TSSinference Manual

Jinhong Dong (Guertin lab, UConn Health)

Table of contents

(note this document was writting in Quarto format [.qmd], but as a text file should be readable on all major editors if its extension is just .md)

1 FASTQ to bigWig

This follows the PRO-seq QC pipeline in Nascent RNA Methods.

Begin with paired-end FASTQs. PE2 reads from PROseq will be the inputs for finding TSSs.

> **Note**
>
> The code blocks have been formatted to be human readable, so you do not have to copy it as it is here. If something breaks, make sure to check if lines are breaking where you don't want to be!

1.1 Set up: tmp memory, seqOutBias, genome annotations

These setup steps generally only need to be done once, except if you process FASTQs with a different read size (see seqOutBias section).

tmp memory

To avoid tmp memory issues, do the following:

```
# create new tmp directory
mkdir /scratch/yourname/tmp

# then in every sbatch script include this line before any of your
# other commands:
export TMPDIR=/scratch/yourname/tmp
```

seqOutBias

Run seqOutBias seqtable on its own prior to working on your samples. This takes a while and must be done in order to allow parallel instances of seqOutBias. This must be done for every unique read size, which varies depending on the type of sequencing.

General note about submitting sbatch scripts: unless you specify `cd`-ing to a specific directory, the directory that you submitted the script in will become the working directory.

```
#! /usr/bin/bash

#SBATCH --job-name=mappability47
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 32
#SBATCH -p general
#SBATCH --qos=general
#SBATCH --mem=56G
#SBATCH --mail-type=ALL
#SBATCH --mail-user=jdong@uchc.edu
#SBATCH -o /home/FCAM/jdong/slurm_out/%x_%j.all_out
#SBATCH -e /home/FCAM/jdong/slurm_out/%x_%j.all_out

# set new tmp directory
export TMPDIR=/scratch/jdong/tmp

echo "Current wd:" $(pwd)
echo "Current node:" $(hostname)
```

```
module load genometools/1.5.10
module load ucsc_genome/2012.05.22
module load rust

seqOutBias=/home/FCAM/jdong/software/seqOutBias #version 1.4.0
genome=/home/FCAM/jdong/human38/hg38.fa
read_size=47

seqOutBias seqtable $genome --read-size=${read_size}
```

Note which directory you are saving your seqOutBias outputs in, because in the main processing script you need to reference the tallymer and table files created in this step:

```
tallymer=/home/FCAM/jdong/ZNF143_PRO_2023/hg38.tal_${read_size}.gtTxt.gz
table=/home/FCAM/jdong/ZNF143_PRO_2023/hg38_${read_size}.4.2.2.tbl
```

Genome Annotations

Set up genome annotations. If you are doing this manually, initialize an interactive node on Xanadu:

```
srun --partition=general --qos=general --mem=24G -N 1 -n 1 -c 3 --pty bash
```

Specifying intron/exon annotations separately is important for several of the quality control measures that will be generated in the main processing script.

The main thing that might change in the future is if you want to use a different release number of the reference gene annotation.

```
release=109
wget http://ftp.ensembl.org/pub/release-${release}/gtf/homo_sapiens/Homo_sapiens.GRCh38.${
gunzip Homo_sapiens.GRCh38.${release}.chr.gtf.gz

#extract all exon 1 annotations
grep 'exon_number "1"' Homo_sapiens.GRCh38.${release}.chr.gtf | \
    sed 's/^/chr/' | \
    awk '{OFS="\t";} {print $1,$4,$5,$14,$20,$7}' | \
    sed 's/";//g' | \
    sed 's/"//g' | sed 's/chrMT/chrM/g' | \
    sort -k1,1 -k2,2n > Homo_sapiens.GRCh38.${release}.tss.bed

#extract all exons
```

```
grep 'exon_number' Homo_sapiens.GRCh38.${release}.chr.gtf | \
    sed 's/^/chr/' | \
    awk '{OFS="\t";} {print $1,$4,$5,$14,$20,$7}' | \
    sed 's/";//g' | \
    sed 's/"//g' | sed 's/chrMT/chrM/g' | \
    sort -k1,1 -k2,2n > Homo_sapiens.GRCh38.${release}.all.exons.bed

#extract all complete gene annotations, sorted for use with join
awk '$3 == "gene"' Homo_sapiens.GRCh38.${release}.chr.gtf | \
    sed 's/^/chr/' | \
    awk '{OFS="\t";} {print $1,$4,$5,$10,$14,$7}' | \
    sed 's/";//g' | \
    sed 's/"//g' | sed 's/chrMT/chrM/g' | \
    sort -k5,5 > Homo_sapiens.GRCh38.${release}.bed

#extract all complete gene annotations, sorted for use with bedtools map
awk '$3 == "gene"' Homo_sapiens.GRCh38.${release}.chr.gtf | \
    sed 's/^/chr/' | \
    awk '{OFS="\t";} {print $1,$4,$5,$10,$14,$7}' | \
    sed 's/";//g' | \
    sed 's/"//g' | sed 's/chrMT/chrM/g' | \
    sort -k1,1 -k2,2n > Homo_sapiens.GRCh38.${release}_sorted.bed

module load bedtools

#merge exon intervals that overlap each other
mergeBed -s -c 6 -o distinct \
    -i Homo_sapiens.GRCh38.${release}.all.exons.bed | \
    awk '{OFS="\t";} {print $1,$2,$3,$4,$2,$4}' | \
    sort -k1,1 -k2,2n > Homo_sapiens.GRCh38.${release}.all.exons.merged.bed

#remove all first exons (so pause region is excluded from
#exon / intron density ratio)
subtractBed -s -a Homo_sapiens.GRCh38.${release}.all.exons.merged.bed \
    -b Homo_sapiens.GRCh38.${release}.tss.bed | \
    sort -k1,1 -k2,2n > Homo_sapiens.GRCh38.${release}.no.first.exons.bed

#extract gene names of exons
intersectBed -s -wb -a Homo_sapiens.GRCh38.${release}.no.first.exons.bed \
    -b Homo_sapiens.GRCh38.${release}.bed | \
    awk '{OFS="\t";} {print $1,$2,$3,$11,$4,$4}' | \
```

```
    sort -k1,1 -k2,2n \
    > Homo_sapiens.GRCh38.${release}.no.first.exons.named.bed

#extract the pause region from the first exons
#aka positions 20 - 120 downstream of the TSS
awk  '{OFS="\t";} $6 == "+" {print $1,$2+20,$2 + 120,$4,$5,$6} \
    $6 == "-" {print $1,$3 - 120,$3 - 20,$4,$5,$6}' \
    Homo_sapiens.GRCh38.${release}.tss.bed  | \
    sort -k1,1 -k2,2n > Homo_sapiens.GRCh38.${release}.pause.bed

#define and name all introns
subtractBed -s -a Homo_sapiens.GRCh38.${release}.bed \
    -b Homo_sapiens.GRCh38.${release}.all.exons.merged.bed | \
    sort -k1,1 -k2,2n > Homo_sapiens.GRCh38.${release}.introns.bed
```

1.2 Confirm UMI length

This is important for removing adapter steps later. Even if you know this number from your experimental protocol it doesn't hurt to confirm it in your data. Before doing any tests, subset some reads from any one of your FASTQs so these tests don't take forever to run. Make sure you are subsetting from a PE1 file!

```
# grab 4 million reads
zcat HEK_CloneZD29_30min_control_rep4_PE1_final.fastq.gz | \
    head -n 40000000 > Z143_UMI_test.fastq

# do this in an interactive node
srun --partition=general --qos=general --mem=36G -N 1 -n 1 -c 12 --pty bash
```

Method 1: Interpreting cutadapt output

```
module load cutadapt

# i think the adapter sequences are the same as last time
# cutadapt arguments:
# -m : minimum length
# -O : [capital letter o] minimum overlap length between
#       adapter and read
# -a : adapter sequence ligated to 3' end, or of the
#       first read in paired data
```

```
# -o : [lowercase o] name of output file, which is a
#       FASTQ with trimmed reads
cutadapt --cores=12 -m 1 -O 1 \
    -a TGGAATTCTCGGGTGCCAAGG Z143_UMI_test.fastq \
    -o Z143_UMI_test_noadapt.fastq > Z143_UMI_test_cutadapt.txt

tail Z143_UMI_test_cutadapt.txt
# 38  32011 0.0 2 28256 3368 387
# 39  1976282 0.0 2 1842220 117943 16119
# 40  81301 0.0 2 67815 12434 1052
# 41  56103 0.0 2 50968 3886 1249
# 42  45902 0.0 2 42491 2676 735
# 43  41906 0.0 2 38847 2339 720
# 44  38556 0.0 2 35661 2301 594
# 45  37035 0.0 2 34420 2048 567
# 46  43034 0.0 2 40439 2117 478
# 47  45296 0.0 2 41996 2565 735
```

Counting how many sequences follow the spike in counts (at length 39): UMI length = 8.

Method 2: Trim and Align

Another method to confirm UMI length is to trim the PE1 reads from the 5' end, one base at a time, and see when a spike in alignment rate to the human genome takes place.

```
module load fastx
cat Z143_UMI_test.fastq | fastx_clipper -Q 33 -a TGGAATTCTCGGGTGCCAAGG \
    -o Z143_UMI_test_clipped.fastq

# Trim by 1 base at a time
for n in {1..9}
do
fastx_trimmer -f $(expr $n + 1) -i Z143_UMI_test_clipped.fastq \
    -o Z143_UMI_test_clipped_trim$n.fastq
done

# -f means the first base to KEEP, so -f 2 will trim the first base
# (from the left/5' side)
head -2 Z143_UMI_test_clipped.fastq
# @NB551647:99:HVY2FBGXN:1:11101:24868:1042 1:N:0:ACTTGA
# ACATAGTACTTTTAAAT
head -2 Z143_UMI_test_clipped_trim1.fastq
# @NB551647:99:HVY2FBGXN:1:11101:24868:1042 1:N:0:ACTTGA
```

```
#  CATAGTACTTTTAAAT
head -2 Z143_UMI_test_clipped_trim2.fastq
# @NB551647:99:HVY2FBGXN:1:11101:24868:1042 1:N:0:ACTTGA
#   ATAGTACTTTTAAAT

# test alignments by aligning in parallel!
```

"Z143_UMI_alignments.sh"

```
#! /usr/bin/bash
#SBATCH --job-name=ZNF143_UMI_alignments
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 24
#SBATCH -p general
#SBATCH --qos=general
#SBATCH --mem=36G
#SBATCH --mail-type=ALL
#SBATCH --mail-user=jdong@uchc.edu
#SBATCH -o /home/FCAM/jdong/slurm_out/%x_%j.all_out
#SBATCH -e /home/FCAM/jdong/slurm_out/%x_%j.all_out

export TMPDIR=/scratch/jdong/tmp
module load bowtie2
genome_index=/home/FCAM/jdong/human38/hg38
directory=/labs/Guertin/ZNF143_PRO
cd $directory

# simple alignment with bowtie2
for x in Z143_UMI_test_clipped*.fastq
    do
    name=$(echo $x | awk -F ".fastq" '{print $1}')
    echo "Now aligning $name"
    bowtie2 -p 24 -t -x ${genome_index} -U ${x} -S ${name}.sam
    done 2> Z143_UMI_test_clipped_stderr.txt
```

```
grep -i "alignment rate" Z143_UMI_test_clipped_stderr.txt
# 65.72% overall alignment rate # 0 bases trimmed
# 74.20% overall alignment rate # 1
# 78.63% overall alignment rate # 2
# 82.87% overall alignment rate # 3
```

```
# 84.89% overall alignment rate # 4
# 86.72% overall alignment rate # 5
# 89.22% overall alignment rate # 6
# 74.36% overall alignment rate # 7
# 95.84% overall alignment rate # 8 -- spike
# 96.61% overall alignment rate # 9
```

Note the large spike in alignment rate when 8 bases are trimmed–this is another point in favor of the UMI length = 8.

1.3 Process FASTQs and get quality control metrics

The main processing script (on Xanadu: `/labs/Guertin/ZNF143_PRO/pro_processing.sh`) can be used as a template for creating individual scripts for each of your FASTQ files, which allows you to submit all of your samples at once to the cluster. This saves a LOT of time compared to running them sequentially.

Details are in Nascent RNA Methods, but briefly, for each sample:

- Remove adapters with cutadapt PE1, PE2
- Calculate PE1 total and PE1 with adapter
  - Begin adding to QC_metrics.txt for this particular sample
- seqtk removes reads < 10 bases from PE1
- fqdedup to deduplicate the PE1 reads and match (fastq_pair) with their PE2 reads to get dedup'd versions of both
- calculate frequency of insert sizes to determine amount of RNA degradation
- trim UMI (seqtk)
  - NOTE!! THIS IS WHERE BOTH PE1 AND PE2 ARE REVERSE-COMPLEMENTED
- align (bowtie2) to rDNA and only keep reads that don't align (fastq_pair to get corresponding PE2)
  - not sure how this works, but -t adjust the hash table for fastq_pair to be the size of the number of reads that don't align to rDNA
- align filtered reads to human genome (include –rf flag bc we reverse-comped all the reads)
  - well really align -> samtools turn into bam -> sort by leftmost coordinate
  - fastq_pair adds ".paired.fq" to the end of your inputs
- calculate alignment rate to rDNA and to the rest of the human genome (literally counting reads and dividing numbers)
- plot and calculate complexity (fqComplexity)

  - What is this doing? Basically telling you how many unique reads you have, which will determine how much read depth is recommended to reach a desired number of "concordant aligned reads". hmm. For ex, at read depth 10 million, 75% uniqueness is recommended.
  - The lower the read depth, the more unique the reads, and the higher the read depth, the more repeats you might see, approaching the "true rate" of PCR duplication for that particular library.

- seqOutBias! make beds and bigWigs separated by strand
- calculate pause indices. Pause index too low means low nuclear run-on efficiency, which suggests library problems
- calculate intron/exon ratios as a guess of nascent RNA purity. If there are lots of nascent RNA, exons and introns should be present at about the same ratios per gene

"pro_processing.sh":

```
#! /usr/bin/bash

#SBATCH --job-name=pro_processing_XXXXXXX.sh     # name for job
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 24
#SBATCH -p general
#SBATCH --qos=general
#SBATCH --mem=36G
#SBATCH --mail-type=ALL
#SBATCH --mail-user=jdong@uchc.edu
#SBATCH -o /home/FCAM/jdong/slurm_out/%x_%j.all_out
#SBATCH -e /home/FCAM/jdong/slurm_out/%x_%j.all_out

cd /labs/Guertin/ZNF143_PRO

echo "Current wd:" $(pwd)
echo "Current node:" $(hostname)

name=XXXXXXX

module load cutadapt/3.5
module load seqtk/1.3
seqOutBias=/home/FCAM/jdong/software/seqOutBias #version 1.4.0
fqdedup=/home/FCAM/jdong/software/fqdedup
flash=/home/FCAM/jdong/software/flash
module load fastq-pair/1.0
```

```
module load samtools/1.16.1
module load genometools/1.5.10
module load ucsc_genome/2012.05.22
module load rust
module load bowtie2
module load bedtools

sizes=/home/FCAM/jdong/human38/hg38.chrom.sizes
annotation_prefix=/home/FCAM/jdong/ZNF143_PRO_2023/Homo_sapiens.GRCh38.109
UMI_length=8
read_size=47
cores=24
genome=/home/FCAM/jdong/human38/hg38.fa
genome_index=/home/FCAM/jdong/human38/hg38
prealign_rdna_index=/home/FCAM/jdong/hum_rDNA/human_rDNA
tallymer=/home/FCAM/jdong/ZNF143_PRO_2023/hg38.tal_${read_size}.gtTxt.gz
table=/home/FCAM/jdong/ZNF143_PRO_2023/hg38_${read_size}.4.2.2.tbl

gunzip ${name}_PE1_final.fastq.gz
gunzip ${name}_PE2_final.fastq.gz

echo 'removing dual adapter ligations and calculating the fraction of'
echo 'adapter/adapters in' $name
cutadapt --cores=$cores -m $((UMI_length+2)) -O 1 \
    -a TGGAATTCTCGGGTGCCAAGG ${name}_PE1_final.fastq \
    -o ${name}_PE1_noadap.fastq \
    --too-short-output ${name}_PE1_short.fastq > ${name}_PE1_cutadapt.txt
cutadapt --cores=$cores -m $((UMI_length+10)) -O 1 \
    -a GATCGTCGGACTGTAGAACTCTGAAC ${name}_PE2_final.fastq \
    -o ${name}_PE2_noadap.fastq \
    --too-short-output ${name}_PE2_short.fastq > ${name}_PE2_cutadapt.txt

PE1_total=$(wc -l ${name}_PE1_final.fastq | awk '{print $1/4}')
PE1_w_Adapter=$(wc -l ${name}_PE1_short.fastq | awk '{print $1/4}')
AAligation=$(echo "scale=2 ; $PE1_w_Adapter / $PE1_total" | bc)
echo -e  "value\texperiment\tthreshold\tmetric" > ${name}_QC_metrics.txt
echo -e "$AAligation\t$name\t0.80\tAdapter/Adapter" >> ${name}_QC_metrics.txt

echo 'removing short RNA insertions in' $name
seqtk seq -L $((UMI_length+10)) ${name}_PE1_noadap.fastq \
    > ${name}_PE1_noadap_trimmed.fastq
```

```
echo 'removing PCR duplicates from' $name
fqdedup -i ${name}_PE1_noadap_trimmed.fastq -o ${name}_PE1_dedup.fastq

PE1_noAdapter=$(wc -l ${name}_PE1_dedup.fastq | awk '{print $1/4}')

fastq_pair -t $PE1_noAdapter ${name}_PE1_dedup.fastq ${name}_PE2_noadap.fastq

echo 'calculating and plotting RNA insert sizes from' $name
flash -q --compress-prog=gzip --suffix=gz ${name}_PE1_dedup.fastq.paired.fq \
        ${name}_PE2_noadap.fastq.paired.fq -o ${name}

insert_size.R ${name}.hist ${UMI_length}

echo 'trimming off the UMI from' $name
seqtk trimfq -b ${UMI_length} ${name}_PE1_dedup.fastq | \
    seqtk seq -r - > ${name}_PE1_processed.fastq
seqtk trimfq -e ${UMI_length} ${name}_PE2_noadap.fastq | \
    seqtk seq -r - > ${name}_PE2_processed.fastq

echo 'aligning' $name 'to rDNA'
bowtie2 -p $((cores-2)) -x $prealign_rdna_index \
    -U ${name}_PE1_processed.fastq 2>${name}_bowtie2_rDNA.log | \
    samtools sort -n - | \
    samtools fastq -f 0x4 - > ${name}_PE1.rDNA.fastq
reads=$(wc -l ${name}_PE1.rDNA.fastq | awk '{print $1/4}')
fastq_pair -t $reads ${name}_PE1.rDNA.fastq ${name}_PE2_processed.fastq

echo 'aligning' $name 'to the genome'
bowtie2 -p $((cores-2)) --maxins 1000 -x $genome_index --rf \
    -1 ${name}_PE1.rDNA.fastq.paired.fq \
    -2 ${name}_PE2_processed.fastq.paired.fq 2>${name}_bowtie2.log | \
    samtools view -b - | samtools sort - -o ${name}.bam

echo 'calculating rDNA alignment rate for' $name
PE1_prior_rDNA=$(wc -l ${name}_PE1_processed.fastq | awk '{print $1/4}')
PE1_post_rDNA=$(wc -l ${name}_PE1.rDNA.fastq | awk '{print $1/4}')
total_rDNA=$(echo "$(($PE1_prior_rDNA-$PE1_post_rDNA))")
concordant_pe1=$(samtools view -c -f 0x42 ${name}.bam)
total=$(echo "$(($concordant_pe1+$total_rDNA))")
rDNA_alignment=$(echo "scale=2 ; $total_rDNA / $total" | bc)
echo -e "$rDNA_alignment\t$name\t0.10\trDNA Alignment Rate" \
```

```
    >> ${name}_QC_metrics.txt

echo 'calculating alignment rate for' $name
map_pe1=$(samtools view -c -f 0x42 ${name}.bam)
pre_alignment=$(wc -l ${name}_PE1.rDNA.fastq.paired.fq | awk '{print $1/4}')
alignment_rate=$(echo "scale=2 ; $map_pe1 / $pre_alignment" | bc)
echo -e "$alignment_rate\t$name\t0.80\tAlignment Rate" \
    >> ${name}_QC_metrics.txt

echo 'plotting and calculating complexity for' $name
fqComplexity -i ${name}_PE1_noadap_trimmed.fastq

echo 'calculating and plotting theoretical sequencing depth'
echo 'to achieve a defined number of concordantly aligned reads for' $name
PE1_total=$(wc -l ${name}_PE1_final.fastq | awk '{print $1/4}')
PE1_noadap_trimmed=$(wc -l ${name}_PE1_noadap_trimmed.fastq | \
    awk '{print $1/4}')
factorX=$(echo "scale=2 ; $PE1_noadap_trimmed / $PE1_total" | bc)
echo 'fraction of reads that are not adapter/adapter ligation products'
echo 'or below 10 base inserts:'
echo $factorX
PE1_dedup=$(wc -l ${name}_PE1_dedup.fastq | awk '{print $1/4}')
factorY=$(echo "scale=2 ; $concordant_pe1 / $PE1_dedup" | bc)
fqComplexity -i ${name}_PE1_noadap_trimmed.fastq -x $factorX -y $factorY

echo 'Separating paired end reads and creating genomic BED and'
echo 'bigWig intensity files for' $name
seqOutBias scale $table ${name}.bam --no-scale \
    --stranded --bed-stranded-positive \
    --bw=$name.bigWig --bed=$name.bed --out-split-pairends --only-paired \
    --tail-edge --read-size=$read_size --tallymer=$tallymer

grep -v "random" ${name}_not_scaled_PE1.bed | grep -v "chrUn" | \
    grep -v "chrEBV" | sort -k1,1 -k2,2n > ${name}_tmp.txt
mv ${name}_tmp.txt ${name}_not_scaled_PE1.bed

echo 'calculating pause indices for' $name
mapBed -null "0" -s -a $annotation_prefix.pause.bed \
    -b ${name}_not_scaled_PE1.bed | \
    awk '$7>0' | sort -k5,5 -k7,7nr | sort -k5,5 -u > ${name}_pause.bed
```

```
join -1 5 -2 5 ${name}_pause.bed $annotation_prefix.bed | \
    awk '{OFS="\t";} $2==$8 && $6==$12 {print $2,$3,$4,$1,$6,$7,$9,$10}' | \
    awk '{OFS="\t";} $5 == "+" {print $1,$2+480,$8,$4,$6,$5} $5 == "-" \
        {print $1,$7,$2 - 380,$4,$6,$5}' | \
    awk  '{OFS="\t";} $3>$2 {print $1,$2,$3,$4,$5,$6}' | \
    sort -k1,1 -k2,2n  > ${name}_pause_counts_body_coordinates.bed
mapBed -null "0" -s -a ${name}_pause_counts_body_coordinates.bed \
    -b ${name}_not_scaled_PE1.bed | awk '$7>0' | \
    awk '{OFS="\t";} {print $1,$2,$3,$4,$5,$6,$7,$5/100,$7/($3 - $2)}' | \
    awk '{OFS="\t";} {print $1,$2,$3,$4,$5,$6,$7,$8,$9,$8/$9}' \
    > ${name}_pause_body.bed

pause_index.R ${name}_pause_body.bed

echo 'Calculating exon density / intron density as '
echo 'a metric for nascent RNA purity for' $name
mapBed -null "0" -s -a $annotation_prefix.introns.bed \
        -b ${name}_not_scaled_PE1.bed | awk '$7>0' | \
        awk '{OFS="\t";} {print $1,$2,$3,$5,$5,$6,$7,($3 - $2)}' \
        > ${name}_intron_counts.bed
mapBed -null "0" -s -a $annotation_prefix.no.first.exons.named.bed \
        -b ${name}_not_scaled_PE1.bed | awk '$7>0' | \
        awk '{OFS="\t";} {print $1,$2,$3,$4,$4,$6,$7,($3 - $2)}' \
        > ${name}_exon_counts.bed

exon_intron_ratio.R ${name}_exon_counts.bed ${name}_intron_counts.bed

echo 'removing intermediate files'
rm ${name}_PE1_short.fastq
rm ${name}_PE2_short.fastq
rm ${name}_PE1_noadap.fastq
rm ${name}_PE2_noadap.fastq
rm ${name}_PE1_noadap_trimmed.fastq
rm ${name}_PE1_dedup.fastq
rm ${name}_PE1_processed.fastq
rm ${name}_PE2_processed.fastq
rm ${name}_PE1_dedup.fastq.paired.fq
rm ${name}_PE2_noadap.fastq.paired.fq
rm ${name}_PE1_dedup.fastq.single.fq
rm ${name}_PE2_noadap.fastq.single.fq
rm ${name}_PE1.rDNA.fastq.paired.fq
```

```
rm ${name}_PE1.rDNA.fastq.single.fq
rm ${name}_PE2_processed.fastq.paired.fq
rm ${name}_PE2_processed.fastq.single.fq
rm ${name}.extendedFrags.fastq.gz
rm ${name}.notCombined_1.fastq.gz
rm ${name}.notCombined_2.fastq.gz

echo 'script complete'
```

After you have a copy of the template in your working directory (also containing all the FASTQs), create individualized copies for each of your samples using the following loop. `sleep 1` is there to ensure that each script is complete before moving on to the next sample.

```
file=pro_processing.sh

for i in *_PE1_final.fastq
do
    nm=$(echo $i | awk -F"/" '{print $NF}' | \
        awk -F"_PE1_final.fastq" '{print $1}')
    echo $nm
    sed -e "s/XXXXXXX/${nm}/g" "$file" > pro_processing_${nm}.sh
    sleep 1
done
```

You can check if the replacement has worked correctly before submitting all the scripts:

```
for i in *_PE1_final.fastq
do
    nm=$(echo $i | awk -F"/" '{print $NF}' | \
        awk -F"_PE1_final.fastq" '{print $1}')
    echo $nm
    sbatch pro_processing_${nm}.sh
    sleep 1
done
```

Use `squeue` to monitor the progress of your jobs.

Now you should have un-normalized bigWigs (created by seqOutBias), named in this pattern: CELL_CONDITION_REPLICATE_PLUS/MINUS_PE2.bigWig

Remember, for TSSinference we want the PE2 reads from PRO-seq.

2 Prepare the bigWigs

Dependencies:

- python3
- Guertin lab github scripts
  - PRO_normalization.sh
  - normalization_factor.R
  - normalize_bedGraph.py
- UCSC tools
  - bedGraphToBigWig
  - bigWigToBedGraph
  - bigWigMerge
- path to hg38.chrom.sizes (download from UCSC)

Once everything is installed and executable, navigate to directory containing all of your bigWigs and run this line of code:

```
PRO_normalization.sh -c /path/to/hg38.chrom.sizes
```

Outputs:

- scaled bedGraphs and bigWigs of each replicate for minus and plus strands
- merged and scaled bigWigs of each condition for minus and plus strands

2.1 Merge all plus and all minus bigWigs for CONTROL conditions only

This step merges the control bigWigs for each strand in order to get accurate TSSs for downstream steps. Adapted from lines 80-90 of PRO_normalization.sh, but with $reps = 8 because we are combining 4 bigWigs per strand.

```
reps=4
pair="_PE2"
name="HEK_CloneZD29_30min_control"
chrSizes="/Users/jinhongdong/fileRef/hg38.chrom.sizes"

plusfiles=$(ls ${name}_*rep*_plus${pair}_scaled.bigWig)
bigWigMerge $plusfiles tmpPlus${pair}.bg
minusfiles=$(ls ${name}_*rep*_minus${pair}_scaled.bigWig)
bigWigMerge -threshold=-10000000000 $minusfiles tmpMinus${pair}.bg
```

```
scaleall=$(bc <<< "scale=4 ; 1.0 / $reps")
normalize_bedGraph.py -i tmpPlus${pair}.bg -s $scaleall \
    -o ${name}_plus${pair}_scaled.bg
normalize_bedGraph.py -i tmpMinus${pair}.bg -s $scaleall \
    -o ${name}_minus${pair}_scaled.bg
sort -k1,1 -k2,2n ${name}_plus${pair}_scaled.bg \
    > ${name}_plus${pair}_scaled_sorted.bg
sort -k1,1 -k2,2n ${name}_minus${pair}_scaled.bg \
    > ${name}_minus${pair}_scaled_sorted.bg
bedGraphToBigWig ${name}_plus${pair}_scaled_sorted.bg $chrSizes \
    ${name}_plus${pair}_scaled.bigWig
bedGraphToBigWig ${name}_minus${pair}_scaled_sorted.bg $chrSizes \
    ${name}_minus${pair}_scaled.bigWig

rm ${name}_plus${pair}_scaled.bg
rm ${name}_minus${pair}_scaled.bg
rm ${name}_plus${pair}_scaled_sorted.bg
rm ${name}_minus${pair}_scaled_sorted.bg
rm tmpPlus${pair}.bg
rm tmpMinus${pair}.bg
```

3 TSSinference

3.1 Preparation

Required:

- TSSinference.R (Guertin lab github)
- R package "bigWig"
- Reference BED files of annotated 1st exons from GENCODE (filtered appropriately)

How to create the annotated 1st exon file (GTF file can be downloaded from the Gencode website):

```
grep 'transcript_type "protein_coding"' gencode.v42.basic.annotation.gtf | \
    grep -v 'tag "readthrough_transcript"' | \
    grep -v 'tag "RNA_Seq_supported_only"' | \
    grep -v 'tag "RNA_Seq_supported_partial"' | \
    awk '{if($3=="exon"){print $0}} ' | \
    grep -w "exon_number 1" | \
```

```
cut -f1,4,5,7,9 | tr ";" "\t" | \
awk '{for(i=5;i<=NF;i++){if($i~/^gene_name/){a=$(i+1)}}\
    print $1,$2,$3,a,"na",$4}' | \
tr " " "\t" | tr -d '"' | \
grep -v "\tENSG" > gencode.hg38.v42.basic.firstExon.latest.filtered.bed
```

Assuming all bigWigs, annotation files, and scripts are in the same working directory:

> **!** Important
>
> When merging the bigWigs from seqOutBias, it is possible that the PE2 files will need to be switched when inputting them as plus or minus (see variables below). The way to check this is to look at the TSS calls in the browser after running TSSinference and primaryTranscriptAnnotation and seeing if they line up with PE2 read pileups on the correct strand.

```
library(bigWig)
# source TSSinference functions
source("TSSinference.R")
# prepare annotation file
gene.exon1 = parse.bed.exon1(
    "gencode.hg38.v42.basic.firstExon.latest.filtered.bed")
# assign bigWig variables
# These two are the ones with control + experimental merged
bw.plus='HEK_CloneZD29_30min_minus_PE2_scaled.bigWig'
bw.minus='HEK_CloneZD29_30min_plus_PE2_scaled.bigWig'
```

3.2 TSSinference options

The basic function with default options is:

```
TSSinference <- function(bed, bw.plus, bw.minus, tssWin = 100,
    top.num.peaks = 20, low.limit.tss.factor = 3, denWin = 100,
    densityFilter = FALSE)
```

Arguments and default values:

- bed: gene.exon1, or the annotated first exon locations
- bw.plus: normalized bigWig for plus strand
- bw.minus: normalized bigWig for minus strand

- tssWin: Distance up/downstream of the `bed` regions in which to search for TSSs. Default = 100
- top.num.peaks: Maximum number of read peaks to find per gene. Default = 20
- low.limit.tss.factor: This will be multiplied by the lowest read count value in the bigWigs to establish the lowest threshold for read counts. Default = 3
- denWin: Distance up/downstream of each peak to use in calculating up/downstream read densities. Default = 100
- densityFilter: If TRUE, will report one predominant TSS per gene, based on downstream > upstream density. If no such peaks are present for a gene, the reported TSS will default to the most upstream position of the annotated first exon for that gene. If FALSE, all the peaks for a gene will be reported. Default = FALSE

```
# predominant TSS
potential.tss2 = TSSinference(gene.exon1, bw.plus, bw.minus,
    densityFilter=TRUE)
```

3.3 Reorganizing TSSinference output

```
# remove chrM genes, TSSs with height NA
temp.tss = potential.tss2[!is.na(potential.tss2$height),]
temp.tss2 = temp.tss[!(temp.tss$chrom == "chrM"),]

# sort by chromosome and start position
predomTSS.sort = temp.tss2[order(temp.tss2$chrom, temp.tss2$start),]

# renumber the rownames
rownames(predomTSS.sort) = seq(nrow(predomTSS.sort))
```

Other ways to run TSSinference

```
# all TSSs (default)
potential.tss2 = TSSinference(gene.exon1, bw.plus, bw.minus,
    densityFilter=FALSE)

# follow up with taking only the TSS with maximum height for each gene
temp.tss = potential.tss2[!is.na(potential.tss2$height),]
temp.tss2 = temp.tss[!(temp.tss$chrom == "chrM"),]
contrTSS = do.call(rbind, lapply( split(temp.tss2,
    as.factor(temp.tss2$gene)), function(x)
    {return(x[which.max(x$height),])})
```

```
    )

# sort by chromosome and start position
contrTSS.sort = contrTSS[order(contrTSS$chrom, contrTSS$start),]
# reassign row names to be row numbers
rownames(contrTSS.sort) = seq(nrow(contrTSS.sort))
```

Save as a BED file:

```
# need: chrom, start, end, gene, value(height), strand
write.table(potential.tss.sort[, c(1,2,3,4,7,6)],
    file = "HEK_CloneZD29_TSSinf.bed", quote = FALSE,
    sep ='\t', row.names = FALSE, col.names = FALSE)
```

3.4 Visualizing TSSinference output

Add tracklines, then upload the BED file to the UCSC genome browser along with tracks for the input bigWigs to see where TSSs are in relation to read pileups.