Tutorial for our model

John L Darcy

19 June 2019

Software Requirements

- `R` (run on version 3.6.0, but likely works fine on earlier versions)
- `ape` R package, used for reading and trimming phylogenies.
- `ggplot2` R package, used for some visualizations.
- `parallel` R package (included in base R), used for parallel processing.
- `assembly_model_functions.r` file, which includes functions for our model.

`ape` and `ggplot2` can be installed in R by simply running:

```r
install.packages("ape")
install.packages("ggplot2")
```

Then, all requirements can be loaded into an R environment with:

```r
library("ape")
library("ggplot2")
library("parallel")
source("assembly_model_functions.r")
```

(make sure the argument to source is the file path of the assembly model functions file.)

Reading in and formatting data

For this tutorial, we will use the Moving Pictures of the Human Microbiome data (Caporaso et al. 2012) used in our paper. Those data have been copied into the `model_inputs` folder, in the same directory where this tutorial is found. Our model needs 3 data objects to run:

- `OTU table`. This is a numeric matrix or data.frame of observations, where columns represent samples and rows represent species (called OTUs for the rest of this tutorial). Must have unique species identities as row names and unique sample identities as column names. For this tutorial, the raw data are stored as tab-delimited text.
- `Tree`. A phylo object (see ?ape) containing all species in OTU table (may contain more). Stored in Newick format.
- `Metadata`. Data.frame object where rows represent samples and columns are different metadata categories. At least one column needs to be a variable that can be ordered to give the temporal order of samples. Stored as tab-delimited text.

The commands below read the raw data files into R:

```r
otutable_raw <- read.table("model_inputs/esv_table.txt", sep = '\t', header = T,
  row.names = 1, comment.char = "")
tree_raw <- read.tree("model_inputs/esv_seqs_filt_aln_pfiltered.fasta.treefile")
metadata_raw <- read.table("model_inputs/movingpix_metadata.txt", stringsAsFactors = F,
  sep='\t', header = T, row.names = 1, comment.char = "")
```

Those raw data files can be sorted and filtered using our `prep_check_inputs` function. This function removes extra tips from the tree, combines phylogenetically identical OTUs, rarefies raw observation data, and ensures that all species and samples are represented in the right objects. In the code below, we use the options `rarefy = TRUE` and `rare_depth = 5000` to rarefy our data to 5000 observations per sample. The argument md_order_col = 11 specifies that the 11th column of our metadata file contains data that are used to define the temporal order of our samples. name = "all_movingpix_inputs" just gives a descriptive name to our data set. This function will print output to the console (or stdout if being run in a shell script), letting you know what it did. Warnings will be given if samples are dropped due to rarefaction.

```
    inputs <- prep_check_inputs(
        otutable = otutable_raw,
        metadata = metadata_raw,
        md_order_col=11,
        tree = tree_raw,
        distmat=NULL, rarefy=TRUE, rare_depth=5000,
        merge_tree_0dists=TRUE, drop_unobserved_otus=TRUE,
        name="all_movingpix"
    )
```

The object inputs we just made is simply a list of sorted objects. Let's take a look at what's inside:

```
  names(inputs)
```

```
## [1] "metadata"     "cum_otutable" "otutable"     "tree"
## [5] "distmat"      "md_order_col" "name"
```

Objects metadata, otutable, and tree are just sorted versions of our inputs. md_order_col is an integer, and we defined it above. distmat is NULL because we did not provide one (a distance matrix can be used as an alternative to a phylogenetic tree). cum_otutable is an accumulated version of otutable, which is a necessary input for the model.

Subsetting data

Our data are read in and sorted, but this data set contains 2 individuals and 4 body sites for each of them. We want to run our model only on the male's gut microbiome. To do that, we can subset our inputs using the subset_inputs function. This function will subset our inputs, and also discard extra leftover data, like OTUs and tree tips that are no longer present in the subset. The subset is designated with the sub_tf argument, which accepts a boolean vector. It is best to generate this vector using logic operations on the sorted metadata, as done below.

```
  male_gut_inputs <- subset_inputs(
    inputs_list = inputs,
    sub_tf = inputs$metadata$body_site == "feces" & inputs$metadata$individual == "M3",
    name="male gut"
  )
```

Let's do this for the male right palm as well.

```
  male_rpalm_inputs <- subset_inputs(
    inputs_list = inputs,
    sub_tf = inputs$metadata$body_site == "R_palm" & inputs$metadata$individual == "M3",
    name="male right palm"
  )
```

Running model

To run our model with minimal user brain damage, we include a workflow script that does the following:

- Calculates and visualizes empirical phylodiversity accumulation
- Runs simulations across a range of D values from -5 to 5 (default n=2000)

- Applies error model to those simulations and estimates empirical D, including visualization
- Runs simulations of neutral model D=0 (default n=2000)
- Tests and visualizes comparison to neutral model
- Stores all model data as an .rdata file
- Returns a list of results

This script is parallelized, which speeds it up significantly. To our knowledge, this parallelization will not work on Windows, but works well on Mac and Linux. Let's run the workflow script on our male gut and male right palm data sets. Progress bars and progress messages are not shown in this tutorial, but they are quite verbose. Here, we run the model with only 500 simulations for both model fitting and significance testing, just so it finishes quickly. We use `ncores = 12` because our machine has 12 processing cores (2x Intel Xeon E5-2643). We have found that using virtual cores (hyper threading) can cause the function to crash, so it is best to use less than or equal to the number of physical cores available on your machine or cluster.

To get a handle on memory use, we recommend running the script with `ncores=1`, monitoring memory use with `top` or similar, and then scaling the number of cores used assuming each will use the same amount of memory as the single-threaded job. For the male right palm analysis below, each of our 12 threads used approximately 4 GB of RAM, totaling 48 GB RAM. Memory use is proportional to the number of OTUs in a data set.

```r
male_gut_results <- pd_assemb_workflow(
  inputs_i = male_gut_inputs,
  n_dispersions=500,
  n_neutral_sim=500,
  ncores = 12
)
male_rpalm_results <- pd_assemb_workflow(
  inputs_i = male_rpalm_inputs,
  n_dispersions=500,
  n_neutral_sim=500,
  ncores = 12
)
```

Folders for model outputs are automatically generated using the names of inputs objects. Take a look in the `male right palm` folder, which contains .pdf plots for model fitting and model testing. It also contains a .rdata file containing the function's output.

Summary information about the run can be found in `$info`, and the actual model fit can be found in `$model_fitting$estimate`.

```r
male_rpalm_results$info
```

```
## $name
## [1] "male right palm"
##
## $out_dir
## [1] "male right palm"
##
## $n_dispersions
## [1] 500
##
## $n_neutral_sim
## [1] 500
##
## $n_otus
## [1] 5851
##
```

```
## $n_samples
## [1] 107
##
## $ncores
## [1] 12
##
## $total_tree_pd
## [1] 324.2613
```

```
male_rpalm_results$model_fitting$estimate
```

```
##                 r          i      ymin          K        est
## mean   -1.830127 -0.3849039 0.3526725 -0.2350800 -0.1630763
## lci025 -1.921859 -0.4139574 0.3446663 -0.2375613 -0.1876676
## hci975 -1.746293 -0.3513376 0.3575605 -0.2322433 -0.1352595
```

a P-value for the test against the neutral hypothesis $D = 0$ can be found in `$neutral_testing$prob`.

```
male_rpalm_results$neutral_testing$prob
```

```
## [1] 0.002
```

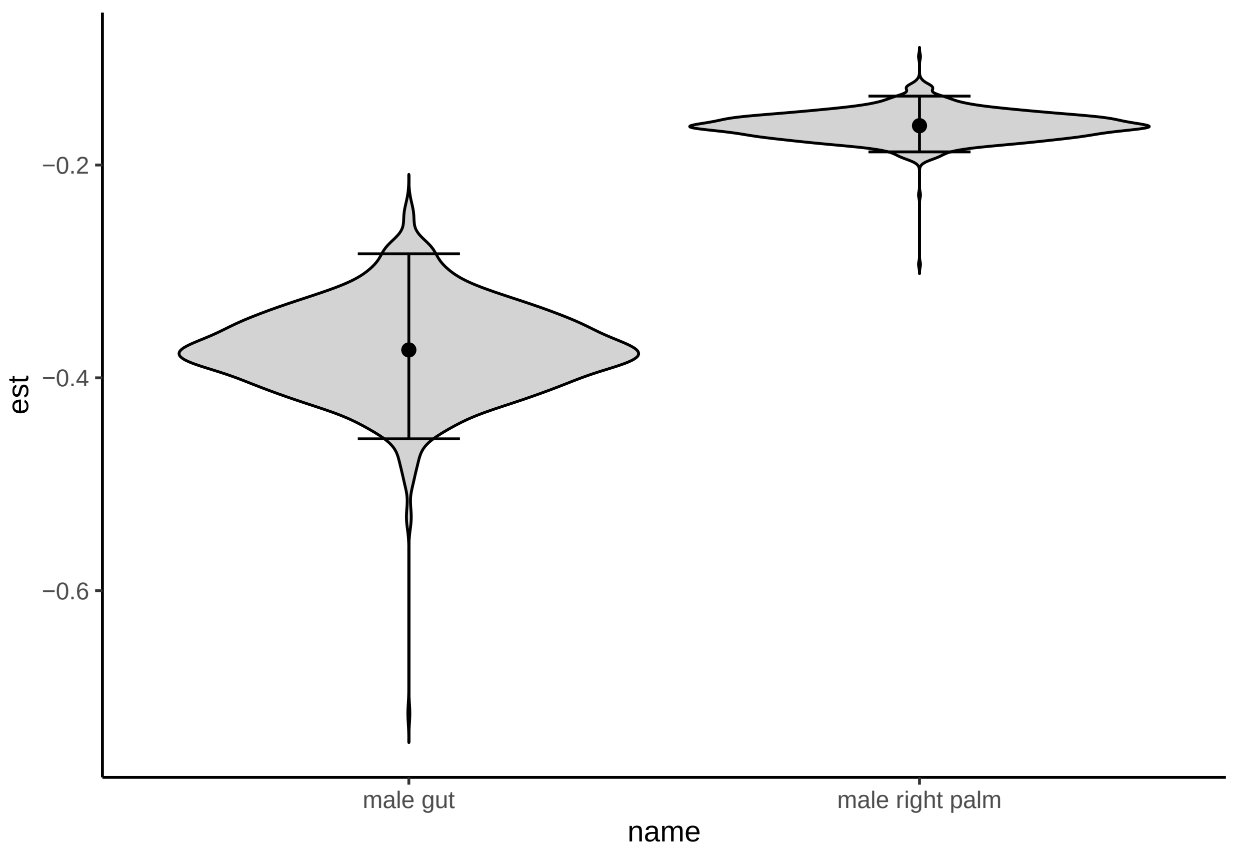
```
male_gut_results$neutral_testing$prob
```

```
## [1] 0.002
```

Visualizing model results

The function `plot_dispersion_ests_violin` takes a list of results lists as generated above, and uses `ggplot2` to generate a violin plot. Each violin is the distribution of D values from bootstrapped error model fits for a given data set. Data set labels are the names that were defined for inputs above. The dot in the center of a violin is the mean, and bars represent 95% confidence intervals for the D estimate.

```
plot_dispersion_ests_violin( list(male_gut_results, male_rpalm_results) )
```


−0.2
−0.4
−0.6
est
male gut
male right palm
name