How-to guide for GT-seq data analysis (pt. 1)

Rachel Voyt

27 February, 2023

Overview

This document represents "Part 1" of the full GT-seq data analysis pipeline, and is meant to be used as a template for the preparation, processing, and genotyping of GT-seq data from Illumina and/or Nanopore sequencing platforms. The "Part 2" template for data analysis is available as a separate file, and includes assessments of locus and sample performance - this step is most useful for optimization stages and troubleshooting.

Many of the steps included in the present pipeline are taken from the "GTscore" pipeline created by Garrett McKinney (see the associated github here). I've modified some of the original perl scripts and rearranged the sections to better fit the current project, and have taken care to indicate when text/scripts were taken directly from the GTscore pipeline.

1 Preparation

1.1 Install programs

R packages

```
library(tidyverse)
library(phylotools)
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("Biostrings")
source("GTscore_modified.R") # NOTE- added an option for "\\[ATGC\\]"="N" (lines 616 and 617) to replac
```

Fastq & MultiQC

Fastq and MultiQC are both needed for sequence quality checks. Use the scripts below to create new environments and install each program. Note that for MultiQC to be up to date, I needed to install it in an environment with Python 3.9 using conda-forge.

Fastq

```
# Create new conda environment
conda create -n fastqc-env

# Activate via:
conda activate fastqc-env

# Install fastqc
conda install -c bioconda fastqc
```

MultiQC

```
# Create new conda environment
conda create -n multiqc-env python=3.9

# Activate via:
conda activate multiqc-env

# Install multiqc
conda install -c bioconda multiqc
```

Perl and modules

The GTscore pipeline includes several Perl scripts as well as Perl modules "Algorithm::Combinatorics" and "Excel-Writer-XLSX". Use the scripts below to install on Linux/Mac computers (more on how to install Perl modules here).

A couple notes–

- It used to be possible to install Perl through Conda (conda install -c conda-forge perl), but this doesn't seem to work well anymore
- Be sure to install both Perl and the modules in the same place - e.g., both in the root directory
- You can also just use "sudo cpan Algorithm::Combinatorics" instead of using the cpanimus command "cpanm" - I had to switch to using the regular "cpan" command because cpanm started putting my modules in odd places

```
# Install perl
curl -L http://xrl.us/installperlnix | bash

# Install cpanminus (makes installing perl modules easier)
cpan App::cpanminus

# Install necessary perl modules
sudo cpanm Algorithm::Combinatorics
## OR
sudo cpan Algorithm::Combinatorics
```

1.2 Prepare sequencing results

Below are steps to prepare sequencing results from both Illumina and Nanopore platforms, including downloading, extracting, and interleaving sequences (both platforms) as well as trimming adapter sequences (Nanopore only).

1.2.1 ILLUMINA

Download sequences from BaseSpace To download all sample files, it's easiest to use the BaseSpace Command Line Interface (CLI).

To install the BaseSpace CLI, follow the scripts provided here: https://developer.basespace.illumina.com/docs/content/documentation/cli/cli-overview

Note that even if it's already installed, you may need to re-authenticate via the command below. Running the command below will give a link; click to open in a browser and complete authentication.

```
bs auth
```

Once authentication is complete, you can download the desired files as follows:

```
# Check *Project IDs* (make note of which you want to download)
bs list projects

# Download *Sample Sequences* for the project of interest
bs download project -i PROJECTID# -o /DESIRED/LOCATION/HERE --extension=fastq.gz

# Download *Unindexed Reads* for the project of interest
bs download project -i PROJECTID# -o /DESIRED/LOCATION/HERE --extension=fastq.gz
```

```
# Navigate to the directory containing the .fastq.gz files from the sequencing run
cd ORIGINAL_SEQS

# Move all .gz files out of their individual folders and into the current directory; remove empty folde
mv */* ./

# Extract all files
gunzip -r .
```

Extract fast.gz files

Interleave files To prep for the GTscore pipeline, we need to combine reads 1 & 2 for each sample into a single interleaved file - leave a set of non-interleaved files in the original folder though to keep for fastq quality checks.

```
# Move to interleaved seq directory
cd INTERLEAVED_SEQS

# Interleave reads 1 & 2 for each sample and place in a new folder
for i in ./ORIGINAL_SEQS/*_R1_001.fastq; do name=$(basename $i _R1_001.fastq); reformat.sh in=00_run3Se

# Copy files to GTscore folder
cp * ../GTSCORE_DIRECTORY/

# Trim filenames to make them more manageable
for f in *.fastq; do   g="${f%%_*}.fastq"; mv "${f}" "${g}"; done
```

1.2.2 NANOPORE

Download sequences *tbd*

```
# Navigate to the directory containing the sequence zip file
cd ALLSEQS_ZIP

# Extract zip file into a new directory - keep the original zip file where it is as a copy
unzip FILE_NAME.zip -d ../NEW_DIRECTORY

# Navigate to the directory containing the .fastq.gz files from the sequencing run
cd DIRECTORY

# Move all .gz files out of their individual folders and into the current directory & remove empty fold
mv */* ./
```

```
# Extract all files
gunzip -r .
```

Extract sequences

Trim adapter sequences For the GTscore pipeline to work, we need to trim off all adapter sequences. Guppy can be set to automatically trim the Nanopore adapters, but our sequences also contain the Illumina overhang sequence - we can use 'cutadapt' in bash to find and remove them, including partial sequences. Step to do this are below:

First create, then navigate to new directory for trimmed sequences

```
mkdir SEQS_TRIMMED
cd SEQS_TRIMMED
```

Then run a loop to do the following: 1) clean up the file names so they're just "barcodeXX.fastq" 2) remove everything before the forward-primer overhang and after the reverse-primer overhang in all of our original sequences 3) create a new set of trimmed sample files

```
for i in ../ORIGINAL_SEQS/*.fastq; do f=${i%%_892ac4c1_03630e12*}; g=${f##*_}; cutadapt -a TCGTCGGCAGC

# Can view what it does by running the following:
for i in ../ORIGINAL_SEQS/*.fastq; do f=${i%%_892ac4c1_03630e12*}; g=${f##*_}; echo cutadapt -a TCGTCG
```

Finally, copy the trimmed files to the GTscore folder (they need to be here for the GTscore scripts to run properly)

```
cp bar* ../GTSCORE_DIRECTORY/
```

1.3 Create GTscore input files

The GTscore pipeline requires two types of input files: 1) a file containing all sample file names, and 2) a file containing primer-probe information. Because our dataset includes two different species with some species-specific loci, I also found it helpful to create separate files for each so that we have a more accurate read count and genotype rates for each sample and locus.

Note that if your dataset includes multiple species, you will need a sample metadata file to create the GTscore sample files.

1.3.1 Sample files

Sample file prep is split into Illumina and Nanopore workflows since Nanopore output has previously included a few extra things that needed to be filtered out.

NOTE - regardless of sequencing platform, I recommend adjusting sample names so that they do NOT include characters such as periods, underscores, dashes, etc. The GTscore perl scripts tend to switch these around, which then requires formatting adjustments within the R script (nothing hard, but a bit of a pain). I haven't taken the time to dig into the perl scripts themselves to adjust them, so at this point it's easiest to name samples (or change sample names) so that they don't include anything aside from text and numbers.

3.1.1 ILLUMINA

All samples Create file for all samples in bash

```
for i in *fastq; do echo $i; done > sampleFiles_fullSet.txt
```

Load into R

```
sf_fullSet <- read.table("sampleFiles_fullSet.txt")
```

Species subsets First we need to load in the metadata for all of the samples and make a few formatting adjustments

```
md <- read.csv("METADATA_FILE.csv") %>%
  mutate(sampleID = gsub("_","-", sampleID)) %>%
  mutate(sampleFile = paste0(sampleID, ".fastq"))
```

Then we can create our sample file species subsets. Note that we're including the negative controls in both species subsets – e.g., "negative control #1" would be listed in the LWED sample file as well as the SIMP sample file.

```
# Create subsets
lwed <- filter(md, is.na(species) | !species == "SIMP")
simp <- filter(md, is.na(species) | !species == "LWED")

sf_lwed <- sf_fullSet %>%
  filter(V1 %in% lwed$sampleFile)

sf_simp <- sf_fullSet %>%
  filter(V1 %in% simp$sampleFile)

# Export sample files
write.table(sf_lwed, file = "sampleFiles_LWED.txt", sep = "\t", row.names = F, col.names = F, quote = F)
write.table(sf_simp, file = "sampleFiles_SIMP.txt", sep = "\t", row.names = F, col.names = F, quote = F)
```

Export sample files Follow the file export scripts below to ensure that sample files are in the correct format for the GTscore pipeline.

```
write.table(sf_fullSet, file = "sampleFiles_fullSet.txt", sep = "\t", row.names = F, col.names = F, quo
write.table(sf_LWED, file = "sampleFiles_LWED.txt", sep = "\t", row.names = F, col.names = F, quote = F)
write.table(sf_SIMP, file = "sampleFiles_SIMP.txt", sep = "\t", row.names = F, col.names = F, quote = F)
```

3.1.2 NANOPORE Nanopore data may require an extra step when creating sample files, since previous MinION output sequences included barcodes that weren't supposed to be there. As such, I opted to create an original set of sample files (including all MinION output), then subset that to include only the barcodes that were used for the samples, and then create separate files by species.

All barcodes Easiest to do this part in bash–

```
for i in bar*; do echo $i; done > sampleFiles_original.txt
```

Barcodes of interest First step is to clean up the metadata file - note that the script below also includes a step to remove duplicates, as some sample entries in the metadata file from a previous MinION run were listed multiple times (might not be necessary for other runs).

```
md <- read.csv("METADATA_FILE.csv") %>%
  filter(!is.na(Barcode)) %>%
  mutate(Barcode = str_replace(Barcode, "BC", "barcode")) %>%
  mutate(sampleID = paste(Barcode, "trimmed.fastq", sep = "_")) %>%
  mutate(Field.id.a = str_replace(Field.id.a, "Leontocebus weddeli", "LWED")) %>%
  mutate(Field.id.a = str_replace(Field.id.a, "Saguinus imperator", "SIMP")) %>%
  dplyr::rename(species = Field.id.a) %>%
  distinct()
```

Then we can subset the original sample files to only the barcodes of interest.

```
# Read in original list of sample files
sf_original <- read.table("sampleFiles_original.txt")

# Subset to barcodes of interst only
sf_fullSet <- sf_original %>%
  filter(V1 %in% md$sampleID)
```

Species subsets Note that when making the species-specific sample files, I chose to include the negative controls in both – e.g., "negative control #1" would be listed in the LWED sample file as well as the SIMP sample file.

```
# Create subsets
lwed <- filter(md, is.na(species) | !species == "SIMP")
simp <- filter(md, is.na(species) | !species == "LWED")

sf_LWED <- sf_fullSet %>%
  filter(V1 %in% lwed$sampleID)

sf_SIMP <- sf_fullSet %>%
  filter(V1 %in% simp$sampleID)
```

Export sample files Follow the file export scripts below to ensure that sample files are in the correct format for the GTscore pipeline.

```
write.table(sf_fullSet, file = "sampleFiles_fullSet.txt", sep = "\t", row.names = F, col.names = F, quo
write.table(sf_LWED, file = "sampleFiles_LWED.txt", sep = "\t", row.names = F, col.names = F, quote = F
write.table(sf_SIMP, file = "sampleFiles_SIMP.txt", sep = "\t", row.names = F, col.names = F, quote = F
```

1.3.2 Primer-probe files

Here I'm loading in primer-probe files that I created for previous runs. I've divided the primer-probe files into three sets, one with all samples (contains all loci), one for LWED samples only (excludes SIMP-specific loci) and one for SIMP samples only (excludes LWED-specific loci).

Each primer-probe file contains the following:

1. Locus - locus name
2. Ploidy - in our case, all are diploid so ploidy = 2
3. SNPpos - position of SNP in the amplicon, assuming that the first base = 0
4. Allele1 - one of the options for the SNP
5. Allele2 - the other option for the SNP
6. Probe1 - an 8 nt sequence overlapping the SNP; contains Allele1
7. Probe2 - same as Probe1, but contains Allele2
8. Primer - the forward primer sequence for each locus

```
pp_fullSet <- read.table("primerProbeFile_fullSet.txt", header = T)
pp_lwed <- read.table("primerProbeFile_LWED.txt", header = T)
pp_simp <- read.table("primerProbeFile_SIMP.txt", header = T)
```

2 Quality checks

This section isn't necessary for amplicon counting, genotyping, etc., but it's a helpful check, particularly in the optimization stages.

To get sequence quality scores for each sample, we can use 'fastqc' paired with 'MultiQC' – 'fastp' is better for paired-end reads, but doesn't play nice when it comes to getting quality scores from the MultiQC report. Note that we'll be running quality checks on reads 1 and 2 separately (vs. interleaved).

Once we have the output from the programs above, we can convert it into something more usable in R using the package 'TidyMultiqc'.

2.1 Fastq + MultiQC

Run the scripts below to create the MultiQC report.

Note - If there are a lot of samples in the run, use "--interactive" to force plots to tell you which line belongs to which sample in the MultiQC plots, otherwise it will remove this option.

```
# Activate fastqc environment
cd ../QUALITY_CHECKS
conda activate fastqc-env

# Run fastqc
for i in ../ORIGINAL_SEQS/*; do fastqc $i; done

# Move fastqc files to quality checks folder
mv ../ORIGINAL_SEQS/*fastqc* .

# Activate multiqc environment
conda activate multiqc-env

# Run MultiQC on all files within the quality checks folder
multiqc --interactive .
```

2.2 TidyMultiqc

The package 'TidyMultiqc' converts the 'multiqc_data.json' file into tidy data frames. For the scripts in this section, I followed this vignette provided by the makers of the TidyMultiqc package.

Load FastQC MultiQC report to R

```
library(multiqc)

MultiQCfastQCpath <- file.path("/home/rachelvoyt/Documents/UT-Grad/Development/repos/tamGenetics_primat

MultiQCfastQC <- TidyMultiqc::load_multiqc(MultiQCfastQCpath, sections = c("general", "raw"))
```

Obtain median sequence quality scores

MultiQC reports do not provide a numerical summary statistic for read quality; they only have mapping quality and pass/fails for the per-base sequence quality. We instead need to pull this data from one of the plots – I'm using the "Per Sequence Quality Scores" plot here:

```
df <- TidyMultiqc::load_multiqc(
  MultiQCfastQCpath,
```

```
  sections = 'plot',
  plots = "fastqc_per_sequence_quality_scores_plot")
```

This provides a nested data frame as a set of x, y pairs. As it's a histogram plot, we know that the `x` value is the quality score, and `y` is the number of times that score has been counted.

We can use tidyr to unnest the data, HistDat to create a HistDat object for each group, then purr to map each plot data frame into a row of summary statistics:

```
df_unNest <- df %>%
  dplyr::mutate(
    purrr::map_dfr(plot.fastqc_per_sequence_quality_scores_plot, function(plot_df){
      hist = HistDat::HistDat(vals=plot_df$x, counts = plot_df$y)
      list(
        mean_qc = mean(hist),
        median_qc = median(hist),
        max_qc = max(hist)
      )
    }),
    plot.fastqc_per_sequence_quality_scores_plot = NULL
  )
```

This dataframe has separate rows for reads 1 & 2; we can rearrange that to make it easier to work with for our purposes. Since we're most interested in median quality, we'll just pull that value for each sample.

```
seqQC_r1 <- df_unNest %>%
  select(c("metadata.sample_id", "median_qc")) %>%
  filter(grepl("R1", metadata.sample_id)) %>%
  rename(medianQC_r1 = median_qc) %>%
  mutate(sampleNo = substr(metadata.sample_id, start = 9, stop = 11))
seqQC_r2 <- df_unNest %>%
  select(c("metadata.sample_id", "median_qc")) %>%
  filter(grepl("R2", metadata.sample_id)) %>%
  rename(medianQC_r2 = median_qc) %>%
  mutate(sampleNo = substr(metadata.sample_id, start = 9, stop = 11))
seqQC <- merge(seqQC_r1, seqQC_r2, by = "sampleNo") %>%
  select(c("sampleNo", "medianQC_r1", "medianQC_r2")) %>%
  arrange(sampleNo)
```

Let's just take a peek at the distribution of sequence quality scores for now; we'll use it in our other analyses later on:

```
ggplot(seqQC, aes(medianQC_r1)) +
  geom_bar(stat = "count") +
  theme_bw()

ggplot(seqQC, aes(medianQC_r2)) +
  geom_bar(stat = "count") +
  theme_bw()
```

3 Count reads for amplicons

This section is the first step of Garrett McKinney's GTscore pipeline). Most of the text below is pulled from the original GTscore files, with some modifications for clarity.

3.1 Info on AmpliconReadCounter.pl

The read counter is written in perl (AmpliconReadCounter.pl), but can be called from R. Running it does the following:

1. Identifies each unique sequence, then counts the number of times each unique sequence occurs within an individual
2. Aligns each unique sequence with primer and probe; if the sequence doesn't align, then it is excluded as an off-target sequence and reports by individual and by locus are given.

- Note: By default, all primers are trimmed to the length of the shortest primer to increase speed. Optionally the full length primer can be used for the primer but this may significantly increase run timing depending on variation in primer lengths across loci.

Input flags for this script are:

- –p a tab delimited file containing primer/probe information for each locus
- –files a text file containing a list of .fastq sequence files to count reads from

Optional flags:

- –prefix optional prefix for output file names
- –inDir option to specify directory containing sequence data
- –inputType fq or fastqgz (defaults to fastqgz)
- –useFullPrimer uses the full primer for counting reads rather than the trimmed primer
- –alleleOrder order of alleles output in locusTable file. Options are original (matches primer-probe file order) or alphabetical (default)
- –printMatched outputs matched reads for each individual
- –printDiscarded outputs discarded reads for each individual

3.2 Run AmpliconReadCounter.pl

To run from the command line, use: perl AmpliconReadCounter.pl -p primerProbeFile.txt –files sampleList.txt

To run from R, use:

```
# All samples, all loci
system2("perl",
        args="AmpliconReadCounter.pl -p primerProbeFile_fullSet.txt --files sampleFiles_fullSet.txt --p

# LWED
system2("perl",
        args="AmpliconReadCounter.pl -p primerProbeFile_LWED.txt --files sampleFiles_LWED.txt --prefix

# SIMP
system2("perl",
        args="AmpliconReadCounter.pl -p primerProbeFile_SIMP.txt --files sampleFiles_SIMP.txt --prefix
```

AmpliconReadCounter.pl outputs a LocusTable file and an AlleleReads file for single-SNP and haplotype data, plus two summary files. We'll only need the single-SNP files for our purposes.

The default names for these files are as follows, with LWED and SIMP appended to files associated with species-specific analyses:

- LocusTable_singleSNPs.txt - locus name, ploidy, and alleles for each SNP
- AlleleReads_singleSNPs.txt - counts for each SNP allele (rows are loci, columns are individuals)
- LocusTable_haplotypes.txt - same as above (doesn't matter for us since we're only using single SNPs)
- AlleleReads_haplotypes.txt - same as above (doesn't matter for us since we're only using single SNPs)
- GTscore_individualSummary.txt - counts by individual of total reads, off-target reads, primer-only reads, and primer probe reads

- GTscore_locusSummary.txt - counts by locus of primer reads and primer probe reads

4 Genotyping

This section is the second step of Garrett McKinney's GTscore pipeline), though I've modified it to include a minimum of 10x coverage for genotyping to occur.

4.1 Identify loci <10x coverage

Genotyping by sequencing generally requires a minimum of 10x coverage to be considered reliable; however, the GTscore genotyping script doesn't include this cutoff, and will provide genotypes with as little as one read per allele for heterozygous calls and 3 reads (vs. 0) for a homozygous call. As such, I'm implementing a filtering step here to recode all loci with <10 reads to "0" so that loci with lower than 10x coverage will not receive genotypes.

Step 1: Recode loci with <10x coverage in the full set of allele read counts

```
# Read in allele counts for the full dataset
readCounts <- read.table("fullSet_AlleleReads_singleSNPs.txt")

# Set up a function to sum the read counts per allele for each locus, using package gsubfn
repl <- function(x) gsubfn("(\\d+),(\\d+)", ~ as.numeric(x) + as.numeric(y), paste(x))

# Then apply the function to readCounts to sum each set of allele reads for each locus
readCounts_sum <- replace(readCounts, TRUE, lapply(readCounts, repl)) %>%
  mutate(across(everything(),as.numeric))

# Recode <10x loci with "0"
readCounts[readCounts_sum < 10] <- "0,0"
```

Step 2: Create lwed & simp subsets & create new AlleleReads_singleSNPs files (note that the reformat step is only necessary if those characters are included in your sample names)

```
# Reformat lists with . vs. -
lwed_point <- lwed %>%
  mutate(sampleID = gsub("-", "\\.", sampleID)) %>%
  select(sampleID) %>%
  as.character()
simp_point <- simp %>%
  mutate(sampleID = gsub("-", "\\.", sampleID)) %>%
  select(sampleID) %>%
  as.character()

# Get lists of lwed- and simp-specific loci
lwed_loci <- read.delim("LWED_LocusTable_singleSNPs.txt",header=TRUE,stringsAsFactors=FALSE)
simp_loci <- read.delim("SIMP_LocusTable_singleSNPs.txt",header=TRUE,stringsAsFactors=FALSE)

readCounts_lwed <- readCounts %>%
  select_(.dots = lwed_point) %>%
  rownames_to_column("Locus") %>%
  filter(Locus %in% lwed_loci$Locus_ID) %>%
  column_to_rownames("Locus")
readCounts_simp <- readCounts %>%
  select_(.dots = simp_point) %>%
  rownames_to_column("Locus") %>%
```

```
  filter(Locus %in% simp_loci$Locus_ID) %>%
  column_to_rownames("Locus")

# Export new AlleleReads_singleSNPs files
write.table(readCounts,"fullSet_AlleleReads_singleSNPs_10x.txt",quote=FALSE,sep="\t")
write.table(readCounts_lwed,"LWED_AlleleReads_singleSNPs_10x.txt",quote=FALSE,sep="\t")
write.table(readCounts_simp,"SIMP_AlleleReads_singleSNPs_10x.txt",quote=FALSE,sep="\t")
```

4.2 Genotyping

Genotyping is accomplished using the polyGen function. The genotyping algorithm is described in McKinney et al. (2018) and is a maximum likelihood algorithm capable of genotyping any number of alleles and ploidy per locus. This allows genoyping of single SNPs as well as microhaplotypes, and loci with elevated ploidy.

Two arguments are required for polyGen, the locusTable and alleleReads files output by AmpliconReadCounter.

Optional arguments for polyGen are: - p_thresh - threshold p-value for likelihood ratio test (default 0.05) - epsilon - error rate for genotyping model (default 0.01)

NOTE that only primer probe reads are used in genotyping!

Note also that I'm making three genotype files, one for all loci vs. all samples and two species-specific sets - this allows our genotyping metrics to be more accurate.

Full set

```
#load locus table and 10x allele reads file
fullSet_singleSNP_locusTable<-read.delim("fullSet_LocusTable_singleSNPs.txt",header=TRUE,stringsAsFacto
fullSet_singleSNP_alleleReads<-read.delim("fullSet_AlleleReads_singleSNPs_10x.txt",header=TRUE,row.name

#generate singleSNP genotypes using the polyGen algorithm
fullSet_polyGenResults_singleSNP<-polyGen(fullSet_singleSNP_locusTable,fullSet_singleSNP_alleleReads)

#write results
write.table(fullSet_polyGenResults_singleSNP,"fullSet_polyGenResults_singleSNP.txt",quote=FALSE,sep="\t

#rubias format
sampleMetaData <- data.frame(sample_type="NA",repunit=NA,collection="NA",indiv=colnames(fullSet_polyGenl
exportRubias(fullSet_polyGenResults_singleSNP,fullSet_singleSNP_locusTable,sampleMetaData,filename="full
```

LWED

```
#load locus table and 10x allele reads file
LWED_singleSNP_locusTable<-read.delim("LWED_LocusTable_singleSNPs.txt",header=TRUE,stringsAsFactors=FALS
LWED_singleSNP_alleleReads<-read.delim("LWED_AlleleReads_singleSNPs_10x.txt",header=TRUE,row.names=1,str

#generate singleSNP genotypes using the polyGen algorithm
LWED_polyGenResults_singleSNP<-polyGen(LWED_singleSNP_locusTable,LWED_singleSNP_alleleReads)

#write results
write.table(LWED_polyGenResults_singleSNP,"LWED_polyGenResults_singleSNP.txt",quote=FALSE,sep="\t")
```

SIMP

```
#load locus table and 10x allele reads file
SIMP_singleSNP_locusTable<-read.delim("SIMP_LocusTable_singleSNPs.txt",header=TRUE,stringsAsFactors=FALS
SIMP_singleSNP_alleleReads<-read.delim("SIMP_AlleleReads_singleSNPs_10x.txt",header=TRUE,row.names=1,st

#generate singleSNP genotypes using the polyGen algorithm
SIMP_polyGenResults_singleSNP<-polyGen(SIMP_singleSNP_locusTable,SIMP_singleSNP_alleleReads)

#write results
write.table(SIMP_polyGenResults_singleSNP,"SIMP_polyGenResults_singleSNP.txt",quote=FALSE,sep="\t")
```

5 Data summaries

The locus and sample summaries are one of the most helpful outputs of the GTscore pipeline, particularly during optimization stages. To make them more useful for our particular project, I've added to this section by creating "complete" summaries that have species-specific metrics and include summary info from other GTscore sections.

4.1 Locus summaries

Summarize single SNP results for loci

The summarizeGTscore command generates summary data for each locus in table form. The summary data includes:

- genotype rate
- average read depth
- minor (least frequent) allele frequency
- major (most frequent) allele frequency
- alleles per locus
- frequency per allele

Minor allele frequency is a common metric for filtering loci that are likely to be uninformative for population genetics; however, loci with haplotype alleles may have an allele with very low frequency but still have appreciable frequency at multiple other alleles. Because of this, the major allele frequency is included in output, as well as the observed frequencies for all alleles at a given locus.

```
#summarize single SNP results
fullSet_singleSNP_summary <- summarizeGTscore(fullSet_singleSNP_alleleReads, fullSet_singleSNP_locusTabl

LWED_singleSNP_summary<-summarizeGTscore(LWED_singleSNP_alleleReads, LWED_singleSNP_locusTable, LWED_pol

SIMP_singleSNP_summary<-summarizeGTscore(SIMP_singleSNP_alleleReads, SIMP_singleSNP_locusTable, SIMP_pol

#write results
write.table(fullSet_singleSNP_summary,"fullSet_singleSNP_summary.txt",quote=FALSE,sep="\t",row.names=FA
write.table(LWED_singleSNP_summary,"LWED_singleSNP_summary.txt",quote=FALSE,sep="\t",row.names=FALSE)
write.table(SIMP_singleSNP_summary,"SIMP_singleSNP_summary.txt",quote=FALSE,sep="\t",row.names=FALSE)
```

"Complete" locus summary

To make the summary files a bit more helpful for our purposes, here I'm creating a "complete" locus summary that 1) combines the single SNP summaries created in the section above with the GTscore locus summary output by AmpliconReadCounter.pl, and 2) provides performance metrics specific to that locus's species.

In doing so, this means that...

- values for LWED-specific loci reflect performance with LWED individuals only
- values for SIMP-specific loci reflect performance with SIMP individuals only
- all other loci reflect performance with LWED and SIMP individuals combined

```
# Load locus summaries
ls <- read.table("fullSet_GTscore_locusSummary.txt", header = T, sep = "\t")
lwed_ls <- read.table("LWED_GTscore_locusSummary.txt", header = T, sep = "\t")
simp_ls <- read.table("SIMP_GTscore_locusSummary.txt", header = T, sep = "\t")

# Separate locus summaries and singleSNP summaries into results from shared and species-specific loci a
## Locus summaries
ls1 <- ls %>%
  filter(!str_detect(Locus, "LWED|SIMP"))
lwed1_ls <- lwed_ls %>%
  filter(str_detect(Locus, "LWED"))
simp1_ls <- simp_ls %>%
  filter(str_detect(Locus, "SIMP"))

## SingleSNP summaries
ss <- fullSet_singleSNP_summary %>%
  filter(!str_detect(Locus_ID, "LWED|SIMP"))
lwed_ss <- LWED_singleSNP_summary %>%
  filter(str_detect(Locus_ID, "LWED"))
simp_ss <- SIMP_singleSNP_summary %>%
  filter(str_detect(Locus_ID, "SIMP"))

# Recombine locus summaries & single snp summary files, then merge the two
ls_recombine <- rbind(ls1, lwed1_ls, simp1_ls)

ss_recombine <- rbind(ss, lwed_ss, simp_ss)
ss_recombine$Locus_ID <- sub("^([^_]*_[^_]*).*", "\\1", ss_recombine$Locus_ID)

ls_ss <- merge(ls_recombine, ss_recombine, by.x = "Locus", by.y = "Locus_ID")

# Export
write.csv(ls_ss, "./summaryFiles/complete_locusSummary.csv", row.names = F)
```

4.2 Sample summaries

Summarize single SNP summaries for samples

The summarizeSamples command generates summary data for each sample in table form, and includes:

- genotype rate
- heterozygosity
- contamination score

Note that the metrics in the sample summaries that I'm creating below are species-specific; i.e., LWED sample performance does not include performance with SIMP-specific loci and vice versa.

Note also that the summarizeSamples function changes the "-" to a "." – be sure to change this back (if needed).

```
# Load species-specific sample summaries from AmpliconReadCounter
LWED_GTscore_individualSummary <-read.delim("LWED_GTscore_individualSummary.txt",header=TRUE,stringsAsF
```

```
SIMP_GTscore_individualSummary <-read.delim("SIMP_GTscore_individualSummary.txt",header=TRUE,stringsAsFa

# Create single SNP summaries
LWED_singleSNP_sampleSummary <- summarizeSamples(LWED_polyGenResults_singleSNP, LWED_singleSNP_alleleRea
  mutate(sample = gsub("\\.", "-", sample))
SIMP_singleSNP_sampleSummary <- summarizeSamples(SIMP_polyGenResults_singleSNP, SIMP_singleSNP_alleleRea
  mutate(sample = gsub("\\.", "-", sample))
```

"Complete" sample summary

The "complete" sample summary combines the single SNP summaries created above with 1) the GTscore individual summary output by AmpliconReadCounter.pl and 2) sample metadata.

Step 1: Combine single-SNP summary with GTscore individual summary

```
# make new copies & add column to note which loci-set the values are based on
LWED_gtIndivSummary <- LWED_GTscore_individualSummary %>%
  mutate(lociSet = "LWED") %>%
  merge(., LWED_singleSNP_sampleSummary, by.x="Sample", by.y = "sample")
SIMP_gtIndivSummary <- SIMP_GTscore_individualSummary %>%
  mutate(lociSet = "SIMP") %>%
  merge(., SIMP_singleSNP_sampleSummary, by.x = "Sample", by.y = "sample")

# Combine LWED & SIMP, adjust sample name
lwedSIMP_gtIndivSummary <- rbind(LWED_gtIndivSummary, SIMP_gtIndivSummary) %>%
  mutate(sampleFile = paste0(Sample, ".fastq"))
```

Step 2: Add sample metadata

```
# Merge metadata & fullSet sample summaries
complete_sampleSummary <- merge(lwedSIMP_gtIndivSummary, md, by = "sampleFile")

# Export
write.csv(complete_sampleSummary, "./summaryFiles/complete_sampleSummary.csv", row.names = F)
```