(6)], prob_choose[c(5)])


if (count == 1) {
  prob_choose[7:12] <- 0
}
if (count == 2) {
  prob_choose[9:12] <- 0
}
if (count == 3 | count == 5) {
  prob_choose[7:8] <- 0
  independent <- 0
}
if (count == 4 | count == 6) {
  independent <- 0
}
```

```
    P.speciation <- parameters(prob_choose[1], prob_choose[1],
                               prob_choose[2], prob_choose[3],
                               "Env_NonD", "Env_D", "For", "Dom")

    P.extinction  <- parameters(prob_choose[4], prob_choose[4],
                                prob_choose[5], prob_choose[6],
                                "Env_NonD", "Env_D", "For", "Dom")


    P.diffusion <- parameters(0, prob_choose[7],
                              prob_choose[8], 0,
                              "Target_For", "Target_Dom",
                              "Source_For", "Source_Dom")

    P.TakeOver <- parameters(prob_choose[9], prob_choose[10],
                             prob_choose[11], prob_choose[12],
                             "Target_For", "Target_Dom",
                             "Source_For", "Source_Dom")
    multiplier <- 1 # always 1 now.
   if (count %in% 1:4) {
        myOut <- RunSimUltimate2(myWorld, P.extinction, P.speciation, P.diffusion, P.Arisal,
    P.TakeOver, nbs, independent, number_of_time_steps, multiplier, silent = TRUE,
    count, resolution = seq(1, number_of_time_steps, 100), P.Arisal0, start = NULL)
                             }
   if (count %in% 5:6) {
        myOut <- RunSimUltimate2.push(myWorld, P.extinction, P.speciation, P.diffusion, P.Arisal,
    P.TakeOver, nbs, independent, number_of_time_steps, multiplier, silent = TRUE,
    count, resolution = seq(1, number_of_time_steps, 100), P.Arisal0, start = NULL)
                             }


  }

}

#############################################################################
coords <- as.matrix(apply(language_centroids[, 3:4], 2, as.numeric)) #coords
conds <- ifelse(suitability2 == 0, 1, 2)
conds[is.na(conds)] <- sample(c(1, 2), sum(is.na(conds)), replace = TRUE)
origins <- language_centroids[, 5]

##### Specify simulation parameters ##################################

number_of_tips <- length(coords[,1])
number_of_time_steps_a <- 50000
#replicate_cycle <- c(1)  #number of replicates
#############################################################################
data("parameters.table")


system.time(
  myWorld <- BuildWorld(coords, conds)
)
```

```
number_of_neighbors <- sample(5:9,1)

nbs <- knn2nb(knearneigh(coords, k = number_of_neighbors, longlat = TRUE),
              sym = TRUE) # 7 symmetric neighbors
n.obs <- sapply(nbs, length)
seq.max <- seq_len(max(n.obs))
nbs <- t(sapply(nbs, "[", i = seq.max))

dim(myWorld)


# NAI <- 1000
args <- commandArgs(trailingOnly = FALSE)
print(args)
NAI <- as.numeric(args[7])
#setwd("~/Box Sync/colliding ranges/Simulations_humans/big world cluster outputs")


# Starting point
start <- sample((1:nrow(language_centroids))[as.logical(language_centroids[, 6])], 1)

sim_run_cluster(replicate_cycle = NAI,
                myWorld, number_of_time_steps = number_of_time_steps_a,
                nbs, number_of_tips = nrow(myWorld), number_of_neighbors= number_of_neighbors, origins=
```

Chapter 7

Running Module 1 and Module 2 on the cluster

7.1 Not for the faint of hart

7.2 Bash scripts

Running an R script on the cluster requires two parts: an R script with the code to be run and a PBS script to control how that R script is run on the cluster.

There are two different clusters at Wustl.edu, an old and a new cluster. This script runs the R code on the new cluster.

```
#!/bin/bash
#PBS -N Four_model_run
#PBS -V
#PBS -l walltime=23:59:00
#PBS -l pmem=1200mb
#PBS -l nodes=1:ppn=1:haswell
#PBS -t 1-1000


echo $PBS_ARRAYID

cd /home/cbotero/mydirectory/Four_model_compare
module load R

export R_LIBS=$HOME/rlibs
#R CMD INSTALL --library=/home/ttuff/rlibs  FARM_1.0.tar.gz

Rscript --vanilla ./FARM_four_model_compare.R ${PBS_ARRAYID}
```

We were regularly causing problems running to many jobs on the new cluster and we were asked to move to the old cluster. This cluster has slower individual processors, but we can run more jobs at one time, so productivity has stayed about the same.

```
#!/bin/bash
#PBS -N FARM_third_run_old
```

```
#PBS -V
#PBS -l walltime=160:00:00
#PBS -l pmem=1200mb
#PBS -q old
#PBS -l nodes=1:ppn=1:nehalem
#PBS -t 1-500


echo $PBS_ARRAYID

cd /home/ttuff/mydirectory/Four_model_compare
module load R

export R_LIBS=$HOME/rlibs
#R CMD INSTALL --library=/home/ttuff/rlibs  FARM_1.0.tar.gz

Rscript --vanilla ./FARM_four_model_compare.R ${PBS_ARRAYID}
```

7.3 Passing arguements to R-script

The final argument in the #PBS script above (#PBS -t 1-500) controls the serial running schema for running many simultanious instances of the R script at a time. This argument is passes to R as an integer value using the following arguements inside R.

```
args <- commandArgs(trailingOnly = FALSE) #7 elements are passed from the PBS

NAI <- as.numeric(args[7]) # the seventh of those elements is the array integer.
```

Here is a working example of how to set up the R script.

```
#install.packages("rfoaas")
library(rfoaas)

##If the PBS script started this code running and passed the number 13
##to this particular run of a larger serial set.


#args <- commandArgs(trailingOnly = FALSE)

NAI <- 13 #as.numeric(args[7])

    sayHello <- function(loop_number){
      print(paste0("I can count to ", loop_number, "!   ", cool(from="Ty")))
    }

    sayHello(NAI)
```

```
## [1] "I can count to 13!   Cool story, bro. - Ty"
```

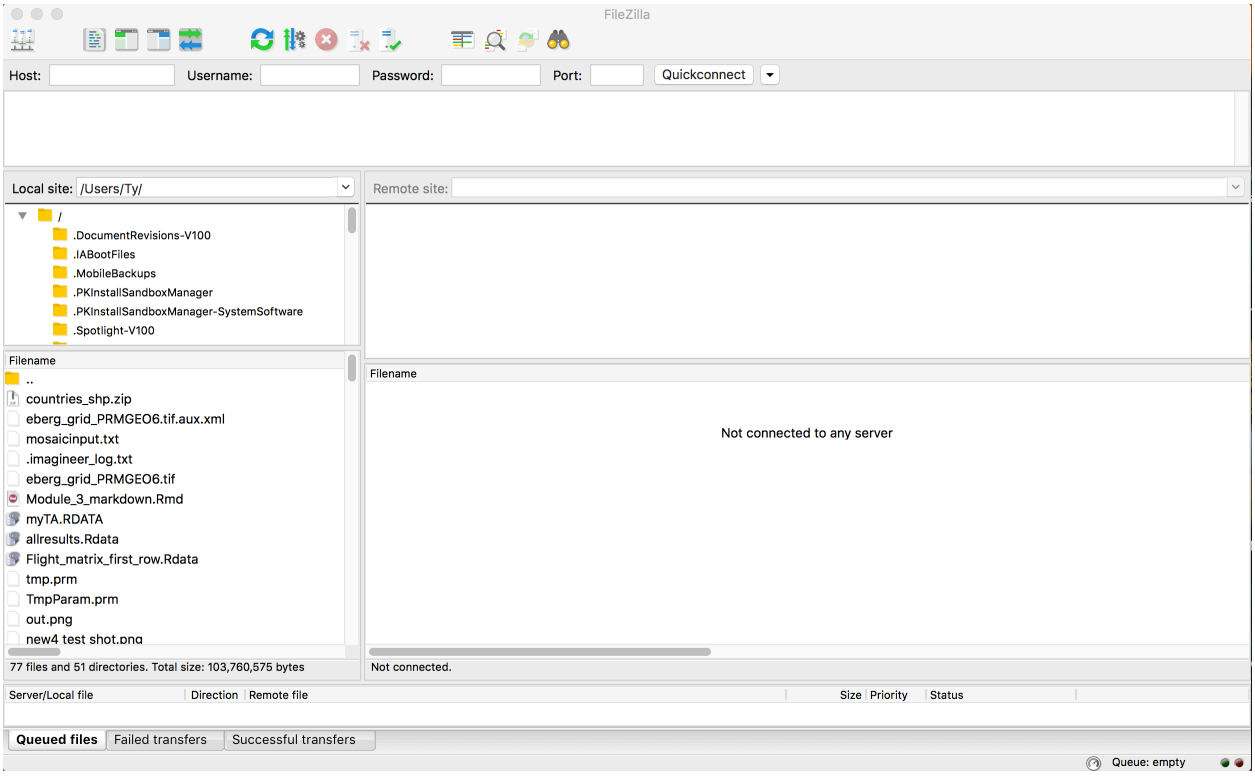


Figure 7.1: A fresh Filezilla window

7.4 Accessing and working with the cluster

You logon to the cluster using linux/unix code from the command line terminal on you computer. Open the terminal and put in you login info.

```
ssh -Y ttuff@login.chpc.wustl.edu
password:_______
```

Upon first login, you will be in a folder called 'HOME' with a series of system files in it. You will want to use an FTP client to view and organize these files. I prefer Filezilla, but there are several other good clients available for free. Download filezilla, make sure it's in your applications folder, and open it. You should see a window that looks like a newer version of this. The left panels are the files on the computer you're working from and the right two panels will show the files on the server once you log in through Filezilla also.

Once logged in, you need to change the directory

```
cd /home/ttuff/mydirectory/Four_model_compare
```

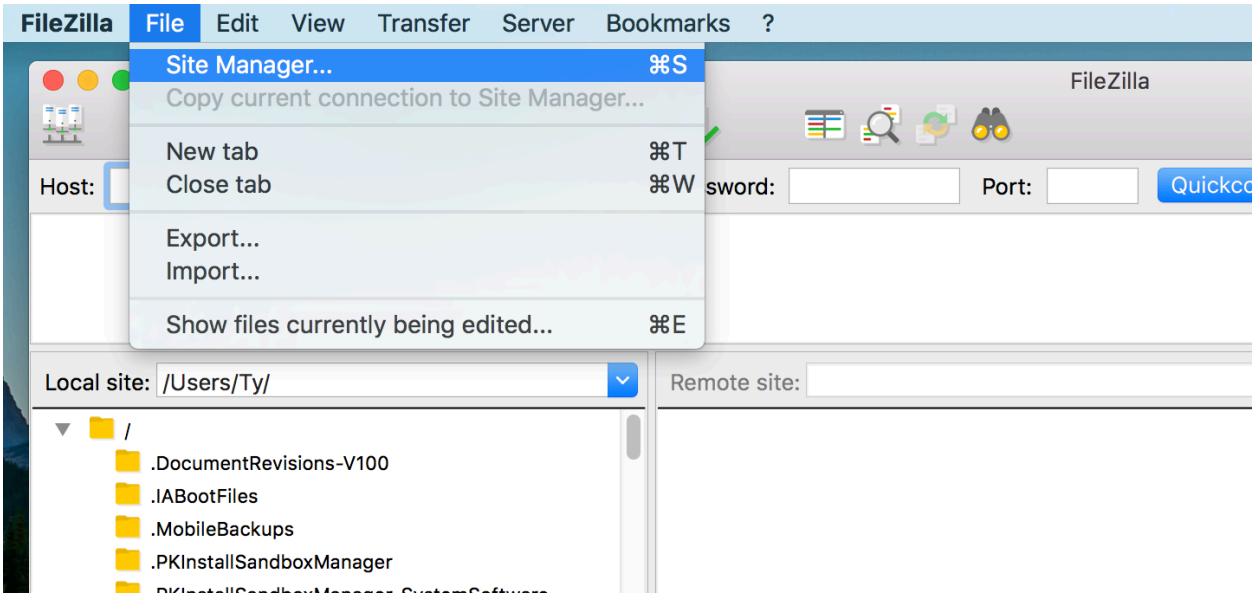


Figure 7.2: Start a new server link

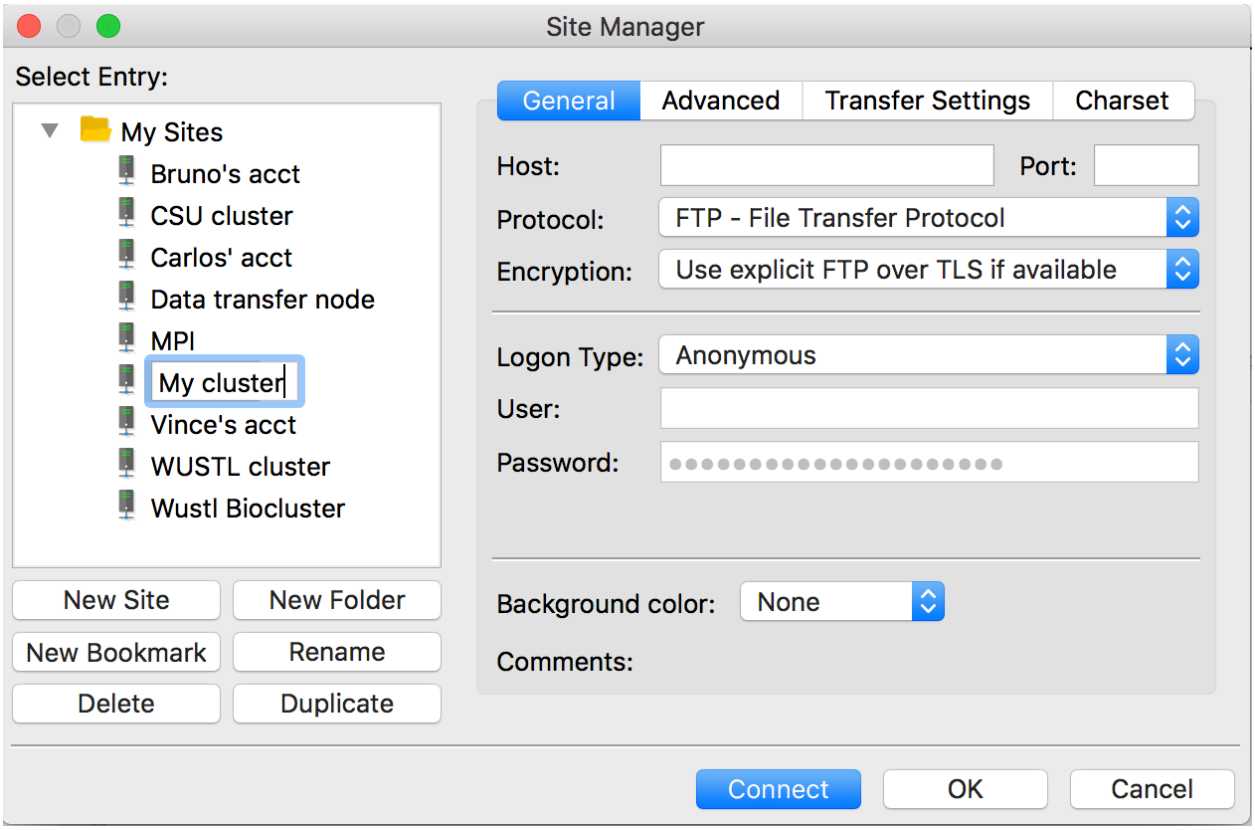


Figure 7.3: Start a new site and name that new site

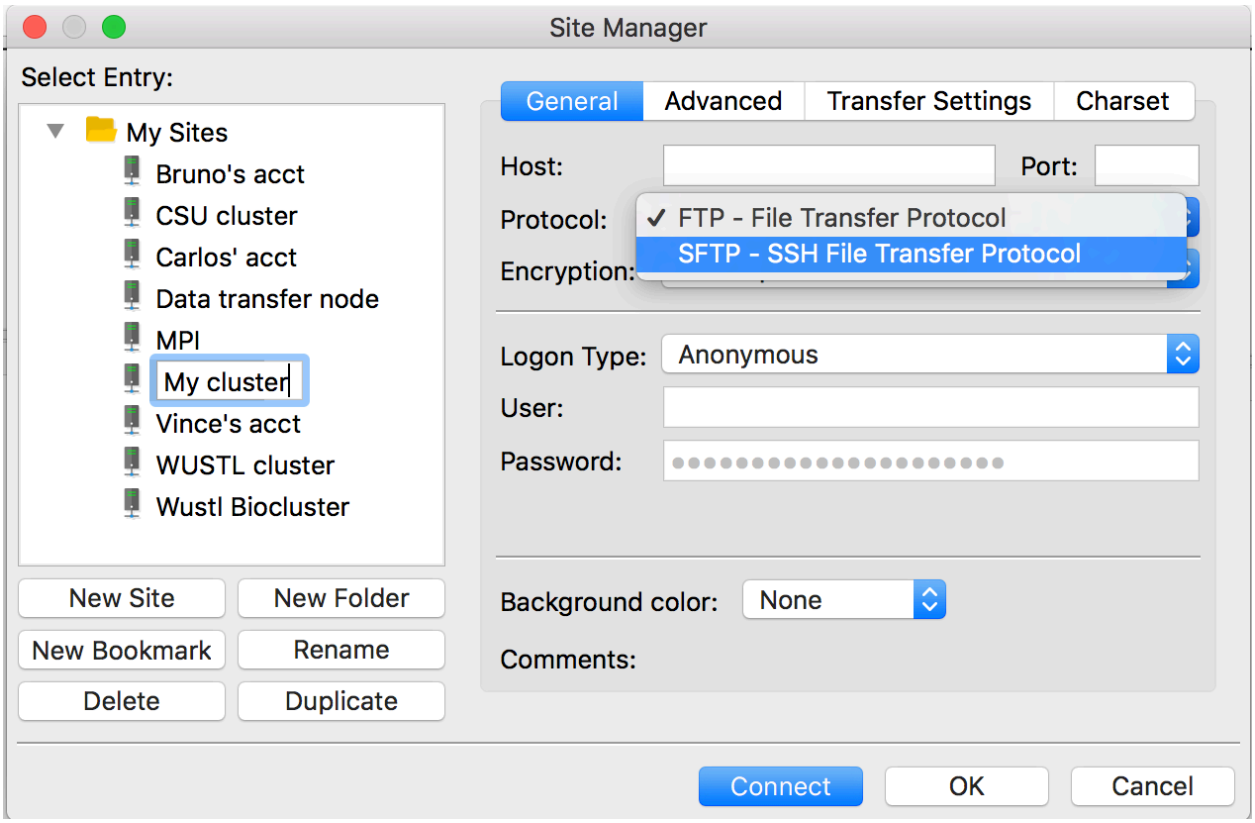


Figure 7.4: Select a secure ssh file transfer protocol

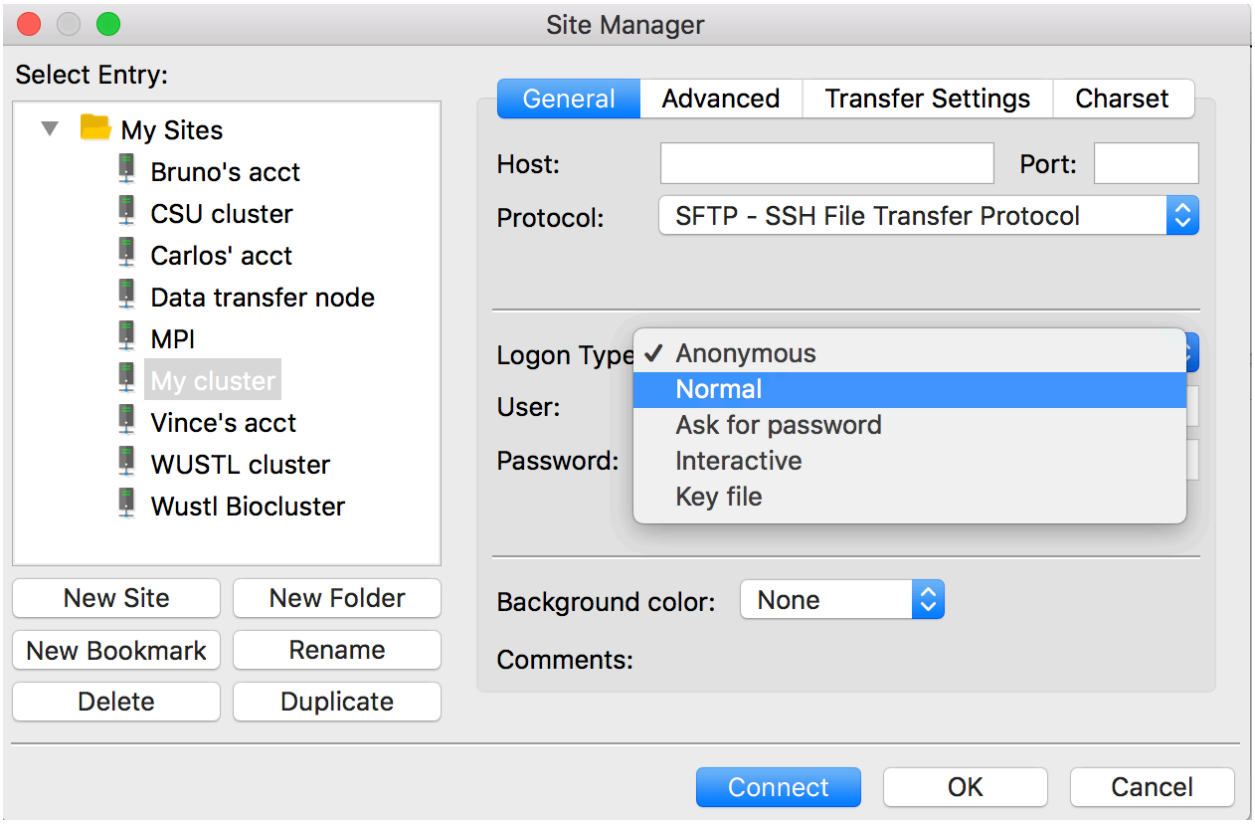


Figure 7.5: Login as normal

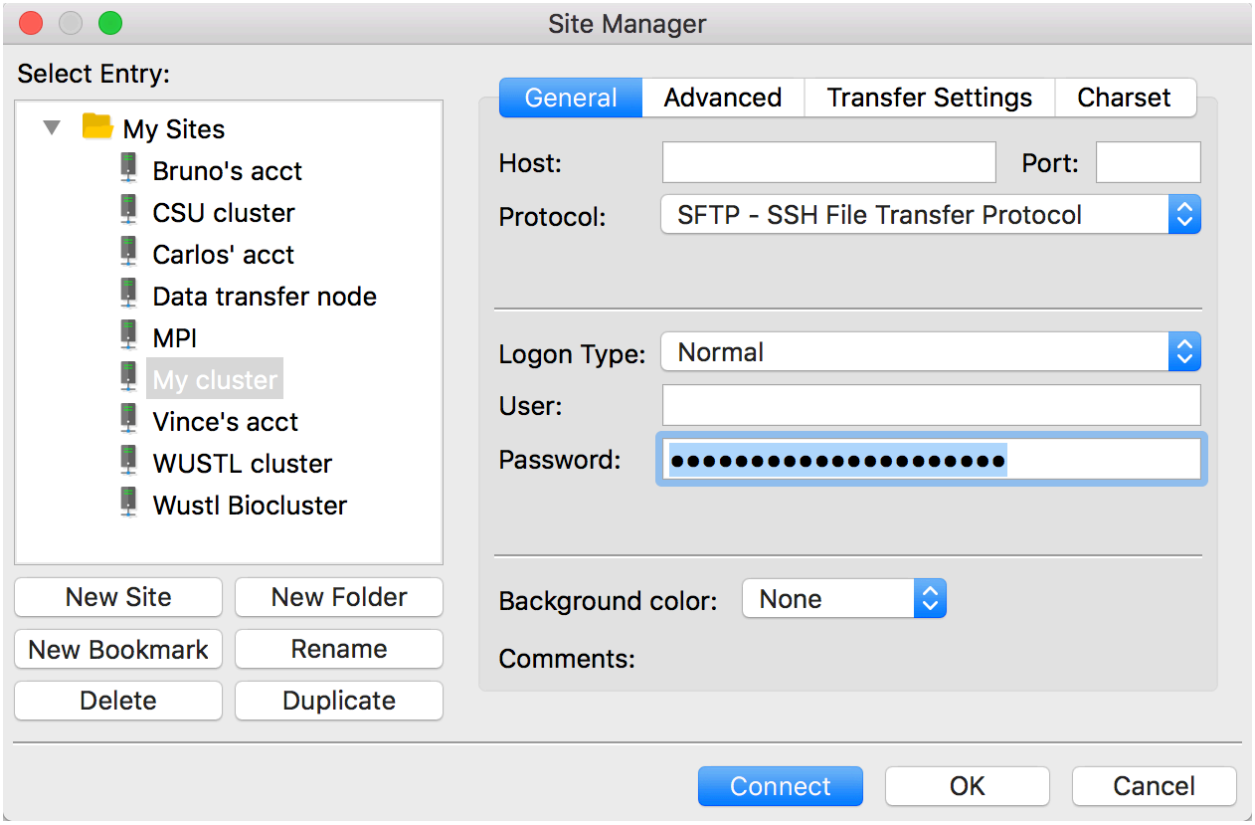


Figure 7.6: Enter password

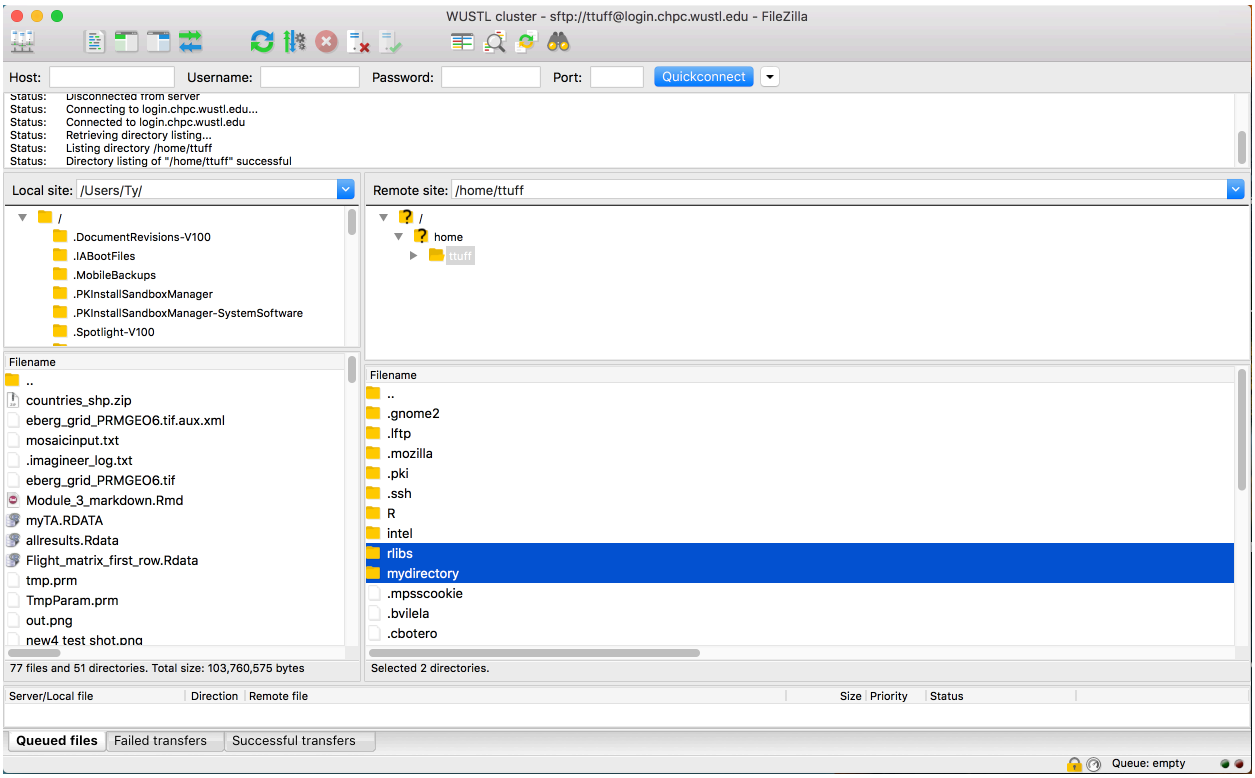


Figure 7.7: You need to create two new directories (folders) for us to work out of. Call one 'rlibs' and the other 'mydirectory'

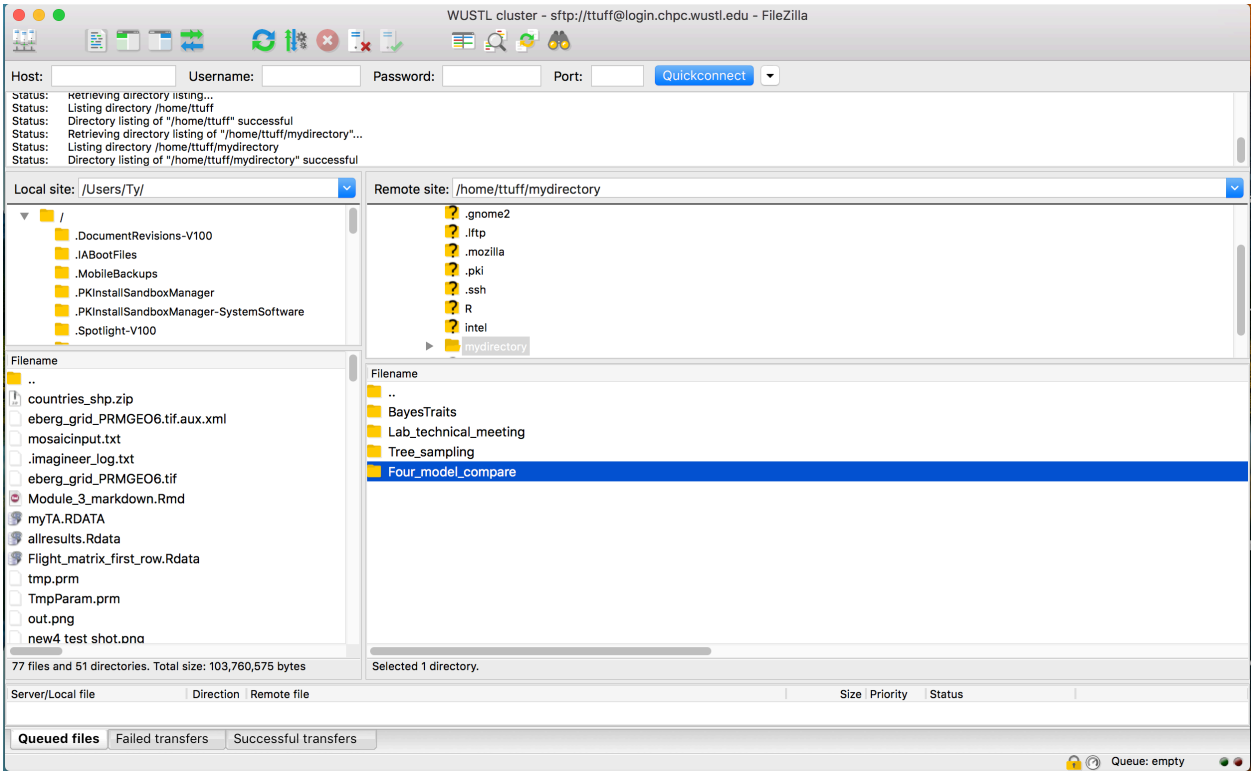


Figure 7.8: Within mydirectory, create another folder called 'Four_model_compare'

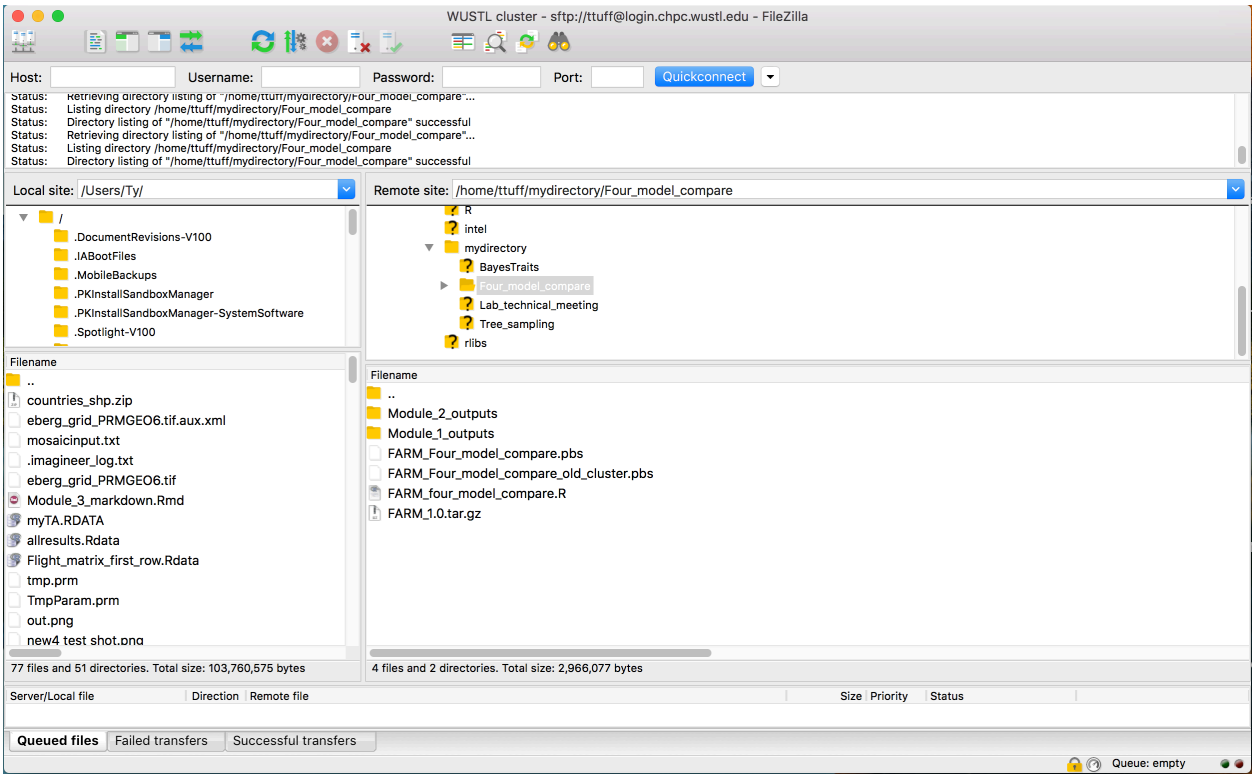


Figure 7.9: Within 'Four_model_compare', create two folders called 'Module_1_outputs' and 'Module_2_outputs'. Then drag three files from your computer files on the left to the server folders on the right: one .R file, one .pbs file, and a zip file with the package FARM in it. These should be named FARM_four_model_compare.R, FARM_four_model_compare_old_cluster.pbs, and FARM_1.0.tar.gz .

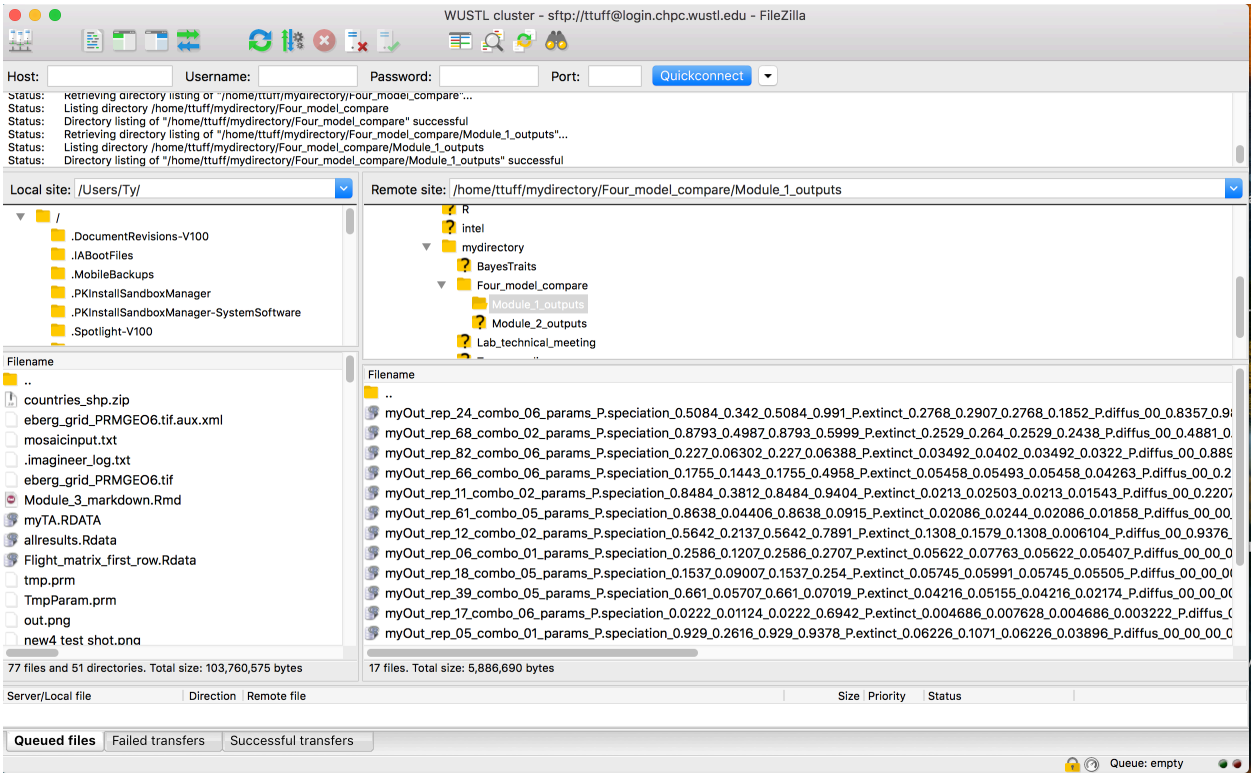


Figure 7.10:

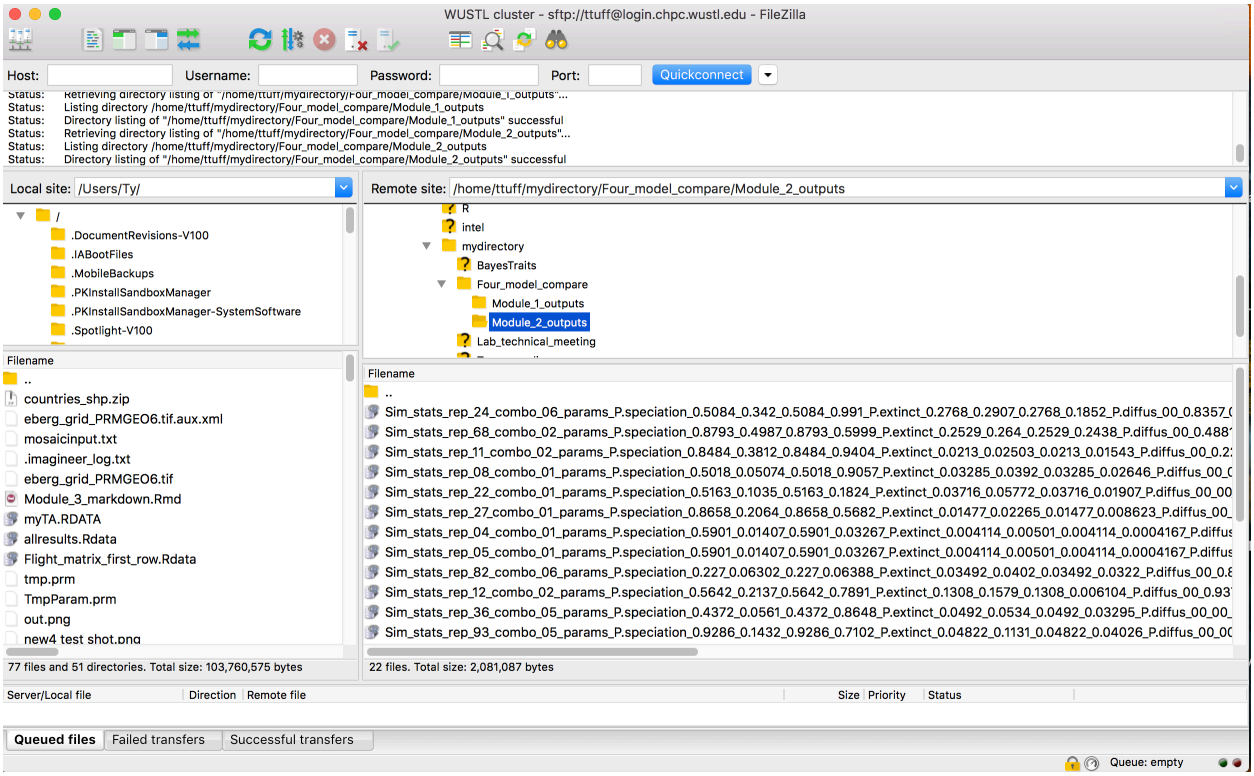


Figure 7.11:

Chapter 8

Module 3: Analysing results produce from Modules 1 and 2

```
library(png)
```

8.1 Organize data

```
# A frequent scenario in our analysis code is that we need to read thousands of
# output files into a single table.
# There are a couple flavors of file name that need to be dealt with and the
# output file needs to include a column with the full file name for reference
# later. We want to save the final table as it's own file to a folder outside of
# the one we just read.

## Here is one flaver where the input is two folders, one with extant
# simulations and one with extinct simulations. Need to read the files and
# produce two tables, one for extinct and one for extant.

## First consolidate the available files into a single table
concatenate <- function(path) {
 available <- list.files(path, full.names = TRUE)
 name <- unlist(strsplit(available[1], split="_"))
 n <- length(available)
 n_name <- length(name) - 1
 load(available[1])
 ncol_files <- length(Sim_statistics[[1]]) + n_name
 files <- matrix(nrow = n,
                 ncol = ncol_files)
 split_names <- function(x){unlist(strsplit(x, split = "_"))}
 name_available <- do.call(rbind, lapply(as.list(available), split_names))
 files[, 1:n_name] <- name_available[, -ncol(name_available)]

 for (i in 1:n) {
   error <- try(load(available[i]), silent = TRUE)
   if (class(error) != "try-error") {
```

```
    files[i, (n_name + 1):ncol_files] <- Sim_statistics[[1]]
  }
}


files <- files[, -c(1, 2, 3, 5, 7, 8, 13, 18, 23, 28, 33, 35)]
last_name <- colnames(Sim_statistics[[1]])
if (is.null(last_name)) {

  files <- files[, -ncol(files)]
}
colnames(files) <-  c("Sim_stats_rep", "combo", paste0("P.speciation", 1:4),
                      paste0("P.extinct", 1:4),  paste0("P.diffus", 1:4),
                      paste0("P.TO", 1:4),  paste0("P.Arisal", 1:4),
                      "timesteps", "NBS", last_name)

Concatenated_data <- as.data.frame(files)

begin <- which(colnames(Concatenated_data) ==  "number_of_branches")
Concatenated_data_stat <- Concatenated_data[, begin:ncol(Concatenated_data)]
Concatenated_data_stat <- apply(Concatenated_data_stat, 2, as.numeric)
remove <- which(is.na(rowSums(Concatenated_data_stat)))
Concatenated_data <- Concatenated_data[-remove, ]

one <- subset(Concatenated_data, combo =="01")
two <- subset(Concatenated_data, combo =="02")
five <- subset(Concatenated_data, combo =="05")
six <- subset(Concatenated_data, combo =="06")

crop <- min(sapply(list(one, two, five, six), nrow))

one <- one[1:crop, ]
two <- two[1:crop, ]
five <- five[1:crop, ]
six <- six[1:crop, ]

Concatenated_data2 <- rbind(one, two, five, six)
res <- list(Concatenated_data, Concatenated_data2, crop)
names(res) <- c("Concatenated_data", "Concatenated_data_crop", "crop")
return(res)

}

path <- "~/Box Sync/Four model compare third run/Module 2"
Concatenated_data0 <- concatenate(path)
print(Concatenated_data0[[3]])
path_save <- "~/Box Sync/Four model compare third run"
Concatenated_data <- Concatenated_data0[[1]]
save(Concatenated_data, file = paste0(path_save,
                                      "/Four_model_compare_results_not_cropped.Rdata"))
Concatenated_data <- Concatenated_data0[[2]]
save(Concatenated_data, file=paste0(path_save,
                                    "/Four_model_compare_results_",
```

```
                                    format(Sys.time(), format="%d_%b_%Y"),
                                    "_crop_to_", Concatenated_data0[[3]],".Rdata"))

### Repeated for extinct
path <- "~/Box Sync/Four model compare third run/Module 2 extinct"
Concatenated_data <- concatenate(path)[[1]]
path_save <- "~/Box Sync/Four model compare third run"

save(Concatenated_data, file=paste0(path_save,
                                    "/Four_model_compare_results_extinct_",
                                    format(Sys.time(), format="%d_%b_%Y"),".Rdata"))
```

8.2 Diagnostics

```
load("Four_model_compare_results_02_Aug_2017_crop_to_6128.Rdata")
extant <- Concatenated_data
extant

 load("Four_model_compare_results_extinct_02_Aug_2017.Rdata")
extinct <- Concatenated_data
extinct

head(extant)
head(extinct)
```

```
for(i in c(3:22)){
    extinct[which(is.nan(as.numeric(as.character(extinct[, i])))) == TRUE), i] <- NA
}

for(i in c(3:22)){
    extant[which(is.nan(as.numeric(as.character(extant[, i])))) == TRUE), i] <- NA
}




xlimit <- c(0,1)
ylimit <- c(0,3000)
maincex <- 0.9

png(file="Global_success_rate_per_parameter.png", width=8.5, height=11, units="in", res=300)

par(mfrow=c(5,4), mar=c(3,3,3,0))


hist(as.numeric(as.character(extinct[,3])), main="speciation of F in F env", col=adjustcolor("firebrick
hist(as.numeric(as.character(extant[,3])), main="speciation of F in F env", col=adjustcolor("cornflower


hist(as.numeric(as.character(extinct[,4])), main="speciation of D in F env", col=adjustcolor("firebrick
hist(as.numeric(as.character(extant[,4])), main="speciation of D in F env", col=adjustcolor("cornflower
```

```
hist(as.numeric(as.character(extinct[,5])), main="speciation of F in D env", col=adjustcolor("firebrick
hist(as.numeric(as.character(extant[,5])), main="speciation of F in D env", col=adjustcolor("cornflowerb

hist(as.numeric(as.character(extinct[,6])), main="speciation of D in D env", col=adjustcolor("firebrick
hist(as.numeric(as.character(extant[,6])), main="speciation of D in D env", col=adjustcolor("cornflowerb

#######

hist(as.numeric(as.character(extinct[, 7])), main="extinction of F in F env", col=adjustcolor("firebric
hist(as.numeric(as.character(extant[, 7])), main="extinction of F in F env", col=adjustcolor("cornflowe


hist(as.numeric(as.character(extinct[, 8])), main="extinction of D in F env", col=adjustcolor("firebric
hist(as.numeric(as.character(extant[, 8])), main="extinction of D in F env", col=adjustcolor("cornflowe

hist(as.numeric(as.character(extinct[, 9])), main="extinction of F in D env", col=adjustcolor("firebric
hist(as.numeric(as.character(extant[, 9])), main="extinction of F in D env", col=adjustcolor("cornflowe

hist(as.numeric(as.character(extinct[, 10])), main="extinction of D in D env", col=adjustcolor("firebri
hist(as.numeric(as.character(extant[, 10])), main="extinction of D in D env", col=adjustcolor("cornflow

######

hist(as.numeric(as.character(extinct[, 11])), main="arisal of F in F env", col=adjustcolor("firebrick",
hist(as.numeric(as.character(extant[, 11])), main="arisal of F in F env", col=adjustcolor("cornflowerblu



hist(as.numeric(as.character(extinct[, 12])), main="arisal of D in F env", col=adjustcolor("firebrick",
hist(as.numeric(as.character(extant[, 12])), main="arisal of D in F env", col=adjustcolor("cornflowerblu

hist(as.numeric(as.character(extinct[, 13])), main="arisal of F in D env", col=adjustcolor("firebrick",
hist(as.numeric(as.character(extant[, 13])), main="arisal of F in D env", col=adjustcolor("cornflowerblu

hist(as.numeric(as.character(extinct[, 14])), main="arisal of D in D env", col=adjustcolor("firebrick",
hist(as.numeric(as.character(extant[, 14])), main="arisal of D in D env", col=adjustcolor("cornflowerblu

######

hist(as.numeric(as.character(extinct[, 15])), main="NOPE -- Diffusion: source F, target F", col=adjustco
hist(as.numeric(as.character(extant[, 15])), main="NOPE -- Diffusion: source F, target F", col=adjustcol



hist(as.numeric(as.character(extinct[, 16])), main="Diffusion: source D, target F", col=adjustcolor("fi
hist(as.numeric(as.character(extant[, 16])), main="Diffusion: source D, target F", col=adjustcolor("cor

hist(as.numeric(as.character(extinct[, 17])), main="Diffusion: source F, target D", col=adjustcolor("fi
hist(as.numeric(as.character(extant[, 17])), main="Diffusion: source F, target D", col=adjustcolor("cor

hist(as.numeric(as.character(extinct[, 18])), main="NOPE -- Diffusion: source D, target D", col=adjustco
hist(as.numeric(as.character(extant[, 18])), main="NOPE -- Diffusion: source D, target D", col=adjustcol
```

```
####

hist(as.numeric(as.character(extinct[, 19])), main="Takeover: source F, target F", col=adjustcolor("fir
hist(as.numeric(as.character(extant[, 19])), main="Takeover: source F, target F", col=adjustcolor("corn



hist(as.numeric(as.character(extinct[, 20])), main="Takeover: source D, target F", col=adjustcolor("fir
hist(as.numeric(as.character(extant[, 20])), main="Takeover: source D, target F", col=adjustcolor("corn

hist(as.numeric(as.character(extinct[, 21])), main="Takeover: source F, target D", col=adjustcolor("fir
hist(as.numeric(as.character(extant[, 21])), main="Takeover: source F, target D", col=adjustcolor("corn

hist(as.numeric(as.character(extinct[, 22])), main="Takeover: source D, target D", col=adjustcolor("fir
hist(as.numeric(as.character(extant[, 22])), main="Takeover: source D, target D", col=adjustcolor("corn


dev.off()
```

8.3 Random Forest Analysis

8.3.1 Training/Building

The procedures described so far have outlined the creation of a dataset that includes the input and output variables for each replicate simulation. When taken as a whole, that dataset describes a distribution of possible numerical outcomes that are possible given the 4 available input types. The purpose of the random forest machine learning algorithm (Breiman 2001, Liaw and Wiener 2015) was to correlate input and output variables as a means of categorizing simulation outputs according to the type of input that had created them. We used 70% of our available simulation data to build the random forest, the remaining 30% to test the forest's accuracy and precision at inferring input categories, and then used that trained and vetted algorithm to infer the most likely input category for real world human output data.

This random forest model was trained by building a forest of decision trees. Each decision tree describes a series of ordered dichotomous decisions that categorize the 12 output statistics into one of 4 input types. Individual trees were built by first taking a randomly sized and randomly selected sample of rows (replicate simulations) and columns (output variables) from the full simulation dataset to create a unique subset dataset. That unique dataset was then subjected to a bagging model to rank the variables in order of importance and build a decision tree off of that rank. We repeated this process, with replacement, to build 3000 individual decision trees sampled from 70% (n = 40000 x 0.7 = 28000) of the total simulation data.

8.3.2 Testing

The random forest was vetted using the remaining 30% of the simulation data reserved for model testing. Rather than building decision trees with these data, we fed each replicate through the recently built forest so it could classify outputs and then compare those inferred input types to the known input types to the to see how frequently the forest returned the correct classification. The accuracy and precision of these tests are reported in a confusion matrix showing the number of matches that were correctly or incorrectly made between inputs and outputs for each model . Our trained random forest algorithm could correctly match output statistics with input mechanisms with very good accuracy . It is noteworthy that there was very little confusion between the two competing hypothesized mechanisms, diffusion and diffusion by takeover. If

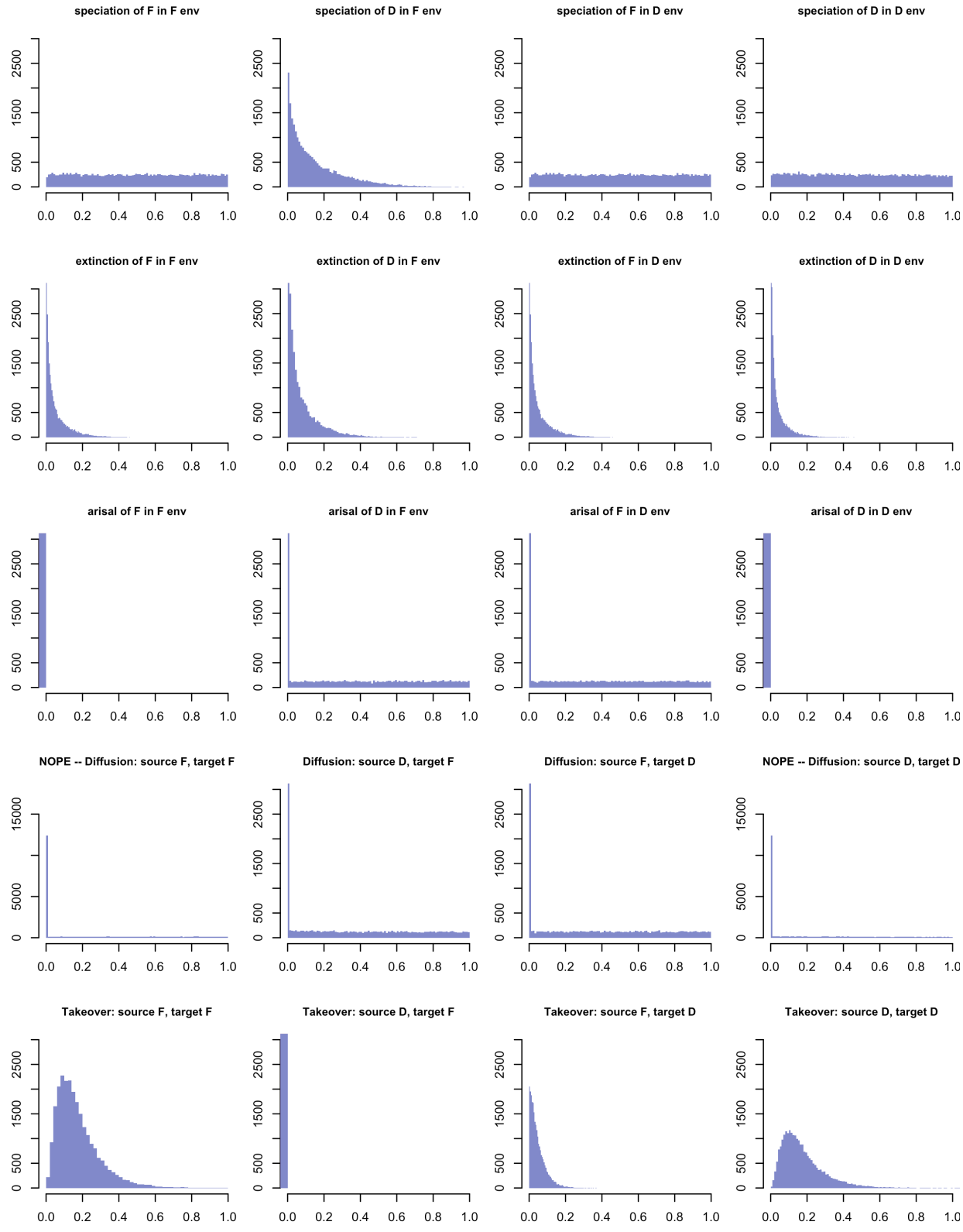

speciation of F in F env
speciation of D in F env
speciation of F in D env
speciation of D in D env
extinction of F in F env
extinction of D in F env
extinction of F in D env
extinction of D in D env
arisal of F in F env
arisal of D in F env
arisal of F in D env
arisal of D in D env
NOPE -- Diffusion: source F, target F
Diffusion: source D, target F
Diffusion: source F, target D
NOPE -- Diffusion: source D, target D
Takeover: source F, target F
Takeover: source D, target F
Takeover: source F, target D
Takeover: source D, target D

Figure 8.1:

the trained random forest algorithm is presented with a simulation created by diffusion, if is very unlikely to be classify it as coming from diffusion by takeover.

8.3.3 Matching

The structure of our simulation outputs where designed to mimic the structure of our real cultural data so that the two could be compared directly. In the same way the output for replicate simulations where processed, the phylogeny and spatial pattern of extant human cultures were summarized using 12 summary statistics passed to the trained random forest algorithm. The random forest assigned the most likely mechanism to have caused that human's known historical trajectory. It does this by asking every dichotomous decision tree in the random forest to vote on which hypothesized mechanism created the output provided by the data and then tallying those votes into a consensus vote. Our consensus vote showed that over 50% of the trees believe that the current configuration of agriculture across the globe is the result of demic diffusion augmented by cultural diffusion. About 35% of the trees assigned known global trend to the basic model that includes only demic diffusion, while the other two models received almost no votes. This suggest that if we are wrong that demic and cultural diffusion contributed to current distributions of agriculturist cultures, the next most likely answer is that it diffusion created these patterns with no help from cultural diffusion.

8.4 Run a single random forest on available outputs

```
# Packages
library(randomForest)

# Radom Forest Function
RF <- function(table_sim, table_real, ntree = 2000, stats_remove = NULL,
               repetitions = 100) {
 # table_sim = table croped

 begin <- which(colnames(table_sim) ==  "number_of_branches")
 table_sim_data <- table_sim[, begin:ncol(table_sim)]
 table_sim_data <- apply(table_sim_data, 2, as.numeric)
 if (any(is.infinite(table_sim_data))) {
   stop("There is infinite values in the simulation statistics")
 }
 rf_data <- data.frame("Model" = table_sim$combo, table_sim_data)
 rf_data_real <- data.frame(table_real)

 if (!is.null(stats_remove)) {
   rf_data <- rf_data[, -stats_remove]
   rf_data_real <- rf_data_real[, -stats_remove]
 }

 fun <- function(x, y, per = .33) {
   sample(which(y$Model == x), round(table(y$Model)[1]*per))
 }

 results <- matrix(nrow = repetitions, ncol = length(table_real) * 6 + 21)
 for (i in 1:repetitions) {
   sub.test <- unlist(lapply(as.list(paste0(0, c(1,2,5,6))), fun,
                             y = rf_data))
   test2 <- rf_data[sub.test, 2:ncol(rf_data)]
```

```
  test1 <- rf_data[sub.test, 1]
  train <- rf_data[-sub.test, ]

  fit <- randomForest(Model ~ ., data = train, xtest = test2,
                      ytest = test1, importance = TRUE,
                      ntree = ntree, keep.forest = TRUE,
                      replace = TRUE)
 #dev.new()
 #plot(fit)
  predictions <- predict(fit, rf_data_real, type = "prob")
  var_import <- importance(fit)

  error <- mean(fit$test$confusion[, 5])
  confusion <- as.numeric(fit$test$confusion[, 1:4])
  names(confusion) <- paste0("Confusion_",
                             rep(colnames(fit$test$confusion[, 1:4]),
                                 each = 4),
                             rownames(fit$test$confusion[, 1:4]))
  predictions_prob <- as.numeric(predictions)
  names(predictions_prob) <- paste0("Prediction_prob", colnames(predictions))
  var_import_vec <- as.numeric(var_import)
  names(var_import_vec) <- paste0("var_imp_",
                                  rep(colnames(var_import),
                                      each = nrow(var_import)),
                                  "_",
                                  rownames(var_import))
  vec <- c(error, confusion, predictions_prob, var_import_vec)
  results[i, ] <- vec
}
colnames(results) <- names(vec)
colnames(results)[1] <- "Error_test"
results <- cbind(1:nrow(results), results)
colnames(results)[1] <- "Replicate"
return(results)
}

# Multiple Random Forest
RF_mult <- function(table_sim, table_real_mult, ntree = 2000,
                    stats_remove = NULL, repetitions = 100) {
n_r <- nrow(table_real_mult)
n <- repetitions * n_r
n_col <- ncol(table_real_mult) * 6 + 22
results <- matrix(nrow = n, ncol = n_col)
x <- 0
for (i in seq(1, n, repetitions)) {
  x <- x + 1
  temp <- RF(table_sim,
             table_real_mult[x, , drop = FALSE],
             ntree = ntree,
             repetitions = repetitions,
             stats_remove = stats_remove)
  results[i:(i + repetitions - 1), ] <- temp
}
```

```
 tree <- rep(1:n_r, each = repetitions)
 results <- cbind(tree, results)
 colnames(results) <- c("Tree", colnames(temp))
 return(results)
}

# Accumulation function
RF_acum <- function(table_sim, table_real_mult, ntree = 2000,
                    stats_remove = NULL, repetitions = 100,
                    resolution = 100, minimun = 100) {
 sequence <- seq(minimun, nrow(table_sim), resolution)
 n_col <- ncol(table_real_mult) * 6 + 24
 n_r <- nrow(table_real_mult)
 n_seq <- length(sequence)
 n <- repetitions * n_r * n_seq
 results <- matrix(nrow = n, ncol = n_col)
 lottery1 <- which(table_sim$combo == "01")
 lottery2 <- which(table_sim$combo == "02")
 lottery5 <- which(table_sim$combo == "05")
 lottery6 <- which(table_sim$combo == "06")
 sub1 <- sample(lottery1, sequence[1]/4, replace = FALSE)
 sub2 <- sample(lottery2, sequence[1]/4, replace = FALSE)
 sub5 <- sample(lottery5, sequence[1]/4, replace = FALSE)
 sub6 <- sample(lottery6, sequence[1]/4, replace = FALSE)
 for (i in 1:n_seq) {
   if (i != 1) {
     sub1 <- c(sub1, sample(lottery1[-sub1], sequence[1]/4, replace = FALSE))
     sub2 <- c(sub2, sample(lottery2[-sub2], sequence[1]/4, replace = FALSE))
     sub5 <- c(sub5, sample(lottery5[-sub5], sequence[1]/4, replace = FALSE))
     sub6 <- c(sub6, sample(lottery6[-sub6], sequence[1]/4, replace = FALSE))
   }
   temp <- RF_mult(table_sim[c(sub1, sub2, sub5, sub6), ], table_real_mult,
                   ntree = ntree, repetitions = repetitions,
                   stats_remove = stats_remove)
   n_temp <- nrow(temp)
   results[1:n_temp + (n_temp * (i - 1)), ] <- cbind(rep(sequence[i],
                                                         n_temp), temp)
 }
 colnames(results) <- c("Subsample_size", colnames(temp))
 return(results)
}

# TEST FINAL (RUN ONLY THIS)
# Load data
setwd("~/Desktop/RF after bruno")
load("Four model compare third runFour_model_compare_results_28_Jul_2017_crop_to_5767.Rdata")
load("real.analysis_mult.RData")

table_sim = Concatenated_data
table_real = real.analysis.mult[[1]][[1]]
table_real_mult = do.call(rbind, lapply(real.analysis.mult,
                                        function(x){x[[1]]}))
```

```
#start time
time_start <- Sys.time()

# Change the parameters as you wish
res_acum <- RF_acum(table_sim, table_real_mult[100:110,,drop=FALSE], ntree = 2000,
                    stats_remove = NULL, repetitions = 1,
                    resolution = 5000, minimun = 1000)

#stop time
time_stop <- Sys.time()
difftime(time_stop, time_start)

save(res_acum, file="~/Desktop/10_more_tree_out.Rdata")

# Change argument FUN to sd to obtain the standard deviation
plot(aggregate(x = res_acum[, 4], by = list((res_acum[, 1])), FUN = mean), type = "n", ylim=c(0,1))

lines(aggregate(x = res_acum[, 21], by = list((res_acum[, 1])), FUN = mean), type = "l", col="orange")
lines(aggregate(x = res_acum[, 22], by = list((res_acum[, 1])), FUN = mean), type = "l", col="blue")
lines(aggregate(x = res_acum[, 23], by = list((res_acum[, 1])), FUN = mean), type = "l", col="pink")
lines(aggregate(x = res_acum[, 24], by = list((res_acum[, 1])), FUN = mean), type = "l", col="darkgreen
```

8.5 Visualize outputs from Random Forest analysis

```
# load outputs from RF runs
load("/Users/Ty/Desktop/small_RF_out.Rdata")
a <- res_acum

load("/Users/Ty/Desktop/10_more_tree_out.Rdata")
b <- res_acum

load("/Users/Ty/Desktop/10_tree_out.Rdata")
c <- res_acum

#object called res_acum
res_acum <- rbind(b,c)

load("Random_Forest_output_data_for_publication_2_February_2018.Rdata")
```

```
png("overall_error_per_sample_size.png", width = 11, height = 8.5, res = 300, units = "in")
plot(aggregate(x = res_acum[, 4], by = list((res_acum[, 1])), FUN = mean), type="l", ylim=c(0,0.5), ylab
dev.off()
```

```
png("predictions.png", width = 11, height = 8.5, res = 300, units = "in")
par(mar=c(5,6,2,1))

# Change argument FUN to sd to obtain the standard deviation
plot(aggregate(x = res_acum[, 4], by = list((res_acum[, 1])), FUN = mean), type = "n", ylim=c(0,1), ylab


basic_mean <- aggregate(x = res_acum[, 21], by = list((res_acum[, 1])), FUN = mean)
diffusion_mean <- aggregate(x = res_acum[, 22], by = list((res_acum[, 1])), FUN = mean)
```

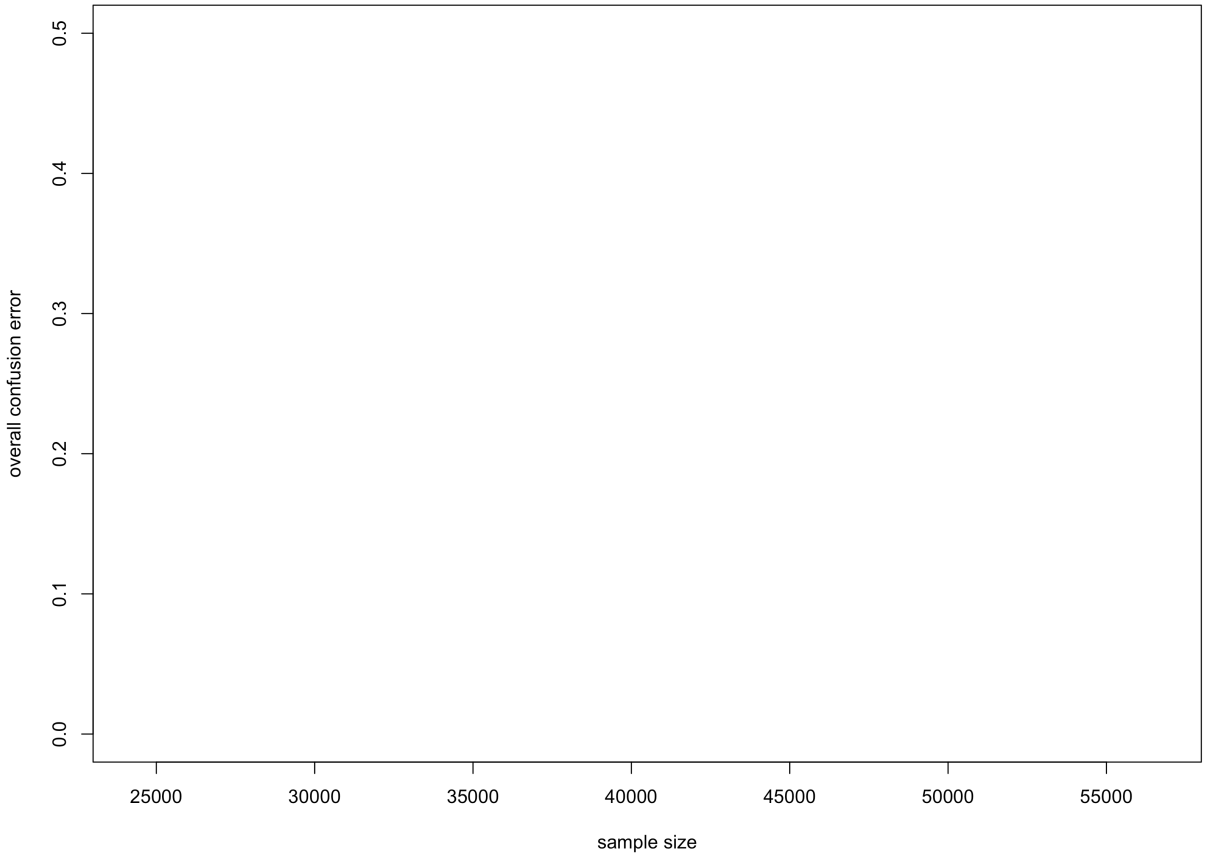

overall confusion error
0.0
0.1
0.2
0.3
0.4
0.5
25000
30000
35000
40000
45000
50000
55000
sample size

Figure 8.2:

```
TO_mean <- aggregate(x = res_acum[, 23], by = list((res_acum[, 1])), FUN = mean)
both_mean <- aggregate(x = res_acum[, 24], by = list((res_acum[, 1])), FUN = mean)

basic_sd <- aggregate(x = res_acum[, 21], by = list((res_acum[, 1])), FUN = sd)
diffusion_sd <- aggregate(x = res_acum[, 22], by = list((res_acum[, 1])), FUN = sd)
TO_sd <- aggregate(x = res_acum[, 23], by = list((res_acum[, 1])), FUN = sd)
both_sd <- aggregate(x = res_acum[, 24], by = list((res_acum[, 1])), FUN = sd)


polygon(x=c(basic_mean[,1], rev(basic_mean[,1])), y=c(basic_mean[,2] + basic_sd[,2], rev(basic_mean[,2]
polygon(x=c(diffusion_mean[,1], rev(diffusion_mean[,1])), y=c(diffusion_mean[,2] + diffusion_sd[,2], re
polygon(x=c(TO_mean[,1], rev(TO_mean[,1])), y=c(TO_mean[,2] + TO_sd[,2], rev(TO_mean[,2] - TO_sd[,2])),
polygon(x=c(both_mean[,1], rev(both_mean[,1])), y=c(both_mean[,2] + both_sd[,2], rev(both_mean[,2] - bo



lines(basic_mean,  col="orange")
lines(diffusion_mean,  col="cornflowerblue")
lines(TO_mean,  col="pink")
lines(both_mean,  col="darkgreen")

labs <- c("Basic", "+Diffusion", "+Takeover", "+Diffusion +Takeover")
legend("topright", legend=labs, col=c("orange", "cornflowerblue", "pink", "darkgreen"), lty=1, cex=1.5,

dev.off()
```

```
png("last_step_predictions.png", width = 11, height = 8.5, res = 300, units = "in")
par(mar=c(5,6,2,1))
this <- length(basic_mean[,1])

plot(x=c(0,5), y=c(0,1), type="n",  xaxt="n", ylab="probability that our known cultural phylogeny came
polygon(x=c(0.75,0.75,1.25,1.25), y=c(basic_mean[this,2] - basic_sd[this,2], basic_mean[this,2] + basic
polygon(x=c(1.75,1.75,2.25,2.25), y=c(diffusion_mean[this,2] - diffusion_sd[this,2], diffusion_mean[thi
polygon(x=c(2.75,2.75,3.25,3.25), y=c(TO_mean[this,2] - TO_sd[this,2], TO_mean[this,2] + TO_sd[this,2],
polygon(x=c(3.75,3.75,4.25,4.25), y=c(both_mean[this,2] - both_sd[this,2], both_mean[this,2] + both_sd[



lines(x=c(0.5, 1.5), c(basic_mean[this,2], basic_mean[this,2]),  col="orange")
lines(x=c(1.5, 2.5), c(diffusion_mean[this,2], diffusion_mean[this,2]),  col="cornflowerblue")
lines(x=c(2.5, 3.5), c(TO_mean[this,2], TO_mean[this,2]),  col="pink")
lines(x=c(3.5, 4.5), c(both_mean[this,2], both_mean[this,2]),  col="darkgreen")

legend("topright", legend=labs, col=c("orange", "cornflowerblue", "pink", "darkgreen"), lty=1, cex=1.5,

dev.off()
```

```
#colnames(res_acum)

long <- length(Confusion_0101_mean[,1])
Confusion_sum_01 <- aggregate(x = res_acum[, 5:8], by = list((res_acum[, 1])), FUN = mean)
Confusion_sum_02 <- aggregate(x = res_acum[, 9:12], by = list((res_acum[, 1])), FUN = mean)
Confusion_sum_03 <- aggregate(x = res_acum[, 13:16], by = list((res_acum[, 1])), FUN = mean)
Confusion_sum_04 <- aggregate(x = res_acum[, 17:20], by = list((res_acum[, 1])), FUN = mean)
```

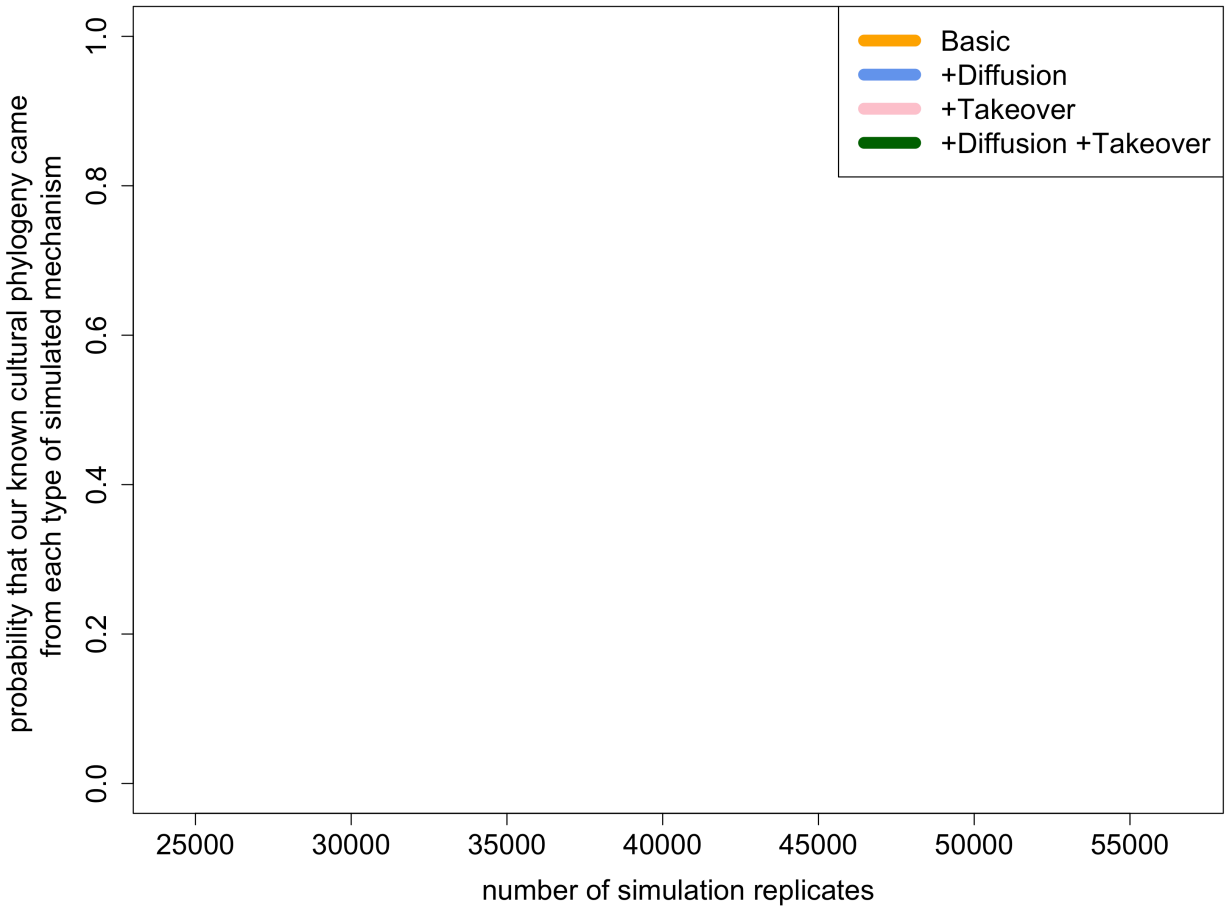

Basic
+Diffusion
+Takeover
+Diffusion +Takeover
probability that our known cultural phylogeny came from each type of simulated mechanism
0.0
0.2
0.4
0.6
0.8
1.0
25000
30000
35000
40000
45000
50000
55000
number of simulation replicates

Figure 8.3:

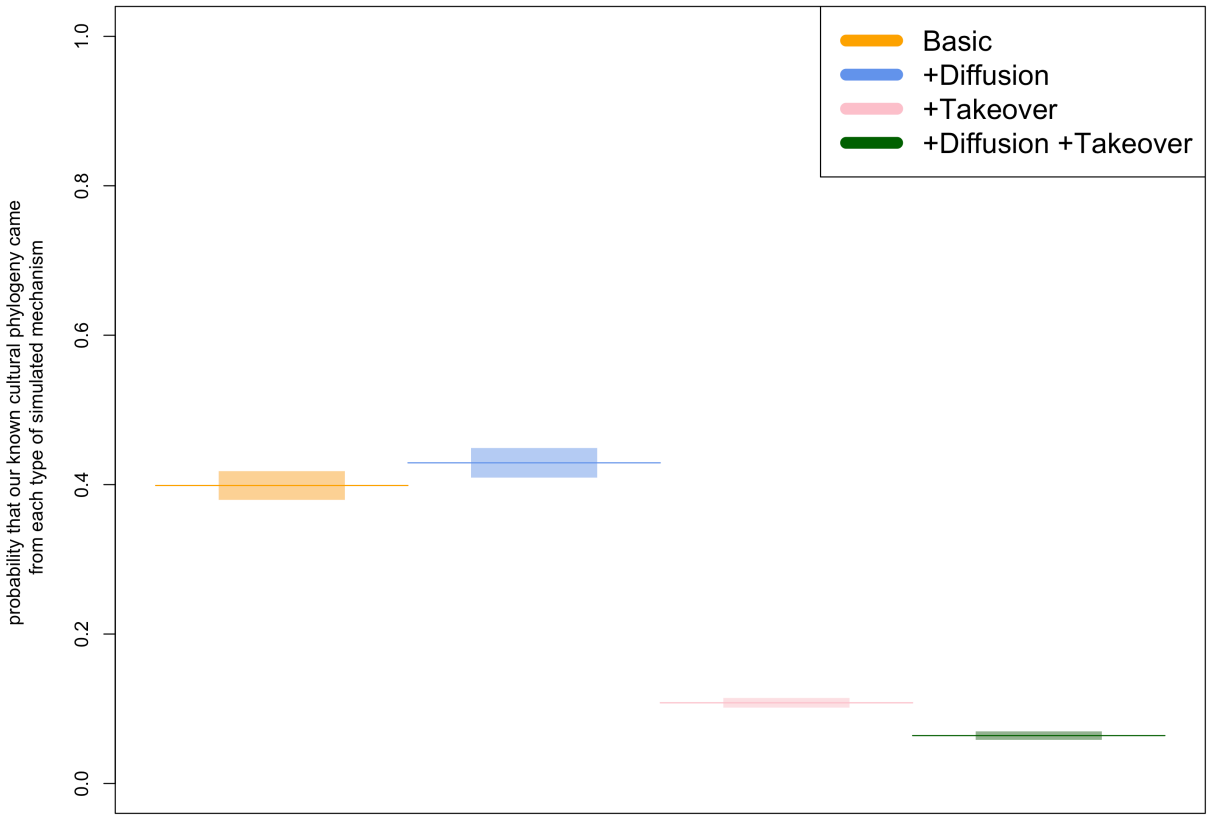

probability that our known cultural phylogeny came from each type of simulated mechanism
0.0
0.2
0.4
0.6
0.8
1.0
Basic
+Diffusion
+Takeover
+Diffusion +Takeover

Figure 8.4:

```
percent_one <- (Confusion_sum_01[,2:5]/sum(Confusion_sum_01[,2:5])) * 100
percent_two <- (Confusion_sum_02[,2:5]/sum(Confusion_sum_02[,2:5])) * 100
percent_three <- (Confusion_sum_03[,2:5]/sum(Confusion_sum_03[,2:5])) * 100
percent_four <- (Confusion_sum_04[,2:5]/sum(Confusion_sum_04[,2:5])) * 100


confusion <- matrix(c(percent_one, percent_two, percent_three, percent_four), 4,4)

labs <- c("Basic", "+Diffusion", "+Takeover", "+Diffusion",  "+Takeover")
colors1 <- colorRampPalette(colors = c("grey95", "grey40"))


png("confusion_matrix.png", width = 8.5, height = 8.5, res = 600, units = "in")
par(mar=c(8,8,1,1))
plot(0,0,xlim=c(-0.2,1.4), ylim=c(-0.2,1.4), xaxt="n", xlab="", yaxt="n", ylab="" , bty="n")
#image(prop, col = colors1(20), axes=FALSE)
axis(1, at=c(0, .4, .8, 1.2, 1.3), labels=labs, tick = FALSE, line = FALSE, cex.axis = 1, pos = -.19, la
axis(2, at=rev(c(-0.05, 0.05, .4, .8, 1.2)), labels=labs, tick = FALSE, line = FALSE, cex.axis = 1, las=
mtext("percent of time that RF identifies input model as each model type", side = 1, padj = 10, cex = 1)
mtext("known model type given to random forest", side = 2, padj = -10, cex = 1)



for(i in 1:4) {
  for(j in 4:1) {
    xs <- c(0, .4, .8, 1.2)[i]
    ys <- rev(c(0, .4, .8, 1.2))[j]
    polygon(x=c(xs-0.2, xs-0.2, xs+0.2, xs+0.2), y=c(ys-0.2, ys+0.2, ys+0.2, ys-0.2), col=colors1(100)[
    if(i == j){text(x = xs, y = ys, paste0(round(as.numeric(confusion[i, j]), 2), "%"), cex = 2.2, col=
  }
}
```

```
# Variables importance

imp <- importance(fit)
imp <- apply(imp, 2, function(x) (x - min(x))/(max(x) - min(x)))
imp <- imp[sort(imp[, 5], index.return = TRUE, decreasing = TRUE)$ix, ]

as.character(replace[, 6])
load(as.character(replace[90, 6]))
imp <- importance(fit)

names <- rownames(imp)
names[names == "spatial.tests.fora"] <- "Space F"
names[names == "spatial.tests.dom"] <- "Space D"
names[names == "sprate"] <- "Sp(ratio)"
names[names == "transition_from_trait_1_to_2"] <- "TR(1-2)"
names[names == "transition_from_trait_2_to_1"] <- "TR(2-1)"
names[names == "Phylogenetic_signal"] <- "PhySig(D)"
names[names == "Evolutionary_distinctiveness_sum"] <- "EDsum"
names[names == "Pylo_diversity_is_sum_of_BL"] <- "PDsum"
names[names == "transition_rate_ratio_1to2_over_2to1"] <- "TR(ratio)"
names[names == "gamma"] <- "Gamma"
names[names == "mean_Phylogenetic_isolation"] <- "MPI"
```

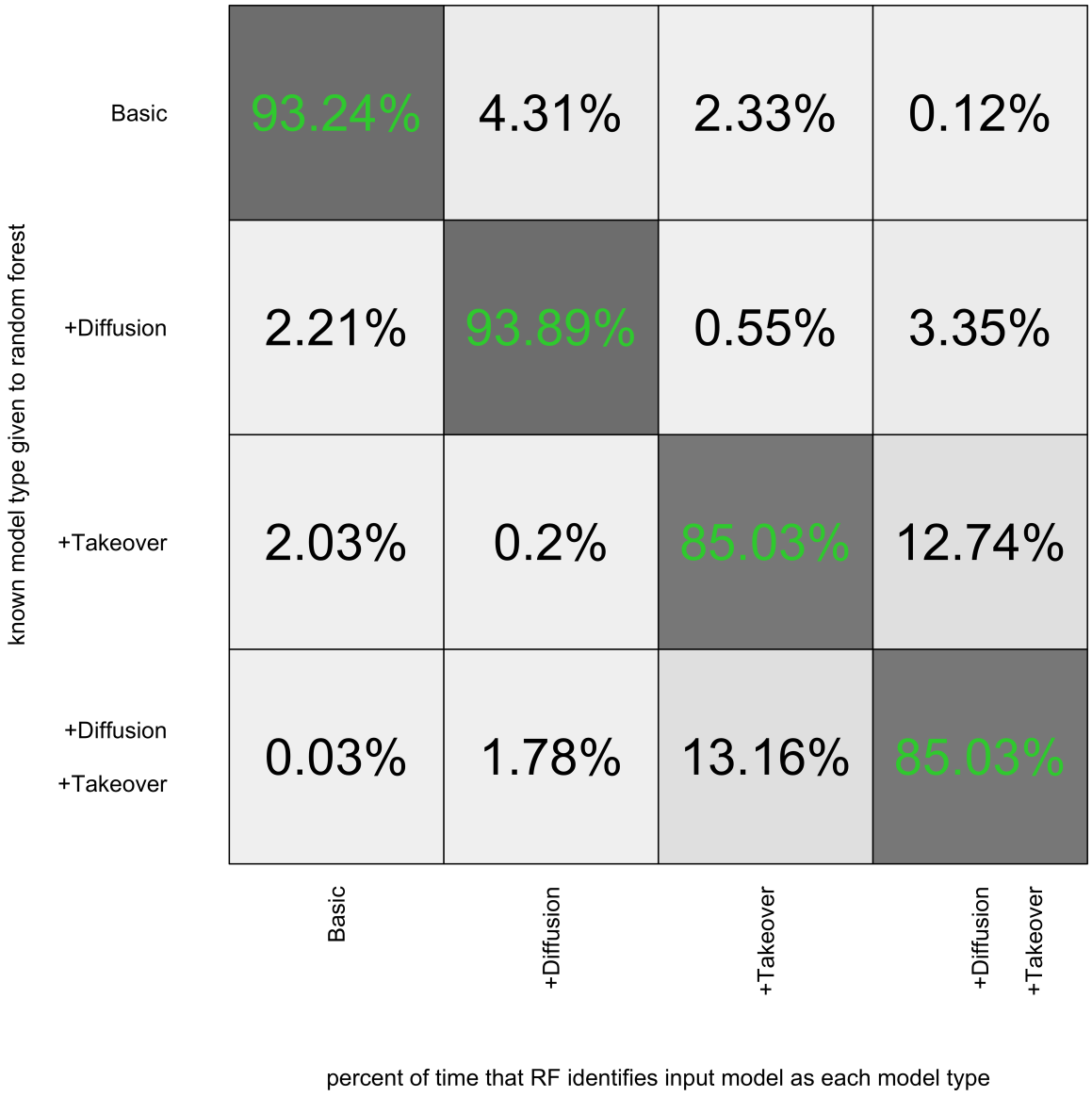

known model type given to random forest
Basic
+Diffusion
+Takeover
+Diffusion
+Takeover
93.24%
4.31%
2.33%
0.12%
2.21%
93.89%
0.55%
3.35%
2.03%
0.2%
85.03%
12.74%
0.03%
1.78%
13.16%
85.03%
Basic
+Diffusion
+Takeover
+Diffusion
+Takeover
percent of time that RF identifies input model as each model type

Figure 8.5:

```
names[names == "extrate"] <- "Ext(ratio)"
names[names == "average_phylogenetic_diversity_is_mean_of_BL"] <- "PDmean"
names[names == "extinction_per_speciation"] <- "DR"
names[names == "variance_Phylogenetic_isolation"] <- "VPI"
names[names == "F_quadratic_entropy_is_sum_of_PD"] <- "F"
names[names == "Mean_pairwise_distance"] <- "MPD"
names[names == "variance_Pylo_diversity_is_variance_of_BL"] <- "PDvar"
names[names == "variance_pairwise_distance"] <- "VPD"


x_length <- length(levels(replace[,1]))

######
# make matrix









#####
#plot from matrix









pdf(file="heat map of variable importance.pdf", width=11, height=8.5)

par(mar=c(4,10,4,1))
plot(x = seq(1, x_length, by=1), y = rep(0, x_length), type="n", ylim=c(0,20), ylab="", yaxt="n", xaxt=

axis(2, label=names, at=seq(0.5,length(names)-.5), las=2)


for(i in 1:length(as.character(replace[, 6]))){

    load(as.character(replace[i, 6]))
    imp <- importance(fit)
#imp <- apply(imp, 2, function(x) (x - min(x))/(max(x) - min(x)))
    #i <- 10

x_range <- c(i, i, i+1, i+1)
percent <- as.numeric(ceiling(imp[,5] ))
percent[which(percent == 0)] <- NA
colors <- cm.colors(100)[percent]
```

```
for(j in 0:19){
    y_range <- c(j, j+1, j+1, j)
polygon(x = x_range, y = y_range, col= colors[j+1], border=NA)
}

}

abline(v=seq(0,1000, by=100))

axis(1, label=format(unique(replace[,7]), format="%d %b %Y"), at=seq(50,950, by=100)[1:length(unique(re
dev.off()
```

```
importance(fit)
```

```
# Variables importance

imp <- importance(fit)
imp <- apply(imp, 2, function(x) (x - min(x))/(max(x) - min(x)))
imp <- imp[sort(imp[, 5], index.return = TRUE, decreasing = TRUE)$ix, ]


names <- rownames(imp)
names[names == "spatial.tests.fora"] <- "Space F"
names[names == "spatial.tests.dom"] <- "Space D"
names[names == "sprate"] <- "Sp(ratio)"
names[names == "transition_from_trait_1_to_2"] <- "TR(1-2)"
names[names == "transition_from_trait_2_to_1"] <- "TR(2-1)"
names[names == "Phylogenetic_signal"] <- "PhySig(D)"
names[names == "Evolutionary_distinctiveness_sum"] <- "EDsum"
names[names == "Pylo_diversity_is_sum_of_BL"] <- "PDsum"
names[names == "transition_rate_ratio_1to2_over_2to1"] <- "TR(ratio)"
names[names == "gamma"] <- "Gamma"
names[names == "mean_Phylogenetic_isolation"] <- "MPI"
names[names == "extrate"] <- "Ext(ratio)"
names[names == "average_phylogenetic_diversity_is_mean_of_BL"] <- "PDmean"
names[names == "extinction_per_speciation"] <- "DR"
names[names == "variance_Phylogenetic_isolation"] <- "VPI"
names[names == "F_quadratic_entropy_is_sum_of_PD"] <- "F"
names[names == "Mean_pairwise_distance"] <- "MPD"
names[names == "variance_Pylo_diversity_is_variance_of_BL"] <- "PDvar"
names[names == "variance_pairwise_distance"] <- "VPD"


png("var_import_all.png", width = 25, height = 25, unit="in", res=300)
par(mar = c(10, 18, 1, 1))
plot(x = rev(imp[, 5]), y = 1:nrow(imp), type = "l", yaxt = "n",
     ylab = "", xlab = "Variable Importance",
     xlim = c(0, 1), lwd = 2, cex.lab = 4)
for (i in 1:nrow(imp)) {
  abline(h = i, lty = 3, col = "gray80")
}
abline(v = seq(0, 1, 1/19), lty = 3, col = "gray80")

lines(x = rev(imp[, 4]), y = 1:nrow(imp), col = "darkgreen", lwd = 2)
```

```
lines(x = rev(imp[, 3]), y = 1:nrow(imp), col = "red", lwd = 2)
lines(x = rev(imp[, 2]), y = 1:nrow(imp), col = "blue", lwd = 2)
lines(x = rev(imp[, 1]), y = 1:nrow(imp), col = "darkorange1", lwd = 2)
lines(x = rev(imp[, 5]), y = 1:nrow(imp), lwd = 3)

points(x = rev(imp[, 4]), y = 1:nrow(imp), col = "darkgreen", cex = 2)
points(x = rev(imp[, 3]), y = 1:nrow(imp), col = "red", cex = 2)
points(x = rev(imp[, 2]), y = 1:nrow(imp), col = "blue", cex = 2)
points(x = rev(imp[, 1]), y = 1:nrow(imp), col = "darkorange1", cex = 2)
points(x = rev(imp[, 5]), y = 1:nrow(imp), pch = 20, cex = 3)


text(y = 1:nrow(imp), x = par("usr")[1] - .17, labels = rev(names),
     srt = 0, pos = 4, xpd = T, cex = 4)
dev.off()
```

```
par(mfrow=c(2,3))

# Box plots
boxplot(spatial.tests.fora ~ Model, data = data.analysis.comp3)
abline(h = a$spatial.tests.fora, col = "red", lty = 2)

boxplot(spatial.tests.dom ~ Model, data = data.analysis.comp3)
abline(h = a$spatial.tests.fora, col = "red", lty = 2)

boxplot(log(sprate) ~ Model, data = data.analysis.comp3, ylim = c(-10, 10))
abline(h = log(a$sprate), col = "red", lty = 2)

boxplot(log(extrate) ~ Model, data = data.analysis.comp3, ylim = c(-10, 10))
abline(h = log(a$extrate), col = "red", lty = 2)

boxplot(log(transition_rate_ratio_1to2_over_2to1) ~ Model, data = data.analysis.comp3)
abline(h = log(a$sprate), col = "red", lty = 2)

boxplot(Phylogenetic_signal ~ Model, data = data.analysis.comp3, ylim = c(0, 1))
abline(h = a$Phylogenetic_signal, col = "red", lty = 2)
```

```
#build a data tracking table to track parameter changes through time

str(fit)

y
```

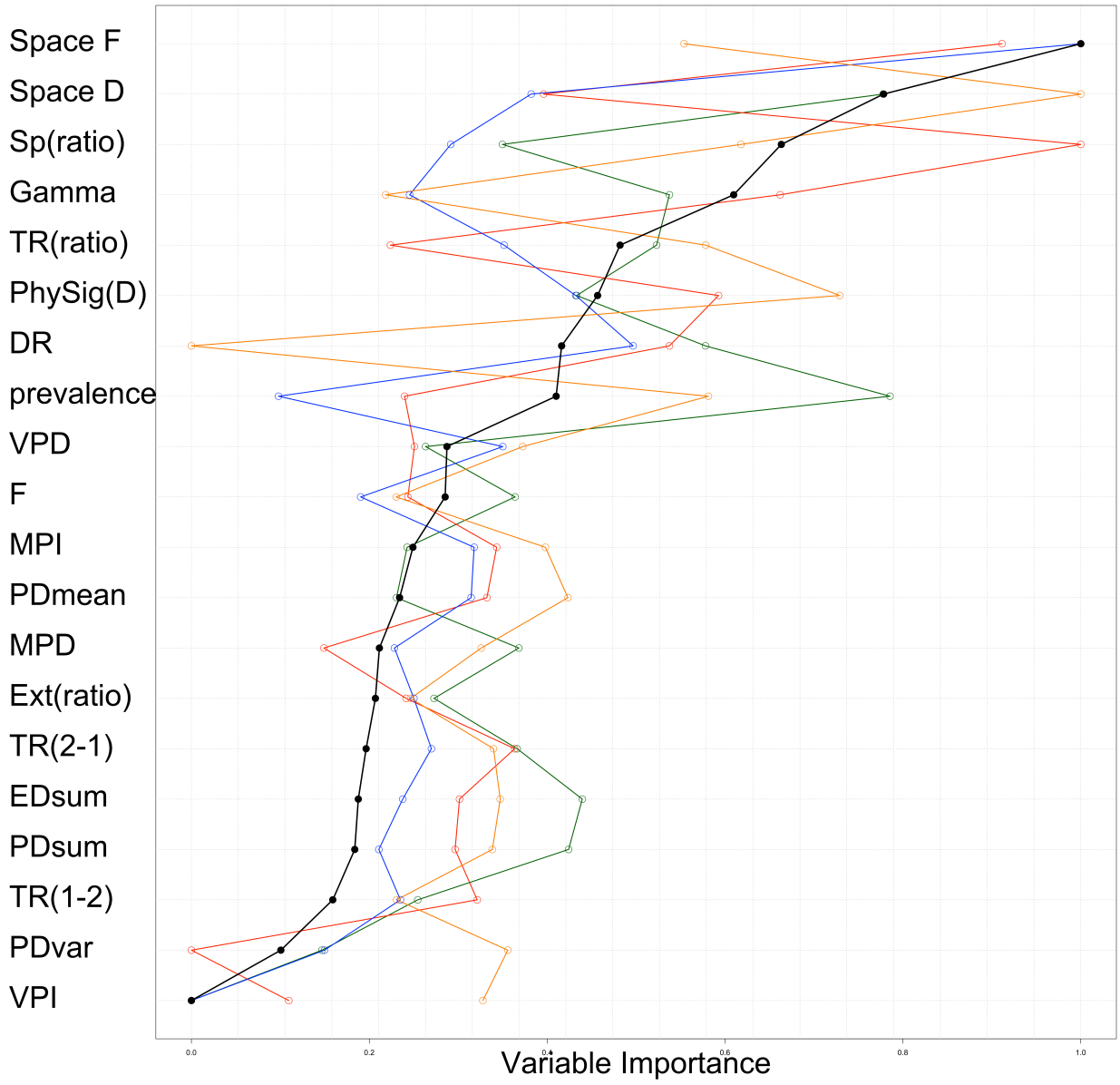

Space F
Space D
Sp(ratio)
Gamma
TR(ratio)
PhySig(D)
DR
prevalence
VPD
F
MPI
PDmean
MPD
Ext(ratio)
TR(2-1)
EDsum
PDsum
TR(1-2)
PDvar
VPI
0.0
0.2
0.4
0.6
0.8
1.0
Variable Importance

Figure 8.6:

Bibliography