concordance=TRUE strip.white=true

An introduction to the EcoPhyl package

Matt Helmus (???) and Will Pearse (wdpearse@umn.edu)

November 2013

Contents

1 Installing EcoPhyl

You can install EcoPhyl by typing `install.packages("EcoPhyl", dependencies=TRUE)`, and get a listing of the functions in the package by typing `library(help=EcoPhyl)`. When loading EcoPhyl into your R session (`library(EcoPhyl)`), you may notice a warning about a package named 'ecoPD'. ecoPD is not maintained on CRAN like many other packages, but can still be installed by typing install.packages("ecoPD", repos="http://R-Fo

If you find any bugs, or have any feature requests for the package, please use the online tracker at `http://github.com/willpearse/XXX/issues`. Indeed, all the package's code is available at `http://github.com/willpearse/XXX/issues`, where you can suggest modifications and alterations as you wish! EcoPhyl is an evolving package; new functions and features will be added as we publish papers describing them.

Please make sure you cite the accompanying paper for EcoPhyl (XXX) when publishing work using it. Also, while `EcoPhyl` contains much novel code, it relies heavily on the *R* ecosystem. Much of the community phylogenetic metric functions are wrappers around existing code (detailed in the help files for each function), in particular code from `picante`, `caper`, and `ecoPD` (REFS XXX)—please cite the authors of these packages so that their hard-work is rewarded!

2 Data formats in EcoPhyl

EcoPhyl functions work with 'comparative community ecology' objects, called `comparative.comm` objects. These are designed to help keep phylogenies, community data matrices,

species trait data, and environmental data all in the same place in a format that makes it easy to work with them. They're much less scary than they sound!

Below we load EcoPhyl, some example data that comes with it, and then make a comparative.comm object. You can examine the phylogeny ('tree'), community data ('comm'), and trait data ('traits') that went into making dataset for yourself, although all the data types are explained in more detail below.

```
library(EcoPhyl)

## Error:  there is no package called 'EcoPhyl'

data(EcoPhyl)

## Warning:  data set 'EcoPhyl' not found

data <- comparative.comm(traits.and.phy$phy,
                         traits.and.phy$comm,
                         traits.and.phy$traits)

## Error:  could not find function "comparative.comm"

data

## function (..., list = character(), package = NULL, lib.loc = NULL,
##     verbose = getOption("verbose"), envir = .GlobalEnv)
## {
##     fileExt <- function(x) {
##         db <- grepl("\\.[^.]+\\.(gz|bz2|xz)$", x)
##         ans <- sub(".*\\.", "", x)
##         ans[db] <- sub(".*\\.([^.]+\\.)(gz|bz2|xz)$", "\\1\\2",
##             x[db])
##         ans
##     }
##     names <- c(as.character(substitute(list(...))[-1L]), list)
##     if (!is.null(package)) {
##         if (!is.character(package))
##             stop("'package' must be a character string or NULL")
##         if (any(package %in% "base"))
##             warning("datasets have been moved from package 'base' to package 'data
##         if (any(package %in% "stats"))
```

```
##             warning("datasets have been moved from package 'stats' to package 'dat
##         package[package %in% c("base", "stats")] <- "datasets"
##     }
##     paths <- find.package(package, lib.loc, verbose = verbose)
##     if (is.null(lib.loc))
##         paths <- c(path.package(package, TRUE), if (!length(package)) getwd(),
##             paths)
##     paths <- unique(paths[file.exists(paths)])
##     paths <- paths[file_test("-d", file.path(paths, "data"))]
##     dataExts <- tools:::.make_file_exts("data")
##     if (length(names) == 0L) {
##         db <- matrix(character(), nrow = 0L, ncol = 4L)
##         for (path in paths) {
##             entries <- NULL
##             packageName <- if (file_test("-f", file.path(path,
##                 "DESCRIPTION")))
##                 basename(path)
##             else "."
##             if (file_test("-f", INDEX <- file.path(path, "Meta",
##                 "data.rds"))) {
##                 entries <- readRDS(INDEX)
##             }
##             else {
##                 dataDir <- file.path(path, "data")
##                 entries <- tools::list_files_with_type(dataDir,
##                   "data")
##                 if (length(entries)) {
##                   entries <- unique(tools::file_path_sans_ext(basename(entries)))
##                   entries <- cbind(entries, "")
##                 }
##             }
##             if (NROW(entries)) {
##                 if (is.matrix(entries) && ncol(entries) == 2L)
##                   db <- rbind(db, cbind(packageName, dirname(path),
##                     entries))
##                 else warning(gettextf("data index for package %s is invalid and wi
##                   sQuote(packageName)), domain = NA, call. = FALSE)
```

```
##             }
##         }
##         colnames(db) <- c("Package", "LibPath", "Item", "Title")
##         footer <- if (missing(package))
##             paste0("Use ", sQuote(paste("data(package =", ".packages(all.available
##                 "\n", "to list the data sets in all *available* packages.")
##         else NULL
##         y <- list(title = "Data sets", header = NULL, results = db,
##             footer = footer)
##         class(y) <- "packageIQR"
##         return(y)
##     }
##     paths <- file.path(paths, "data")
##     for (name in names) {
##         found <- FALSE
##         for (p in paths) {
##             if (file_test("-f", file.path(p, "Rdata.rds"))) {
##                 rds <- readRDS(file.path(p, "Rdata.rds"))
##                 if (name %in% names(rds)) {
##                   found <- TRUE
##                   if (verbose)
##                     message(sprintf("name=%s:\t found in Rdata.rds",
##                       name), domain = NA)
##                   thispkg <- sub(".*/([^/]*)/data$", "\\1", p)
##                   thispkg <- sub("_.*$", "", thispkg)
##                   thispkg <- paste0("package:", thispkg)
##                   objs <- rds[[name]]
##                   lazyLoad(file.path(p, "Rdata"), envir = envir,
##                     filter = function(x) x %in% objs)
##                   break
##                 }
##                 else if (verbose)
##                   message(sprintf("name=%s:\t NOT found in names() of Rdata.rds, i
##                     name, paste(names(rds), collapse = ",")),
##                     domain = NA)
##             }
##             if (file_test("-f", file.path(p, "Rdata.zip"))) {
```

```
##                 warning("zipped data found for package ", sQuote(basename(dirname(
##                   ".\nThat is defunct, so please re-install the package.",
##                   domain = NA)
##                 if (file_test("-f", fp <- file.path(p, "filelist")))
##                   files <- file.path(p, scan(fp, what = "", quiet = TRUE))
##                 else {
##                   warning(gettextf("file 'filelist' is missing for directory %s",
##                     sQuote(p)), domain = NA)
##                   next
##                 }
##             }
##             else {
##                 files <- list.files(p, full.names = TRUE)
##             }
##             files <- files[grep(name, files, fixed = TRUE)]
##             if (length(files) > 1L) {
##                 o <- match(fileExt(files), dataExts, nomatch = 100L)
##                 paths0 <- dirname(files)
##                 paths0 <- factor(paths0, levels = unique(paths0))
##                 files <- files[order(paths0, o)]
##             }
##             if (length(files)) {
##                 for (file in files) {
##                   if (verbose)
##                     message("name=", name, ":\t file= ...", .Platform$file.sep,
##                       basename(file), "::\t", appendLF = FALSE,
##                       domain = NA)
##                   ext <- fileExt(file)
##                   if (basename(file) != paste0(name, ".", ext))
##                     found <- FALSE
##                   else {
##                     found <- TRUE
##                     zfile <- file
##                     zipname <- file.path(dirname(file), "Rdata.zip")
##                     if (file.exists(zipname)) {
##                       Rdatadir <- tempfile("Rdata")
##                       dir.create(Rdatadir, showWarnings = FALSE)
```

```
##                         topic <- basename(file)
##                         rc <- .External(C_unzip, zipname, topic,
##                           Rdatadir, FALSE, TRUE, FALSE, FALSE)
##                         if (rc == 0L)
##                           zfile <- file.path(Rdatadir, topic)
##                       }
##                       if (zfile != file)
##                         on.exit(unlink(zfile))
##                       switch(ext, R = , r = {
##                         library("utils")
##                         sys.source(zfile, chdir = TRUE, envir = envir)
##                       }, RData = , rdata = , rda = load(zfile,
##                         envir = envir), TXT = , txt = , tab = ,
##                         tab.gz = , tab.bz2 = , tab.xz = , txt.gz = ,
##                         txt.bz2 = , txt.xz = assign(name, read.table(zfile,
##                           header = TRUE, as.is = FALSE), envir = envir),
##                         CSV = , csv = , csv.gz = , csv.bz2 = ,
##                         csv.xz = assign(name, read.table(zfile,
##                           header = TRUE, sep = ";", as.is = FALSE),
##                           envir = envir), found <- FALSE)
##                     }
##                     if (found)
##                       break
##                   }
##                   if (verbose)
##                     message(if (!found)
##                       "*NOT* ", "found", domain = NA)
##               }
##               if (found)
##                   break
##           }
##           if (!found)
##               warning(gettextf("data set %s not found", sQuote(name)),
##                   domain = NA)
##       }
##       invisible(names)
## }
```

```
## <bytecode: 0x7fcc5aad9758>
## <environment: namespace:utils>
```

EcoPhyl is conservative; if you give it trait data for only half of the species in your community data, the comparative.comm object will only contain data on those species that have both trait data and community data. The same goes for the phylogeny, and for sites with environmental data. EcoPhyl will warn you about the loss of species or traits when you print the object to screen, and while it's making the comparative.comm object (unless you set the argument warm=FALSE).

You can also subset your comparative.comm object to exclude certain species or sites, in much the same way you can a data.frame. Note that EcoPhyl will not (by default) warn you if this operation drops out certain species or sites. For example:

```
data[1:5,]

## Error:  object of type 'closure' is not subsettable

data[,1:3]

## Error:  object of type 'closure' is not subsettable

data[,1:3,warn=TRUE]

## Error:  object of type 'closure' is not subsettable
```

2.1 Phylogenies

EcoPhyl uses the `phylo` format in the `ape` package to store phylogenues. You can load your own phylogenies using the ape functions 'read.tree' and 'read.nexus'.

2.2 Community data

EcoPhyl uses the same community data format as the `vegan` package: a matrix or data.frame with sites in the rows and species in the columns. The elements of the community matrix can be species abundances or presence/absence (1/0). Not all the species in your matrix have to be present in a site, i.e. there can be empty columns in your data. This is particularly important when using the dispersion measures (see below). Your data should be named, with row names have correspond to sites, and column names that correspond to species.

2.3 Trait data

Trait data should be a data.frame with row names that correspond to the species in the phylogeny, and named columns for each separate trait.

2.4 Environemntal data

Environmental data should be a data.frame with row names that correpsond to the sites in your community data, and separate (named) columns for each kind of environmental data.

3 Plotting and exploring data

EcoPhyl comes with a few functions that are intended to make exploring your data slightly easier. For instance, you can plot out graphs of species abundances in communities

```
cc.dotplot(data)

## Error:  could not find function "cc.dotplot"

#cc.barplot(data, c(""))
```

4 Community phylogenetic metrics

EcoPhyl splits community phylogenetic metrics into four functions according to the scheme outlined by Pearse et al. (XXX): `shape`, `evenness`, `dispersion`, and `dissimilarity`. In brief, *shape* metrics ignore abundances, *evenness* metrics incorporates abundances, *dispersion* metrics compare observed communities with source pools of potential species, and *dissimilarity* metrics compare one community with another.

You can calculate all metrics within a class at the same time (which is what we recommend), or you can pick a particular one. Below we show how to calculate the metrics, and give examples of how to work with their output.

```
shape.output <- shape(data)

## Error:  could not find function "shape"

shape.output

## Error:  object 'shape.output' not found

shape.output$mpd

## Error:  object 'shape.output' not found

dissimilarity.output <- dissimilarity(data, metric="phylosor")

## Error:  could not find function "dissimilarity"

plot(hclust(dissimilarity.output$phylosor))

## Error:  object 'dissimilarity.output' not found
```

5 Eco-evolutionary regression (EcoPhyl)

EcoPhyl is intended to replace and improve upon earlier Visual Basic/C programs called EcoPhyl. You can regress the relative coexistence of species in your dataset against those species phylogenetic (`eco.phy.regression`) and trait (`eco.trait.regression`) dissimilarity, as well as shared habitat preferences based on environmental tolerances (`eco.env.regression`). Note that you can examine all traits at the same time, or each trait separately. Below are some examples.

```
model <- eco.trait.regression(data, method="lm")

## Error:  could not find function "eco.trait.regression"

model

## Error:  object 'model' not found

more.complex.model <- eco.trait.regression(data, method="mantel", altogether=FALSE, p

## Error:  could not find function "eco.trait.regression"
```

```
more.complex.model
```

```
## Error:  object 'more.complex.model' not found
```

```
more.complex.model[[1]]
```

```
## Error:  object 'more.complex.model' not found
```

```
plot(eco.phy.regression(data, method="quantile"))
```

```
## Error:  could not find function "eco.phy.regression"
```

However, you can also regress the phylogenetic signal of traits in your dataset against the output from an `eco.trait.regression` on those traits. This allows you to generate figures similar to those of Cavendar-Bares et al. (2004; figure 4 XXX), only now with more modern measures of phylogenetic signal in traits (accessibly through the function `phy.signal`). For instance:

```
model <- jcb.regression(data, eco.permute=100)
```

```
## Error:  could not find function "jcb.regression"
```

```
plot(model)
```

```
## Error:  object 'model' not found
```

6 Phylogenetic Generalised Linear Mixed Models

Matt, could you write this?