Processing fungal nanopore sequences

Code will set quarto rendering related options

```r
# Allows to use different font sizes inside code chunks
def.chunk.hook  <- knitr::knit_hooks$get("chunk")
knitr::knit_hooks$set(chunk = function(x, options) {
  x <- def.chunk.hook(x, options)
  ifelse(options$size != "normalsize",
         paste0("\n \\", options$size,"\n\n", x, "\n\n \\normalsize"), x)})
# Some global options
knitr::opts_chunk$set(warning = F, message = F)
knitr::opts_chunk$set(fig.align = 'center')
knitr::opts_chunk$set(warning = F)
silent_library <- function(pkg) {
  suppressPackageStartupMessages(library(pkg, character.only = T))
}
```

R base and Bioconductor related libraries

```r
# Load libraries
library(tidyverse)
library(kableExtra)
library(ggthemes)
library(dada2)
library(mia)
library(ggsci)
library(ShortRead)
library(patchwork)
```

Preprocessing

The Nanopore basecaller lacks support for demultiplexing dual indexes located on both the 5' and 3' ends of reads. Additionally, ligated libraries may contain reads in either orientation. To address these, we use `Cutadapt` for demultiplexing and `SeqKit` for reverse-complementing reads in the reverse orientation. The sequences are then merged, and PCR primers are trimmed.

A Python wrapper script, BCO_demux_tool has been created to automate these steps. Script outputs demultiplexed fastq.gz read files.

Read quality

Nanopore sequencing quality scores differ from other sequencing technologies because it is impossible to determine accurate per-base Phred quality values directly from the electrical signal. Instead, these scores are estimated based on the confidence of the basecalling model, rather than directly reflecting the probability of a sequencing error. This should be considered when analyzing Nanopore data, as traditional quality filtering approaches may not always be appropriate.

The code below reads already trimmed sample reads. A dataframe consisting of average Phred Q-values is subsequently visualised by violin plot.

```r
#Define file path
source_dir <- "raw/"
# A function to read and calculate phred Q-values
extract_qscores <- function(file) {
    # Read sequence file
    fq <- readFastq(file)
    # Create matrix from quality values
    qmat <- as(quality(fq), "matrix")
    # Calculate avg q-score per read
    avg_qscores <- apply(qmat, 1, mean, na.rm = T)
    # Trim file names
    sampleid <- sub("_trimmed\\.fastq\\.gz$", "", basename(file))
    # Create a dataframe
    qdata <- tibble(
        average_qscore = avg_qscores,
        sampleid = sampleid)
    return(qdata)
}

# Create list of suitable files from source directory
files <- list.files(source_dir, pattern = "\\.fastq\\.gz$", full.names = T)
# Combined data frame by using map_dfr and applying function
qscore_data <- map_dfr(files, extract_qscores)

# Save results to rds files
saveRDS(qscore_data, "qiime2/average_phred.rds")
```

Plot results

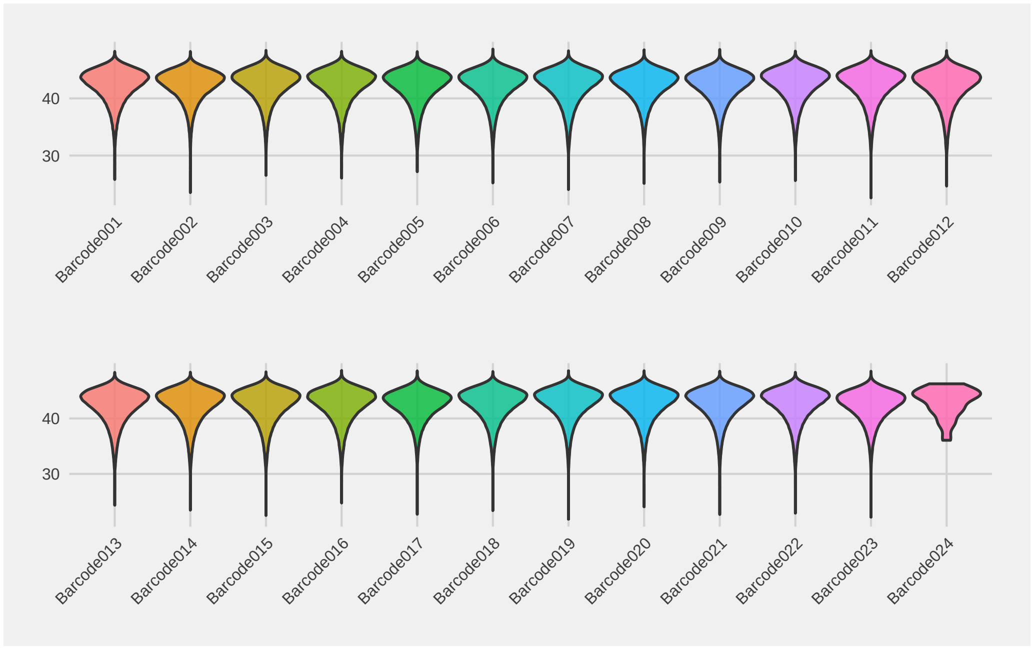
```r
qdata <- readRDS("qiime2/average_phred.rds")

# Determine the number of sample batches (each batch has max 12 samples)
sample_list <- unique(qdata$sampleid)
sample_groups <- split(sample_list, ceiling(seq_along(sample_list) / 12))  # Split into groups of 12

# Generate a separate plot for each group
plots <- map(sample_groups, function(group_samples) {
  ggplot(filter(qdata, sampleid %in% group_samples),
         aes(x = sampleid, y = average_qscore, fill = sampleid)) +
    geom_violin(scale = "width", alpha = 0.8) +
    theme_fivethirtyeight(base_size = 8) +
    labs(x = "Sample", y = "Average Q-score") +
    theme(legend.position = "none", axis.text.x = element_text(angle = 45, hjust = 1))  # Rotate labels for readability
})

# Combine all plots into a single vertical layout
final_plot <- wrap_plots(plots, ncol = 1)

# Show the plot
print(final_plot)
```



ONT, however, calculates estimated cumulative error rate instead and converts value to average quality per read.

```
#Define paths
source_dir <- "raw/"
# A function to read and calculate average read quality
extract_nano_qscores <- function(file) {
    # Read fastq
    fq <- readFastq(file)
    # Extract quality values to a matrix
    qmat <- as(quality(fq), "matrix")
    # Convert phred values to error probabilities
    error_probs <- 10^(-qmat/10)
    # Compute number of expected errors per read
    total_errors <- rowSums(error_probs, na.rm = T)
    # Length per read
    read_lenghts <- rowSums(!is.na(qmat))
    # Compute ONT-style Q-score
    ont_qscores <- -10*log10(total_errors/read_lenghts)
    # Trim sample name
    sampleid <- sub("_trimmed\\.fastq\\.gz$", "", basename(file))

    # Create a dataframe
    qdata <- tibble(
        nanopore_qscore = ont_qscores,
        sampleid = sampleid)
    return(qdata)
```

```r
}

# Create list of file names
files <- list.files(source_dir, pattern = "\\.fastq\\.gz$", full.names = T)
# Combined data frame by using map_dfr and applying function
qscore_data <- map_dfr(files, extract_nano_qscores)
# Save results into a rds file
saveRDS(qscore_data, "qiime2/nano_quality.rds")
```

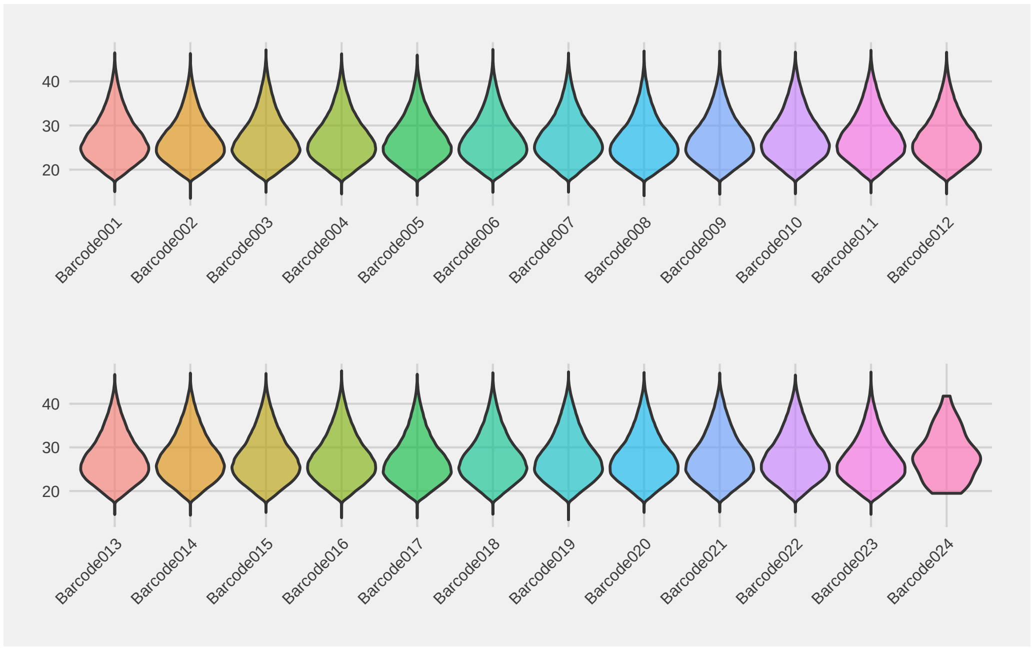
```r
qdata <- readRDS("qiime2/nano_quality.rds")

# Determine the number of sample batches (each batch has max 12 samples)
sample_list <- unique(qdata$sampleid)
sample_groups <- split(sample_list, ceiling(seq_along(sample_list) / 12))  # Split into groups of 12

# Generate a separate plot for each group
plots <- map(sample_groups, function(group_samples) {
  ggplot(filter(qdata, sampleid %in% group_samples),
         aes(x = sampleid, y = nanopore_qscore, fill = sampleid)) +
    geom_violin(scale = "width", alpha = 0.6) +
    theme_fivethirtyeight(base_size=8) +
    labs(x = "Sample", y = "Nanopore Q-score") +
    theme(legend.position = "none", axis.text.x = element_text(angle = 45, hjust = 1))  # Rotate labels for readability
})

# Combine all plots into a single vertical layout
final_plot <- wrap_plots(plots, ncol = 1)

# Show the plot
print(final_plot)
```



ITSxpress (standalone)

`Itsxpress` is used to trim ITS region. The code extracts full ITS region. Cutadapt is used to remove short sequences.

```
#!/bin/bash
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"
source activate base

# Define the input and output directories
input_dir="/homedir04/msuokas/r_projects/env_fungi/raw"
output_dir="/homedir04/msuokas/r_projects/env_fungi/xpress"

# Create output subdirectories if they don't exist
mkdir -p "$output_dir"

# Loop over each file in the input directory
for file in "$input_dir"/*
do
    # Get the base name of the file (without path and extension)
    base_name=$(basename "$file" .fastq.gz)
    # Process with all regions option
    itsxpress --single_end --fastq "$file" --threads 20 \
    --region ALL --outfile "$output_dir/${base_name}_full.fastq.gz"

    cutadapt -m 50 -o "$output_dir/${base_name}_full_trimmed.fastq.gz" \
    "$output_dir/all/${base_name}_all.fastq.gz"
done

echo "Processing complete! Check the output directory for results."
```

Itsxpress sometimes leaves fastq sequence headers without DNA sequence. Such files will cause error when using vsearch to dereplicate sequences. We use support script to fix problematic files.

```
bash fix_fastq.sh /homedir04/msuokas/r_projects/env_fungi/xpress/
```

Process extracted ITS region sequences

Import reads to qiime2

```
# Activate conda environment
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"
source activate qiime-2024.10
# Run command
qiime tools import --type 'SampleData[SequencesWithQuality]' \
--input-path xpress/manifest.csv --output-path qiime2/full_its.qza \
--input-format SingleEndFastqManifestPhred33
```

Dereplicate sequences

```
# Activate conda environment
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"
source activate qiime-2024.10

# Dereplicate
qiime vsearch dereplicate-sequences \
--i-sequences qiime2/full_its.qza \
--o-dereplicated-table qiime2/derep_table.qza \
--o-dereplicated-sequences qiime2/derep_seqs.qza \
--verbose
```

Pick denovo otus at 97 % identity level using `vsearch` plugin

```
# Activate conda environment
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"
source activate qiime-2024.10

# Pick otus
qiime vsearch cluster-features-de-novo \
--i-sequences qiime2/derep_seqs.qza \
--i-table qiime2/derep_table.qza \
--p-perc-identity 0.97 --p-strand plus \
--p-threads 20 --output-dir qiime2/denovo
```

Filter rare features

```
# Activate conda environment
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"
source activate qiime-2024.10

# QIIME 2 command to filter rare features
qiime feature-table filter-features \
--i-table qiime2/denovo/clustered_table.qza \
--p-min-frequency 10 \
--o-filtered-table qiime2/denovo/f10_table.qza

qiime feature-table filter-seqs \
--i-data qiime2/denovo/clustered_sequences.qza \
--i-table qiime2/denovo/f10_table.qza \
--o-filtered-data qiime2/denovo/f10_sequences.qza
```

Identify chimeric sequences

```
# Activate conda environment
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"
source activate qiime-2024.10

# Detect chimeras
qiime vsearch uchime-denovo \
--i-sequences qiime2/denovo/f10_sequences.qza \
--i-table qiime2/denovo/f10_table.qza \
--o-chimeras qiime2/denovo/chimeras.qza \
--o-stats qiime2/denovo/uchime-stats.qza \
--o-nonchimeras qiime2/denovo/nonchimeras.qza
```

Filter chimeric features from the table

```
# Activate conda environment
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"
source activate qiime-2024.10

# Keep nonchimeric features in the table
qiime feature-table filter-features \
--i-table qiime2/denovo/f10_table.qza \
--m-metadata-file qiime2/denovo/nonchimeras.qza \
--o-filtered-table qiime2/otu_table.qza
```

Filter chimeric sequences

```
# Activate conda environment
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"
source activate qiime-2024.10

# Keep nonchimeric sequences
qiime feature-table filter-seqs \
--i-data qiime2/denovo/f10_sequences.qza \
--i-table qiime2/otu_table.qza \
--o-filtered-data qiime2/otu_sequences.qza
```

Import feature table and metadata to TSE

```r
# Import feature table and sort sample names alphabetically
tse <- importQIIME2(featureTableFile = "qiime2/otu_table.qza")
tse <- tse[, sort(colnames(tse))]

# Import metadata file and add data to colData
metadata <- data.frame(read_tsv("qiime2/meta.tsv",
show_col_types = F))
metadata <- column_to_rownames(metadata, "Sampleid")
colData(tse) <- DataFrame(metadata)

# Check dimensions
tse
```

```
class: TreeSummarizedExperiment
dim: 8292 24
metadata(0):
assays(1): counts
rownames(8292): a5010f85c93eb041d28b76b98645d95d6dfaa7db
  3d4bab7221d32b82095a0ebc4cc8e45909aa1e21 ...
  40ef4e0761d396974af1ce8c18fdd6fc9aa7c559
  1dc8b1f77b4dd8807445d704b3deceab60a33885
rowData names(0):
colnames(24): Barcode001 Barcode002 ... Barcode023 Barcode024
colData names(5): Labc Area Veg Soil Type
reducedDimNames(0):
mainExpName: NULL
altExpNames(0):
rowLinks: NULL
rowTree: NULL
colLinks: NULL
colTree: NULL
```

Import sequences and rearrange them to correspond assay table

```r
# Import squence file
ref_sequences <- importQZA("qiime2/otu_sequences.qza")
ref_ids <- names(ref_sequences)
tse_ids <- rownames(tse)
# Check if all rownames are present in the reference IDs
if (!all(tse_ids %in% ref_ids)) {
stop("Not all rownames from tse are present in the reference sequences.")
}
# Reorder `ref_sequences` to match the order of `tse` rownames
ref_sequences_ordered <- ref_sequences[match(tse_ids, ref_ids)]
all(names(ref_sequences_ordered) == rownames(tse))
```

```
[1] TRUE
```

```r
# Included ordered sequences to data object
referenceSeq(tse) <- ref_sequences_ordered
```

Classify taxonomy. This step is computionally heavy.

```r
# Unite reference file
unite <- "/homedir04/msuokas/reference/sh_general_release_dynamic_s_04.04.2024.fasta"
# Assign taxonomy using dada2 assignTaxonomy function
taxa <- assignTaxonomy(referenceSeq(tse),unite, minBoot=60, multithread=10)
# Save result to rds file
saveRDS(taxa, "qiime2/taxonomy.rds")
```

Process and include taxonomic results

```
# Read results
taxa <- data.frame(readRDS("qiime2/taxonomy.rds"))
# Remove taxa prefixes from each column
taxa <- as.data.frame(lapply(taxa, function(x) sub("^[a-z]__","", x)))
#Add taxonomy to rowData
rownames(taxa) <- NULL
rowData(tse) <- DataFrame(taxa)
# Rename rows
rownames(tse) <- paste0("OTU_", seq_len(nrow(tse)))
```

Prune non-fungal taxa

```
# Non-fungal taxa
tse <- tse[rowData(tse)$Kingdom %in% "Fungi",]
# Final dimensions
tse
```

```
class: TreeSummarizedExperiment
dim: 8292 24
metadata(0):
assays(1): counts
rownames(8292): OTU_1 OTU_2 ... OTU_8291 OTU_8292
rowData names(7): Kingdom Phylum ... Genus Species
colnames(24): Barcode001 Barcode002 ... Barcode023 Barcode024
colData names(5): Labc Area Veg Soil Type
reducedDimNames(0):
mainExpName: NULL
altExpNames(0):
rowLinks: NULL
rowTree: NULL
colLinks: NULL
colTree: NULL
referenceSeq: a DNAStringSet (8292 sequences)
```

Write results to files

RDS

```
saveRDS(tse, "qiime2/tse.rds")
```

Abundance table

```
#FeatureID will be rowname
abd <- data.frame(FeatureID = rownames(tse),assays(tse)$counts)
#Write
write_tsv(abd, "results/feature_table.tsv")
```

Taxonomy table

```
#FeatureID will be rowname
taxt <- data.frame(FeatureID = rownames(tse), rowData(tse))
#Write
write_tsv(taxt, "results/taxonomy.tsv")
```

Feature sequences

```
# Write fasta file
writeXStringSet(referenceSeq(tse), "results/repseq.fasta",
append = F, compress = F,
format = "fasta")
```

Metadata file

```
metadf <- data.frame(colData(tse)) %>% rownames_to_column(var="Sampleid")
#write
write_tsv(metadf, "results/metadata.tsv")
```

Summary table

```
# Create data frame with counts information
summary_df <- data.frame(Samples = colData(tse)$Labc,
                         Reads = colSums(assay(tse, "counts")))
rownames(summary_df) <- NULL
kable(summary_df, caption = "ITS sequence summary") %>%
kable_styling(latex_options = c("HOLD_position", "striped", "scale_down"),
font_size = 11) %>% row_spec(0, background = "teal",
color = "white")
```

Table 1: ITS sequence summary

Samples	Reads
R08	198378
R16	232812
R07	252158
R37	132975
R33	252766
R13	271416
M06	190069
R38	240816
R05	319647
R34	217338
R32	210455
R04	197552
M04	261328
M07	258038
M01	242923
R21	260505
R18	202929
R40	188609
R42	279398
M05	223169
M02	261025
R31	222414
R23	266089
negative	26

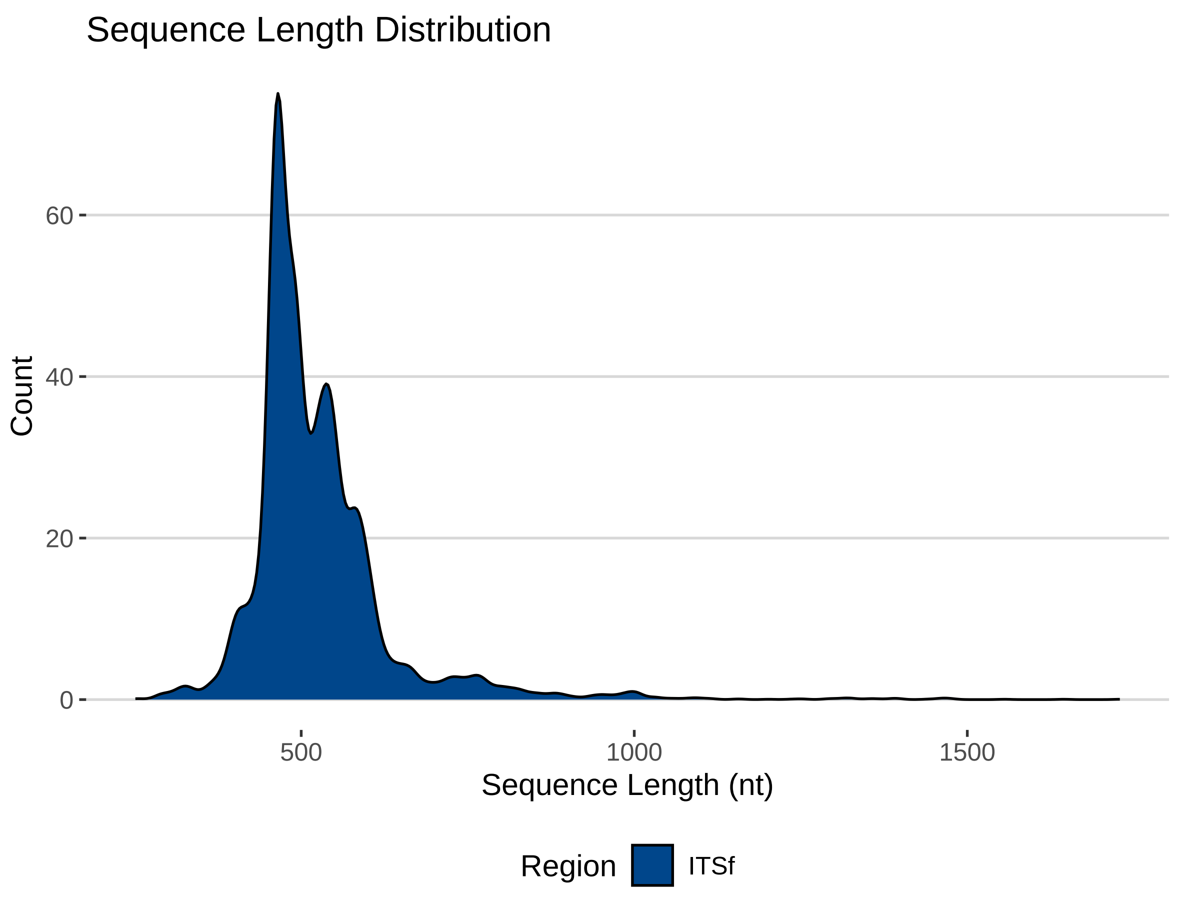
Sequence length distribution

```
# Get sequences
seqset <- referenceSeq(tse)

# Calculate sequence lengths
sequence_lengths <- nchar(seqset)

# Combine lengths into a single data frame in long format
its_df <- data.frame(
  Length = sequence_lengths,
  Region = factor(rep("ITSf", length(sequence_lengths))))

# Plot the density distribution
ggplot(its_df, aes(x = Length, fill = Region)) +
  geom_density(aes(y = ..count..), alpha = 1) +
  labs(title = "Sequence Length Distribution",
       x = "Sequence Length (nt)", y = "Count") +
  theme_hc() +
  scale_fill_lancet()
```


Sequence Length Distribution
Count
Sequence Length (nt)
Region
ITSf