DADA2 analysis of 16S rRNA gene amplicon sequencing reads

Jianshu Zhao

2021-03-27

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(fig.width = 10)
knitr::opts_chunk$set(tidy.opts=list(width.cutoff=60),tidy=TRUE)
```

Introduction

Implementing DADA2 pipeline for resolving sequence variants from 16S rRNA gene amplicon ***paired-end*** sequencing reads, adopting the tutorial from https://benjjneb.github.io/dada2/tutorial.html and https://benjjneb.github.io/dada2/bigdata_paired.html with minor adjustments. This report captures all the workflow steps necessary to reproduce the analysis.

Load R packages:

```
## Loading required package: Rcpp
```

```
## [1] '1.14.1'
```

```
## Loading required package: Biostrings
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##     dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which, which.max, which.min
```

```
## Loading required package: S4Vectors
```

```
## Loading required package: stats4
##
## Attaching package: 'S4Vectors'
## The following object is masked from 'package:base':
##
##     expand.grid
## Loading required package: IRanges
## Loading required package: XVector
##
## Attaching package: 'Biostrings'
## The following object is masked from 'package:base':
##
##     strsplit
##
## Attaching package: 'gridExtra'
## The following object is masked from 'package:BiocGenerics':
##
##     combine
## Loading required package: ape
##
## Attaching package: 'ape'
## The following object is masked from 'package:Biostrings':
##
##     complement
## /Users/jianshuzhao/Github/dada2/scripts
```

Get list of input fastq files.

```
# Variable 'input.path' containing path to input fastq files
# directory is inherited from wrapper script dada2_cli.r.

input.file.list <- grep("*fastq", list.files(input.path), value = T)
# input.path <- normalizePath('input/')

# List of input files

# Sort ensures forward/reverse reads are in same order
fnFs <- sort(grep("_R1.*\\.fastq", list.files(input.path), value = T))
fnRs <- sort(grep("_R2.*\\.fastq", list.files(input.path), value = T))

# Extract sample names, allowing variable filenames; e.g.
# *_R1[_001].fastq[.gz]
sample.names <- gsub("_R1.*\\.fastq(\\.gz)?", "", fnFs, perl = T)
sample.namesR <- gsub("_R2.*\\.fastq(\\.gz)?", "", fnRs, perl = T)
if (!identical(sample.names, sample.namesR)) stop("Forward and reverse files do not match.")

# Specify the full path to the fnFs and fnRs
```

```
fnFs <- file.path(input.path, fnFs)
fnRs <- file.path(input.path, fnRs)
```

Generate quality plots for FWD and REV reads and store in Read_QC folder.

```
# Create output folder NOTE: variable 'output.dir' containing
# name of output folder is inherited from wrapper script
# dada2_cli.r.
cwd <- getwd()

readQC.folder <- file.path(cwd, output.dir, "Read_QC")
ifelse(!dir.exists(readQC.folder), dir.create(readQC.folder,
    recursive = TRUE), FALSE)
```

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

```
# Generate plots and save to folder in multi-page pdf

# Forward reads
fwd.qc.plots.list <- list()
for (i in 1:length(fnFs)) {
    fwd.qc.plots.list[[i]] <- plotQualityProfile(fnFs[i])
    rm(i)
}
# Save to file
pdf(paste0(readQC.folder, "/FWD_read_plot.pdf"), onefile = TRUE)
marrangeGrob(fwd.qc.plots.list, ncol = 2, nrow = 3, top = NULL)
```

```
dev.off()
```

```
## pdf
##   2
```

```
rm(fwd.qc.plots.list)

# Reverse reads
rev.qc.plots.list <- list()
for (i in 1:length(fnRs)) {
    rev.qc.plots.list[[i]] <- plotQualityProfile(fnRs[i])
    rm(i)
}
# Save to file
pdf(paste0(readQC.folder, "/REV_read_plot.pdf"), onefile = TRUE)
marrangeGrob(rev.qc.plots.list, ncol = 2, nrow = 3, top = NULL)
```

```
dev.off()
```

```
## pdf
##   2
```

```
rm(rev.qc.plots.list)
```

Trim and filter reads.

```
# Create filtered_input/ subdirectory for storing filtered
# fastq reads
filt_path <- file.path(cwd, output.dir, "filtered_input")
ifelse(!dir.exists(filt_path), dir.create(filt_path, recursive = TRUE),
    FALSE)
```

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

```
# Define filenames for filtered input files
filtFs <- file.path(filt_path, paste0(sample.names, "_F_filt.fastq.gz"))
filtRs <- file.path(filt_path, paste0(sample.names, "_R_filt.fastq.gz"))

# Filter the forward and reverse reads: Note that: 1. Reads
# are both truncated and then filtered using the maxEE
# expected errors algorighm from UPARSE.  2. Reverse reads
# are truncated to shorter lengths than forward since they
# are much lower quality.  3. _Both_ reads must pass for the
# read pair to be output.  4. Output files are compressed by
# default.

rd.counts <- as.data.frame(filterAndTrim(fnFs, filtFs, fnRs,
    filtRs, truncLen = c(240, 200), maxN = 0, maxEE = c(1, 2),
    truncQ = 2, rm.phix = TRUE, compress = TRUE, multithread = TRUE))
# Table of before/after read counts
rd.counts$ratio <- round(rd.counts$reads.out/rd.counts$reads.in,
    digits = 2)
rd.counts
```

```
##                              reads.in reads.out ratio
## Orwoll_BI0023_BI_R1.fastq.gz    62724     43895  0.70
## Orwoll_BI0056_BI_R1.fastq.gz    55342     39185  0.71
## Orwoll_BI0131_BI_R1.fastq.gz    55144     38609  0.70
## Orwoll_BI0153_BI_R1.fastq.gz    44610     33146  0.74
## Orwoll_BI0215_BI_R1.fastq.gz    48227     33882  0.70
## Orwoll_BI0353_BI_R1.fastq.gz    54271     43034  0.79
```

```
# Write rd.counts table to file in readQC.folder
write.table(rd.counts, paste0(readQC.folder, "/Read_counts_after_filtering.tsv"),
    sep = "\t", quote = F, eol = "\n", col.names = NA)
```

Learn the error rates.

The DADA2 algorithm depends on a parametric error model (err) and every amplicon dataset has a different set of error rates. The learnErrors method learns the error model from the data, by alternating estimation of the error rates and inference of sample composition until they converge on a jointly consistent solution. As in many optimization problems, the algorithm must begin with an initial guess, for which the maximum possible error rates in this data are used (the error rates if only the most abundant sequence is correct and all the rest are errors).

```
set.seed(100)
# Filtered forward reads
errF <- learnErrors(filtFs, nread = 1e+06, multithread = TRUE)
```

```
## Warning in learnErrors(filtFs, nread = 1e+06, multithread = TRUE): The nreads
## parameter is DEPRECATED. Please update your code with the nbases parameter.
```

```
## 55620240 total bases in 231751 reads from 6 samples will be used for learning the error rates.
```

```
# Filtered reverse reads
errR <- learnErrors(filtRs, nread = 1e+06, multithread = TRUE)
```

```
## Warning in learnErrors(filtRs, nread = 1e+06, multithread = TRUE): The nreads
## parameter is DEPRECATED. Please update your code with the nbases parameter.
```

```
## 46350200 total bases in 231751 reads from 6 samples will be used for learning the error rates.
```

```
# Visualize the estimated error rates Forward
# plotErrors(errF, nominalQ=TRUE) Save to file
ggsave(paste0(readQC.folder, "/Error_rates_per_sample_FWD.pdf"),
    plotErrors(errF, nominalQ = TRUE), device = "pdf")
```

```
## Saving 10 x 4.5 in image
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

```
# Reverse plotErrors(errR, nominalQ=TRUE) Save to file
ggsave(paste0(readQC.folder, "/Error_rates_per_sample_REV.pdf"),
    plotErrors(errR, nominalQ = TRUE), device = "pdf")
```

```
## Saving 10 x 4.5 in image
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

The error rates for each possible transition (eg. A->C, A->G, ...) are shown. Points are the observed error rates for each consensus quality score. The black line shows the estimated error rates after convergence. The red line shows the error rates expected under the nominal definition of the Q-value. If the black line (the estimated rates) fits the observed rates well, and the error rates drop with increased quality as expected, then everything looks reasonable and can proceed with confidence.

Infer Sequence Variants

This step consists of dereplication, sample inference, and merging of paired reads

Dereplication combines all identical sequencing reads into into "unique sequences" with a corresponding "abundance": the number of reads with that unique sequence. DADA2 retains a summary of the quality information associated with each unique sequence. The consensus quality profile of a unique sequence is the average of the positional qualities from the dereplicated reads. These quality profiles inform the error model of the subsequent denoising step, significantly increasing DADA2's accuracy.

The sample inference step performs the core sequence-variant inference algorithm to the dereplicated data.

Spurious sequence variants are further reduced by merging overlapping reads. The core function here is mergePairs, which depends on the forward and reverse re.samples being in matching order at the time they were dereplicated.

```
# Sample inference of dereplicated reads, and merger of
# paired-end reads
mergers <- vector("list", length(sample.names))
names(mergers) <- sample.names
names(filtFs) <- sample.names
names(filtRs) <- sample.names
for (sam in sample.names) {
    cat("Processing:", sam, "\n")
    derepF <- derepFastq(filtFs[[sam]])
```

```
    ddF <- dada(derepF, err = errF, multithread = TRUE)
    derepR <- derepFastq(filtRs[[sam]])
    ddR <- dada(derepR, err = errR, multithread = TRUE)
    merger <- mergePairs(ddF, derepF, ddR, derepR)
    mergers[[sam]] <- merger
}
```

```
## Processing: Orwoll_BI0023_BI
## Sample 1 - 43895 reads in 11469 unique sequences.
## Sample 1 - 43895 reads in 12878 unique sequences.
## Processing: Orwoll_BI0056_BI
## Sample 1 - 39185 reads in 7614 unique sequences.
## Sample 1 - 39185 reads in 9266 unique sequences.
## Processing: Orwoll_BI0131_BI
## Sample 1 - 38609 reads in 7877 unique sequences.
## Sample 1 - 38609 reads in 9784 unique sequences.
## Processing: Orwoll_BI0153_BI
## Sample 1 - 33146 reads in 8186 unique sequences.
## Sample 1 - 33146 reads in 9228 unique sequences.
## Processing: Orwoll_BI0215_BI
## Sample 1 - 33882 reads in 8421 unique sequences.
## Sample 1 - 33882 reads in 10008 unique sequences.
## Processing: Orwoll_BI0353_BI
## Sample 1 - 43034 reads in 8611 unique sequences.
## Sample 1 - 43034 reads in 10577 unique sequences.
```

```
rm(derepF)
rm(derepR)
```

Construct sequence table

We can now construct a "sequence table" of our samples, a higher-resolution version of the "OTU table" produced by classical methods:

```
seqtab <- makeSequenceTable(mergers)
dim(seqtab)
```

```
## [1]   6 957
```

```
# Inspect distribution of sequence lengths
table(nchar(getSequences(seqtab)))
```

```
##
## 252 253 254
##  91 856  10
```

```
# The sequence table is a matrix with rows corresponding to
# (and named by) the samples, and columns corresponding to
# (and named by) the sequence variants.
```

Remove chimeras

The core dada method removes substitution and indel errors, but chimeras remain. Fortunately, the accuracy of the sequences after denoising makes identifying chimeras simpler than it is when dealing with fuzzy OTUs: all sequences which can be exactly reconstructed as a bimera (two-parent chimera) from more abundant sequences.

```
# Remove chimeric sequences:
seqtab.nochim <- removeBimeraDenovo(seqtab, method = "consensus",
    multithread = TRUE, verbose = TRUE)
```

```
## Identified 387 bimeras out of 957 input sequences.
```

```
dim(seqtab.nochim)
```

```
## [1]   6 570
```

```
# ratio of chimeric sequence reads
1 - sum(seqtab.nochim)/sum(seqtab)
```

```
## [1] 0.08023029
```

```
# write sequence variants count table to file
write.table(t(seqtab.nochim), paste0(output.dir, "/all_samples_SV-counts.tsv"),
    sep = "\t", eol = "\n", quote = F, col.names = NA)
# write OTU table to file
saveRDS(seqtab.nochim, paste0(output.dir, "/seqtab_final.rds"))
```

IMPORTANT: Most of your **reads** should remain after chimera removal (it is not uncommon for a majority of **sequence variants** to be removed though). If most of your reads were removed as chimeric, upstream processing may need to be revisited. In almost all cases this is caused by primer sequences with ambiguous nucleotides that were not removed prior to beginning the DADA2 pipeline.

Track reads through the pipeline

As a final check of the progress, look at the number of reads that made it through each step in the pipeline. This is a great place to do a last sanity check. Outside of filtering (depending on how stringent you want to be) there should no step in which a majority of reads are lost. If a majority of reads failed to merge, you may need to revisit the truncLen parameter used in the filtering step and make sure that the truncated reads span your amplicon. If a majority of reads failed to pass the chimera check, you may need to revisit the removal of primers, as the ambiguous nucleotides in unremoved primers interfere with chimera identification.

```
getN <- function(x) sum(getUniques(x))
track <- cbind(rd.counts, sapply(mergers, getN), rowSums(seqtab),
    rowSums(seqtab.nochim))
colnames(track) <- c("input", "filtered", "ratio", "merged",
    "tabled", "nonchim")
rownames(track) <- sample.names
# print table
track
```

```
##                  input filtered ratio merged tabled nonchim
## Orwoll_BI0023_BI 62724    43895  0.70  40920  40920   38155
## Orwoll_BI0056_BI 55342    39185  0.71  37368  37368   33217
## Orwoll_BI0131_BI 55144    38609  0.70  36920  36920   35661
## Orwoll_BI0153_BI 44610    33146  0.74  31151  31151   28775
## Orwoll_BI0215_BI 48227    33882  0.70  31540  31540   28866
## Orwoll_BI0353_BI 54271    43034  0.79  40435  40435   36143
```

```
# save to file
write.table(track, paste0(readQC.folder, "/Read_counts_at_each_step.tsv"),
    sep = "\t", quote = F, eol = "\n", col.names = NA)
```

Align sequences and reconstruct phylogeny

Multiple sequence alignment of resolved sequence variants is used to generate a phylogenetic tree, which is required for calculating UniFrac beta-diversity distances between microbiome samples.

```
# Get sequences
seqs <- getSequences(seqtab.nochim)
names(seqs) <- seqs  # This propagates to the tip labels of the tree
# Multiple seqeuence alignment
mult <- msa(seqs, method = "ClustalOmega", type = "dna", order = "input")
```

```
## using Gonnet
```

```
# Save msa to file; convert first to phangorn object
phang.align <- as.phyDat(mult, type = "DNA", names = getSequences(seqtab.nochim))
write.phyDat(phang.align, format = "fasta", file = paste0(output.dir,
    "/msa.fasta"))

# Call FastTree (via 'system') to reconstruct phylogeny
system(paste("FastTreeMP -gtr -nt ", output.dir, "/msa.fasta > ",
    output.dir, "/FastTree.tre", sep = ""))


detach("package:phangorn", unload = TRUE)
detach("package:msa", unload = TRUE)
```

Assign taxonomy

The assignTaxonomy function takes a set of sequences and a training set of taxonomically classified sequences, and outputs the taxonomic assignments with at least minBoot bootstrap confidence. Formatted training datasets for taxonomic assignments can be downloaded from here https://benjjneb.github.io/dada2/training.html.

`assignTaxonomy( ... )` implements the RDP naive Bayesian classifier method described in Wang et al. 2007. In short, the kmer profile of the sequences to be classified are compared against the kmer profiles of all sequences in a training set of sequences with assigned taxonomies. The reference sequence with the most similar profile is used to assign taxonomy to the query sequence, and then a bootstrapping approach is used to assess the confidence assignment at each taxonomic level. The return value of assignTaxonomy(...) is a character matrix, with each row corresponding to an input sequence, and each column corresponding to a taxonomic level.

```
# Assign taxonomy:
taxa.gg13_8 <- assignTaxonomy(seqtab.nochim, "/Users/jianshuzhao/Github/dada2/reference_dbs_16S/gg_13_8_
    multithread = TRUE, tryRC = TRUE)

# Print first 6 rows of taxonomic assignment
unname(head(taxa.gg13_8))
```

```
##      [,1]          [,2]               [,3]
## [1,] "k__Bacteria" "p__Bacteroidetes"  "c__Bacteroidia"
## [2,] "k__Bacteria" "p__Proteobacteria" "c__Gammaproteobacteria"
## [3,] "k__Bacteria" "p__Bacteroidetes"  "c__Bacteroidia"
## [4,] "k__Bacteria" "p__Firmicutes"     "c__Clostridia"
## [5,] "k__Bacteria" "p__Bacteroidetes"  "c__Bacteroidia"
## [6,] "k__Bacteria" "p__Bacteroidetes"  "c__Bacteroidia"
##      [,4]                  [,5]                    [,6]
## [1,] "o__Bacteroidales"    "f__Bacteroidaceae"     "g__Bacteroides"
```

```
## [2,] "o__Enterobacteriales" "f__Enterobacteriaceae" "g__Klebsiella"
## [3,] "o__Bacteroidales"     "f__Bacteroidaceae"     "g__Bacteroides"
## [4,] "o__Clostridiales"     "f__Ruminococcaceae"    "g__Faecalibacterium"
## [5,] "o__Bacteroidales"     "f__Bacteroidaceae"     "g__Bacteroides"
## [6,] "o__Bacteroidales"     "f__Bacteroidaceae"     "g__Bacteroides"
##      [,7]
## [1,] "s__"
## [2,] "s__"
## [3,] "s__caccae"
## [4,] "s__prausnitzii"
## [5,] "s__ovatus"
## [6,] "s__ovatus"
```

```
# Replace NAs in taxonomy assignment table with prefix
# corresponding to tax rank
taxa.gg13_8.2 <- replaceNA.in.assignedTaxonomy(taxa.gg13_8)

# Write taxa table to file
write.table(taxa.gg13_8.2, paste0(output.dir, "/all_samples_GG13-8-taxonomy.tsv"),
    sep = "\t", eol = "\n", quote = F, col.names = NA)
```

Merge OTU and GG13-8 taxonomy tables

```
otu.gg.tax.table <- merge(t(seqtab.nochim), taxa.gg13_8.2, by = "row.names")
rownames(otu.gg.tax.table) <- otu.gg.tax.table[, 1]
otu.gg.tax.table <- otu.gg.tax.table[, -1]

write.table(otu.gg.tax.table, paste0(output.dir, "/all_samples_SV-counts_and_GG13-8-taxonomy.tsv"),
    sep = "\t", eol = "\n", quote = F, col.names = NA)
```

For RDP and Silva, taxonomic assignment to species level is a two-step process. Fast and appropriate species-level assignment from 16S data is provided by the `assignSpecies( ... )` method. `assignSpecies( ... )` uses exact string matching against a reference database to assign Genus species binomials. In short, query sequence are compared against all reference sequences that had binomial genus-species nomenclature assigned, and the genus-species of all exact matches are recorded and returned if it is unambiguous.

The convenience function `addSpecies( ... )` takes as input a taxonomy table, and outputs a table with an added species column. Only those genus-species binomials which are consistent with the genus assigned in the provided taxonomy table are retained in the output. See here for more on taxonomic assignment https://benjjneb.github.io/dada2/assign.html.

Asign SILVA and RDP taxonomies and merge with OTU table

```
# Assign SILVA taxonomy
taxa.silva <- assignTaxonomy(seqtab.nochim, "/Users/jianshuzhao/Github/dada2/reference_dbs_16S/silva_nr
    multithread = TRUE)

# Replace NAs in taxonomy assignment table with prefix
# corresponding to tax rank
taxa.silva.2 <- replaceNA.in.assignedTaxonomy(taxa.silva)

# OMIT APPENDING SPECIES FOR SILVA DUE TO MEMORY CONSTRAINTS
# Append species. Note that appending the argument
# 'allowMultiple=3' will return up to 3 different matched
```

```
# species, but if 4 or more are matched it returns NA.
# taxa.silva.species <- addSpecies(taxa.silva,
# '/n/huttenhower_lab/data/dada2_reference_databases/silva_species_assignment_v128.fa.gz')

# Merge with OTU table and save to file
otu.silva.tax.table <- merge(t(seqtab.nochim), taxa.silva.2,
    by = "row.names")
rownames(otu.silva.tax.table) <- otu.silva.tax.table[, 1]
otu.silva.tax.table <- otu.silva.tax.table[, -1]

write.table(otu.silva.tax.table, paste0(output.dir, "/all_samples_SV-counts_and_SILVA-taxonomy.tsv"),
    sep = "\t", eol = "\n", quote = F, col.names = NA)

# Assign RDP taxonomy
taxa.rdp <- assignTaxonomy(seqtab.nochim, "/Users/jianshuzhao/Github/dada2/reference_dbs_16S/rdp_train_
    multithread = TRUE)
```

```
## Warning in .Call2("fasta_index", filexp_list, nrec, skip, seek.first.rec, :
## reading FASTA file /Users/jianshuzhao/Github/dada2/reference_dbs_16S/
## rdp_train_set_18.fa.gz: ignored 9 invalid one-letter sequence codes
```

```
# Replace NAs in taxonomy assignment table with prefix
# corresponding to tax rank
taxa.rdp.2 <- replaceNA.in.assignedTaxonomy(taxa.rdp)

# OMIT APPENDING SPECIES FOR RDP DUE TO MEMORY CONSTRAINTS
# Append species. Note that appending the argument
# 'allowMultiple=3' will return up to 3 different matched
# species, but if 4 or more are matched it returns NA.
# taxa.rdp.species <- addSpecies(taxa.rdp,
# '/n/huttenhower_lab/data/dada2_reference_databases/rdp_species_assignment_16.fa.gz')


# Merge with OTU table and save to file
otu.rdp.tax.table <- merge(t(seqtab.nochim), taxa.rdp.2, by = "row.names")
rownames(otu.rdp.tax.table) <- otu.rdp.tax.table[, 1]
otu.rdp.tax.table <- otu.rdp.tax.table[, -1]

write.table(otu.rdp.tax.table, paste0(output.dir, "/all_samples_SV-counts_and_RDP-taxonomy.tsv"),
    sep = "\t", eol = "\n", quote = F, col.names = NA)
```

Session Info:

```
sessionInfo()
```

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS  10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] ape_5.4-1           gridExtra_2.3       Biostrings_2.54.0
##  [4] XVector_0.26.0      IRanges_2.20.2      S4Vectors_0.24.4
##  [7] BiocGenerics_0.32.0 ggplot2_3.3.3       dada2_1.14.1
## [10] Rcpp_1.0.4.6        knitr_1.28
##
## loaded via a namespace (and not attached):
##  [1] lattice_0.20-38             png_0.1-7
##  [3] Rsamtools_2.2.3             assertthat_0.2.1
##  [5] digest_0.6.26               R6_2.4.1
##  [7] GenomeInfoDb_1.22.1         plyr_1.8.6
##  [9] ShortRead_1.44.3            evaluate_0.14
## [11] pillar_1.4.6                zlibbioc_1.32.0
## [13] rlang_0.4.8                 Matrix_1.3-2
## [15] rmarkdown_2.7               labeling_0.4.2
## [17] BiocParallel_1.20.1         stringr_1.4.0
## [19] igraph_1.2.5                RCurl_1.98-1.2
## [21] munsell_0.5.0               DelayedArray_0.12.3
## [23] compiler_3.6.3              xfun_0.22
## [25] pkgconfig_2.0.3             htmltools_0.5.1.1
## [27] tidyselect_1.1.0            SummarizedExperiment_1.16.1
## [29] tibble_3.0.4                GenomeInfoDbData_1.2.2
## [31] codetools_0.2-16            quadprog_1.5-8
## [33] matrixStats_0.56.0          crayon_1.3.4
## [35] dplyr_0.8.5                 withr_2.3.0
## [37] GenomicAlignments_1.22.1    bitops_1.0-6
## [39] grid_3.6.3                  nlme_3.1-144
## [41] gtable_0.3.0                lifecycle_0.2.0
## [43] formatR_1.7                 magrittr_1.5
## [45] scales_1.1.1                RcppParallel_5.0.3
## [47] stringi_1.4.6               farver_2.0.3
## [49] hwriter_1.3.2               reshape2_1.4.4
## [51] latticeExtra_0.6-29         ellipsis_0.3.1
## [53] vctrs_0.3.4                 fastmatch_1.1-0
## [55] RColorBrewer_1.1-2          tools_3.6.3
## [57] Biobase_2.46.0              glue_1.4.2
```

```
## [59] purrr_0.3.4                jpeg_0.1-8.1
## [61] yaml_2.2.1                 colorspace_1.4-1
## [63] GenomicRanges_1.38.0
```