

Lab 2 write-up

your_name_here

Due on XXXX

Week 2 Lab

Homework for Next Week

- Fill in all of the answer blocks and coding tasks in this lab notebook.
- Complete all tasks in the Homework section (bottom of the Rmd).

Learning Objectives

- Understand the features and formatting of common genomic data files.
 - Explore how genome sequence features influence the accuracy/quality of an assembly.
 - Get comfortable manipulating DNA sequences in R.
 - Create custom functions in R to perform analyses using less code.
-

Overview

This week we will get acquainted with real sequence data, and try out some approaches to assembling a few individual genes. Then we'll look at our large, whole genome files and take some time to understand exactly what they tell us. We'll also start writing our own functions in R. Data analysis can get repetitive. If you find yourself writing the same code over and over again, making a function to do it for you saves time and makes your code more readable.

HW 1 (algorithms, reading)

Brief instructor-led discussion.

Assembly in R

For local (non RStudio Cloud) users If you are using RStudio locally (as in, downloaded on your computer and not in RStudio Cloud), you need to do a little extra installation and loading of packages and files. From Canvas (in the Week 2 module), download `lab2_assembly.R` and `package_loading.R` and save them into the same folder where you've saved this Rmd (`lab2_BIS23B.Rmd`).

If everything is in the right place, when you run the next lines of code, you will first install any missing packages, then load up some data and functions for today's lab.

```

# you only need to run these lines if you are using RStudio on your home
# computer; in RStudio Cloud the scripts have already been loaded.
source("package_loading.R")
source("lab2_assembly.R")

```

Checking out our data Whether you are using RStudio Cloud or you ran the scripts locally on your computer, you should now see a few objects in your environment (top right). These include several custom functions as well as 7 sets of mystery sequences. These functions require the package `Biostrings` to be loaded. (This was done behind the scenes on RStudio Cloud, or by using the script `package_loading.R` for people using a downloaded version of RStudio.)

Now let's get to our new mystery data. We can begin by looking at `mystery1`.

```
mystery1
```

```
## [1] "GATTACGACTCACT" "TTACGACTCACTGGA" "TCACTGAAGTATCGCG" "ATCGCGATCATCAAC"
```

```
length(mystery1)
```

```
## [1] 4
```

```
nchar(mystery1)
```

```
## [1] 14 15 16 15
```

With these three lines of code we first viewed the raw data, counted the number of separate entries (strings) in the vector, then counted the number of characters in each string. In this case, the reads are not all the same length. But that won't matter for the `assemble()` function.

Let's use this `assemble()` function. We'll talk about this together as a class before setting you loose on some other data.

```

a1 <- assemble(mystery1, min_overlap = 5)
a2 <- assemble(mystery2, min_overlap = 5)

# Let's calculate coverage (how many reads overlap each
# site in the assembled sequence, on average)

# This is a measure of quality -- more coverage == more
# confidence that this is a correctly-assembled sequence.

# to find the coverage, we get the total length of our assembled contig(s)
len <- nchar(a1)
# then we get the total length of all our reads combined (which will be
# bigger than len, since some overlap)
total <- sum(nchar(mystery1))

# we divide the total length of all reads by the length of the contigs
coverage <- total/len
coverage

```

[1] 1.764706

After you've assembled a set of sequences, you often want to make some calculations about the quality of your assembly. *Did they all overlap well enough to form a single sequence (contig)? How long is the contig? What's the coverage (the average number of reads for any given site in the contig)?* Next week you'll get more in depth with assessing assembly quality – we need to assess the quality of our halophile data.

Pause here for an instructor-led discussion.

(Slightly) larger data sets We still won't get anywhere near the size of a typical FASTQ file of sequencing reads, but let's assemble some of our other sets of reads.

A scavenger hunt to complete with in your breakout room

1. Two of these mystery vectors are actually different sets of reads from the same exact sequence. Which ones are they? How do you know?
2. One of these sets of reads is from a complete, circular sequence of DNA. Which one? How do you know?
3. Does the minimum overlap make much of a difference for the assembly? What happens if it's very small, or large relative to your read lengths?
4. Do you get the same assembly every time if you run the exact same code? Check with group to see if your assemblies look the same.

Answer block – scavenger hunt answers (1-2 sentences each.)

Write your answers below

- 1.
 - 2.
 - 3.
 - 4.
-

```
# Use this space to write the code for assembling the mystery sequences
# Assemble mystery2, 3, 4, 5, 6, and 7, and store them as
# a2, a3, a4, a5, a6, a7
```

```
#####
### Your code below      ###
#####
```

```
# example for mystery2
a2 <- assemble(mystery2, min_overlap = 5)
```

```
#####
#####
```

Pause here for some discussion of our assembly findings

How can functions help us?

We have assembled 7 different sets of mystery reads. We'll want to calculate some statistics about each of them, for example the counts of particular nucleotides. Let's create some functions to do this work for us, so that we don't have to rewrite the same code many times.

```
# Practice creating functions

# we first specify the function's name and the name of the arguments (inputs)

contig_num <- function(assembly)
{
  # this function, contig_num, takes an assembly
  # we'll assume the assembly is a vector of strings

  # the number of contigs is the number of different strings
  # in the vector, which is given by the length() function
  len <- length(assembly)

  # at the end of a function, either just put name of the output, (here
  # it's len), or use return() with your output
  return(len)
}

# This was not a useful function. length() does the job by itself.
contig_num(a1)
```

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

```
length(a1)
```

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

```
# But maybe we want to count the number of "A" base pairs in each contig.  
# That requires a bit more code
```

```
count_A <- function(assembly)  
{  
  # this function, count_A, takes an assembly (vector of strings)  
  # and will return a vector of integers (# of As in each contig)  
  
  # split the string(s) into a list, then into a vector  
  assembly_vec <- unlist(strsplit(assembly, split = ""))  
  
  # note: we nested two functions, unlist() and strsplit(), to write  
  # more compact code  
  
  #check which entries equal A  
  A_total <- sum(assembly_vec == "A")  
  
  return(A_total)  
}
```

```
# Run the lines above, and the function should now appear the  
# Functions section in your environment
```

```
count_A(a1)
```

```
## [1] 10
```

```
# now let's look at a mystery function  
# read the code and guess what it does  
# then we can add comments together to explain it
```

```
mystery_func <- function(assembly,reads)  
{  
  genome_len <- sum(nchar(assembly))  
  total_read_len <- sum(nchar(reads))  
  
  cov <- total_read_len/genome_len  
  
  return(cov)  
}
```

```
# example output when mystery2 and its assembly are used  
mystery_func(a2, mystery2)
```

```
## [1] 10.97561
```

Functions can be very useful. You may even find yourself writing some short utility functions, then calling them inside of larger functions you write. If you read the code for `assemble()`, you'll see that it calls several

simpler functions. This follows the principle of modular programming, where you write small programs to handle each distinct computational task, then combine them together in the right order inside of a larger function or loop.

```
# Here you have a pre-written function to calculate GC content.
# But, the lines of code ARE IN THE WRONG ORDER!
# Read it over, move the code into the right order,
# and add comments to explain each step.

#####
### Your code below      ###
#####

count_gc <- function(assembly)
{
  GC <- C_total + G_total

  C_total <- sum(assembly_vec == "C")

  assembly_vec <- unlist(strsplit(assembly, split = ""))

  G_total <- sum(assembly_vec == "G")

  return(GC)
}

# When that's complete, use the code from count_gc,
# but add a few lines so that you return the fraction
# of all nucleotides that are GC, not the total count
# I.e. it should be a number between 0 and 1.

fraction_gc <- function(assembly)
{
}

# VALIDATION
# your answer for fraction_gc(a1) should be 0.47 (rounded)
# your answer for fraction_gc(a2) should be 0.59 (rounded)

#####
#####
```

Pause here for a short break, then we'll look at some data together.

What does genomic data look like?

We are mostly going to avoid dealing with the raw sequencing read data (FASTQ files). These files are very large and messy to work with. It turns out that R doesn't manage its memory efficiently for gigantic data sets, so genome assembly programs tend to be written in other languages.

Let's focus first on FASTA files. These are really just text files, but with a specific format. Your instructor will open the file `strain1926.fasta` with a text editor. If you want to look at FASTA files, it's good practice to have a lightweight text editor installed on your computer. Some examples are atom or Notepad++ (windows only). Some hardcore people like VIM, but it's not as easy for starting out.

As we start looking through the strains, you'll each have two strains to focus on. One that is yours alone, and one that you share with a group.

Link to google doc has been removed for student privacy and data protection.

Homework for Week 3, due Wednesday, 4/15, at 10:59pm

Besides the data entry in the google doc, all work should be completed inside of this Rmd file and uploaded to Canvas as a pdf or html file.

Finish up labwork (2 pt)

Make sure that any short responses or coding snippets above have been filled in with your work, and your other data is entered into the google doc.

Online discussion (part of Week 2 participation points)

Write a follow-up or reply to another student's follow-up on the Week 2 discussion post on Piazza.

Readings and response (3 pts)

Read over these three very short blog posts: one from Keith Bradnam (he was formerly at UC Davis) summarizing an effort to compare genome assembly approaches, and two from Elin Videvall (she is now a postdoc in conservation genomics at the Smithsonian) explaining the N50 statistic.

- <https://haldanessieve.org/2013/01/28/our-paper-making-pizzas-and-genome-assemblies/>,
- <https://www.molecularecologist.com/2017/03/whats-n50/>,
- <https://www.molecularecologist.com/2017/04/the-first-problem-with-n50/>.

Response prompt In week 3 we'll spend some time assessing the quality of the assemblies of our halophile genomes. Assembly quality is often evaluated using statistics like N50, or simple counts like the number of contigs, the total summed length of all the contigs (compared with a known genome of a similar organism), etc.

With assembly quality in mind, what are 1-2 main take-aways from the Assemblathon 2 effort, and what are some things to be concerned about when calculating statistics like N50?

Your response below (100-200 words)

Data Exploration (5 points total)

Data You have each been assigned a single strain from our BIS 23A data (*link to google doc has been removed for student privacy and data protection*). The data are all FASTA files, which we briefly looked at during Lab 2. These files contain nothing but the DNA sequences of all of the *contigs*. Remember, in most genome assemblies you don't succeed in combining all the reads into one long genome. Instead, you are typically able to combine the reads into a number of longer sections, but there will still be gaps. Because of this, a FASTA file will have multiple sequences: one for each contig (contig stands for contiguous sequence). These files are loaded into the Lab 2 assignment on RStudio Cloud.

IMPORTANT: READ CLOSELY. If you're working using a downloaded version of RStudio, you'll need to download your primary and secondary strain FASTA files from Canvas (in the "data" folder in the Files tab). In the folder where you have your Lab 2 Rmd saved, make a new folder called "data" and save the files inside of there. The folder should be called "data" will all lowercase. This will save you headaches later on in the course by ensuring that we all have standardized names for our folders. **END OF EXPLANATION FOR LOCAL (DOWNLOADED) RSTUDIO USERS.**

As part of your homework, you will do a baseline check to see if your primary strain is truly a pure isolate, or is a mix of several species. One simple way to do this is to choose a gene that is typically only found as a single copy in a genome, and search for how many copies you can find. If you find more than one copy, and the copies are different from each other, that means that the genome is probably not a single pure isolate. Instead, there might be several different species of microbe that were living in the same sample, and when you sequenced them you built some contigs of DNA from one species and some contigs from another.

Complete the task below and enter data into the google doc (2 of the 5 points) There's a simple online tool to check for multiple copies of the 16S rRNA gene (more on that in the Week 3 lecture). This tool is called ContEst16S. Navigate to the page and click "Bacteria", then click "Upload FASTA" and select your FASTA file from your computer. It might take a minute or two to upload. When it's done uploading, you click "Run ContEst16S". After a minute or two, it will produce an output lower down on the page. There are a few possible outcomes:

- It might find exactly one 16S sequence (probably a pure isolate)
- It might find no 16S sequences (purity unclear)
- It might find several 16S sequences that are very similar (possibly pure)
- It might find several 16S sequences that are very different (definitely not pure)

In the google doc, record the "Decision". Also, if the tool finds any 16S fragments, scroll down the page to "Sequences" and copy and paste the first sequence (the actual nucleotides, not the species name) into the "16S Sequence 1" column in our google doc and paste the second sequence (if relevant) into the "16S Sequence 2" column. If there are more than two 16S sequences, only paste the first two, and make a note that there are more in the "Notes" column.

Strain 1926 has already been completed as an example of what we are looking for.

Coding (3 of the 5 points)

Complete short tutorial on Biostrings We have created a short tutorial on using the Biostrings package. This package allows for easy manipulation of FASTA sequence data (hooray, no more `unlist(strsplit())` shenanigans). The tutorial is designed to run on the command line in RStudio Cloud. Even if you plan to complete your RMarkdown work in a downloaded form, it is probably still easier to do this tutorial using RStudio Cloud. For best results, use the Google Chrome browser.

If you are using RStudio Cloud, you can simply run the command `swirl()` in your console (lower left). Then enter your name, select “Genome Sequence Data in R”, then select the “Working with Biostrings” lesson.

One minor point, use the left arrow, `<-`, not the equals sign, `=`, when storing variables for this tutorial. That will help the automatic system understand your code.

Note: if you cannot make this tutorial work on RStudio Cloud, contact your instructors and we can help you install the tutorial locally. It’s a little finicky, but not too hard.

Try loading your own primary strain to check a few genomic features You can use the function `readDNASTringSet()` with your FASTA file as the input to create a `DNASTringSet` object in your environment. Use functions like `width()` (the length of all contigs) and `mean()` to calculate the following:

- The length (AKA width) of the largest contig in your strain’s assembly,
- The mean length of all the contigs in your assembly,
- The total number of contigs (after loading, you can just look at environment to see the dimensions of your `DNASTringSet`).

Enter these data into the appropriate columns of the google doc. All the FASTA files are available in the Rmd file for Lab 2 on RStudio Cloud. If you are using a downloaded version of RStudio, you should have already saved your data from Canvas into a folder called “data” that is located in the same place as your Lab 2 Rmd. The code below models how to get started.

```
#####  
### Your code below ###  
#####  
# use this space to load your FASTA file and get the data requested above.  
  
# template for loading your data as a DNASTringSet  
# you'll need to edit the filename to match the FASTA file for  
# your strain  
my_strain <- readDNASTringSet("data/ENTER_YOUR_STRAIN.fasta")  
  
# for the example (strain 1926), we used the command  
# strain_1926 <- readDNASTringSet("data/strain1926.fasta")  
# we commented the line above so it won't automatically run  
  
#####  
#####
```

Code Appendix

This includes all of the inline code you wrote in the coding sections while you were exploring in this week's lab. You don't need to run anything – it will automatically fill itself in.

```
# Use this space to write the code for assembling the mystery sequences
# Assemble mystery2, 3, 4, 5, 6, and 7, and store them as
# a2, a3, a4, a5, a6, a7
```

```
#####
### Your code below      ###
#####
```

```
# example for mystery2
a2 <- assemble(mystery2, min_overlap = 5)
```

```
#####
#####
```

```
# Here you have a pre-written function to calculate GC content.
# But, the lines of code ARE IN THE WRONG ORDER!
# Read it over, move the code into the right order,
# and add comments to explain each step.
```

```
#####
### Your code below      ###
#####
```

```
count_gc <- function(assembly)
{
  GC <- C_total + G_total

  C_total <- sum(assembly_vec == "C")

  assembly_vec <- unlist(strsplit(assembly, split = ""))

  G_total <- sum(assembly_vec == "G")

  return(GC)
}
```

```
# When that's complete, use the code from count_gc,
# but add a few lines so that you return the fraction
# of all nucleotides that are GC, not the total count
# I.e. it should be a number between 0 and 1.
```

```
fraction_gc <- function(assembly)
{
}
}
```

```

# VALIDATION
# your answer for fraction_gc(a1) should be 0.47 (rounded)
# your answer for fraction_gc(a2) should be 0.59 (rounded)

#####
#####
#####
### Your code below          ###
#####
# use this space to load your FASTA file and get the data requested above.

# template for loading your data as a DNASTringSet
# you'll need to edit the filename to match the FASTA file for
# your strain
my_strain <- readDNASTringSet("data/ENTER_YOUR_STRAIN.fasta")

# for the example (strain 1926), we used the command
# strain_1926 <- readDNASTringSet("data/strain1926.fasta")
# we commented the line above so it won't automatically run

#####
#####

```