D-place FARM documentation: Master script

Ty Tuff, Bruno Vilela, and Carlos Botero

project began: 15 May 2016, document updated: 04 July 2017

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.

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
```

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=
```