D-place FARM documentation: Module 1

Ty Tuff, Bruno Vilela, and Carlos Botero

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

Module 1: Simulation to produce a world and a tree

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

```
library(FARM)
```

```
## Warning: replacing previous import 'apTreeshape::is.binary.phylo' by
## 'ape::is.binary.phylo' when loading 'FARM'

## Warning: replacing previous import 'ape::plot.mst' by 'spdep::plot.mst'
## when loading '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"
```

Inputs

Module 1 functions

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.

```r
# 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.

```r
#==================================================================
# 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))
}
```

Push versions

```r
# 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)
}
```

```r
#==================================================================
# 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))
}
```

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.

```r
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)
}
```

```r
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])
```

```r
    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

```r
# 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)
}
```

```r
#===================================================================
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)
```

```r
      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))
}
```