16S Amplicon Sequencing Data Analysis
April 8, 2015

Introduction

Amplicon analysis is a multi-step process. There are decisions to be made in analysis and implementation. We will go through an example touching on a couple of the issues in amplicon analysis: compatible of reading files in different programs, different ways to bin sequences into OTUs, and whether to use paired or single-end reads. We will then briefly touch on ways to explore the data using R.

The working environment

We are going to use QIIIME for this workshop, but there are other software programs available (see reference section). Each of these works in a similar way, where some generous folk(s) wrote a bunch of scripts, and you invoke those scripts to analyze your data. QIIME for the most part is just a compilation of python scripts, and the QIIME website offers many tutorials for data analysis.

The Desktop contains two folders that come with the QIIME virtual image, Before_you_start and Shared_Folder (See http://qiime.org/install/virtual_box.html). Raw data is located in the Amplicon_Workshop Folder, and FastQC is a java script that we will use to visually explore our data quality.

Preparing the sequences

Open terminal, and change directory ("cd") to the Amplicon_Workshop directory

```
$ cd Desktop/Amplicon_Workshop
```

You can see what files are in that folder with the following command. "ls" lists the contents of the directory, and the flags "-lh" specifies a long list where each item is a row (the "l") and print sizes in human readable format (the "h").

```
$ ls –lh
```

Paired sequences from the UC Berkeley facility are returned already separated by barcodes, each with a forward read (R1) and a reverse read (R2). Begin by concatenating all the R1 reads together; this just links all the sequences from the different files in series.

```
$ cat *R1*.fastq > R1.fastq
```

Repeat for the R2 reads.

```
$ cat *R2*.fastq > R2.fastq
```

A check on whether the correct files were concatenated is that the two files should be exactly the same size. What are the sizes of the files?

R1.fastq	
R2.fastq	

We are now going to use FastQC, to check on the quality of reads. This will inform if/where you want to trim sequences.

```
$ fastqc
```

This opens a graphic user interface (GUI). Click File then Open and navigate to the Amplicon_Workshop directory. Open your newly concatenated R1.fastq file. After loading, you can view Per Base Quality .png graph. Repeat for the R2.fastq file.

Describe the quality of the reads. Would you use both directions or just one? Would you trim?

R1.fastq	
R2.fastq	

There are several separate, stand-along programs that trim the ends of reads (e.g. Trimmomatic, FastX-Toolkit).

Working in QIIME

Back in Terminal, check that QIIME in configured correctly.

```
$ print_qiime_config.py –t
```

Later on in the analysis, we'll need a reference database, taxonomy, and phylogenetic tree for bacteria. In this case, we'll use Greengenes. Here we are defining them as environment variables using their paths. Use the tab feature to auto-complete directories.

```
$ export reference_seqs=/home/qiime/qiime_software/gg_otus-
13_8- release/rep_set/97_otus.fasta
$ export reference_tree=/home/qiime/qiime_software/gg_otus-13_8-
release/trees/97_otus.tree
$ export reference_tax=/home/qiime/qiime_software/gg_otus-
13_8- release/taxonomy/97_otu_taxonomy.txt
```

Joining Paired Reads

Now we are going to take the forward and reverse Illumina reads and join them. This command tries to pair the forward and reverse sequences from each amplicon on the flowcell, where it uses the sequence header to identify what pair should be joined. One implementation of joining reads in QIIME uses the "fastq-join" method (-m) , and outputs the resulting files to the "fastq_joined" directory (-o).

```
$   join_paired_ends.py –f R1.fastq –r R2.fastq –m
fastq-join –o fastq_joined
```

How many reads were successfully paired? Use "word count" and "lines" to check the number of sequences that were input and how many were successfully paired. If every single pair of sequences was joined, the number of sequences in the original files and the joined file would be identical. Since fastq files contain four pieces of information for each sequence (sequence header; sequence; filler; and quality information), the actual number of sequences is the number of lines in the fastq file divided by 4. (On the other hand, fasta files have two pieces of information for each entry, the sequence header and then the sequence.)

```
$ wc –l R1.fastq
```

How many input sequences were there? How many were successfully paired? Is this a good percentage? Do you want to move forward?

R1.fastq	
R2.fastq	
Paired reads	
Percentage of reads paired	

QIIME has another option for joining reads called SeqPrep (-m SeqPrep). There are also stand-alone programs implemented elsewhere (PANDASeq, UPARSE). Should you use one of those methods of pairing reads instead?

The sequence header for each sequence needs to be changed to work with downstream QIIME analysis. Right now, the file has sequence identifiers as given by the sequencing facility, for example:

```
@M02248:9:000000000-AB7RF:1:1101:16073:1341  1:N:0:19
```

```
@M02248:9:000000000-AB7RF:1:1101:16073:1341  1:N:0:19
```

The first box is unique for each amplicon. The second box indicates the read; in this case, this is the forward, R1. The third box is the sample (i.e. associated with a barcode/index). To work with QIIME, sequences headers need to read a particular way. Sample name is followed by an underscore, and then followed by a unique amplicon (see http://qiime.org/scripts/add_qiime_labels.html). For example:

```
@Sample10A_16073:1341
```

The sequencing's facility 19 is known to me as Sample10A. The unique identifier stays the same. I have a crude python script that changes these facility-generated sequences headers into a form that QIIME can read, based on a sort of mapping text file (not to be confused the

with QIIME metadata mapping file). Open up the script and check it out. Then, run it by typing into Terminal:

```
$ python rewrite_header_paired.py
```

OTU Binning

The next step requires a fasta file, so first convert the fastq file to fasta.

```
$ convert_fastaqual_fastq.py –c fastq_to_fastaqual –f
joined.fastq –o fasta/
```

In the interest of time, we'll be doing a closed reference OTU picking, that is, only those sequences that cluster with known bacterial taxa in the reference data will be included. This is appropriate for real data sets in limited situations only. We specify a parameters file that facilities a fast processing of OTUs. Let's get this started and then talk about some methods for OTU binning.

```
$ pick_closed_reference_otus.py –o otus/ –i fasta/joined.fna
–r $reference_seqs –t $reference_tax –p fast_params.txt
```

Look through the tutorials on the QIIME website (http://qiime.org/tutorials/index.html). QIIME specifies many methods for OTU binning. Describe some of the features of each, noting appropriateness for different environments and speed. What are some issues with sequences that must be considered when doing *de novo* OTU picking that may not be relevant for closed reference? (Chimeras and singletons)

Closed reference	
Open reference	
De novo	
Multi---step	

The primary output that we can about from this command is the *OTU table*, or the number of times each operational taxonomic unit (OTU) is observed in each sample. QIIME uses the Biological Observation Matrix (BIOM) format for representing these files, and these can be converted to .txt files: http://biom---format.org/documentation/biom_conversion.html

```
$ biom summarize–table –i otus/otu_table.biom –o
otu_table_summary.txt
```

One thing to look at in the summary is the value to which all samples should be subsampled. Some samples have a greater number of reads than others, and to ensure that there is no bias due to uneven sequencing depth, it's customary to rarefy, or subsample, all samples to a given number of sequences. In practice, some samples have a very low number of reads and may

need to be excluded. You have to weight the balance of including as many samples as possible with ensuring a sufficient number of sequences per sample.

Based on the results of the table summary, how many sequences per sample would you rarefy to?

Rarefaction value	

Now we are going to see how the bacterial compositions in the different samples relate to each other – so called 'beta diversity' analyses. There are many ways to compare the composition across samples, and here we use Unifrac, one that takes into account phylogenetic relationships. Which metric(s) to use are specified in the parameter file. We will also subsample to a particular rarefaction value.

```
$ beta_diversity_through_plots.py –i otus/otu_table.biom –e
60000 – o bdiv/ –t $reference_tree –m metadata_map.txt –p
bdiv_params.txt
```

Look at the unweighted unifrac plot by opening the "index.html" file in a web browser.

To Pair or Not to Pair?

Many researchers do not pair reads, for a myriad of reasons. In this exercise, we will briefly explore if our conclusions of beta-diversity between samples changes whether we used paired reads or just one of the directions.

Working from the original, raw files, this step creates one fasta file with the appropriate sample names and does basic quality control

```
$ split_libraries_fastq.py –i
2A_S3_L001_R1_001.fastq,3A_S5_L001_R1_001.fastq,4A_S7_L001_
R1_001.f astq,61A_S121_L001_R1_001.fastq ––sample_ids
Sample2A,Sample3A,Sample4A,Sample61A –o R1_slout/ –m
metadata_map_single.txt ––barcode_type 'not–barcoded' –q 19
```

Then repeat the OTU---picking and beta---diversity calculations.

```
$ pick_closed_reference_otus.py –i R1_slout/seqs.fna –o
R1_otus/ –r $reference_seqs –t $reference_tax –p
fast_params.txt

$ biom summarize–table –i R1_otus/otu_table.biom –o
R1_otu_table_summary.txt

$ beta_diversity_through_plots.py –i R1_otus/otu_table.biom
–e 59000 –o R1_bdiv/ –t $reference_tree –m metadata_map.txt
–p bdiv_params.txt
```

Now compare the two distance matrices to see how the samples are related across the two analysis pipelines.

```
$ transform_coordinate_matrices.py -i
bdiv/unweighted_unifrac_pc.txt,R1_bdiv/unweighted_unifrac_pc
.txt -r 100 -o procrustes_output

$ make_emperor.py -c -i procrustes_output/ -o
procrustes_output/plots/ -m metadata_map.txt
```

Based on the results, would you use paired reads or not?

Beginning data analysis

To explore the OTU table in Excel, you can convert the .biom table to a classic txt file. For large data sets, these tables can be tricky to view in Excel.

```
$ biom convert -i otus/otu_table.biom -o otu_table.txt --
header-key taxonomy -b
```

For this tutorial, we used a reduced data set for teaching purposes, but for the exploration of data analysis, we will use a full data set. The full data set consist of 72 soil samples, which was extracted in two replicates A & B (except for 1 sample, which just has an A). With the negative control, there are a total of 144 samples in this library. The soil samples span six collection sites. Some are surface samples and others were collected in mammal burrows.

We will be using R for some basic data analysis. R is a statistical environment that requires running analyses using command-line like instructions. There are basic capabilities that come in the "base" working environment. Additional computation capabilities are embedded within "libraries" that you install within the R environment.

Enter the R environment through Terminal.

```
$ R
```

We are going to use two libraries, and they come installed with this R but you need to load them. These two are used for analysis for ecological data.

```
> library(labdsv)
> library(vegan)
```

We need to navigate to the directory where are files are located.

```
> setwd("~/Desktop/Amplicon_Workshop/Full_Data_Set/")
```

There are libraries that deal with reading in BIOM tables (biom, qiimer). Here we'll just read in the otu table in text format using the base functionality in R.

First read in the entire biom table that was converted to a text file. In the table, the first column is the OTU ID, and the last column is the assigned taxonomy for that OTU. Intermediate columns are the different samples.

```
>
otu.tab.wtax=read.table("otu_table.txt",header=TRUE,row.names=1,sep='\
t', check.names=FALSE,skip=1,comment.char="")

> help(read.table)
```

[and q to exit]

The format of the otu table at this point is where the rows are OTUs and the columns are samples. For ecological analysis, the table will need to be transposed (samples are rows and OTUs are columns) but for now this format is easier for some basic quality control.

Quality control

The 16S bacterial genes can amplify organelles, since those are bacteria inside other cells. In some instances, you might want to exclude those OTUs. First, look to see if there are OTUs with the taxonomy assignment that includes 'mitochondria' or 'chloroplast'. I happen to know that in the Greengenes database, Chloroplast is capitalized while mitochondria is not, so I add the flag to ignore the case so that the search is not case sensitive. This command searchers (or greps) for the 'phrase' with in the taxonomy column of the OTU table.

```
> length(grep('chloroplast',otu.tab.wtax$taxonomy,ignore.case=TRUE))
> length(grep('mitochondria',otu.tab.wtax$taxonomy,ignore.case=TRUE))
> length(grep('chloroplast',otu.tab.wtax$taxonomy,ignore.case=TRUE)) +
length(grep('mitochondria',otu.tab.wtax$taxonomy,ignore.case=TRUE))
```

How many OTUs match these types?

chloroplast	
mitochondria	
combined	

Note that you could also use the grep command to look for OTUs of a taxonomic class of interest (for example, Pseudomonads).

Here is one way to exclude those rows (or OTUs) that have a particular keyword in the taxonomy column. First I'm creating a new matrix with the mitochondria excluded, and then from that new matrix, I'm excluding the chloroplast.

```
> otu.tab.wtax.filt= otu.tab.wtax[-
c(grep('mitochondria',otu.tab.wtax$taxonomy,ignore.case=TRUE))
,]
> otu.tab.wtax.filt= otu.tab.wtax.filt[-
c(grep('chloroplast',otu.tab.wtax.filt$taxonomy,ignore.case=TRUE))
,]
```

You can check on the dimensions of the old table, and given the number of OTUs that match these types above, then confirm that your new table has the intended number of OTUs.

```
>    dim(otu.tab.wtax)
>    dim(otu.tab.wtax.filt)
```

Original OTU table	
Filtered OTU table	

The dimension command returns the number of rows by the number of columns. You can return just one of those dimensions by specifying which entry of the dimension, or you can use a command that calculates the number of rows and columns. The next command determines the number of rows (OTUs) of the original OTU table matrix then subtracts the number of rows form the filtered matrix. This should match the combined value above. You can also ask R to check whether they are identical.

```
> nrow(otu.tab.wtax)-nrow(otu.tab.wtax.filt)
```

Another issue is what do with the sequences in the negative control. Even in the negative control looked empty in a gel, it is likely that sequencing the lane will show sequences. There are a couple of routes that people take. Some exclude any OTUs that are present in negative controls. Others "subtract" the number of OTUs in the negative from all over samples.

First, you can see how many OTUs were in the negatives.

```
> length(which(otu.tab.wtax.filt$SampleNeg!=0))
```

Number of OTUs in negative control	

You could see what taxa are in the negative controls, but this is often not that informative, since it depends on the distribution of those taxa across the dataset. In my experience, many of the taxa in negative controls appear to be spillover from other lanes (potentially arising from "mistagging"). Therefore, my referred method is to subtract the number of OTUs in the negative from all other samples.

First, I exclude the taxonomy column.

```
> otu.tab.filt  =  subset(otu.tab.wtax.filt,  select=-taxonomy)
```

Then, put the Sample Negative in the last column and sort by decreasing value of the Sample Negative – this just makes it easier for the next step.

```
> otu.tab.filt2=otu.tab.filt[,c((1:ncol(otu.tab.filt))[-
(grep("SampleNeg",names(otu.tab.filt)))],
grep("SampleNeg",names(otu.tab.filt)))]
```

```
> otu.tab.filt3=otu.tab.filt2[order(-otu.tab.filt2$SampleNeg),]
```

Now, subtract the last column from every other column, and replace negative values with zeros.

```
> otu.tab.filt4=otu.tab.filt3[,1:(ncol(otu.tab.filt3)-
1)]- otu.tab.filt3[,ncol(otu.tab.filt3)]
> otu.tab.filt4[otu.tab.filt4<0]  = 0
```

Just to be tidy, remove empty OTUs that may have fallen out in the cleaning process.

```
> otu.tab.filt5=otu.tab.filt4[rowSums(otu.tab.filt4)>0,]
```

Now, you have an OTU table (OTUs as rows and samples as columns) that has been been stripped of mitochondria/chloroplast as well as reads in the negative control.

Some basic data exploration

Most of the analysis in R requires that the OTU table be transposed from the way it is now; that is, R expects the rows are samples and the columns are species.

```
> otu.tab=t(otu.tab.filt5)
```

How many bacterial OTUs were identified in this dataset?

Number of OTUs	
Given our choice of OTU clustering choices, is this value on the low end or high end of estimates?	

For some analysis, you will need to use a rarefied data table to remove uneven sampling depth. You could see the lowest and highest samples by number of reads using this command. It's quite a range!

```
> c(head(sort(rowSums(otu.tab))),  tail(sort(rowSums(otu.tab))))
```

Based on that, what would you choose to rarefy your samples to? Remember, you are trying to weigh sufficient representation of each sample and not excluding too many samples.

Rarefaction value	

And take a moment to realize how amazing that! You are describing the bacteria community in a sample based on **thousands** of sequences, and you are doing that for over a hundred samples. Previously, this would be done with cloning then Sanger sequencing individual PCR products. This also means you are "throwing away" thousands of sequences.

Exclude those samples that are less than the rarefaction value, then rarefy to that number.

```
> otu.tab.rar= otu.tab[which(rowSums(otu.tab)  > 46000), ]
> otu.tab.rar=rrarefy(otu.tab.rar,46000)
```

Without using R, think about what the minimum and maximum row sums for your new data table would be.

Minimum row sum value	
Maximum row sum value	

What percentage of sequences from the unrarefied OTU table did you retain in the rarefied OTU table?

Percentage	

Now, read in the table of metadata.

```
> meta=read.table("metadata_map.txt",header=T)
```

The tables need to be lined up, so that the rows (ie. Samples) are the same in both tables). Since the OTU table has fewer samples (we excluded the Negative, as well as any samples that were excluded after rarefaction). Create a "rarefied" metadata data according to the rows of the OTU table, then check that it worked.

```
> meta.rar=meta[rownames(otu.tab.rar),]
> identical(rownames(otu.tab.rar),rownames(meta.rar))
```

A common technique is to visualize potential differences among samples. Here, we'll use non---metric multidimensional scaling (NMDS) to plot samples on a common scale. First, we'll create a distance matrix of pairwise distances between samples, using one of many indices available in R. Then we'll NMDS to turn those distances into 2-dimensional points.

```
> distBray <- vegdist(otu.tab.rar,  'bray')
> nmdsBray <- nmds(distBray)
```

Plot the results, coloring by certain parameters in the metadata table.

```
>plot(nmdsBray,pch=21,cex=1.5,bg=meta.rar$Site)
legend("topleft",legend=levels(factor(meta.rar$Site)),pch=21,pt.bg=lev
els (factor(meta.rar$Site)))
```

R is a powerful program, and there are plenty of online tutorials to help you get familiar with the syntax.

Some other tools for amplicon---based analysis
These will have varying levels of documentation and help.

- mothur (http://www.mothur.org/wiki/Main_Page)
- UPARSE (http://drive5.com/uparse/)
- FastX Toolkit (http://hannonlab.cshl.edu/fastx_toolkit/)
- Oligotyping
- PANDAseq (https://github.com/neufeld/pandaseq)

Some additional reading

Goodrich, J.K., Di Rienzi, S.C., Poole, A.C., Koren, O., Walters, W.A., Caporaso, J.G., Knight, R. and Ley, R.E. (2014) Conducting a microbiome study, *Cell*, **158**, 250---262.

Edgar, R.C. (2013) UPARSE: highly accurate OTU sequences from microbial amplicon reads, *Nat. Methods*, **10**, 996–998.

Lundberg, D.S., Yourstone, S., Mieczkowski, P., Jones, C.D. and Dangl, J.L. (2013) Practical innovations for high---throughput amplicon sequencing, *Nat Methods*, **10**, 999--1002.

Nguyen, N.H., Smith, D., Peay, K. and Kennedy, P. (2014) Parsing ecological signal from noise in next generation amplicon sequencing, *The New phytologist*, 10.1111/nph.12923.

Bokulich, N.A., Subramanian, S., Faith, J.J., Gevers, D., Gordon, J.I., Knight, R., Mills, D.A. and Caporaso, J.G. (2013) Quality-filtering vastly improves diversity estimates from Illumina amplicon sequencing, *Nat Methods*, **10**, 57---59.