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

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

February 21, 2012

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.

2.3 Import from QIIME

QIIME is a free, open-source OTU clustering and analysis pipeline written for Unix (mostly Linux) [1]. It is distributed in a number of different forms (including a pre-installed virtual machine), and relevant links for obtaining and using QIIME should be found at:

`http://qiime.org/`

2.3.1 Input

One QIIME input file (sample map), and two QIIME output files (`"otutable.txt"`, `".tre"`) are recognized by the `import_qiime()` function. Only one of the three input files is required to run, although an `"otutable.txt"` file is required if `import_qiime()` is to return a complete experiment object.

In practice, you will have to find the relevant QIIME files among a number of other files created by the QIIME pipeline. A screenshot of the directory structure created during a typical QIIME run is shown in Figure 1.

2.3.2 Output

The class of the object returned by `import_qiime()` depends upon which filenames are provided. The most comprehensive class is chosen automatically, based on the input files listed as arguments. At least one argument needs to be provided.

2.3.3 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 QIIME pipeline.

```
> otufilename <- "../data/ex1_otutable.txt"
> mapfilename <- "../data/ex1_sampleData.txt"
> trefilename <- "../data/ex1_tree.tre"
> MyExpmt1 <- import_qiime(otufilename, mapfilename, trefilename)
```

A separate object containing taxonomic data is not necessary, because this information is included in the "otu_table.txt" file, and parsed into a proper `taxonomyTable` component object by `import_qiime()`.

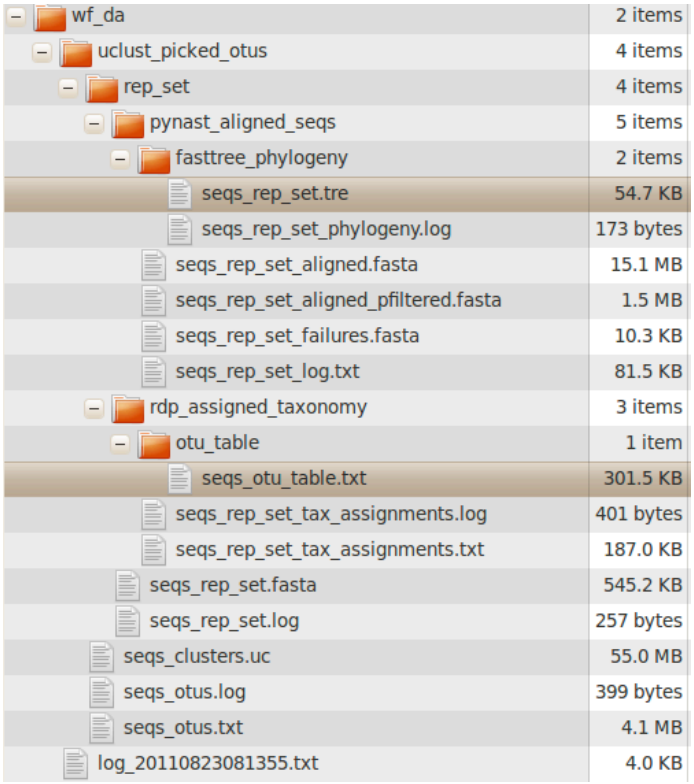

Figure 1: **A** typical QIIME output directory. The two output files suitable for import by *phyloseq* are highlighted. A third file describing the samples, their barcodes and covariates, is created by the user and required as *input* to QIIME. It is a good idea to import this file, as it can be converted directly to a `sampleData` object and can be extremely useful for certain analyses.

2.4 Import from mothur

The open-source, platform-independent, locally-installed software package, "mothur", can also process barcoded amplicon sequences and perform OTU-clustering [2]. It is extensively documented on a wiki at the following URL:

`http://www.mothur.org/wiki/`

2.4.1 Input

Currently, there are three different files produced by the *mothur* package (Ver 1.22.0) that can be imported by *phyloseq*. At minimum, a user must supply a `".list"` file, and at least one of the following two files: `".groups"` or `".tree"`

The group file is produced by *mothur*'s `"make.group()"` function. Details on its use can be found at:
`http://www.mothur.org/wiki/Make.group`
The tree file is a phylogenetic tree calculated by *mothur*.

2.4.2 Output

The output from `import_mothur()` depends on which file types are provided. If all three file types are provided, an `"otuTree"` object is returned (which contains both an OTU abundance table and its associated phylogenetic tree).

2.4.3 Example

The path on your machine may (probably will) vary, but here is an example import with all three files:

```
> mothur_list_file  <- "~/Downloads/mothur/Esophagus/esophagus.an.list"
> mothur_group_file <- "~/Downloads/mothur/Esophagus/esophagus.good.groups"
> mothur_tree_file  <- "~/Downloads/mothur/Esophagus/esophagus.tree"
> # # Actual examples follow:
> show_mothur_list_cutoffs(mothur_list_file)
> MyExpmt1 <- import_mothur(mothur_list_file, mothur_group_file, mothur_tree_file)
```

2.5 Import from PyroTagger

PyroTagger is an OTU-clustering pipeline for barcoded 16S rRNA amplicon sequences, served and maintained by the Department of Energy's (DOE's) Joint Genome Institute (JGI). It can be used through a straightforward web interface at:

`http://pyrotagger.jgi-psf.org/`

PyroTagger takes as input the untrimmed sequence (`.fasta`) and sequence-quality (`.qual`) files, as well as a sample mapping file that contains the bar code sequence for each sample and its name. It uses a 97% identity threshold for defining OTU clusters (approximately species-level of taxonomic distinction), and provides no options for specifying otherwise. It does allow users to modify the threshold setting for low-quality bases.

2.5.1 Input

PyroTagger returns a single excel spreadsheet file (`.xls`) containing both abundance and taxonomy data, as well as some associated confidence information related to each taxonomic assignment. This spreadsheet also reports on potential chimeric sequences.

This single output file is sufficient for `import_RDP_tab()`, provided the file has been converted to a tab-delimited plain-text format. Any spreadsheet application should suffice. No other changes should be made to the `.xls` file.

2.5.2 Output

`import_RDP_tab()` returns an object from the `"otuTax"` class, containing the OTU abundance table and taxonomy table. To my knowledge, PyroTagger does not calculate a tree of the representative sequences from each OTU cluster, nor a distance object, so analyses like `tipglom()` are not applicable.

2.5.3 Example

Here is an example, importing a PyroTagger file:

```
>   pyrotagger_tab_file <- "path/to/my/filename.txt"
>   myData1 <- import_pyrotagger_tab(pyrotagger_tab_file,
+     strict_taxonomy=FALSE, keep_potential_chimeras=FALSE)
```

For completeness, the optional arguments were shown with their default values selected. If you do not need to modify the optional arguments, the import command simplifies to:

```
>   myData1 <- import_pyrotagger_tab(pyrotagger_tab_file)
```

2.6 Import from RDP pipeline

The Ribosomal Database Project (RDP [3]; `http://rdp.cme.msu.edu/`) provides a web-based barcoded 16S rRNA amplicon sequence processing pipeline called the "RDP Pyrosequencing Pipeline" (`http://pyro.cme.msu.edu/`). A user must run all three of the "Data Processing" steps sequentially through the web interface in order to acquire the output from Complete Linkage Clustering, the approach to OTU clustering used by the RDP Pipeline. Note that this import function assumes that the sequence names in the resulting cluster file follow a particular naming convention with underscore delimiter. (See Section 2.6.3, below.)

2.6.1 Input

The output from the Complete Linkage Clustering, `".clust"`, is the only input to the RDP pipeline importer:

```
>   myOTU1 <- import_RDP_cluster("path/to/my/filename.clust")
```

2.6.2 Output

This importer returns an `otuTable` object.

2.6.3 Expected Naming Convention

The RDP cluster pipeline (specifically, the output of the complete linkage clustering step) has no formal documentation for the ".clust" file structure or its apparent sequence naming convention.

The cluster file itself contains the names of all sequences contained in the input alignment. If the upstream barcode and aligment processing steps are also done with the RDP pipeline, then the sequence names follow a predictable naming convention wherein each sequence is named by its sample and sequence ID, separated by a "_" as delimiter:

"sampleName_sequenceIDnumber"

This import function assumes that the sequence names in the cluster file follow this convention, and that the sample name does not contain any "_". It is unlikely to work if this is not the case. It is likely to work if you used the upstream steps in the RDP pipeline to process your raw (barcoded, untrimmed) fasta/fastq data.

2.7 Example Data (included)

There are multiple example data sets included in *phyloseq*. Many are from published investigations and include documentation with a summary and references, as well as some example code representing some aspect of analysis available in *phyloseq*. In the package index, go to the names beginning with "data-" to see the documentation of currently available example datasets.

To load example data into the working environment, use the `data()` command:

```
> data(ex1)
> data(esophagus)
> data(enterotype)
> data(soilrep)
> data(GlobalPatterns)
```

Similarly, entering `?enterotype` will reveal the documentation for the so-called "enterotype" dataset.
See the Example Data page on the phyloseq GitHub wiki at:
`https://github.com/joey711/phyloseq/wiki/Example-Data`

2.8 phyloseq object summaries

In small font, the following is the summary of the `GlobalPatterns` dataset that prints to the terminal. These summaries are consistent among all `phyloseq-class` objects. Although the components of `GlobalPatterns` have many thousands of elements, the command-line returns only a short summary of each component. 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 more of a particular component, use an accessor function (see Table 2, Section 3).

```
> data(GlobalPatterns)
> GlobalPatterns

phyloseq-class experiment-level object
OTU Table:          [19216 species and 26 samples]
                     species are rows
Sample Map:         [26 samples by 7 sample variables]:
Taxonomy Table:     [19216 species by 7 taxonomic ranks]:
Phylogenetic Tree:  [19216 tips and 19215 internal nodes]
                     rooted
```

2.9 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
`sampleData`	data.frame	`sampleData` 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	phyloseq-class, "experiment-level" object
`merge_phyloseq`	2 or more component or phyloseq-class objects	Combined instance of phyloseq-class

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 <- sampleData(raw_sample_data.frame)
> tax1 <- taxTab(raw_taxonomy_matrix)
> tre1 <- read.nexus(my_nexus_file)
```

2.10 `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_sampleData, my_taxonomyTable, my_tree)
```

You do not need to have all four data types in the example above in order to combine them into one validity-checked experiment-level phyloseq-class object. The `phyloseq()` method will detect which component data classes are present, and build accordingly. Downstream analysis methods will access the required components using emphphyloseq's accessors, and throw an error if something is missing. For most downstream methods you will only need to supply the combined, phyloseq-class object (the output of `phyloseq()`), usually as the first argument.

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

Whenever an instance of the phyloseq-class is created by *phyloseq* — for example, when we use the `import_qiime()` function to import data, or combine manually imported tables using `phyloseq()` — the row and column indices representing taxa or samples are internally checked/trimmed for compatibility, such that all component data describe exactly (and only) the same species and samples.

2.11 `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, sampleData, 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'
`getTaxa`	Get a unique vector of the observed taxa at a particular taxonomic rank
`getVariable`	Returns an individual sample variable vector/factor
`nsamples`	Get the number of samples described by an object
`nspecies`	Get the number of species (taxa) described by an object
`otuTable`	Build or access otuTable objects
`rank.names`	Get the names of the available taxonomic ranks
`sampleData`	Build or access sampleData objects
`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
`sample.variables`	Returns the names of sample variables in an object
`speciesSums`	Returns the total number of individuals observed from each species
`speciesAreRows`	Returns the orientation of the abundance table
`taxTab`	Build or access taxTab objects
`tre`	Access the tree contained in a phyloseq object

Table 2: Accessor functions for *phyloseq* objects.

4 Trimming, subsetting, filtering phyloseq data

4.1 Trimming: `prune_species()` and `prune_samples()`

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.

In general, most trimming should be accomplished using the S4 methods `prune_species()` or `prune_samples()`.

4.2 Simple filtering example

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)
```

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"
```

4.3 Arbitrarily complex abundance filtering

The previous example was a relatively simple filtering in which we kept only the most abundant 20 in the whole experiment. But what if we wanted to keep the most abundant 20 taxa of each sample? And of those, keep only the taxa that are also found in at least one-third of our samples?

For this more complicated filtering *phyloseq* contains a function, `genefilterSample()`, that takes as an argument a *phyloseq* object, as well as a list of one or more filtering functions that will be applied to each sample in the abundance matrix (`otuTable`), as well as an integer argument, `A`, that specifies for how many samples the filtering function must return `TRUE` for a particular taxa to avoid removal from the object. A supporting function `filterfunSample` is also included in *phyloseq* to facilitate creating a properly formatted function (enclosure) if more than one function is going to be applied simultaneously. `genefilterSample` returns a logical vector suitable for sending directly to prune_species() for the actual trimming.

Here is an example on a completely fabricated `otuTable` called `testOTU`.

```
> testOTU <- otuTable(matrix(sample(1:50, 25, replace=TRUE), 5, 5), speciesAreRows=FALSE)
> f1  <- filterfunSample(topk(2))
> wh1 <- genefilterSample(testOTU, f1, A=2)
> wh2 <- c(T, T, T, F, F)
> prune_species(wh1, testOTU)
> prune_species(wh2, testOTU)
```

Here is a second example using the included dataset, `ex1`. The most abundant taxa are kept only if they are in the most abundant 10% of taxa in at least half of the samples in dataset `ex1`. Note that it is not necessary to subset `ex1` in order to do this filtering. The S4 method prune_species() subsets each of the relavent component objects, and returns the complex object back.

```
> data(ex1)
> f1  <- filterfunSample(topp(0.1))
> wh1 <- genefilterSample(ex1, f1, A=(1/2*nsamples(ex1)))
> sum(wh1)

[1] 734

> ex2 <- prune_species(wh1, ex1)

>   print(ex2)

phyloseq-class experiment-level object
OTU Table:          [734 species and 21 samples]
                     species are rows
Sample Map:         [21 samples by 2 sample variables]:
Taxonomy Table:     [734 species by 9 taxonomic ranks]:
Phylogenetic Tree:  [734 tips and 732 internal nodes]
                     unrooted
```

If instead of the most abundant fraction of taxa, you are interested in the most abundant fraction of individuals (aka sequences, observations), then the `topf` function is appropriate. For steep rank-abundance curves, `topf` will seem to be much more conservative (trim more taxa) because it is based on the cumulative sum of relative abundance. It does not guarantee that a certain number or fraction of total taxa (richness) will be retained.

```
> data(ex1)
> f1  <- filterfunSample(topf(0.9))
> wh1 <- genefilterSample(ex1, f1, A=(1/3*nsamples(ex1)))
> sum(wh1)
> prune_species(wh1, ex1)
```

4.4 subset_samples(): subset by sample variates

It is possible to subset the samples in a *phyloseq* object based on the sample variables using the `subset_samples()` function. For example to subset `ex1` such that only *Gender A* is present, the following line is needed (the related tables are subsetted automatically as well):

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

> ex3

phyloseq-class experiment-level object
OTU Table:          [7077 species and 11 samples]
                     species are rows
Sample Map:         [11 samples by 2 sample variables]:
Taxonomy Table:     [7077 species by 9 taxonomic ranks]:
Phylogenetic Tree:  [7077 tips and 7070 internal nodes]
                     unrooted
```

For this example only a categorical variable is shown, but in principle a continuous variable could be specified and a logical expression provided just as for the `subset` function. In fact, because `sampleData` component objects are an extension of the data.frame class, they can also be subsetted with the `subset` function:

```
> subset(sampleData(ex1), Gender=="A")
```

```
Sample Data:        [11 samples by 2 sample variables]:
     Gender Diet
sa2       A    0
sa4       A    1
sa5       A    1
sa6       A    1
sa7       A    0
sa10      A    1
sa11      A    0
sa12      A    0
sa14      A    1
sa17      A    1
sa21      A    0
```

4.5 subset_species(): subset by taxonomic categories

It is possible to subset by specific taxonomic category using the `subset_species()` function. For example, if we wanted to subset `ex1` so that it only contains data regarding the phylum *Firmicutes*:

```
> ex4 <- subset_species(ex1, Phylum=="Firmicutes")
```

```
> ex4
```

```
phyloseq-class experiment-level object
OTU Table:          [4960 species and 21 samples]
                     species are rows
Sample Map:         [21 samples by 2 sample variables]:
Taxonomy Table:     [4960 species by 9 taxonomic ranks]:
Phylogenetic Tree:  [4960 tips and 4954 internal nodes]
                     unrooted
```

4.6 random subsample abundance data

Can also randomly subset, for example a random subset of 100 taxa from the full dataset.

```
> randomSpecies100 <- sample(species.names(ex1), 100, replace=FALSE)
> ex5 <- prune_species(randomSpecies100, ex1)
```

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. 2. 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 (`sampleData`), 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 [4]) 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 `sampleData` 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*).

The phyloseq-class can be considered an "experiment-level class" and should 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 the `phyloseq()` constructor will stop with an error if the arguments do not include an `otuTable`. There are a number of common methods that require either an `otuTable` and `sampleData` combination, or an `otuTable` and phylogenetic tree combination. These methods can operate on instances of the phyloseq-class, and will stop with an error if the required component data is missing.

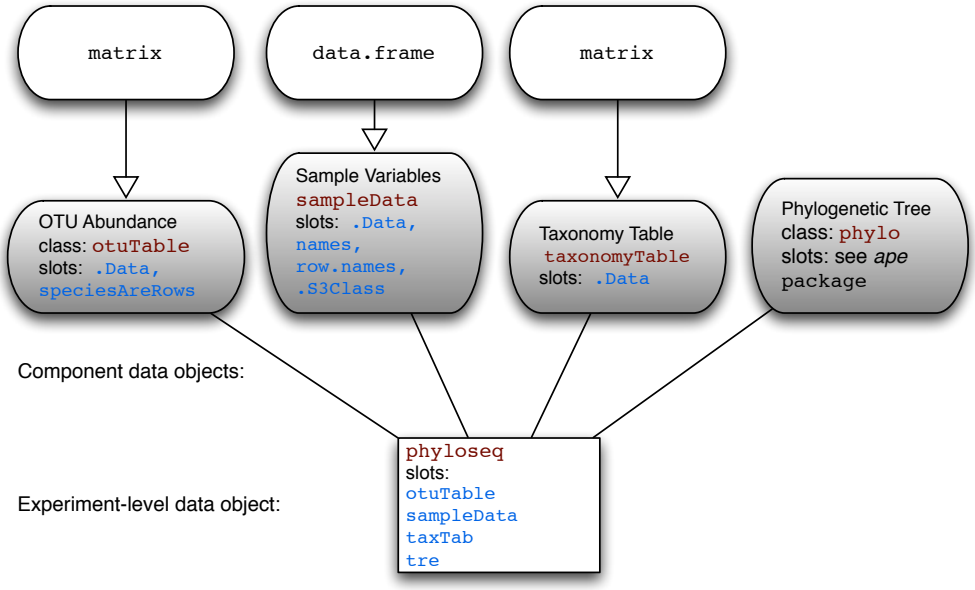

Figure 2: **C**lasses and inheritance in the *phyloseq* package. Core data classes are shown with grey fill and rounded corners. The class name and its slots are shown with red- or blue-shaded text, respectively. Inheritance is indicated graphically by arrows. Lines without arrows indicate that a higher-order class contains a slot with the associated data class as one of its components.

B Installation

B.1 Bioconductor Version

The main release version of *phyloseq* is available from Bioconductor (`http://bioconductor.org/packages/devel/bioc/html/phyloseq.html`), and can be installed from the `R` console with the following two commands:

```
>         source("http://bioconductor.org/biocLite.R")
>         biocLite("phyloseq")
```

If the above fails it is probably because of a mismatch with your current version of R and that required. One option is to update your version of R; not necessarily recommended if this means jumping out of the current stable version. You can always install the development version from GitHub (see below).

B.2 GitHub Development Version

`R` has tools to install packages directly from GitHub. The GitHub repository of *phyloseq* is the most frequently updated development version, and also the place to suggest or contribute code. Use the following command(s) from within `R` to perform the installation:

Install the Bioconductor dependencies (genefilter is optional):

```
>         source("http://bioconductor.org/biocLite.R")
>         biocLite("multtest")
>         biocLite("genefilter")
```

Install the CRAN dependencies (doParallel is optional):

```
>   install.packages("ape")
>   install.packages("doParallel")
>   install.packages("foreach")
>   install.packages("ggplot2")
>   install.packages("igraph")
>   install.packages("picante")
>   install.packages("vegan")
>   install.packages("RJSONIO")
>   install.packages("plyr")
```

Finally, install phyloseq using GitHub as an alternative repository with the `install_github` command in the devtools package.

```
>   # Make sure you have devtools installed
>   install.packages("devtools")
>   # Load the devtools package
>   library("devtools")
>   # Build and install phyloseq
>   install_github("phyloseq", "joey711")
```

Also, make sure to check the "Installation" page on the phyloseq wiki at GitHub: `https://github.com/joey711/phyloseq/wiki/Installation` if you have any problems, as this is likely to be the first place news about installation will be posted. Also check out the rest of the development homepage on GitHub (`https://github.com/joey711/phyloseq`), as this is the best place to post issues, bug reports, feature requests, contribute code, etc.

For running parallel implementation of functions/methods in *phyloseq* (e.g. `UniFrac(ex1, parallel=TRUE)`), you will need also to install a function for registering a parallel "backend". Only one working parallel backend is needed, but there are several options, depending on the details of your particular system any one of the following may be sufficient:

```
>   install.packages("doParallel")
>   install.packages("doMC")
>   install.packages("doSNOW")
>   install.packages("doMPI")
```

C Bibliography

References

[1] Caporaso, J. G. *et al.* QIIME allows analysis of high-throughput community sequencing data. *Nature Methods* **7**, 335–336 (2010).

[2] Schloss, P. D. *et al.* Introducing mothur: Open-Source, Platform-Independent, Community-Supported Software for Describing and Comparing Microbial Communities. *Applied and Environmental Microbiology* **75**, 7537–7541 (2009).

[3] Cole, J. R. *et al.* The Ribosomal Database Project: improved alignments and new tools for rRNA analysis. *Nucleic Acids Research* **37**, D141–5 (2009).

[4] Gentleman, R. C., Carey, V. J., Bates, D. M. & others. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology* **5**, R80 (2004).