Basic storage, access, and manipulation of phylogenetic sequencing data with *phyloseq*

Paul J. McMurdie
Holmes Group
Statistics Department, Stanford University
Stanford, CA

September 15, 2011

Contents

1 Introduction

There are already several ecology and phylogenetic packages available in R, including the adephylo, vegan, ade4, picante, ape, phangorn, phylobase, and OTUbase packages. These can already take advantage of many of the powerful statistical and graphics tools available in R. However, at present a user must devise their own methods for parsing the output of their favorite OTU clustering application, and, as a consequence, there is also no standard within Bioconductor (or R generally) for storing or sharing the suite of related data objects that describe a phylogenetic sequencing project. The phyloseq package[1] seeks to address these issues by providing a related set of S4 classes that internally manage the handling tasks associated with organizing, linking, storing, and analyzing phylogenetic sequencing data. *phyloseq* additionally provides some convenience wrappers for input from common clustering applications, common analysis pipelines, and native implementation of methods that are not available in other R packages.

[1] Throughout this vignette we use regular or *italics* font for packages/applications with names that are capitalized or uncapitalized, respectively. We further use a `courier` style font for R code, including function and class names.

2 Load *phyloseq* and import data

2.1 Load *phyloseq*

To use *phyloseq* in a new R session, it will have to be loaded. This can be done in your package manager, or at the command line using the `library()` command:

```
> library("phyloseq")
```

2.2 Import data

An important feature of *phyloseq* are methods for importing phylogenetic sequencing data from common taxonomic clustering pipelines. These methods take file pathnames as input, read and parse those files, and return a single object that contains all of the data.

As an example, the following lines of code would create a `phyloseqTaxTree` object (see Appendix A for class definitions) from files on your computer, had they been created by the the QIIME pipeline.

```
> otufilename <- "../data/ex1_otutable.txt"
> mapfilename <- "../data/ex1_samplemap.txt"
> trefilename <- "../data/ex1_tree.tre"
> ex1 <- readQiime(otufilename, mapfilename, trefilename)
```

An example data set is included in *phyloseq*, called "`ex1`". It is derived from a preliminary investigation of human intestinal microbiome. The user need only invoke the `data()` command to bring this object into the environment:

```
> data(ex1)
```

2.3 phyloseq object summaries

In small font, the following is the summary of object `ex1` that prints to the terminal. These summaries are consistent among all object classes defined by *phyloseq*. Although the components of `ex1` have many thousands of elements, the command-line returns only a short summary of each by default. This encourages you to check that an object is still what you expect, without needing to let thousands of elements scroll across the terminal. In the cases in which you do want to see all elements of an object, use `print()`.

```
> ex1

otuSamTaxTree Object

Sample Map [21 by 2]:
Samples: sa1, sa2 ... sa20, sa21
Variables: Diet Gender
    Diet Gender
sa1    0      B
sa2    0      A
sa3    0      B
...

OTU Table [7077 by 21]:
Species: otuID_3, otuID_4 ... otuID_9997, otuID_10000
Samples: sa1, sa2 ... sa20, sa21
        sa1 sa2 sa3 sa4
otuID_3   0  16   0   1
otuID_4   0   0   0   0
otuID_5   2   0   0   1
...

Taxonomy Table [7077 by 9]:
Species: otuID_3, otuID_4 ... otuID_9997, otuID_10000
Taxonomic Level: Root, Domain ... Species, Strain
        Root   Domain     Phylum           Class
otuID_3 "Root" "Bacteria" "Actinobacteria" "Actinobacteria"
otuID_4 "Root" "Bacteria" "Firmicutes"     "Clostridia"
otuID_5 "Root" "Bacteria" "Firmicutes"     "Clostridia"
...

<<< tree >>>
"phylo4"-class phylogenetic tree with
7077 tips, and 7070 internal nodes.
Tips: otuID_4769 otuID_8891 otuID_4100 ...
Unrooted.
<<< tree >>>
```

2.4 Convert raw data to phyloseq components

Suppose you have already imported raw data from an experiment into R, and their indices are labeled correctly. How do you get *phyloseq* to recognize these tables as the appropriate class of data? And further combine them together? Table 1 lists key functions for converting these core data formats into specific component data objects recognized by *phyloseq*. These will also

Functions for building component data objects

Function	Input Class	Output Description
`otuTable`	numeric matrix	`otuTable` object storing taxa abundance
`otuTable`	data.frame	`otuTable` object storing taxa abundance
`sampleMap`	data.frame	`sampleMap` object storing sample variables
`taxTab`	character string	`taxonomyTable` object storing taxonomic identities
`tre`	file path char	phylo4-class tree, read from file
`tre`	phylo-class tree	phylo4-class tree, converted from argument
`read.table`	table file path	A matrix or data.frame (Std Rcore function)
`read.tree`	Newick file path	phylo-class tree object (ape)
`read.nexus`	Nexus file path	phylo-class tree object (ape)
`readNexus`	Nexus file path	phylo4-class tree object (phylobase)

Functions for building complex data objects

Function	Input Class	Output Description
`phyloseq`	2 or more component objects	Complete experiment object
`merge_phyloseq`	2 or more component/complex objects	Complete experiment object

Table 1: Constructors: functions for building *phyloseq* objects.

The following example illustrates using the constructor methods for component data tables.

```
> otu1 <- otuTable(raw_abundance_matrix, speciesAreRows = FALSE)
> sam1 <- sampleMap(raw_sample_data.frame)
> tax1 <- taxTab(raw_taxonomy_matrix)
> tre1 <- read.nexus(my_nexus_file)
```

2.5 `phyloseq()` function: building complex phyloseq objects

Once you've converted the data tables to their appropriate class, combining them into one object requires only one additional function call, `phyloseq()`:

```
> ex1b <- phyloseq(my_otuTable, my_sampleMap, my_taxonomyTable,
+     my_tree)
```

You do not need to have all four data types in the example above in order to combine them. The `phyloseq()` method will create an object of whichever class is appropriate, according to the data you provide. Downstream analysis methods will know which class and which data types are required, and throw a warning if something is missing. For most downstream methods you will only need to supply the combined object returned by `phyloseq()` (usually as the first argument) and appropriate options.

```
> ex1c <- phyloseq(my_otuTable, my_sampleMap)
```

We refer to these classes that contain more than one component data type as *higher-order* classes. Whenever an instance of these classes is created by *phyloseq* — for example, when we use the `readQiime()` function to import data, or combine manually imported tables using the `phyloseq()` — the row and column indices

representing taxa or samples are internally checked/trimmed for compatibility, such that all component data describe exactly the same species and samples.

2.6 `merge_phyloseq()` function: merge multiple phyloseq objects

What if you have multiple objects describing parts of the same experimental project (say, because they came from different files)? What if you had already built a combined object for the earlier trials with the `phyloseq()` function, but now want to add additional data tables to that new object?

For all of these merging situations, the suggested function is `merge_phyloseq()`.

3 Accessor functions

Once you have a phyloseq object available, many accessor functions are available to query aspects of the data set. The function name and its purpose are summarized in Table 2.

Function	Description
`[`	Standard extraction operator. works on otuTable, sampleMap, and taxonomyTable
`access`	General slot accessor function for phyloseq-package
`getslots.phyloseq`	Return the slot names of phyloseq objects
`getSpecies`	Returns the abundance values of sample 'i' for all species in 'x'
`getSamples`	Returns the abundance values of species 'i' for all samples in 'x'
`nsamples`	Get the number of samples described by an object
`nspecies`	Get the number of species (taxa) described by an object
`otuSam`	Subset just the otuSam portion of a H.O. object
`otuTree`	Subset just the otuTree portion of a H.O. object
`otuTax`	Subset just the otuTax portion of a H.O. object
`otuSamTax`	Subset just the otuSamTax portion of a H.O. object
`otuSamTree`	Subset just the otuSamTree portion of a H.O. object
`otuSamTaxTree`	Subset just the otuSamTaxTree portion of a H.O. object
`otuTable`	Build or access otuTable objects
`sampleMap`	Build or access sampleMap objects
`taxTab`	Build or access taxTab objects
`tre`	Access the tree contained in a phyloseq object
`sample.names`	Return the names of the samples described by an object
`species.names`	Return the names of the species described by an object
`sampleSums`	Returns the total number of individuals observed from each sample
`speciesSums`	Returns the total number of individuals observed from each species
`speciesAreRows`	returns the orientation of the abundance table

Table 2: Accessor functions for *phyloseq* objects.

4 Trimming, filtering phyloseq data

Trimming high-throughput phylogenetic sequencing data can be useful, or even necessary, for certain types of analyses. However, it is important that the original data always be available for reference and reproducibility; and that the methods used for trimming be transparent to others, so they can perform the same trimming or filtering steps on the same or related data.

To facilitate this, *phyloseq* contains many ways to trim/filter the data from a phylogenetic sequencing project. Because matching indices for taxa and samples is strictly enforced, subsetting one of the data components automatically subsets the corresponding indices from the others. Variables holding trimmed versions of your original data can be declared, and further trimmed, without losing track of the original data.

For example, lets make a new object that only holds the most abundant 20 taxa in the experiment. To accomplish this, we will use the `prune_species()` function.

```
> data(ex1)
> most_abundant_taxa <- sort(speciesSums(ex1), TRUE)[1:topN]
> ex2 <- prune_species(names(most_abundant_taxa), ex1)
```

An alternative, replacement-style approach is also supported by *phyloseq*:

```
> species.names(ex2) <- names(most_abundant_taxa)
```

Now we can ask the question, "what taxonomic Family are these OTUs?" (Subsetting still returns a taxonomyTable object, which is summarized. We will need to convert to a vector)

```
> topFamilies <- taxTab(ex2)[, "Family"]
> as(topFamilies, "vector")

 [1] "Bacteroidaceae"  "Bacteroidaceae"  "Bacteroidaceae"  "Lachnospiraceae"
 [5] "Ruminococcaceae" NA                "Bacteroidaceae"  "Ruminococcaceae"
 [9] "Bacteroidaceae"  "Prevotellaceae"  "Ruminococcaceae" "Prevotellaceae"
[13] "Bacteroidaceae"  "Lachnospiraceae" "Lachnospiraceae" "Ruminococcaceae"
[17] "Lachnospiraceae" "Rikenellaceae"   "Bacteroidaceae"  "Prevotellaceae"
```

Can subset directly to keep only *Gender A*, for example

```
> ex3 <- ex1
> sampleMap(ex3) <- subset(sampleMap(ex3), Gender == "A")
```

```
> ex3

otuSamTaxTree Object

Sample Map [11 by 2]:
Samples: sa2, sa4 ... sa17, sa21
Variables: Diet Gender
    Diet Gender
sa2    0      A
sa4    1      A
sa5    1      A
...

OTU Table [7077 by 11]:
Species: otuID_3, otuID_4 ... otuID_9997, otuID_10000
Samples: sa2, sa4 ... sa17, sa21
         sa2 sa4 sa5 sa6
otuID_3  16   1   0   0
otuID_4   0   0   0   0
otuID_5   0   1   0   0
...

Taxonomy Table [7077 by 9]:
Species: otuID_3, otuID_4 ... otuID_9997, otuID_10000
Taxonomic Level: Root, Domain ... Species, Strain
        Root   Domain     Phylum           Class
```

```
otuID_3 "Root" "Bacteria" "Actinobacteria" "Actinobacteria"
otuID_4 "Root" "Bacteria" "Firmicutes"     "Clostridia"
otuID_5 "Root" "Bacteria" "Firmicutes"     "Clostridia"
...

<<< tree >>>
"phylo4"-class phylogenetic tree with
7077 tips, and 7070 internal nodes.
Tips: otuID_4769 otuID_8891 otuID_4100 ...
Unrooted.
<<< tree >>>
```

Or by specific taxonomic category

```
> ex4 <- ex1
> species.names(ex4) <- species.names(ex1)[taxTab(ex1)[, "Phylum"] ==
+     "Firmicutes"]
```

```
> ex4

otuSamTaxTree Object

Sample Map [21 by 2]:
Samples: sa1, sa2 ... sa20, sa21
Variables: Diet Gender
    Diet Gender
sa1    0      B
sa2    0      A
sa3    0      B
...

OTU Table [4960 by 21]:
Species: otuID_4, otuID_5 ... otuID_9997, otuID_10000
Samples: sa1, sa2 ... sa20, sa21
        sa1 sa2 sa3 sa4
otuID_4   0   0   0   0
otuID_5   2   0   0   1
otuID_6   0   0   0   0
...

Taxonomy Table [4960 by 9]:
Species: otuID_4, otuID_5 ... otuID_9997, otuID_10000
Taxonomic Level: Root, Domain ... Species, Strain
        Root   Domain     Phylum       Class
otuID_4 "Root" "Bacteria" "Firmicutes" "Clostridia"
otuID_5 "Root" "Bacteria" "Firmicutes" "Clostridia"
otuID_6 "Root" "Bacteria" "Firmicutes" "Clostridia"
...

<<< tree >>>
"phylo4"-class phylogenetic tree with
4960 tips, and 4954 internal nodes.
Tips: otuID_4769 otuID_8891 otuID_4100 ...
Unrooted.
<<< tree >>>
```

Can also randomly subset, for example a random subset of 100 OTUs

```
> ex5 <- ex1
> species.names(ex5) <- sample(species.names(ex5), 100, replace = FALSE)
```

5 Transform abundance data

Sample-wise transformation can be achieved with the `transformsamplecounts()` function. It requires two arguments, (1) the *phyloseq* object that you want to transform, and the function that you want to use to perform the transformation. Any arbitrary function can be provided as the second argument, as long as it returns a numeric vector with the same length as its input. In the following trivial example, we create a second object, `ex2`, that has been "transformed" by the identity function such that it is actually identical to `ex1`.

```
> data(ex1)
> ex2 <- transformsamplecounts(ex1, I)
```

For certain kinds of analyis we may want to transform the abundance data. For example, for RDA we want to transform abundance counts to within-sample ranks, and to further include a threshold beyond which all taxa receive the same rank value. The ranking for each sample is performed independently, so that the rank of a particular taxa within a particular sample is not influenced by that sample's total quantity of sequencing relative to the other samples in the project.

The following example shows how to perform such a thresholded-rank transformation of the abundance table in the complex *phyloseq* object `ex1` with an arbitrary threshold of 500.

```
> ex4 <- transformsamplecounts(ex1, threshrankfun(500))
```

6 Phylogenetic smoothing

6.1 `taxglom()` method

Suppose we are skeptical about the importance of species-level distinctions in our dataset. For this scenario, *phyloseq* includes a taxonomic-agglommeration method, `taxglom()`, which merges taxa of the same taxonomic category for a user-specified taxonomic level. In the following code, we merge all taxa of the same Genus, and store that new object as `ex6`.

```
> ex6 <- taxglom(ex1, taxlevel = "Genus")
```

6.2 `tipglom()` method

Similarly, our original example object (`ex1`) also contains a phlyogenetic tree corresponding to each OTU, which we could also use as a means to merge taxa in our dataset that are closely related. In this case, we specify a threshold patristic distance. Taxa more closely related than this threshold are merged. This is especially useful when a dataset has many taxa that lack a taxonomic assignment at the level you want to investigate, a problem when using `taxglom()`. Note that for datasets with a large number of taxa, `taxglom` will be noticeably faster than `tipglom`. Also, keep in mind that `tipglom` requires that its first argument be an object that contains a tree, while `taxglom` instead requires a `taxonomyTable` (See Appendix A).

```
> ex7 <- tipglom(ex1, speciationMinLength = 0.05)
```

Command output not provided here to save time during compilation of the vignette. The user is encouraged to try this out on your dataset, or even this example, if interested. It may take a while to run on the full, untrimmed data.

A *phyloseq* classes

The class structure in the *phyloseq* package follows the inheritance diagram shown in Fig. 1. The *phyloseq* package contains multiple inherited classes with incremental complexity so that methods can be extended to handle exactly the data types that are present in a particular object. Currently, *phyloseq* uses 4 core data classes. They are the taxonomic abundance table (`otuTable`), a table of sample data (`sampleMap`), a table of taxonomic descriptors (`taxonomyTable`), and a phylogenetic tree (`phylo4`, *phylobase* package). The `otuTable` class can be considered the central data type, as it directly represents the number and type of sequences observed in each sample. `otuTable` extends the numeric matrix class in the R base, and has a few additonal feature slots. The most important of these feature slots is the `speciesAreRows` slot, which holds a single logical that indicates whether the table is oriented with taxa as rows (as in the *genefilter* package in Bioconductor [**?**]) or with taxa as columns (as in *vegan* and *picante* packages). In *phyloseq* methods, as well as its extensions of methods in other packages, the `speciesAreRows` value is checked to ensure proper orientation of the otuTable. A *phyloseq* user is only required to specify the `otuTable` orientation during initialization, following which all handling is internal.

The `sampleMap` class directly inherits R's `data.frame` class, and thus effectively stores both categorical and numerical data about each sample. The orientation of a `data.frame` in this context requires that samples/trials are rows, and variables are columns (consistent with *vegan* and other packages). The `taxonomyTable` class directly inherits the `matrix` class, and is oriented such that rows are taxa (e.g. *species*) and columns are taxonomic levels (e.g. *Phylum*).

We use the term "higher-order classes" for those that contain two or more of the previously-described core data classes. We assume that *phyloseq* users will be interested in analyses that utilize their abundance counts derived from the phylogenetic sequencing data, and so all higher-order classes contain an `otuTable` slot. There are a number of common methods that require either an `otuTable` and `sampleMap` combination, or an `otuTable` and phylogenetic tree combination. These methods can operate on instances of the `otuSam` or `otuTree` classes, respectively, or their children. In addition, a virtual class has been defined, `phyloseqFather`, that is inherited by all other higher-order classes. In many cases a method will only need to be defined for one of these classes in order to work properly for other relevant classes as well.

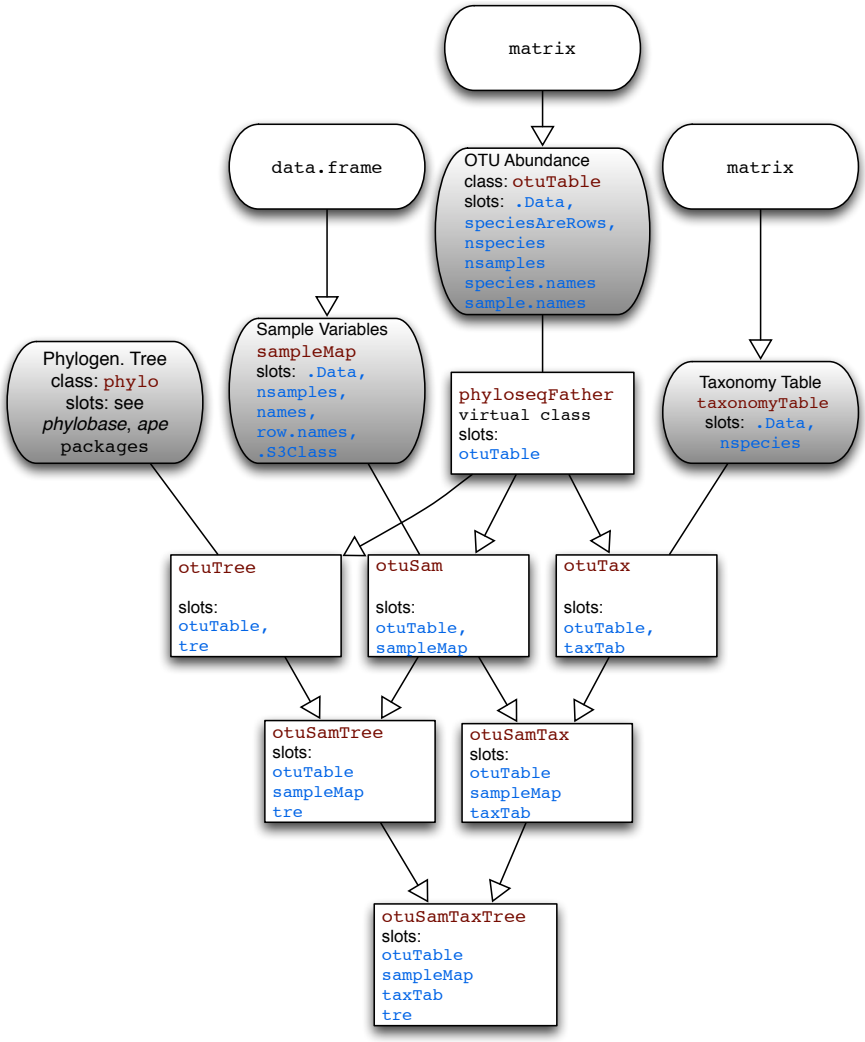

Figure 1: **C**lasses and inheritance in the *phyloseq* package. Core data classes are shown with grey fill and rounded corners. The class name and its slots are shown with red- or blue-shaded text, respectively. Inheritance is indicated graphically by arrows. Lines without arrows indicate that a higher-order object contains a slot with the associated class as one of its components.

B Install Development Version

For development version of the *phyloseq* package when it is not yet available from Bioconductor, see the development homepage on GitHub (`https://github.com/joey711/phyloseq`). This is also the best place to post issues, bug reports, feature requests, etc. The most convenient way to install this and other R packages available on GitHub is to first make sure you have the latest R development tools installed (*devtools* package), and then using the special github variant of the `install()` command, `install_github()`. Step-by-step:

```
> install.packages("devtools")
> library("devtools")
> install("ape")
> install("ggplot2")
> install("igraph")
> source("http://bioconductor.org/biocLite.R")
> biocLite("multtest")
> install("phylobase")
> install("picante")
> install("vegan")
> install_github("phyloseq", "joey711")
```

Note that the above instructions include installation of a number of additional packages upon which *phyloseq* is dependent. This may take some time to install. If there is a portion of these initial instructions not working as listed, please notify me by e-mail or at the github page. For running parallel implementation of functions/methods in *phyloseq* (e.g. parallel wUniFrac), you will need also to install the "Rmpi" package:

```
> install("Rmpi")
```

C Bibliography